PPC

I N T R O D U C T I O N

The information contained in this document is made available with the understanding that it is not supported by Hewlett-Packard. The custom microprocessor used by the HP-75 was not intended to be supported with technical details made available to the user community. When you read this document you will quickly notice that there is no effort to explain to non-design team members how or what is going on. At the time of developement there was no plan to do this. While this document may stimulate more questions than it answers that situation must be accepted "as is". See the NOMAS statement stamped below. This material is being made available to the user community through PPC because we believe that information in this form is better than none at all. PPC offers this information as a service to the community. Additional copies may be ordered from PPC at the address below.

Another source for HP-75 technical information is the Computer Journal of PPC and other volumes in the 75 NOMAS series. Contact PPC for details if desired.

PPC - POB 9599
Fountain Valley, CA 92728-9599 USA

Telephone: (714) 754-6226

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

W H A T   I S   P P C ?

PPC is the Personal Programming Center Inc., a California non-profit public benefit corporation. PPC is the oldest personal computer users group - founded in June 1974. PPC is dedicated to supporting machines that meet the concept of a personal computer as being small enough and portable enough to be conveniently always with the user. The ideal machine hasn't been designed yet. PPC's major activity is to support those products that are available and provide design ideas for better products. This is accomplished by active information exchange on a personal basis.

PPC is a unique organization in that it is non-commerical. It is dedicated to the individual and his or her persuit of making the always too slow or too low capacity machine do the task desired. PPC is a world wide organization composed of over 5,000 members in 63 countries. A chapter organization is supported with 50 chapters worldwide. PPC is an educational organization that gathers and disseminates user information on the machines PPC supports. PPC publishes two regular 'Journals', one dedicated to calculators (our heritage is in the HP-65 Programmable Scientific Calculator) and the other is dedicated to computers. PPC offers a wide range of services and 'products' to its members and the community at large. PPC is a member supported organization, we do not accept advertising, and if we make a profit on certain "commerical" products it is for the purpose keeping membership costs as low as possible. Our 'products' are self generated and are intended to be sold to our members. We do not buy commerical products and resell them. This concept is fundamental to the PPC way of serving its members and the community. We are not in the Hardware or Software business. We supplement the manufacturing, academic, technical, and consumer communities - we do not compete with them. We are applications and state-of-the-art oriented. We contribute our ideas freely and we have a very carefully defined operating philosophy with regards to commerical activity. We encourage and support commerical activity, but remain non-commerical in our exchange of ideas, solutions, and problems. In this regard we are unique.

PPC is managed by a board of seven directors. The directors are Emmett Ingram, Chariman; Richard Nelson, President; Richard Schwartz, Secretary; Fernando Lopez Lopez, Treasurer; Douglas Mecham, Director; John Kennedy, Director; and Bruce Murdock, Director. PPC is an open corporation in that its policies are continuously discussed and debated. PPC is comitted to providing a true users forum for ideas, comments, and suggestions regarding any topic related to equipment, software, or standards. Within the framework of good taste and supportative viewpoints any member or member of the community may present his or her own feelings and ideas. PPC is not a consumer pressure group. All contributions must stand on their own merit in the market place of ideas. PPC members want to hear about what has been done, not what someone may do? Dreams are nice, but PPC members are interested in performance not specifications. We are a meat and potatoes group of people and are not too interested in the parsley.

A typical PPC member is a technical, multi-disciplined, independent thinking user who takes great delight in solving problems. The small, truely personal machines that PPC members use creates a special psychology between user and machine that provides one of the binding forces of the organization. Interested users may send a large self addressed envelope with three ounces of postage (or IRC's) attached to:

PPC - POB 9599
Fountain Valley, CA 92728-9599 USA

CONTENTS

CONTENTS - Continued Page

Programmers Initials - At the end of many entries are the programmers initials following a dash. The programmers are:

GKC - Gary K. Cutler
RY - Raan Young
SkA - Seth D. Alford
MJH - Mary Jo Hornberger
JA - Jack Aplin IV
RH - Robert Heckendorn
MR - Mark Rowe

*Page 140 to 177 is an italic type style.

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

ALLOCATION DOCUMENTATION

Gary K. Cutler

9:17 PM THU., 3 AUG., 1978

Program Pointer Allocation	CHAPTER 1

1.1 Overview

Definition: Pointer allocation is a process which builds a block of information (VPA entry) within the Variable Pointer Area (VPA) for each variable in the program, replaces each variable name in the program with a relative pointer to its VPA entry and replaces the line numbers in GOSUB, GOTO and IF statements with the relative addresses of the target statements.

Calculator vs. Noncalculator programs: The continued existence of calculator variables after the execution of a calculator statement, necessitates a differentiation of Pointer Allocation for these two modes.

Noncalculator program:
a VPA (variable pointer area) will be created.
all variable names will be replaced by pointers to their name form and other data in the VPA.
the PCB (Program Control Block) is given values as follows:
P.LEN - the length of the program including the PCB. If this value is nonzero this indicates that the program has been allocated.
P.PLEN - the length of the VPA
P.ELEN - the total length of the environment not including the ECB.

Calculator program:
the VPA will be augmented by what ever new variable references there are.
variables will be replaced by pointers as for a program.
The PCB and a few system parameters are modified as follows:
P.LEN - the length of the program as for an ordinary program
P.PLEN - the length of the VPA
P.ELEN - the total length of the environment not including the ECB
LSTPLN - the value of the P.PLEN before allocation (set at beginning of PTRALO if in calc; zero otherwise)

LSTELN - the value of P.ELEN before we allocate (set at the beginning of PTRALO if in calc; zero otherwise)

Should pointer allocation fail, the deallocator, DALLOC, is called to remove the VPA that was just created.

occurrence: Pointer allocation is the first procedure in a three step process. The culmination of these three steps is the execution of a program or calculator mode statement. The three steps, in succession, are Pointer Allocation, Environmental Allocation and Execution.

Handi Calls: If a basic token, created by an external ROM, requires some unique type of Pointer Allocation, such token must be assigned a primary attribute greater than or equal to 57 (octal). This attribute will generate the V.ALLO handi event at Pointer Allocation time (refer to the V.ALLO event in the HANDI call documentation for further information).

1.2 Variable Name Form

FRMNAM: FRMNAM is the routine which translates the ASCII name of a variable into an internal format. This format not only preserves the ASCII name but also provides all the necessary information, concerning the variable, for Allocation and Execution. The internal structure consists of a two byte form with bit patterns representing the variables status.

1.2.1 Internal Structure

	7	6	5	4	3	2	1	0
byte 1:	T3	T2	T1	T0	N3	N2	N1	N0
byte 2:	A1	A0	F0	L4	L3	L2	L1	L0

T3 -- 0 --> numeric
1 --> string

T2 -- 0 --> simple
1 --> array

T1,T0 -- 0 --> real
1 --> integer
2 --> short
3 not used

N3--N0 -- 0-9 --> numeric portion of ASCII name
10 --> no numeric in name (blank)

F0 -- 0 --> variable
1 --> function

A1,A0 -- 0 --> local variable
1 --> remote variable parameter passing for CALL
2 --> parameter variable

L4--L0 -- variable name rank.(A=1, B=2, C=3, etc.)

Examples:

local simple numeric variable: C1

byte 1: 0 0 0 0 0 0 0 1 01 hex value

byte 2: 0 0 0 0 0 0 1 1 03 hex value

remote string array variable: F$(i,j)

byte 1: 1 1 0 0 1 0 1 0 CA hex value

byte 2: 0 1 0 0 0 1 1 0 46 hex value

1.3 Variable Pointer Entry

In the following block descriptions, each member of an entry in the VPA (Variable Pointer Area) is 2 bytes long.

I. Numeric variable: Ex. Z9

Length: 4 bytes

Entry form: | Name | Rel Ptr |

II. String variable: Ex. S3$

Length: 6 bytes

Entry form: | Name | Max Len | Rel Ptr |

III. Array variable: Ex. V(5)

Length: 10 bytes

Entry form: | Name | Total Len | Max Row | Max Col* | Rel Ptr |

* In the case of a 1-dimensional array, the maximum column entry will be initialized to -1 (FFFF hex internally).

IV. User defined function (numeric): Ex. FNF(a,b)

Length: 6 bytes

Entry form: | Name | Rel value Ptr | Rel Exp Addr |

V. User defined function (string): Ex. FNP1$(A$)

Length: 8 bytes

Entry form: | Name | Max Len | Rel value Ptr | Rel Exp Addr |

1.4 Program Control Block

The PCB is a block of 10 bytes that contains information about an allocated file. The PCB directly precedes the first line of any lined file in memory and does not exist for unlined files (LIF1 files have no PCB). For any deallocated file, the PCB will contain all zeros. For further information regarding the PCB, refer to the Memory Management Documentation.

PCB: The PCB (Program Control Block) is structured as follows.

```
byte 0/1   P.LEN  -- length of pgm and PCB (0 if deallocated)
byte 2/3   P.PLEN -- length of VPA
byte 4/5   P.CLEN -- spare location
byte 6/7   P.ELEN -- total size of environment not including ECB
byte 8/9   P.SPAR -- spare location (for parameter passing)
```

1.5 Deallocated Vs. Allocated Program

The following program will be followed by two forms of its internal structure; Deallocated and Allocated. The important differences are:

Deallocated	Allocated
PCB set to 0	All PCB entries are valid
Variables are ASCII	Variables replaced by relative pointer
No VPA	VPA begins directly succeeding endline

```
Program:  10 A4=PI

          20 B$='hi there'

          30 M(1,1)=3*A4/2
```

Deallocated Structure (internal)

```
            P.LEN     P.PLEN    P.CLEN    P.ELEN    P.SPAR
           --------  --------  --------  --------  --------
PCB:        00   00   00   00   00   00   00   00   00   00


Line 10:    10   00   06   11 > 34   41 < C9   08   0E
                              num var A4

Line 20:    20   00   0F   13 > 20   42 < 96   08   68
                              str var B$

            69   20   74   68   65   72   65   07   0E


Line 30:    30   00   19   12 > 20   4D < 1A   01   00
                              num array M

            00   1A   01   00   00   0A   1A   03   00

            00 > 34   41 < 2A   1A   02   00   00   2F
```



```
                 num var A4
            08   0E

End line:   99   A9   02   8A   0E
```

** No VPA **

Allocated structure (internal)

```
            P.LEN     P.PLEN    P.CLEN    P.ELEN    P.SPAR
           --------  --------  --------  --------  --------
PCB:        47   00   14   00   00   00   F2   03   00   00


Line 10:    10   00   06   11 > 47   00 < C9   08   0E
                             rel ptr to
                             A4 name form in VPA

Line 20:    20   00   0F   13 > 4B   00 < 96   08   68
                           rel ptr to B$
                           name form in VPA

            69   20   74   68   65   72   65   07   0E


Line 30:    30   00   19   12 > 51   00 < 1A   01   00
                             rel ptr to M
                             name form in VPA

            00   1A   01   00   00   0A   1A   03   00

            00 > 47   00 < 2A   1A   02   00   00   2F
               rel ptr to
               A4 name form in VPA

            08   0E


Endline:    99   A9   02   8A   0E
```


** VPA **

```
             name      val ptr
var: A4     04   01 | 1E   00


             name      max len     val ptr
var: B$     8A   02 | 20   00 | 26   00


             name      tot len **  max row     max col     val ptr
var: M(,)   4A   0D | C8   83 | 0A   00 | 0A   00 | 48   00
```

** The most significant bit of the first byte of the total length denotes the OPTION BASE.

total length negative --> OPTION BASE 0 (upper bit set)

total length positive --> OPTION BASE 1 (upper bit off)

1.6 Major Routines

PTRALO: This routine allocates all variable pointers in the file named in R40.

Input: R40 - the file to be pointer allocated.

Internal: R24 - token pointer that moves through out the program

R26 - points to beginning of variable search area

R30 - points to end of variable search area. if in pgm mode for allocation this is incremented as pointers to values are stored. when the last line of code is allocated, R12 is reset to TOS and ERRSTP is set to 0.

Notes: The routine is entered by a JSB and saves and restores the DCM (binary/BCD) status. It requires input other than the global system pointers, however ALL registers beginning with R22 are considered volatile.

NXTONE: obtains;
token (R23)
token class (R36)
current line (R45/46)
length of line (R47)
conditional;
tk class >= 30 non-allocatable, loop
tk class < 30 pass control to allocation routine

XALL1: exit code for allocation if EOF
: calculates and stores; P.CLEN (length of common area) P.PLEN (length of VPA)
: on return restores registers 20-77 (EVIL)

1.7 Procedure

Procedure: Pointer allocation works from a table of token classes. Each token, having at least one attribute, has its token class determined by the routine GETNXT (token class is the octal number defined by the two least significant digits of the primary attribute). This token class keys the appropriate allocation routine for each particular token. If a token has class < 30 (octal) then that token is allocatable. Non-allocatable tokens have token class >= 30 (octal). Tokens which are created in the future and require allocation different from existing routines, should assign a primary attribute >= 57 (octal). This will generate the V.ALLO handi event. For further information on the V.ALLO handi event refer to the Handi Call Documentation. Here follows the pointer allocation table; giving, the allocation routine, the class of tokens which that routine is responsible for, and a description of the type of tokens found in that class.

POINTER ALLOCATION TABLE

Routine	Class	Token
INIROM	-1	ROM class > 56
XALL1	0	End-of-line
VALOC	1	Fetch variable
BININT	2	Integer constant
SVAL	3	Store variable
SKPCON	4	Real constant
SKPCON	5	String constant
FUNCAL	6	User function call
LINEAL	7	Jump true line
LINEAL	10	Goto, Gosub
RELJMP	11	Jump relative
DEFFN	12	User define function
DEFEND	13	User function end def
EROM	14	External ROM (obsolete)
OPTION	15	Option base
DEFEND	16	Function return
FNASN	17	Function let
SKPNXT	20	Data
DIM	21	Dim
SHORT	22	Short
INT	23	Integer
INIROM	24	Handi call
LINEAL	25	Else jump line
RELJMP	26	Else jump relative
LINEAL	27	Using line

Example: The following example illustrates the structure of a user defined function, deallocated and allocated. Because of the complexity inherent in the allocation of DEF FN statements, the routine responsible for each step in the allocation process is included. It should be noted that the parameters of a user defined function are translated into internal format during Parsing.

40 DEF FNM(A,B)=MAX(ABS(A),ABS(B))

DEALLOCATED	DESCRIPTOR	ALLOCATED	ROUTINE
40 00	line number	40 00	
87	def fn	87	
20 4D	ASCII blank M	2 byte addr of M in VPA	DEFFN
---- ----	jump past fn end	2 byte rel pos of EOL	FNEND
04	param type/count (see note 1)	04	
0A 01	internal name form A	0A 01	
---- ----	var value ptr	2 byte rel loc of A in environment	DEFFN
0A 02	internal name form B	0A 02	
---- ----	var value ptr	2 byte rel loc of B in environment	DEFFN
---- ----	rel posit of PCR	2 byte rel pos of def fn statement	DEFFN
01	fet var	01	
20 41	ASCII blank A	2 byte rel addr of A in def fn statement	VALLOC
B3	abs value	B3	
01	fet var	01	
20 42	ASCII blank B	2 byte rel addr of B in def fn statement	VALLOC
B3	abs value	B3	
AE	max	AE	
AF	inv fn end	AF	

---- ----	pos of store value ptr	2 byte rel addr of M in VPA	FNEND (FNR3)
0E	eol	0E	

NOTE 1: the param count/type is formatted as:
BIT 7-BIT 1 number of parameters in definition (X2)
BIT 0 type of function, 0=numeric, 1=string

1.8 Globals

Name	Location	Description
CSTAR	8385	nested array counter
CURFUN	8285	location of DEF FN
DFPAR1	8386	start of FN definiton
DFPAR2	8388	end of FN definition
DIMFLG	838E	type of variable being allocated
ERRSTP	8391	loc of variable that resulted in alloc error
FILEND	8381	end of file to be allocated
FWVARS	8251	beginning of environment
LSTELN	83C3	last P.LEN value
LSTPLN	83C1	last P.PLEN value
OPTBAS	8282	0 -- option base 1; 1 -- option base 0
PCR	824D	program counter
PRFILE	8243	location of file to be allocated
PRNAME	8263	name of file to be alocated
RNFILE	8247	location of current running file
RSTAR	8383	current alloc point for array parameters
TOS	8257	current top of stack (R12)
VARPTR	838F	next available loc in environment

1.9 Handi Calls

V.ALLO -- token class >= 57 (octal)

1.10 Cross References

Memory Management Document	RH"MEM
Internal Code Examples Document	RH"ICE
Handi Call Document	RH"HDI
Source File	RH&PAL
Global File	KR&GLO

Program Environment Allocation	CHAPTER 2

2.1 Definition

Definition: Environmental allocation is the second and last step in program allocation. This process has the responsibility of creating an environment (variable value area and a control block) for the program named in PRNAME.

The environmental structure for each type of variable is as follows:

Numerics:
- Real -- eight byte value field
- Short -- four byte value field
- Integer -- three byte value field
- Array -- field length is determined by the number of elements in the array times field length allotted for the type of variable.

Strings: two byte actual length field followed by the character field, whose length is defined in the Max Len field of the VPA entry.

2.2 Routines and Pointers

Pointers:

NXTMEM - the top of the environment stack. This is one byte above where the next environment will be inserted.

FWVARS - a pointer to the ECB of the current environment

Routines:

ENVALO - allocates the environment for the program in PRNAME. Calls PUSHEN to allocate room and build the ECB, then goes to INIVLP to initialize the variable value area.

PUSHEN - allocates space for the environment at NXTMEM and builds the ECB.

INIVLP - determines the nature of the variable and initializes the variable's environment appropriately (zeros the field and sets the most significant byte to -1 for numeric variables, blanks the field and sets the length to -1 for string variables).

2.3 Environmental Control Block

The ECB is a block within an environment that contains information about the allocated file. The ECB is the first 30 bytes of the environment and is built and validated by the routine PUSHEN. The following illustrates the structure of the ECB.

Bytes	Name	Description
0/1	E.LEN	length of environment including the ECB.
2/3	E.PREV	length of previous block.
4/5	E.RMEM	reserved memory (RESMEM) allocated for program
6	E.FCNT	FOR/NEXT count
7	E.GCNT	GOSUB count
8/9	E.EREX	address of code to be executed upon an ON ERROR
10/11	E.ERPC	Kangaroo PC after an ON ERROR
12/13	E.ROM	ROM number of mother program
14/21	E.MOM	name of the mother program
22/23	E.RTN	R10 for cont/run
24/25	E.PCR	PCR for cont/run
26	E.STAT	current status (R16)
27	E.DATA	location in current DATA line
28/29	E.DATL	pointer to current DATA line

2.4 Allocated Structure

To demonstrate the internal structure of an allocated program, the following Basic program and its allocated form are supplied.

```
10 A4=PI
20 B$='hi there'
30 M(1,1)=3*A4/2
```

```
ALLOCATED STRUCTURE (internal)

            memory location:   8593 h

                    ** PCB **

 P.LEN     P.PLEN     P.CLEN     P.ELEN     P.SPAR
--------  --------   --------   --------   --------
49   00   14   00    00   00    F2   03    00   00


            memory location:   859D h

                  ** PROGRAM **

Line 10:  10   00   06   11  > 49   00 <  C9   08   0E
                               A4 ptr

Line 20:  20   00   0F   13  > 40   00 <  96   08   68   69
                               B$ ptr

          20   74   68   65   72   65   07   0E


Line 30:  30   00   1B   12  > 53   00 <  1A   01   00   00
                               M(,) ptr

          1A   01   00   00   0A   1A   03   00   00   01
```



```
  > 49   00 <  2A   1A   02   00   00   2F   08   0E
    A4 ptr


ENDLINE:  99   A9   02   8A   0E


              memory location:  85DB h

                   ** VPA **

        Name        Ptr
       --------   --------
A4:     04   01  |  1E   00

        Name       Max Len      Ptr
       --------   --------   --------
B$:     8A   02  |  20   00  |  26   00

        Name       Tot Len     Max Row     Max Col      Ptr
       --------   --------   --------   --------   --------
M(,):   4A   0D  |  C8   83  |  0A   00  |  0A   00  |  48   00


             memory location:   8634 h --> FWVARS

                   ** ECB **

   E.LEN     E.PREV     E.RMEM   E.FCNT  E.GCNT   E.EREX
  --------  --------   --------  ------  ------  --------
  11   04   1F   00    00   00     00      00    00   00

   E.ERPC    E.ROM                    E.MOM
  --------  --------   -----------------------------------------
  00   00   00   00    77   6F   72   6B   66   69   6C   65

   E.RTN     E.PCR     E.STAT  E.DATA   E.DATL
  --------  --------   ------  ------  --------
  47   00   43   00      02      00    00   00

  ------------------------------------------------------------

                ** Environment **

Val A4:     00   00   59   53   26   59   41   31
```

```
                 Len
              --------
Val B$:       08    00  | 68    69    20    74    68    65
              72    65    20    20    20    20    20    20
              20    20    20    20    20    20    20    20
              20    20    20    20    20    20    20    20
              20    20

Val M(0,0):   00    00    00    00    00    00    00    FF
    M(0,1):   00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
    M(0,5):   00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
    M(0,10):  00    00    00    00    00    00    00    FF
    M(1,0):   00    00    00    00    00    00    00    FF

    M(1,1):   00    00    39    80    89    38    12    47

      .       00    00    00    00    00    00    00    FF
      .       00    00    00    00    00    00    00    FF
```

Refer to the Memory Management Documentation for a discussion on the existence and structure of the environmental stack.

2.5 Globals

GLOBALS

Name	Location	Description
FWVARS	8251	pointer to current environment
LEEWAY	83BE	variable memory leeway
LSTELN	83C3	change in environment size
LSTPLN	83C1	last value of P.PLEN
NXTMEM	8253	next byte in available user memory
PRFILE	8243	loc of parameter file
PRNAME	8263	name of parameter file
ROMPTR	82A3	rel loc of current ROM
XTNDLW	83BD	extended LEEWAY flag

2.6 Handi Calls

V.EALO -- This handi call is generated under two separate circumstances.
1) the current variable is remote
2) the current variable is a string array

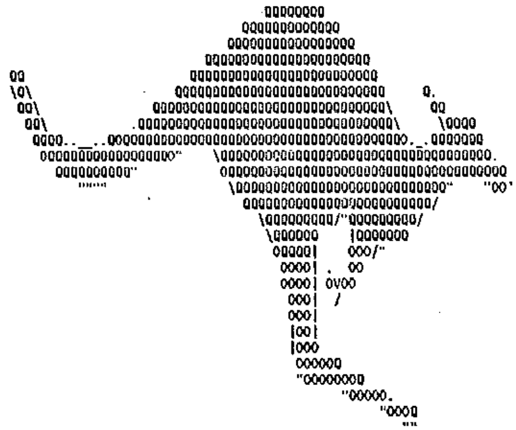
2.7 Cross References

Memory Management Document	RH"MEM
Handi Call Document	RH"HDI
Source File	RH&EAL
Global File	KR&GLO

Table of Contents

The Buzzer Driver Software

Raan Young
07/09/82



Buzzer driver

The beep code lives in two files (KR-BEE & KR-BE'). It was split up to allow BE' to be put in switching rom (freeing up more non-switching space).

The beeper is accessed in two ways:

1) the system routine STBEEP; STBEEP sets up default parameters and calls the BEEPER routine which does the actual driving of the beeper. STBEEP is used for the error beep by the system.
2) the user command BEEP; BEEP takes the user input, translates it to internal form and calls BEEPER.

BEEP parsing expects parameters of form:

1) keyword ON or OFF.
2) none; frequency and duration are same as STBEEP.
3) 1 number; interpreted as frequency, duration same as STBEEP.
4) 2 numbers; frequency and duration.

Beep runtime handles the parameters as:

1) set or clear BEEPOK flag.
2) get default frequency and duration, beep.
3) get default duration, beep.
4) beep.

Duration is converted (using the ONE?PR routine) to a 5-byte binary number which is added to the current real-time clock value to get the absolute time to quit beeping. The BEEPER routine checks this value against the current time every half cycle to see if we have exceeded the absolute time (if so, we quit beeping). The value is saved in WDBR1C to allow a clock rollover to adjust the absolute time appropriately (see KR"CMP for more info on rollover).

Frequency is converted to half-cycle counts with the formula C=28300/F-16. This returns a count for the number of cycles of the inner loop in BEEPER. The 16 is subtracted to compensate for the overhead of the outer loop between half cycles. The routine KEY? which is called by STOP? (called in BEEPER) has been specially constructed so that both possible paths through it take the same length of time. This prevents the pitch from changing if a user hits a key while we are beeping. The numbers 28300 and 16 were found algebraically and fine tuned by experimentation. They should produce tones which are reasonably accurate in the low freqencies. Above about 1000 Hz the accuracy begins to degrade. the maximum frequency is approximately 1700 Hz.

Mary Jo Hornberger
7/12/82

Going to sleep in Kangaroo

There are several ways to go to sleep in Kangaroo:

method	corresponding 'key'
a) type BYE	BYEKEY
b) use BYE in a program	no key (Basic runtime routine)
c) let machine time out	NAPKEY
d) get very low batteries	NAPKEY
e) press shift-ATTN	NAPKEY

Sleep mode can be entered either from the interpreter (running the BYE token) or the mode switcher (NAPKEY or BYEKEY seen any time we're looking for input).

If our power is ok, there are 3 things that could keep us from going straight to sleep after entering the sleep code:

happening	action
a) pending timers	will exit the sleep code and jump to where timer was set up to go
b) appointments due and nothing running	will exit the sleep code, take care of the appointment and then call the sleep code again
c) sleep HANDI call	could possibly be set up to not return to sleep code (see warning below)

There are also a few other obscure ways to keep us from going straight to sleep. These include modifying the V.LOOP HANDI call in the HPIL frame-sending routine so it doesn't return, or changing any of the interrupt service routines. SUCH CHANGES SHOULD ONLY BE ATTEMPTED BY QUALIFIED PERSONS WITH A FULL UNDERSTANDING OF THE COMPLICATIONS INVOLVED.

Following is the sequence of steps we take when we're putting Kangaroo to sleep, from either the BYEKEY or the NAPKEY.

A) If BYE typed in calc mode, stall the calc program.

If the user types BYE in calc mode, we want to stall the calc program, so appointments will be processed. (Appointments aren't processed if we go to sleep while something is running.) To do this, we stall the calc program, set up a BYEKEY as the pending key, and return to the interpreter (ie, exit the BYE token that we're executing). The interpreter returns to EDIT mode input, which sees the BYEKEY, and returns to the mode switcher. The mode switcher sees the BYEKEY and sends us to sleep mode again, this time with nothing running.

B) Turn the HPIL loop off.

After we have taken care of BYE in calc mode, the BYE and NAP keys function the same. First we set the 'test loop before using' flag, in case the user has turned off the HPIL devices on the loop. This keeps us from waiting forever (or until the batteries run down) for a frame to return from a dead loop when we're in STANDBY ON mode.

If there are any DISPLAY IS devices active, we need to unlisten them. Before sending the unlisten, we test the integrity of the loop with an IDY. If the loop is ok, we clear the 'test before using' bit and send out an unlisten. (The 'test before using' flag will be set again as soon as anything times out on the loop.)

C) Take care of timers and appointments.

(If we are being forced to sleep because of low power, we skip this part.)

First we trigger any devices that have set up a comparator interrupt. If there are any pending timers, we set up an EDIT key, and return to caller. This will cause Kangaroo to execute whatever the timer was set up to do, then go to Edit mode when done. (This does NOT return to sleep mode, until another BYEKEY or NAPKEY is done.)

If we're not running a program, we look for and process any unprocessed pending appointments. If we find one, we load up a NAPKEY, and return to the caller. This will cause Kangaroo to come back through the sleep code when it is done with the appointment.

Note that we will never process both a timer and an appointment, since timers are only active when a program is running, and appointments are only processed when a program is NOT running.

D) Disable the comparator interrupt.

We now disable the comparator interrupt, to minimize the chance of another comparator interrupt being serviced in the interrupt service routine, but not getting triggered before we go to sleep. This would have the effect of that interrupt (let's say a timer) being ignored until the user woke us up again. If the user had been depending on the timer to wake his machine up, this could be very frustrating.

Should a comparator interrupt come due between the time the comparator interrupt is disabled and the time we go to sleep, it will cause the hardware to override us when we try to tell the power supply to put us to sleep. Instead, we will return to the wakeup code, almost as if the user had hit an ATTN key immediately after sending us to sleep. The difference between this and a regular wakeup is that the diagnostic rom will not be called, and the lcd will get cleared a little later than usual.

E) Do the Sleep HANDI call.

We do a HANDIO call (VAL V.SLEE) to let the roms know that we're going to sleep. (If any roms want to do anything with the comparator, or don't plan on returning to the sleep code, it is up to them to re-enable the comparator interrupt.)

F) Put out the low batteries warning if we have low power.

If we were sent to sleep because of low power (PWRFLG = nonzero), we put out the low batteries warning message.

G) Check the HPIL loop again and turn off the HPIL chip oscillators.

We set the 'check loop before using' flag again, and resend the unlisten sequence if there are active DISPLAY IS devices. This is done so we have the most current information possible when we go to sleep.

From now on, no frames will be sent on the HPIL loop until we wake up, so we turn off the chip oscillators.

H) Do miscellaneous housekeeping.

First we clear the LETSEE flag, which is the flag that tells Kangaroo that there is something in the LCD. Leaving this flag set would cause Kangaroo to not issue the prompt at wakeup, believing that there was something in the LCD that the user wanted to see.

The global interrupts are then disabled, and the comparator interrupt is reenabled.

The 'key waiting' bit in SVCWRD is cleared, just in case someone hit a key after the BYE, so that key doesn't pop up in the LCD when we wake up.

I) Save registers and compute the checksum

We save a place for the checksum on R6, and push our wakeup address. We then save registers 6,10,12, and 16 in TMPMM2. We now calculate the checksum from 10 bytes past the R6 stack to the last word of available memory (LWAMEM) and store the checksum in the space reserved on R6.

J) Put us to sleep.

And last, but not least, we write the proper values to the power supply status byte (PSSB) to set us to light and deep sleep.

KR"BYE page 3

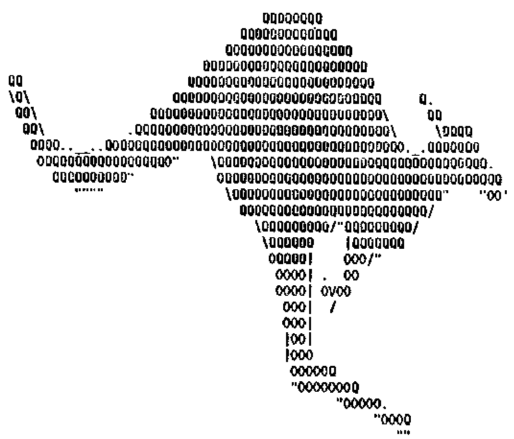
Things used:

```
Globals:
   LETSEE  1 byte     =nonzero if something in the LCD
   PWRFLG  1 byte     =nonzero if low power
   LWAMEM  2 bytes    =address of last word of available memory
   SVCWRD  bit#0      =1 if key waiting to be seen
   PLSTAT  bit#0      =1 if must test loop before using

Equates:
   V.SLEE  equ  03H   =event number for going-to-sleep HANDI call
   BYEKEY  equ  FEH   =key generated by BYE
   NAPKEY  equ  A0H   =key generated by shift-ATTN
   EDITKY  equ  83H   =key generated by EDIT

I/O addresses:
   PSSB    dad  FF82H  =power supply status byte
   GINTDS  dad  FF01H  =global interrupt disable address
   CMPSB   dad  FF80H  =comparator status byte

Major routines:
   BYE.    in  KR&ZZZ  entry point for BYE token runtime routine
   ZZZZZZ  in  KR&ZZZ  entry point for NAPKEY
                          and BYEKEY when seen in mode switcher

Related routines:
   WAITKY  in  KR&IO   loads NAPKEY if timeout when waiting for input
   MODEKY  in  KR&MOD  (the mode switcher) if NAP or BYE key seen,
                          sends us to sleep code
   PWRSRV  in  KR&LOW  sets up power flag to indicate low power

Related documents:
   See Roo Chip ERS for more information about the PSSB and CMPSB
```

KR"BYE page 4

Checksum computation

Raan Young
07/09/82



BOUND

Registers: R34/35 pointer to ram (2K increments)
R36/37 scratch

Globals: TOPROM a label in KR&TOK which is the first byte of code in the top rom (56K)

Subroutines: none

Description: Starting at 32K and incrementing by 2K steps, get 2 bytes from ram. One's complement the first byte, write them back out, and compare what was written with what is actually there. If they are not the same we have found the first byte of non-ram and are done. Otherwise, one's complement the first byte again, and restore the original ram contents. Then increment and try again if not at the top of ram. NOTE: the complementing of the first byte is done because, if there is no ram we will read back the high order byte of the address twice. We want to make sure the two bytes we write out are different so that (if there is no ram) we will read back the same (2nd) byte twice and get an error when we do the compare.

CHECK

Registers:
- R00/01 pointer to end of ram space to be summed (physical end of memory found by BOUND)
- R02/03 scratch
- R14/15 pointer to beginning of ram space to be summed (current stack pointer + 10 bytes)
- R46/47 accumulator for checksum

Globals: LWAMEM pointer in KR&GLO to last word of ram (+1)

Subroutines: SUMIT+ summing routine in KR&CRD

Description: Get the begin and end points for the checksum and call the summing routine. The summing routine is a loop which gets 2 bytes at a time and adds them to a 2 byte accumulator with wrap around carry. If we are summing an odd number of bytes, the extra byte on the end is cleared before the add. The loop continues until our ram pointer is >= the end location.

General Operation:
On coldstart, the system calls BOUND to establish the initial ram size. Each time the system goes to sleep, it calls CHECK to checksum all ram (minus a small part of the system stack) and saves the result on the stack. Each time the system wakes up, it calls BOUND to determine if the user has changed the amount of ram. If it is less than last time a coldstart is forced (since things could be damaged). If it is the same or more, then CHECK is called to compute the checksum of the amount there last time the system went to sleep; the result is compared to the result saved on the stack while going to sleep. If the results are not the same a coldstart is forced (something has been damaged). If the results are the same memory is adjusted as needed for the new ram, the new ram size is saved, and we go on waking up.

HP-75 Instruction Set

System Command P	BASIC Statement P	BASIC Function P
ADJST	ASSIGN #	ABS
ALARM OFF	BEEP	ACOS
ALARM ON	CALL	ANGLE
ASSIGN IO	DATA	ASIN
AUTO	DEF FN	ATN
BEEP OFF	DIM	CAT$
BEEP ON	DISP	CEIL
BYE	DISP USING	CHR$
CAT	END	COS
CAT ALL	END DEF	COT
CAT CARD	FOR...TO...STEP	CSC
CLEAR LOOP	GOSUB	DATE
CLEAR VARS	GOTO	DATE$
CONT	IF...THEN...ELSE	DEG
COPY...TO	IMAGE	EPS
DEFAULT OFF	INPUT	ERRL
DEFAULT ON	INTEGER	ERRN
DEF KEY	LET	EXP
DELAY	LET FN	FLOOR
DELETE	NEXT	FP
DISPLAY IS	OFF ERROR	INF
EDIT	OFF TIMER #	INT
ENDLINE	ON ERROR	IP
EXACT	ON TIMER #	KEY$
EXTD	ON...GOSUB	LEN
FETCH	ON...GOTO	LOG
FETCH KEY	OPTION BASE	LOG 10
INITIALIZE	POP	MAX
LIST	PRINT	MEM
LIST IO	PRINT #	MIN
LOCK	PRINT USING	MOD
MARGIN	PUT	NUM
MERGE	RANDOMIZE	PI
NAME	READ	POS
OFF IO	PUT	RAD
OPTION ANGLE DEGREES	RANDOMIZE	RMD
OPTION ANGLE RADIANS	READ	RND
PACK	READ #	SEC
PLIST	REAL	SGN
PLIST	REM	SIN
PRINTER IS	RESTORE	SQR
PROTECT	RESTORE #	STR$
PURGE	RETURN	TAB
PWIDTH	SHORT	TAN
RENAME...TO	STOP	TIME
RENUMBER	WAIT	TIME$
RESET		UPRC$
RESTORE IO		VAL
RUN		VER$
SET		
STANDBY OFF		
STANDBY ON		
STATS		
TRACE FLOW		
TRACE OFF		
TRACE VARS		
TRANSFORM...INTO		
UNPROTECT		
WIDTH		
YEAR		

Comparator driver software

Raan Young
07/09/82

The comparator is a hardware device which interacts with the real time clock and causes an interrupt whenever the value in the clock is >= the value in the comparator. This allows the software to initiate events at a time specified by the user. The pocket secretary, time mode, and ON TIMER statement are examples of the things which can be done.

The hardware only handles one value at a time. This means that if more than one comparator device (appointment, timer, etc) wants to use it, there is a conflict. This code resolves this conflict. It presents the comparator devices with a comparator machine which handles input in the form of tagged entries (indicating ownership) and informs the owner when that entry is active. The machine can handle 8 different devices (5 are used by the mainframe, see extension section for more), and also handles the rollover of the RTC (once a year). The initialization of the RTC and absolute time is handled by TIME mode (see KR"TMC for more).

Usage of the machine consists of sending an entry to it, flagged with your owner number (see CMPENT section for more). When that entry becomes the next value to be matched, it is loaded into the comparator hardware, and when the interrupt happens, the flag cooresponding to that entries owner is set. CMPCHK checks these flags and calls the trigger routine for the owner of each entry needing service. Entries can be absolute or relative time, depending on the owner, and some have automatic reentry.

COMPARATOR SETUP:
The comparator is setup at coldstart to have all entries in the table 0 except rollover, which is set to 1 year. The rollover value is loaded into the comparator hardware.

COMPARATOR ENTRY:
The comparator entry is setup by calling CMPENT with R40 containing the owner # and R41/47 containing the appropriate value. Owners are:

```
 0 (CT.RLV) rollover entry (not used by CMPENT) [absolute]
 7 (CT.APT) pocket secretary - time of next appointment
            [absolute]
14 (CT.RPA) repeating alarm - time of next beep [absolute,
            auto inc]
21 (CT.C/S) clock/stopwatch - time of next tick [absolute,
            auto inc]
28 (CT.TMR) ON TIMER - time of next timer [absolute]
35 (CT.DM1) unused - see extension section [absolute]
42 (CT.ADJ) clock adjust - time of next adjust to RTC [in-
            terval, auto inc]
49 (CT.DM2) unused - see extension section [interval]
56 (CT.DM3) unused - see extension section [interval]

absolute = time is compared directly to clock
interval = time is added to current clock to get absolute
           value
auto inc = entry is automatically updated by some interval
           1 sec for CT.C/S, 15 sec for CT.RPA, and the
           value stored in TMEINC (by CMPENT) for CT.ADJ.
```

An entry of 0 value will turn off the designated device.

After the table entry has been added, the table is searched for the lowest value, and this is loaded into the comparator hardware.

COMPARATOR INTERRUPT:
The comparator interrupt routine (CMPSRV) handles the processing of the interrupt. This consists of finding out which entry caused the interrupt (since the comparator can not be read, the owner number is kept in CMPPNT). If the entry was rollover, then the rollover processing is initiated at interrupt time (see rollover for more). Otherwise, the comparator bit is set in SVCWRD to indicate service is needed (this will cause SPY to call CMPCHK). Then, the increment value is fetched from TMEINC for clock adjusts, or from the service table for repeating alarms and clock/stopwatch. The increment is added to the current table value (if appropriate) and the new absolute time (or 0 for non-auto inc entries) is placed in the table. Then, the next entry is loaded into the comparator hardware. The service table has the increment value, or 0, for each entry, and the flag mask for that entry. It is indexed by the owner number. The flag mask is added to the current contents of CMPFLG to indicate to CMPCHK which device or devices need service. This completes the interrupt service, the guts of each device is handled when CMPCHK gets called by the system (when its safe).

Rollover:
This is a special case of interrupt processing, which happens once a year for a machine that has not been coldstarted more recently. Rollover handles the problem of the RTC hardware overflowing its count buffer by resetting the buffer to 0 (hardware can count about 2.5 years). The setting of 1 year was chosen for a nice boundary. The current RTC value is saved, and the RTC is cleared. The value is used to adjust WOBRTC (used by WAIT, DELAY, and BEEP for the absolute time to quit). The value is also added to the time base. The value is then subtracted from each of the entries in the comparator table (if the value is < 0 then the minimum value of 1 is loaded). The routine TMRRLV is called to adjust the timers, and then the rollover value is reloaded.

COMPARATOR ENABLE/DISABLE:
The comparator interrupt is enabled (CMPENA) or disabled (CMPDSA) without affecting any of the other bits in the comparator status. This is because the bit which controls the buzzer is in this status, and the state of it should not be changed by the interrupt change.

COMPARATOR CHECKING:
The flags which indicate comparator devices that need servicing are kept in CMPFLG. CMPCHK checks these flags (if NOCHEK is 0) and calls the apppropriate routine to service the device. CMPCHK also clears the device flag (with interrupts disabled -- to prevent possible setting during the clearing) and, if all devices have been serviced, clears the comparator bit in SVCWRD. All registers (R20-77) are saved before the device routine is called, and restored afterwords. After the routine returns, a check is made for power failure or a key pending. If so, we exit CMPCHK (this is to allow the user to type while comparator interrupts are coming in too fast to service [time mode to a printer, or multiple alarms]). Otherwise loop around again to see if any other devices need service.

DEVICE SERVICE ROUTINES:
In the mainframe, there are 5 service routines:

Repeating alarm:	CNTRIG	(see Pocket Secretary documentation)
Alarm:	APTRIG	(see Pocket Secretary documentation)
Timers:	TMTRIG	(see KR"TIM)
Clock:	CSTRIG	(see KR"TMC and Pocket Secretary)
Adjust:	AJTRIG	(see KR"TMC and following text)

An example routine is AJTRIG, which handles the clock adjustments set up by time mode. AJTRIG checks the bit in PSSTAT which indicates the type of clock adjustment (incrementing or decrementing). It then sends the appropriate status to the RTC hardware. This is all that needs to be done, because the setup for the next comparator interrupt to trigger this is done automatically by the comparator interrupt routine (adjust is an auto-increment entry).

COMPARATOR EXTENSIONS:
The comparator can be extended in two ways:

* The comparator intercept (CMPINT) can be used to modify or change the interrupt service routine (subject to rom switching restrictions).
* The trigger HANDI call can be used to pick up triggers for non-mainframe events. This is done by setting up an entry using one of the unused entry codes. When this event comes due, CMPCHK will call HANDI with event # V.ETRG. R20 will contain a pointer such that (R20+1-DVCTBL) will indicate the external entry code used (1,2, or 3). The rom using the comparator must decide if this is its external device, service it if so, and return with HANDLD set (if serviced).

GENERAL FLOW:
The general flow of comparator control is as follows:

* An entry is set up for the comparator and loaded into the hardware.
* The comparator interrupt happens.
* The interrupt service routine flags the event and loads the next event.
* CMPCHK detects the flag and calls the service routine.
* The service routine does its thing (update display, load next entry ...).
* Life goes on.

Card reader driver

Raan Young
07/13/82

The card reader has PROTECT, UNPROTECT, COPY ... CARD/PCRD, and CAT CARD as its key words. The general flow of each command is given here and then key subroutines will be discussed.

CARD LAYOUT:
The card is layed out as:

```
+---------+---------------+-----------+------+
| HP head | write protect | file head | data |
+---------+---------------+-----------+------+
```

Where each field is ended by at least one 0 and starts with a special code (the trailing 0 is not part of the field [it can't be read], and is not included in the count). There is a gap between each field, and at both ends. The special code and gap are handled by the hardware (see the card reader hardware ERS for more).

HP head: This indicates the type of card we have. Currently the only type is HPCV(700D)00. This 8 byte code means HPCV type card, 700 bytes maximum storage (length of card after write protect). The 0's are reserved for future extension.

Write protect: This indicates whether or not the card can be written on. Only the first 2 bytes of this 4 byte field is used for the flag, 0000 = unprotected, FFFF = protected.

File head: This contains all the identification and directory info for this card, in addition to the checksums.

byte	purpose
1	Sub-Format. Indicates a sub-format (ie, 2 data fields) 0=R00
2	# of tracks. # of tracks in file
3	Track #. # of this track
4-5	Track size. # of bytes in track
6-7	File size. # of bytes in entire file
8-9	File type. Basic program, text, lex, LIF, etc.
10-17	Filename. 8 character filename
18-21	Password. 4 character string
22-25	Time/date. Date and time of file creation (2^-14 secs from century)
26-27	File checksum. To make copy unique but multiple cards mixable
28	Partial status. Status of partial statement information 0=R00
29-30	Partial 1st statement. Information about first partial statement
31-32	Partial next statement. Information about next partial statement
33-34	Card checksum. Checksum of card data
35	Header checksum. Checksum for file header.

Data: This contains the data stored on the card. The size of this field is 1-650 bytes.

GENERAL FLOW:
PROTECT/UNPROTECT: the code for both of these commands is the same, only the flag is different (0=unprotected, #0=protected). The code calls STRTUP to get things started (save registers. etc.), then calls STRTCD to disable interrupts, prompt the user for the card, and read/check the HP header. Finally, WRITCR is called to write the flag on the card. INTENA reenables the interrupts, and ENDIT cleans up and leaves.

CAT CARD: the code for this command is invoked by the code for the CAT <filename> command. When the command determines that the device to be cataloged is the card reader, it calls CRDEXM (which does the work). STRTUP gets things started, then GOCARD prompts the user for the card, read/checks the HP header, and reads the write protect field. We are now ready to read the file header, which is done by READCR. Then we test the header with HEDSUM to make sure it was read correctly. If it was, we enable interrupts, check the sub-format, and (if correct) display the track number, number of tracks, and filename (this can be recalled with shift-FET). Then CATLIN outputs the standard catalog for the file, and waits until SIGNIF returns to clean up with ENDIT.

COPY ... CARD/PCRD: the copy code recognizes two devices, CARD and PCRD. These are the same, except that, in the case of PCRD, the file on the target end is made private. The copy command invokes the card reader code when it encounters one (two is an error) card reader device. The code enters through CRDCPY which determines the direction of the copy (from card=load, to card=store) and calls the appropriate routine (LOAD or STORE). CRDCPY also handles the privatization of the target file by calling the routine MKEPRV as needed.

Load: this routine calls STRTUP to get things rolling, then calls ALCALL to get as much memory as is available. The table for tracks loaded is next initialized to indicate no tracks loaded. (This table consists of 38 bytes of 1, when a track is loaded, the byte corresponding to that track number is changed to 0.) GOCARD gets the card going and read/checks the HP header and write protect. Next, read the file header (READCR) and check the possible error conditions (CHKHED - too big, wrong file, password, etc). If all is ok, compute the location for the data track in the file (track#-1 times FULTRK + file start) and copy the data into file using READCR. Then check the data checksum to make sure it was read ok, enable interrupts, and check to see if need any more tracks. If so, put out the track message and get the next one. Otherwise, straighten up the file directory with CRTFLE (release unneeded memory), and finish up with ENDIT.

Store: STRTUP begins the store, then HEDBLD builds up the file header to be written on the card. Tell the user how many tracks they need, and see if the card name is different than the RAM name (if so use new creation date). Calculate the location of the data to be stored, and checksum it. Then, if not a retry of the same track, scan for the statement that crosses the boundary between this track and the next track. Save the overlap count for the next track and add the number of bytes overlap into this track to the file header. Next, calculate the checksum of the header. Now GOCARD prompts the user for the card, read/checks the HP header, and reads the write protect. If not write protected, then WRITCR is called to write the file header, and then again to write the data. After enabling interrupts, check to see if verify is on (always is for the mainframe), and call VERIFY if needed. Reenable interrupts after the call to VERIFY. Finally if not the last track, display the track message and go do the next track. Otherwise, clean up with ENDIT.

Verify: VERIFY saves some info to allow recursive call of some routines, and then calls COCARD to get things going. The file header is read into TMPMM2, checksummed (against the original), and then RDVFY is called. RDVFY reads in the data and computes a flying checksum (the data is never stored anywhere, just summed up). The result of this checksum is compared to the original. If either checksum is bad, E is set to indicate failure.

ERROR HANDLING:
There are two types of card reader errors: hard and soft. Hard errors are errors which abort the entire operation (for example, wrong password -- the user must redo the copy command). Soft errors are errors which restart the current track (for example, card pulled too fast -- user is prompted for same track again).

Since errors can happen at several different places, and unknown levels of the R6 stack, an ABORT routine is used to restore things to a known location. Several items are saved at the beginning of the operation to enable this: SUBPNT saves the current R6 pointer; RTNSVE save the address of where to go for a retry; and ABIFLG is initialized to a 0.

When an error is encountered, ABIFLG is set to 1 if the error is hard, and the error # is passed to the ABORT routine. ABORT restores R6 to SUBPNT and checks the error type. If hard, check to see if doing a load (if yes, purge the partial file), and call ENDIT to cleanup. If soft, report the error with WARN.R, put RTNSVE on the stack, and return to it.

SETUP & CLEANUP:
The setup is handled by STRTUP. This calls EVIL to save all registers used by the code, saves R6 in SUBPNT, and sets NOCHEK to 1 (this prevents CMPCHK from doing anything while waiting for the user to hit RTN). The cleanup is handled by ENDIT. This routine trashes its return (return to the caller of the caller of ENDIT), clears the error buffer, sets NOCHEK to 0, and restores the registers saved in EVIL.

GETTING THE CARD STARTED:
STRTCD handles the startup of the card (GOCARD does a STRTCD, reads the write protect, and sets up for the file header). ABIFLG is initialized to 0 (soft error), the first part of the card prompt (for example 'COPY TO') is written to the LCD, and then the rest of the prompt is written. Then, wait for a key with SIGNIF. If the key is ATTN or shift-ATTN, cause a hard error with no error message. Kill the key, and if it is RTN, start reading the card. Otherwise, wait for another key. To start, save the current delay, write out the pull message with no delay, and restore the delay. Disable interrupts from the keyboard, comparator, and PIL (power is left enabled in case the battery dies during the card reader), and turn on the card reader hardware. Read the HP header, and check to see if power is ok (if not, cause hard error with no message -- if the power is ok then assume it will last for the rest of the card). See if the header matches the expected header (the call to HPINTC allows plugin roms to change this header). If ok, all done. Otherwise, report the soft error (unrecognized header) and restart the track.

READING/WRITING THE CARD:
The hardware communicates via two bytes of information. Data is the first byte, and status is the second byte (all reads and writes are 2 byte hunks reached via FF10H I/O address). The status is laid out as:

RD/WRT	BIT#0	Set indicates a read operation, clear indicates a write operation.
CDRDST	BIT#1	Indicates state ("ON"/"OFF") of card reader. Set by CPU to initiate card reader, cleared by CPU to terminate card reader.
STRSTP	BIT#2	Set by CPU to start card reader operation, cleared by CPU to stop operation.
unused	BIT#3	This bit is not used for anything.
SPDHI	BIT#4	Set by card reader to indicate speed error was on fast side, clock count too low. Cleared by CPU.
SPDLO	BIT#5	Set by card reader to indicate speed error was on slow side, clock count too high. Cleared by CPU.
GENERR	BIT#6	Set to indicate a general card error, such as overflow. Cleared by CPU.
READY?	BIT#7	When set, indicates data buffer is ready for next access (read from/write to), cleared by CPU to indicate access completed.

Data is passed between the CPU and the card via an 8-bit data buffer. For reads, READY? is set to indicate that 8 bits have been copied from the card to the buffer, ready for access by the CPU, which then clears READY? to indicate access finished. For writes, READY? is set to indicate that 8 bits have been copied to the card from the buffer, which is ready for the next 8 bits. The CPU clears READY? to indicate that the buffer has been reloaded.

READCR and WRITCR are the general routines for talking to the card reader hardware. They take as input the starting point and number of bytes to be read (the starting point is the ram location data is to be stored in/taken from). They put the hardware in the appropriate mode, and set E to indicate the mode (E=1 if read, 0 if write). For reads, the count is adjusted by -1 to compensate for the hardware writing the last byte after it is shutdown (the same loop is used for both read and write). For writes, the count is adjusted +4 to make sure there are at least 3 bytes of padding for Titan and 1 byte of 0 for the hardware to end it's read on. The parameters start address, data end address, and field end address (includes padding, etc) are computed and passed to the gut level routines GETBYT/NXTBYT and GORDWR. (Note: Titan required the pad bytes at one time, but does not any longer. However, code had already been frozen by the time this was known.)

GETBYT/NXTBYT sends the card reader the status/data, and goes into a loop to wait for the READY bit to be set. This loop decrements a timeout count (initialized by INTOSA for the first time and then restored to internal count after each byte -- this allows a long count to get the first byte and then a short one for all following bytes). If the count goes to 0, then we have a timeout problem and a soft error is reported. When READY is set, the status/data is read and the ready bit is cleared. Next the error bits are checked (if any set then ERRS determines which one and reports the error -- if this is the last byte of a write, the errors are ignored). NXTBYT also checks to see if we are at the last byte of data, and clears the START/STOP bit if so (this will stop the read/write after the current byte is handled).

GORDWR is a loop which checks to see if we have done the last byte of data. If so write 0, ignore read; otherwise, save/get the data. If the card reader was stopped on the last byte, then exit, otherwise call NXTBYT and go around again.

RDVFY is a special read routine used by the VERIFY operation. It uses GETBYT/NXTBYT to read the card, but instead of storing the data in ram, it builds up a two byte number and adds it to an accumulator (with wrap around carry) to build up a checksum of the data on the card. When the end of the field is reached, it returns with the computed checksum.

DATA LOCATION:
The starting point for data in RAM is found using the DTACLC routine, which uses FULTRK (maximum card length) to determine the location:

(FULTRK*(FLHEAD-1)+<file start>)

The length of the field is also set up by DTACLC.

CHECKSUMS:
The checksums are all done as two byte numbers added to an accumulator with wraparound carry. The header checksum takes the two byte result and adds the top byte to the bottom byte for a one byte result. For the header this is done by VYHDSM/HEDSUM (VYHDSM uses TMPMM2 instead of the normal file header ram). The data checksum is handled by DTASUM. All routines (and the RAM checksum routine [see KR"CHK for more]) use the SUMIT routines for the actual computation. The SUMIT routines use R0 for an end value, R14 for a pointer, and R46 for the accumulator.

MESSAGES TO THE USER:
In addition to the prompt, and error messages, 3 messages are displayed for the user (all can be retrieved with shift-FET). These are the size message (number of tracks needed for the copy), the catalog message (track # and # of tracks), and the track message (what track was just handled). These are handled by SZEMSG, EXMMSG, and TRKMSG respectively. All routines use NUMOUT to translate numbers to ASCII, and MVBYTS to move message portions to output. EROUT- is used to put the message out for the user.

INFORMATION CHECKING:
Information from the file header is checked during a read to make sure that:

- the header was read correctly, soft error
- the sub-format is ok, soft error
- the track is part of the file, soft error
- in addition, if this is the first track read:
  - the filename (if given) matches the card, soft error
  - the password (if given) matches the card, hard error
  - the file will fit in RAM, hard error
  - the types (if special [APPT]) match, hard error
  - and the track id # is loaded (file checksum)

This is done by CHKHED. At the end of the data, the data is checksumed and tested by CHKEND. CHKEND also updates the track load table and finds the next needed track for the track message.

EXTENDING THE CARD READER:
There are 3 ways provided to extend the functions of the card reader.

1) V.CARD HANDIO call. This is called from TRKMSG before the message is written out. It allows plugin roms to do post processing of a track of data before the user is prompted for the next one. For example, to allow more than 36 tracks, this could be used to adjust the track number for the user message, and move the track from a buffer to the real location (this assumes the track was written with a track number < 37).

2) HPINTC intercept. This is a ram intercept which gets called when the HP header and/or sub-format is checked. If the HP header is being checked then R40/41 contains "HP" and R40/47 contains the HP header. If it is the sub-format, then R40/41 contains 0 and R47 contains the sub-format. If the rom wants to do something, it can either return O/NO flag and let the code carry on, or it can grab control and never return.

3) New device name. This will cause the COPY or whatever code to do a V.FILE (or V.SPEC) HANDI call, which the rom can pickup and process. The card reader routines are available for use. For an example of this, see the BCRD device in RY-APR.

GLOBALS USED:

```
    ABTFLG    Flag to indicate type of abort desired.
    CRDSTS    Status bits for card reader machine
    FULTRK    Maximum track size




    HPHEAD    HP header buffer
    HPINTC    Intercept for HP header
    LSTTRK    Size of last track in bytes
    RTNSVE    Where to go after an error
    SUBPNT    Pointer to stack at entry to card reader
    TRKTBL    Table of tracks to be loaded.
    WRTPRO    Write protect buffer

    File header buffer:
      SUBFRM    Sub-format of card
      FLHEAD    Track # and # of tracks
      TRKSZE    Track size in bytes
      FILSZE    File size in bytes
      FILTYP    Type of file
      TMEDTE    Time/date of file creation
      FILNME    Filename
      PASWRD    File password
      FLECHK    Checksum of entire file including directory
      PRTSTS    Partial statement status
      PRT1ST    Length of partial statement on this track
      PRTNXT    Length of partial statement on next track
      CHKSUM    Computed checksum from card
      HDCKSM    File header checksum
```

COMMAND SYNTAX:

COPY "<filename1>:{CARD|PCRD}[/<password>]" TO "<filename2>"
Copies card to memory, if memory is available. Copy will be restarted if filename on card does not match the name in the header. Mismatched passwords will also abort the copy. If the reserved word "CARD" is used in place of the filename1 specification then filename checking will by passed. CARD device will copy normally, PCRD device will make the file private.

COPY "<filename1>" TO "<filename2>:{CARD|PCRD}[/<password>]"
Copies memory to card. Password will secure the card. If the reserved word "CARD" is used in place of the filename2 specification then filename on card will be the same as ram. CARD device will copy normally, PCRD device will make the file private.

PROTECT
Writes a write-protect code before the header. This requires two passes of the card, (first writes data, second writes protect).

UNPROTECT
Erases write protect on card.

CAT {CARD|":CARD"|":PCRD"}
Displays card header, does not copy file.

These commands can be entered for immediate execution or

be included in a program for programmatic execution. All commands prompt the user until all tracks have been processed. All commands can be aborted when prompting for a card by typing ATTN. Anything other than RTN, SHIFT-ATTN, or ATTN is ignored.

Catalog card flowchart:

```
+-----------------+
|    CAT CARD     |
+-----------------+
         |
         v
+-----------------+
|     STRTUP      |
| save registers  |
| save stack pntr |
| suppress CMPCHK |
+-----------------+
         |
         v
+-----------------+
| save return adr |
+-----------------+
         |
         v
+-----------------+
|     GOCARD      |
| prompt user     |
| disable intrpts |
| read HP header  |
| check HP header |
| read wpr flag   |
+-----------------+
         |
         v
+-----------------+
|     READER      |
| read file head  |
+-----------------+
         |
         v
+-----------------+
|     HEDSUM      |
| check headr sum |
+-----------------+
         |
         v
+-----------------+
|     INTENA      |
| enable intrpts  |
+-----------------+
         |
         v
+-----------------+
```

```
|     FRMCHK      |
| check subformat |
+-----------------+
         |
         v
+-----------------+



|     EXMMSG      |
| display track   |
+-----------------+
         |
         v
+-----------------+
|     CATLIN      |
| display catalog |
+-----------------+
         |
         v
+-----------------+
|     SIGNIF      |
| wait for key    |
+-----------------+
         |
         v
+-----------------+
|      ENDIT      |
| restore regs    |
|     return      |
+-----------------+
```

Protect/Unprotect card flowchart:

```
+----------------+                      +----------------+
|    PROTECT     |                      |   UNPROTECT    |
| set msg & flag |                      | set msg & flag |
+----------------+                      +----------------+
        |                                       |
        +------------------>+<------------------+
                            |
                            v
                  +-----------------+
                  |     STRTUP      |
                  | save registers  |
                  | save stack pntr |
                  | suppress CMPCHK |
                  +-----------------+
                            |
                            v
                  +-----------------+
                  | save return adr |
                  +-----------------+
                            |
                            v
                  +-----------------+
                  |     STRTCD      |
                  | prompt user     |
                  | disable intrpts |
                  | read HP header  |
                  | check HP header |
                  +-----------------+
                            |
                            v
                  +-----------------+
                  |     WRITER      |
                  | write wpr flag  |
                  +-----------------+
                            |
                            v
                  +-----------------+
                  |     INTENA      |
                  | enable intrpts  |
                  +-----------------+
                            |
                            v
                  +-----------------+
                  |     ENDIT       |
                  | restore regs    |
                  |    return       |
                  +-----------------+
```

Copy card controller flowchart:

```
                          +----------------+
                          | COPY ... CARD  |
                          |   (CRDCPY)     |
                          +----------------+
                                  |
                                  v
                                  *
                                *   *
              no              *       *              yes
       +-------------------*    load?    *-------------------+
       |                      *       *                      |
       |                        *   *                        |
       |                          *                          |
       v                                                     v
       *                                            +----------------+
     *   *            yes                           |      LOAD      |
   *  store?  *---------+                           |  do load card  |
     *   *              |                           +----------------+
       *                v                                    |
       |        +----------------+                           |
    no |        |     STORE      |                           v
       |        | do store card  |                           *
       |        +----------------+                         *   *
       |                |                                *       *
       |                v                     yes      *           *
       |        +----------------+        +---------*    private?    *
       |        |     return     |        |            *           *
       |        +----------------+        |              *       *
       v                                  |                *   *
+----------------+                        |                  *
| must be private|                        |                  | no
|     store      |                        |                  v
+----------------+                        |         +----------------+
        |                                 |         |     return     |
        v                                 |         +----------------+
+----------------+                        |
|     MKEPRV     |                        |
| make basic priv|                        |
| (warn if not)  |                        |
+----------------+                        *
        |                               *   *
        v                             * load  *      no
+----------------+                  * aborted?  *-------------+
```

```
|     STORE      |            *       *               |
| do store card  |              *   *                 |
+----------------+                *                   v
        |                         | yes       +----------------+
        v                         v           |     MKEPRV     |
+----------------+        +----------------+  | make basic priv|

| restore access |        |     return     |  | (warn if not)  |
+----------------+        +----------------+  +----------------+
        |                                             |
        v                                             v
+----------------+                            +----------------+
|     return     |                            |     return     |
+----------------+                            +----------------+
```

Store card flowchart:

```
                 +-----------------+
                 |   Store card    |
                 +-----------------+
                          |
                          v
                 +-----------------+
                 | GET STARTED:    |
                 | save reg-STRTUP |
                 | build file      |
                 |   header-HEDBLD |
                 | tell user       |
                 |   size - SZEMSG |
                 +-----------------+
                          |
                          v
                          *
                        *   *
                      *       *
                    *  source/  *   yes
                  *  target name  *-----+
                    *   same?   *       |
                      *       *         |
                        *   *           |
                          *             |
                          | no          |
                          v             |
                 +-----------------+    |
                 |     CRTMDT      |    |
                 | get new creat   |    |
                 | date for target |    |
                 |     CRTMDT      |    |
                 +-----------------+    |
                          |             |
                          v             |
   +--------------------->+<------------+
   |                      |
   |                      v
   |             +-----------------+
   |             | FINISH HEADER:  |
   |             | get track size  |
   |             | get data chksum |
   |             +-----------------+
   |                      |
   |                      v
   |                      *
   |                    *   *
   |                  *  new   *     no
```

```
|            *  track?   *---------+
|              *       *           |
|                *   *             |
|                  *               |
|                  | yes           |
|                  v               |


|        +---------------+         |
|        | compute partial |       |
|        | statement info  |       |
|        +---------------+         |
|                |                 |
|                v                 |
|                +<----------------+
|                |
|                v
|        +---------------+
|        |    HEDSUM      |
|        | compute header |
|        | checksum       |
|        +---------------+
|                |
|                v
|        +---------------+
|        |    GOCARD      |
|        |get card started|
|        +---------------+
|                |
|                v
|                *
|              *   *
|            *       *
|          *   write   *   yes
|        *  protected?  *-----------+
|          *           *            |
|            *       *              v
|              *   *          +---------------+
|                *            | Error, restart |
|                | no         +---------------+
|                v
|        +---------------+
|        | PROCESS CARD:  |
|        | write file     |
|        |   header-WRITCR|
|        | write data     |
|        |   WRITCR       |
|        |enable interrupt|
|        |   INTENA       |
|        +---------------+
|                |
|                v
|                *
|              *   *
|            *       *
|          *   verify  *
|        *      on?      *--------------+
```

```
|                *       *               |
|                  *   *                 |
|                    *                   |
|                    |                   |
|                    v                   |


|            +---------------+           |
|            | verify card    |          |
|            |  VERIFY        |          |
|            |enable interrupt|          |
|            |  INTENA        |          |
|            +---------------+           |
|                    |                   |
|                    v                   |
|                    +<------------------+
|                    |
|                    v
|                    *
|                  *   *
|                *  last  *   yes
|              *  track?   *-----------+
|                *       *             |
|                  *   *               v
|                    *         +---------------+
|                    |         | clean up, leave |
|                    |         |    ENDIT        |
|                    |         +---------------+
|                    |
|                    v
|            +---------------+
|            | show track msg |
|            |  TRKMSG        |
|            +---------------+
|                    |
|                    |
+-------------------+
```

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

```
|            | show track msg |
|            |  TRKMSG        |
|            +---------------+
|                    |
|                    |
+-------------------+
```

Load card flowchart:

```
                +---------------+
                |   Load card    |
                +---------------+
                        |
                        v
                +---------------+
                | GET STARTED:   |
                | save reg-STRTUP|
                | get ram-ALCALL |
                | init track table|
                +---------------+
                        |
                        v
|  +------------------->+
|                       |
|                       v
|               +---------------+
|               |    GOCARD      |
|               |get card started|
|               +---------------+
|                       |
|                       v
|               +---------------+
|               | PROCESS CARD:  |
|               | read  file     |
|               |  header-READCR |
|               | check header   |
|               | read  data     |
|               |  READCR        |
|               |enable interrupt|
|               |  INTENA        |
|               +---------------+
|                       |
|                       v
|                       *
|                     *   *
|                   *  last  *   yes
|                 *  track?   *-----------+
|                   *       *             |
|                     *   *               v
|                       *         +---------------+
|                       |         | clean up, leave |
|                       |         |    ENDIT        |
|                       |         +---------------+
|                       |
|                       v
|               +---------------+
```

Verify card flowchart:

```
        +---------------+
        |  Verify card   |
        +---------------+
                |
                v
        +---------------+
        | GET STARTED:   |
        | save old restart|
        | information    |
        +---------------+
                |
                v
        +---------------+
        |    GOCARD      |
        |get card started|
        +---------------+
                |
                v
        +---------------+
        | PROCESS CARD:  |
        | read  file     |
        |  header-READCR |
        | verify header  |
        |  VYHDSM        |
        | verify data    |
        |  RDVFY         |
        +---------------+
                |
                v
        +---------------+
        | restore old    |
        |  restart info  |
        +---------------+
                |
                v
        +---------------+
        |    return      |
        +---------------+
```

```
 .----------------.______
(      _ _  o          _ )
 ( _ _ ‾welcome to  _ _‾ )
  (   deallocation   ‾‾ )
  (_________o           )
            |\---------'
            | |
            | |
            | |
            | |
          \ | | /
       ...\\| |//.... ..
          ... .. ...
             ....
```


Gary K. Cutler

DEALLOCATION	CHAPTER 1

1.1 Definition

Definition: Deallocation is a clean up process. When certain Basic commands are executed, which involve a parameter file(s) that is (are) allocated or an error in allocation has occurred, the process of Deallocation is initiated. Deallocation deletes the program's environment (if necessary), replaces all relative pointers in the program with variable names or BCD line numbers, zeros the values in the PCB (Program Control Block) and deletes the VPA (Variable Pointer Area).

1.2 Run/Call

At the conclusion of a RUN or CALL statement, the environment of the parameter file is removed from the environment stack by the routine POPENV. However, the Variable Pointer Area and the Program Control Block are still valid. This allows repeated execution of a program without repeated construction of the VPA and PCB.

1.3 Invocation

Basic Commands that Deallocate: In order to ensure the integrity of the system, some Basic commands require that the parameter file(s) be deallocated. To optimize these cases, a routine, SAFE', was created. SAFE' opens the parameter file, checks the file for an allocated state and, if so, deallocates the file by calling the routine, DALLOC. If the parameter file is currently running, then SAFE' calls DALLAL to deallocate all files in memory. Following is a list of the Basic Commands which deallocate the file(s) in memory if necessary:

Command

COPY
DELETE
EDIT
MERGE
PRINT
PURGE
RENAME
TRANSFORM

1.4 Maintaining the Environment

At the conclusion of program execution, the program's environment, in many cases, is removed from the environment stack. The routine POPENV, which removes the programs environment, is called when an END statement is executed at runtime. Thus, in order to maintain the environment, prevention of execution of an END statement is necessary. This can be accomplished in three ways.

1) an error occurs at runtime (not during allocation)

2) insertion of a STOP statement before the END statement

3) hitting the ATTN key before executing an END statement

1.5 Variable Names

Name form: At the invocation of the deallocator, all variables have been replaced by relative pointers to their entry in the VPA. The routine VDALOC has the responsibility of locating the variable name form in the VPA and converting this into ASCII code. To accomplish this the routine ASCNAM is called. ASCNAM ands out all bits except for the ASCII name information (see Pointer Allocation Documentation) and subsequently replaces the relative pointer with the original ASCII code.

1.6 Deallocation Routines

DALLAL: DALLAL is the entry point for deallocating all files in memory. The sequence of operation is as follows:
1) pending error information is saved
2) locates directory and
   a) opens file
   b) checks for allocated state
   c) if allocated, calls DALLOC to deallocate
   d) checks for end of directory
3) cleans up the environment stack (except for calculator environment)
4) restores pending error information

DALLOC: DALLOC is responsible for deallocating the file in R40. This consists of:

1) clearing the flags DFPAR1, DCLCOM, NXTCOM and DIMFLG

2) initialize OP1BAS to the undefined state (8001h)

3) replace all relative pointers for variables with their ASCII name

4) replace all relative pointers for line numbers with their BCD value

5) initialize all values in the PCB to zero

6) delete the VPA

Input:

ERRSTP - If zero, deallocates all variables in the current program
- If non-zero, then deallocation ceases at the token pointed to by ERRSTP
- If non-zero, then allocation was halted by an error. This should be the only time ERRSTP is set.

P.LEN - the length of the program before allocation

P.CLEN - the length of VPA before PAL started

(used in recovery of errors from calcprog allocation)

NEXT: NEXT is the main loop for variable pointer deallocation. Deallocation (like allocation) is keyed by the class of the current token. Tokens with class >= 30 (octal) are non-allocatable and therefore deallocation is unnecessary. Tokens with class < 30 (octal) have been allocated and thus some type of deallocation is needed. GETNXT is the routine which obtains the token class for each token seen and NEXT then initiates the appropriate deallocation routine (if necessary) by table addressing. Here follows the deallocation table, with the deallocation routine name, the token class this routine is responsible for and a description of the types of tokens in the particular class.

1.7 Deallocation Table

Routine	Class	Descriptor
INIROM	-1	ROM class > 56
XDALL1	0	End-of-line
VDALOC	1	Fetch variable
BININT	2	Integer constant
SVALD	3	Store variable
SKPCON	4	Real constant
SKPCON	5	String constant
DALFNC	6	FN call
LINEDA	7	Jump true line
LINEDA	10	Goto, Gosub
RELJMP	11	Jump relative
DALFN	12	DEF FN statement
FRET	13	DEF END statement
ROMCLA	14	Ext ROM (obsolete)
FRRET	15	Option base (RTN)
FRET	16	Function return
FRRET	17	Function let (RTN)
SKPNXT	20	Data
DIMD	21	Dim (RTN)
SHORTD	22	Short (RTN)
INTD	23	Integer (RTN)
COMMD	24	(just a RTN)
LINEDA	25	Else jump line
RELJMP	26	Else jump relative
LINEDA	27	Using line

1.8 Handi Call

The V.DALO handi event is generated when a token class > 56 (octal) is found. If a basic token, created by an external ROM, requires a unique deallocation routine not found in the system deallocator, then the primary attribute of this token should be > 56 (octal).

1.9 Globals

Name	Location	Description
DFPAR1	83B6	beginning of user defined function
DINFLG	838E	type of variable (integer,short or real)
DIRECT	854A	beginning of directory
ENDLIN	A999	internal endline #
ERLIN	8378	line # in which error occurred
ERRSTP	8391	location at which PAL quit
ERRTMP	836E	temporary for error information
OPTBAS	8282	option base flag
PCR	824D	program counter
PRFILE	8243	loc of parameter file
PRNAME	8263	name of parameter file
ROMOFF	82A5	offset to make ROMPTR absolute
ROMPTR	82A3	relative pointer to current ROM
VARPTR	838F	pointer to variable environment location

1.10 Cross References

Memory Management Document	RH"MEM
Pointer Allocation Document	GC"ALO
Environment Allocation Document	GC"ALO
Handi Call Document	RH"HDI
Source File	RH&DAL
Global File	KR&GLO

Table of Contents

DECOMPILER

Gary K. Cutler

DECOMPILER	CHAPTER 1

1.1 INTRODUCTION

Decompiling is the process of listing a program or statement. Internally, it requires the reconstruction of input code. The tokens, which have been parsed into RPN and distributed in the system, must be re-assembled into infix notation. Thus decompiling is actually the reverse of parsing and compiling.

Decompiling is a two-stack operation. An expression stack is used to reconstruct expressions from RPN to their original form, and an output stack is used to buffer the output. R12 is used as the expression stack pointer.

In decompilation there are two important parameters of a given operator-- its precedence and its position on the R12 stack. Following is list of operators and their precedences:

OPERATORS	PRECEDENCES (octal)
any operator enclosed in ()'s	100
^	14
* or /	12
+ or - or unary -	7
log. bin. rel	6
log. and	4
log. or	2

When a statement is decompiled, first the line number and then (if present) a LET token are put on the output buffer, because they don't change. The rest of the procedure is performed with the expression (R12) stack. When variables are parsed, their order is never changed. So, in decompilation, the variables are put on the expression stack as they appear in the parsed statement, with each variable name preceded by a stack marker (0E) and its token. As operators are encountered, they are inserted in the most recent position. When an operator is inserted, its precedence and location is checked against the stored precedences and locations of the previous operators that were inserted. The routine OPTST then tests the current operator's information against all previous operators' information to determine if parentheses are needed (see appendix for more detailed information).

In decompiling, the system processes each token and uses its class (the token class is the two right most octal digits of the token) to determine how the token is to be decompiled. The token class is code for the routine which decompiles the particular class, i.e., a table look-up is implied-- each class has its own routine(s) to decompile that class. Here are some common token classes, and their action:

CLASS	TOKEN-TYPE	ACTION
0	End-of-line	Unstack (dump line out of stack).
1	Fetch variable	Send to expression stack.
2	Integer	Send to expression stack.
3	Store variable	Send to expression stack.
4	Numeric constant	Send to expression stack.
5	String constant	Send to expression stack.
32	Subscript e.g. A(3)	() to expression if token odd, else (,) to expression stack.
34	DIM subscript	Action same as class 32.
36	Prints	Unstack, and push , to output.
41	Other reserved words	If : then unstack; else output reserved, and unstack.
42	Miscellaneous output	If then push to expression stac and unstack; otherwise output
44	Not seen by user	Ignore e.g. implied LET token
50	Unary operator	Insert after most recent stack marker and determine the necessity of parentheses.
51	Binary operator	Replace most recent missing operator in expression stack and determine the necessity of parentheses.
52	String unary operator	Same action as class 50.
53	String binary operator	Same action as class 51.
55	System function	Compute the number of parameters, locate the parameter string, insert the function name and parentheses if

		necessary after the most recent stack marker.
56	String system function	Same action as class 55.

The number of operators already decompiled is stored in PRECNT. The precedences of the operators and their positions on the expression (R12) stack are saved in an array addressed by LAVAIL.

The following routines are important to the decompiler:

1) DECOM	6) FETVAR
2) DCLINN	7) STOVAR
3) BSCLN	8) BINCON
4) SYSFUN	9) EXPAND
5) BINOP	10) UNSTAK

1) DECOM
This routine is the entry point into the decompiler, and clears PRECNT and then calls DCLINN (PRECNT contains the number of operators decompiled).

2) DCLIN#
This decompiles the line number-- the line number is converted to decimal, normalized, and put in the decompilation buffer, followed by a blank. The pointer into the decompilation buffer (R30) is then saved, and control is passed to BSCLN.

3) BSCLN
BSCLN (and ensuing subroutine calls) decompiles the rest of the line. As each token is obtained--via GETNEXT--its token class (right most two octal digits of token) is checked, and the proper decompilation for that class is called.

4) SYSFUN
SYSFUN decompiles system functions, by looking in the functions ASCII table for the ASCII name and parameter count of the function. The ASCII name is pushed on the expression stack, and then the closing parentheses must be added; space is inserted in the expression stack (by EXPAND) and the parenthesis is inserted in the proper location.

5) BINOP
BINOP decompiles binary operator analogously to SYSFUN. The ASCII for the operator is inserted in the expression stack (in the most recent spot between two variables or constants), determination of the necessity of parentheses is made, PRECNT is incremented by one, the precedence and location of the operator are then stored at the location pointed to by LAVAIL.

6) BINCON
BINCON converts a three byte integer into a normalized real constant for output to the LCD.

7) EXPAND
EXPAND is called when space is needed on the expression stack between TOS and the R12 stack pointer. To do this, EXPAND calls ALLOC, which in turn, call ADJUST and COPY.

8) FETVAR
FETVAR decompiles variable tokens that are used in an expression. FETVAR pushes E, token class, and the variable name (E is a marker on the expression stack for any variable) on the expression stack.

9) STOVAR
STOVAR does the same thig as FETVAR for variables being assigned by the statement.

10) UNSTAK
UNSTAK is the routine that processes an end-of-line or token class 41, by dumping the expression stack into the decompilation buffer.

1.2 GLOBALS

Name	Location	Description
DCMLEN	837F	decompile length (not greater than 96)
DCOVFL	82C8	overflow flag
EDFILE	8245	location of current edit file
LAVAIL	825B	pointer to operators' location and prec
NXTMEM	8253	next in available user memory
ONFLG	83D3	on gosub goto flag
PRECNT	8364	# of decompiled operators
PRFILE	8243	loc of current parameter file
RM.ASC	0004	offset to ROM ASCII table
ROMOFF	82A5	offset to make ROMPTR absolute
ROMPTR	82A3	rel pointer to current ROM
TOS	8257	current top of stack

1.3 HANDI CALLS

V.DEC -- token with attribute >= 57 (octal)

1.4 CROSS REFERENCES

Handi Call Document	RK"HDI
Internal Code Examples Document	RK"ICE
Source File	KR&DEC
Global File	KR&GLO

CLOSER INSPECTION	CHAPTER 2

2.1 REGISTERS

R12: expression stack pointer

R30: decompilation buffer pointer

R76(LAVAIL): ptr to precedence and location of all previous decompiled operators in current line

R75(PRECNT): ptr to count of decompiled operators in current line

R23: current token

R24: ptr into current line

R36: initially, token class

R45/46: current line number

R47: number of bytes left in current line

TOS: beginning of decompiled line on R12 stack

PRECNT: # of decompiled operators in current line

LAVAIL: precedence and location of all previous decompiled operators in current line

2.2 EXPAND and OPERATORS

EXPAND: The routine EXPAND is called when space is needed on the R12 stack intermediate to the TOS and the stack ptr. At entry, R36 contains the location in the R12 stack at which the required space will be allocated, and R32 contains the number of bytes of space needed. The actual expansion is done in the routine ALLOC which also calls ADJUST and COPY. This routine

takes the contents of each address from the initial point down through the R12 stack and copies them sequentially starting at the inputted (R32) number of bytes after the initial address (R36).

EXAMPLE: Expand three bytes at R36 ptr.

```
R12 stack pre-expand              R12 stack post-expand

       A                                 A
       B                                 B
       *  <--R36 ptr                     * <--R36 ptr
       ^                                 ^

       2                                 2
      SIN <--R34 ptr                     * <--R34 ptr
       F                                 ^
          <--R12 ptr
                                         2
                                        SIN
                                         F
                                           <--R12 ptr
```

Note: The only ptr into the R12 stack which is adjusted is the R12 stack ptr.(R36 and all other pointers remain absolute, not relative, to the stack)

STACK MARKERS: E: place marker for unary and binary operators, system functions and user def fns
F: keys end for the routine UNSTAK

PRECEDENCE:

operator	precedence(octal)
(any op)	100
^	14
*,/	12
+,-,unary -	7
log bin rel	6
log and	4
log or	2

In decompilation there are always two characteristics of an operator under consideration, precedence and position (in R12 stack). Two important values in this procedure are PRECNT (the number of operators already decompiled) and LAVAIL (all decompiled operators precedence and location on the R12 stack). After each new operator is obtained and its position on the R12 stack is located, tests are made, relative to each preceding operator's precedence and location, to determine if parenthesis are needed for a nested expression.(Note: if user used parenthesis in an expression where not required, the above test, in many cases, will fail and after decompilation no parenthesis will appear)

EXAMPLES: not required preserved — not required eliminated — not required eliminated

USER:	x=a*(b/c)	x=(a*b)/c	x=a*(b^c)
PARSE:	x a b c / *	x a b * c /	x a b c ^ *
DECOM:	x=a*(b/c)	x=a*b/c	x=a*b^c

PREC LOC STORAGE: (parsed expression) A B C + * E F 2 ^ / -
(user expression) A*(B+C)-E/F^2

```
 ___________
|     7     |<--R76(LAVAIL) (decreasing stack)
|___________|
|2 byte loc |
|   of -    |
|___________|
|    12     |
|___________|
|2 byte loc |
|   of /    |
|___________|
|    14     |
|___________|
|2 byte loc |
|   of ^    |
|___________|
|    12     |
|___________|
|2 byte loc |
|   of *    |
|___________|
|    100    |
|___________|
|2 byte loc |
|   of +    | <--original LAVAIL
|___________|
```

2.3 DECOMPILATION TABLE

Routine	Class	Token
DEF INIROM	-1	ROM class > 56
DEF EOL	0	end of line
DEF FETVAR	1	fetch variable
DEF BINCON	2	bin integer
DEF STOVAR	3	store variable
DEF CONST	4	num float or str const
DEF SCNST	5	string constant
DEF UFNCAL	6	user function call
DEF JMPLN	7	conditional jmp line N
DEF GOLINE	10	goto and gosub
DEF JMPREL	11	jmp relative
DEF UFNDEF	12	user fn def
DEF FNEND	13	fn end
DEF EXTROM	14	external ROM
DEF RESWD	15	option base
DEF FNRTN	16	user fn return
DEF FNASGN	17	fn assign
DEF RESWD	20	data
DEF RESWD+	21	dim/real
DEF RESWD+	22	short
DEF RESWD+	23	integer
DEF INIROM	24	undefined
DEF EJMPN	25	else jmp line N
DEF EJMPR	26	else jmp relative
DEF ULINN	27	using line N
DEF ON	30	on
DEF PU=	31	store
DEF SUBSCR	32	subscripts
DEF DEFKY	33	def key
DEF DIMSUB	34	dim subscripts
DEF PRNEOP	35	print EOL
DEF PRINTS	36	print stuff
DEF INPUT	37	input w/w/o a prompt
DEF RETURN	40	immed execute
DEF RESWD	41	other reserved words
DEF MISC	42	misc output
DEF MSTOR	43	multi store
DEF RETURN	44	misc ignore
DEF PRTFUN	45	print functions
DEF SYSFUN	46	numeric pseudo-function
DEF SYSFUN	47	special filenames
DEF UNOP	50	num unary operator
DEF BINOP	51	num binary operator

DEF UNOP$	52	str unary operator
DEF BINOP$	53	str binary operator
DEF RETURN	54	immed execute
DEF SYSFUN	55	num function
DEF SYSFN$	56	str function

ACTION OF DECOMPILER ON A BASIC PROGRAM STATEMENT

USER: 10 A=X*(Z+SIN(Y))^2/T

PARSE: 10 A X Z Y SIN + 2 ^ T / * [STORE TOK] E

DECOM, the system decompiler, is our entry for each line to be decompiled. DECOM clears PRECNT, ONFLAG, and COMFLG then jumps to DCLINN to decompile the line number. The line number A,0 is converted to decimal, normalized and placed in the decompilation buffer, followed by a blank byte. The ptr (R30) to the decom buffer is saved, pointing to the second byte after the line number. Control is now passed to BASCLN which will direct the remainder of the decompilation of this line. As each token is obtained (GETNXT) its token class (right most two digits of the primary attribute) keys the proper decompilation routine. In the above example A with token class 3 directs control to STOVAR. STOVAR replaces A's token (11) with the token 1 (numeric variable) and then pushes a stack marker (E), the token (1) and the variable name (A) onto the R12 stack. X with token class 1 passes control to FETVAR. FETVAR examines the type of variable (array, numeric) and then pushes E, 1, and X onto the stack. In a comparitive manner the variables Z and Y are also manipulated, leaving the R12 stack, at this time, as:

R12 stack: [tos] E 1 A E 1 X E 1 Z E 1 Y

SIN, token D8, with token class 55, directs control to the systems function decompile routine, SYSFUN. SYSFUN utilizes the token D8 to locate the ASCII name from the routines ASCII table and the functions parameter count. The ASCII name is pushed to the decom buffer and the location of the name (on the R12 stack) is determined by the parameter count. Since, in this case, the count is one, the highest addressed E is located on the stack and EXPAND is called to allocate three bytes of space for the insertion of SIN. Once the appropriate space has been allocated SIN is pushed into the stack. (Note: In the case of a system or user defined function, the stack marker is NOT overwritten. This same marker must be preserved as a locator for the appropriate unary or binary operator.) Before leaving SYSFUN, however, the parameter must be inclosed. The appropriate location for the parenthesis are located, the R12 stack is expanded and the parenthesis are placed. The next token 2B (+), with class 51, directs control to BINOP. BINOP locates the ASCII name and saves it on the decom buffer. The highest addressed E is located and the + overwrites the E on the stack. PRECNT is

incremented (now 1) and the precedence and location are stored beginning at LAVAIL. 2 (token 1A) with class 2, sends control to BINCON. BINCON converts the three byte integer to an eight byte normalized real and replaces the token 1A with the token 4 (real constant). E, 4, 2 (eight byte real) are then pushed onto the R12 stack. The token 30 (^) and its class, 51, are examined and control passes to BINOP. BINOP obtains the ASCII name and places the name in the decom buffer. The correct marker (E) is located and the ^ overwrites into the stack. Since PRECNT > 0 the previous operators are recalled from LAVAIL, one by one, and each precedence and location is compared to that of the current operator, in this case ^. If any comparison indicates the existence of a nexted expression, the location is determined and the parenthesis are inserted. In this case a nested expression is recognized, the R12 stack is expanded and parenthesis are appropriate placed. BINOP then replaces the precedence of the nested operator (in our case +) to 100 (octal), increments PRECNT and stores the pertinent information at LAVAIL.

The following token 1 (T) and its class, result in E, 1, T being placed on the R12 stack. 2F (/) with class 51 is now examined and control is directed to BINOP. The ASCII name is found and stored on the decom buffer, while the next stack marker E is located. / overwrites the marker and the precedence and location comparisons are made. All fail, so PRECNT is incremented and all operators precedence and location are restored. The last operator *, token 2A with class 51, is examined and control is again passed to BINOP. As above, the procedure results in the * overwriting the appropriate E, comparisons made and failed, PRECNT incremented, and the operators precedence and location restored. The next token 8 (store value) sends control to PU=, where the next available stack marker is located and overwritten with an =.

The end of line token (E) is obtained, which eventually directs control to UNSTAK. UNSTAK trashes the first stack marker, trashes all variable tokens, replaces any remaining stack markers with commas, converts all numeric values to decimal form and finally places the decompiled statement onto the decompilation buffer.

Status of R12 stack and decompilation buffer before EOL:

R12 stack: [tos] E A = X * (Z + SIN (Y)) ^ 2 / T

Decom buffer: 10

ACTION OF THE DECOMPILER ON THE PROGRAM STATEMENT

USER: 20 DEF FNR=SQR(X^2+Y^2)

PARSER: 20 [DEF FN] R X 2 ^ Y 2 ^ + [SQR]

In the above example, decompilation initiates with DECOM and then directs control to BASCLN. BASCLN gets the line number (decimal) and places this on the decompilation buffer. The next token is obtained (GETNXT) and control is passed to UFNDEF (user def fn decom routine). UFNDEF places the ASCII code for DEF FN on the decom buffer and sends control to FNNAMM. FNNAMM locates the name of the user def fn (R) and sends this to the decom buffer. Control returns to UFNDEF, which trashes the relative jump and obtains the parameter count/type byte. The number of parameters is determined and stored (R34) and the relative PCR is trashed. In our case (no parameters) an end of line token is tested. If not, then an = is pushed to the buffer and control is returned.

Decom buffer: 20 DEF FNR=

Subsequently, a stack marker (E), the token (1), and the variable name (X) is now pushed to the R12 stack. The next token (1A) sends control to BINCON, which converts the three byte integer to an eight byte real and places this on the stack, preceded by an E and the token 4 (real constant). 30 (^) initiates a jump to BINOP, which locates the ASCII name and stores this on the decom buffer. The proper stack marker is located and overwritten by ^. PRECNT is incremented and the precedence and location of ^ are stored beginning at LAVAIL. E,1,Y are pushed to the stack, 2 is converted to an eight byte real and pushed after E and 4 and the next token 30 (^) is obtained. Control once more returns to BINOP. ^ overwrites the appropriate marker, its precedence and location are compared the the previous operators, which, in this case, causes no further action other than incrementing PRECNT and storing the operators' precedence and location. The final operator is obtained and examined. BINOP again resumes control and overwrites the marker with +. The comparisons are made, no nested expresions are recognized, PRECNT is incremented and the operators' precedence and location are stored. (SQR) is examined and control is passed to the system's function routine SYSFUN. The ASCII code is obtained and stored on the buffer. The stack marker is located and EXPAND is called to allocate three bytes of space on the R12 stack. SQR is inserted into the stack the beginning and ending of the parameter are located, two bytes are allocated

and the parenthesis are inserted. The end of line token directs control to EOL which in turn passes control to UNSTACK. UNSTACK trashes the initial stack marker (recall that DEF FNR= is already on the decom buffer with the ptr (R30) after the =) on the R12 stack, trashes all variable tokens, and pushes the remaining decompiled statement onto the decompilation buffer.

Status of R12 stack and decompilation buffer befor EOL:

R12 stack: [tos] E SQR (1 X ^ 4 [2] + 1 Y ^ 4 [2])

Decom buffer: 20 DEF FNR=

ACTION OF THE DECOMPILER ON THE PROGRAM STATEMENT

USER: 30 IF L$=CHR$(7*5) THEN 180 ELSE L$= "="

PARSER: 30 L$ 7 5 * CHR$ = [JTRUE] 150 [JREL] L$ "=" [STORE]

Dispensing with the initial action upon the line number we turn to the action upon the statement itself. The variable L$ is decompiled by the routine FETVAR, the constants 7 and 5 are handled by BINCON and the operator, *, is acted upon by BINOP. The result of this decompilaton is

R12 stack: [tos] E 1 $ L E 4 [7] * 4 [5]

The following token C2 (CHR$) and its class 55, directs control to SYSFUN. The sequence of action is as follows; the ASCII name for CHR$ is obtained and saved on the buffer, space is allocated for insertion of the name, CHR$ is pushed in, the number of parameters is determined, the parameters are located and space is allocated for the parenthesis, and finally the parenthesis are inserted. 35 (=) signifying a binary op sends control to BINOP which locates the next available E and pushes the = to the stack. Since the precedence of = is 6, no tests are made for a nested expression.

R12 stack: [tos] E 1 $ L = CHR$ (4 [7] * 4 [5])

The jump line # token, 18, sends control to JMPLN. Action is now directed towards the decom buffer. IF is placed on the buffer and control is passed to UNSTACK, which unstacks the R12 stack and places this part of the statement on the buffer, followed by then and line #.

Decom buffer: IF L$=CHR$(7*5) THEN 180

Control returns and 1C, jump relative, is examined. The routine initiated is EJMPR. First a check for ELSE LINE # is performed. The key is not found so the routine assumes ELSE STATEMENT. The ELSE is placed on the decom buffer (the number of bytes for rel jump is trashed). Control is returned and the remaining statement is decompiled and pushed on the R12 stack. The EOL is obtained, the remaining portion is cleaned up and placed on the decom buffer.

Status of R12 stack and decompilation buffer before the ??

R12 stack: [tos] E 1 $ L = 5 " = "

Decom buffer: 30 IF L$=CHR$(7*5) THEN 180 ELSE

Table of Contents

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

2:20 PM THU., 15 JULY, 1982

xviii

WHAT IS ICE

Ice is frozen water, something looking like frozen water, diamonds, or a serving of ice cream in Great Britan. ICE is the Internal Code Examples. In this document one will find a list of all the mainframe tokens, a collection of examples of the uses of all the tokens, all the data types that appear on the R12 stack, and a table of the attribute routines for decompile, pointer allocation and deallocation.

HOW TO READ THE TOKEN TABLE FORMAT

```
   The file that contains the runtime code  ___________________________
                                                                       |
   The token's attributes                   ________________           |
                                                            |          |
   The name of the runtime routine          __________      |          |
                                                      |     |          |
   The name of the parse time routine       _____     |     |          |
   In this case there is no parse time           |    |     |          |
   routine.                                      |    |     |          |
                                                 |    |     |          |
   ,______________ token number                  |    |     |          |
   |               in hex                        |    |     |          |
   |                                             |    |     |          |
   |     ,________ name of the                   |    |     |          |
   |     |         token                         |    |     |          |
   |     |                                       |    |     |          |
   |     |     ,__ ASCII string                  |    |     |          |
   |     |     |   for the token                 |    |     |          |
   |     |     |   (in this case                 |    |     |          |
   |     |     |   no string)                    |    |     |          |
   v     v     v                                 v    v     v          v
8  store num  ''                           <none> STOSV 0,31        RH&VAR
```

input from R12
   numval - value to be stored   <--- top value on the R12 sta
   nameform - name of target variable
   (dimflag) - if tracing, tracing information <-- value conditionally
   (col) - if tracing, tracing information   on R12
   (row) - if tracing, tracing information

   numvalptr - pointer to target value
output on R12
   (numval) - if multistore, value being stored

```
      ^
      |
      |__________ list of data objects placed on and removed from
                  the R12 stack
```

LIST OF DATA OBJECTS ON THE R12 STACK

nameform - is the internal representation of a variable. The two bytes that make up the name form are layed out as follows:

	bits							
	7	6	5	4	3	2	1	0
byte 1-	T3	T2	T1	T0	N3	N2	N1	N0
byte 2-	A1	A0	F0	L4	L3	L2	L1	L0

T3 - 0 = numeric
   - 1 = string

T2 - 0 = simple
   - 1 = array

(T1,T0) - 0 = real
        - 1 = integer
        - 2 = short
        - 3 = (not used)

(N3,N2,N1,N0) - 0-9 = 0 through 9 after letter in variable name
              - 10 = blank
              - 11 = greek variable

(A1,A0) - 0 = local variable
        - 1 = remote variable   \
        - 2 = parameter variable } remote access
        - 3 = common variable   /

F0 - 0 = variable
   - 1 = function

(L4,L3,L2,L1,L0) - (first letter)-"A"+1 = first letter of variable

dimflag - 1 byte that is 1 if 1 dimensional array and 0 if two dimensional array
row - 2 byte size of the row dimension
col - 2 byte size of the column dimension
numval - 8 byte numeric value
string - strptr on top the R12 stack followed by a strlen
strlen - 2 byte value of the number of characters in the string

strptr - 2 bytes pointer (may be relative to R12) that points to a strval
bseptr - a strptr that points to the beginning of a string from which a substring will be extracted.
FNparam - 4 or 8 bytes as determined by the FNparmtype that is at R10

LIST OF DATA OBJECTS IN THE CODE STREAM (from R10)

strlen1 - 1 byte length of the string
VPAptr - 2 byte pointer relative to the beginning of the file
reljmpaddr - 2 byte pointer relative to the beginning of the file pointing to the code to be executed next
FNblockptr - 2 byte pointer relative to the beginning of the file that points to the function parameter block
FNparmcount - 1 byte count of the number of parameters
FNparmtype - 1 byte flag indicating the type of the function parameter expected (80 if numeric and 81 if string)
strval - n bytes where n is determined by context this is the actual string of characters.

THE TOKEN TABLE

```
0  ERROR      ''          <none>  ERRORX  0,44   KR&TOK

1  num value  ''          <none>  FTSVL   0,1    RH&VAR
```

input from R10
   VPAptr - relative pointer to variable that contains the desired valu
output to R12 stack
   numval - the value of the variable

```
2  get VPA addr ''        <none>  SVADR   0,1    RH&VAR
```

input from R10
   VPAptr - relative pointer to variable that contains the desired valu
output to R12
   VPAptr - same relative pointer

```
3  get string  ''         <none>  FTSTL   0,1    RH&VAR
```

input from R10
   VPAptr - relative pointer to variable in VPA
output to R12
   strptr - pointer to string value
   strlen - length of that string

```
4  num constant ''                 <none>  ICONST  0,4     RH&VAR

 input from R10
    numval - numeric constant to be placed on the stack
 output to R12
    numval - the numeric value just fetched


5  "quoted str  ''                 <none>  SCONST  0,5     RH&VAR

 input from R10
    strlenl - the length of the string
    strval - the characters in the sting
 output to R12 stack
    strptr - location of value
    strlen - length of string


6  unquoted str ''                 <none>  SCONST  0,5     RH&VAR

 input from R10
    strlenl - the length of the string
    strval - the characters in the sting
 output to R12 stack
    strptr - location of value
    strlen - length of string


7  store string ''                 <none>  STOST   0,31    RH&VAR

 input from R12 stack
    strptr - source string address
    strlen - source string length
    strptr - target string address
    strlen - target string length
    VPAptr - pointer to target string VPA
    bseptr - pointer to base addr of target <len field>

 output to R12 stack
    (strptr) - if multistore, source string pointer
    (strlen) - if multistore, source string length

8  store num    ''                 <none>  STOSV   0,31    RH&VAR

 input from R12
    numval - value to be stored
    nameform - name of target variable
    (dimflag) - if tracing, tracing information
    (col) - if tracing, tracing information
    (row) - if tracing, tracing information
    numvalptr - pointer to target value
 output
    (numval) - if multistore, value being stored


9  1-DIM ADR    ''                 <none>  AVADR1  0,32    RH&VAR
```

```
 input from R12

    row - dimension of array
    VPAptr - variable to be indexed
 output
    nameform - name of variable being referenced
    (dimflag) - if tracing, tracing information
    (col) - if tracing, tracing information
    (row) - if tracing, tracing information
    numvalptr - pointer to array element


A  2-DIM ADR    ''                 <none>  AVADR2  0,32    RH&VAR

 input from R12
    col - column dimension of array
    row - row dimension of array
    VPAptr - variable to be indexed
 output
    nameform - name of variable being referenced
    (dimflag) - if tracing, tracing information
    (col) - if tracing, tracing information
    (row) - if tracing, tracing information
    numvalptr - pointer to array element


B  1-dim value  ''                 <none>  AVVAL1  0,32    RH&VAR

 input from R12
    col - column dimension of array
    row - row dimension of array
    VPAptr - variable to be indexed
 output to R12
    numval - value of array element

C  2-dim value  ''                 <none>  AVVAL2  0,32    RH&VAR

 input from R12
    col - column dimension of array
    row - row dimension of array
    VPAptr - variable to be indexed
 output to R12
    numval - value of array element


D  carriage rtn ''                 <none>  ERRORK  0,44    KR&TOK


E  end of line  ''                 <none>  GORTN   0,0     RH&VAR


F               ''                 <none>  ERRORK  0,44    KR&TOK


10 invisible RTN''                 <none>  INVRTN  0,44    RH&EAL


11 numeric addr ''                 <none>  FTADR   0,3     RH&VAR
```

```
 input from R10


    VPAptr - variable to be fetched
 output to R12
    nameform - name form of the variable
    numvalptr - pointer to the value


12 get num addr ''                 <none>  SVADR+  0,3     RH&VAR

 note: calls SVADR with different attribute
 input from R10
    VPAptr - relative pointer to variable that contains the desired valu
 output to R12 stack
    VPAptr - same relative pointer


13 SAVE STR     ''                 <none>  FTSTLS  0,3     RH&VAR

 input from R10
    VPAptr - relative pointer to variable in VPA
 output
    strptr - pointer to string value
    strlen - length of that string
    VPAptr - relative pointer to variable in VPA
    bseptr - pointer to base address of target <len field>


14 MULTI STO.   ''                 <none>  STOSVM  0,43    RH&VAR

 note: calls STOSV with different attribute
 input from R12
    numval - value to be stored
    nameform - name of target variable
    (dimflag) - if tracing, tracing information
    (col) - if tracing, tracing information
    (row) - if tracing, tracing information
    numvalptr - pointer to target value
 output
    (numval) - if multistore, value being stored


15 MULTI STO$   ''                 <none>  STOSTM  0,43    RH&VAR

 note: calls STOST with different attribute
 input from R12 stack
    strptr - source string address
    strlen - source string length
    strptr - target string address
    strlen - target string length
    VPAptr - pointer to beginning of target string
    bseptr - pointer to base address of target <len field>

 output to R12 stack
    (strptr) - if multistore, source string pointer
    (strlen) - if multistore, source string length
```

```
16 num FN call  ''                 <none>  FNCAL.  0,6     RH&VAR


 note: same as FNCAL$
 input from R10
    FNblockptr - relative pointer to FN variable block
    FNparmcount - function parameter count
    (FNparmtype) - function parameter type 1
    (FNparmtype) - function parameter type 2
          .
          .
          .
    (FNparmtype) - function parameter type n

 input from R12
    (FNparam) - function parameter n
    (FNparam) - function parameter n-1
          .
          .
          .
    (FNparam) - function parameter 1


17 str FN call  ''                 <none>  FNCAL$  0,6     RH&VAR

 note: same as FNCAL.

 input from R10
    FNblockptr - relative pointer to FN variable block
    FNparmcount - function parameter count
    (FNparmtype) - function parameter type 1
    (FNparmtype) - function parameter type 2
          .
          .
          .
    (FNparmtype) - function parameter type n

 input from R12
    (FNparam) - function parameter n
    (FNparam) - function parameter n-1
          .
          .
          .
    (FNparam) - function parameter 1



18 JMP  TRUE    ''                 <none>  JTRUE#  0,7     RH&FOR

 input from R10:
    reljmpaddr - relative address to jump to
 input from R12:
    numval - value to be tested


19  input tail  ''                 <none>  ITAIL.  0,44    IV&INP
```

```
1A INT CONST   ''                      <none> INTCON 0,2    RH&VAR

 input from R10
    intval - integer constant to be placed on the stack
 output to R12 stack
    numval - integer in numval form

1B JMP FALSE   ''                      <none> JFALSR 0,11   KR&TOK

 input from R10:
    reljmpaddr - relative address to jump to

 input from R12:
    numval - value to be tested

1C JMP REL     ''                      <none> JMPREL 0,26   KR&TOK

 input from R10:
    reljmpaddr - relative address to jump to

1D 1 substring ''                      <none> SUBST1 0,34   RH&VAR

 input from R12
    numval - the substring character number
    strptr - pointer to string value
    strlen - string length
 output to R12
    strptr - new pointer to string value
    strlen - new string length

1E 2 substring ''                      <none> SUBST2 0,34   RH&VAR

 input from R12
    numval - the second substring subscript number
    numval - the first substring subscript number
    strptr - pointer to string value
    strlen - string length
 output to R12
    strptr - new pointer to string value
    strlen - new string length

1F ELSE JM     ''                      <none> EJMPM  0,25   KR&DEC

 note: EJMPM is just JMPLNM with different attributes

 input from R10:
    reljmpaddr - relative address to jump to

20             ''                      <none> ERRORX 0,44   KR&TOK

21             ''                      <none> ERRORX 0,44   KR&TOK
```

```
22 Array PRINTM ''                     <none> PMARRAY 0,36  MJ&TXT

  input from R12:
     VPAptr - array to be printed (fetched using FETSET)

  notes: does a fetnum and prMval for each item

23              ''                     <none> ERRORX 0,44   KR&TOK

24 Array READM  ''                     <none> RMARRAY 0,44  MJ&TXT

 input from R12:
    VPAptr - array to be read (fetched using FETSET)

 notes: does a read (NUM) for each element

25              ''                     <none> ERRORX 0,44   KR&TOK

26 & concatinate'&'                    <none> CONCA. 7,53   KR&FUN

 input on R12:
    string - second string
    string - first string
 output on R12:
    string - first string concatinated to second string

27 ;            ';'                    <none> PRSEM. 0,41   MJ&TXT

28 (            '('                    <none> ERRORX 0,44   KR&TOK

29 )            ')'                    <none> ERRORX 0,44   KR&TOK

2A *            '*'                    <none> MPYROI 12,51  KR&MTH

 input on R12:
    numval -   factor
    numval -   factor
 output on R12
    numval -   product

2B +            '+'                    <none> ADDROI 7,51   KR&MTH

 input on R12
    numval -   addend
    numval -   addend
 output on R12
    numval -   sum
```

```
2C ,           ','                     <none> ERRORX 0,44   KR&TOK

2D - diadic    '-'                     <none> SUBROI 7,51   KR&MTH

 input on R12:
    numval - Y
    numval - X
 output on R12:
    numval - X-Y

2E .           '.'                     <none> ERRORX 0,44   KR&TOK

2F /           '/'                     <none> DIV2   12,51  KR&MTH

 input on R12:
    numval - Y
    numval - X
 output on R12:
    numval - X/Y

30 ^ X^Y       '^'                     <none> YTX5   14,51  KR&MTH

 input on R12
    numval - Y
    numval - X
 output on R12:
    numval - X^Y

31 #           '#'                     <none> UNEQ$. 6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare

32 <=          '<='                    <none> LEQ$.  6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare

33 >=          '>='                    <none> GEQ$.  6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare
```

```
34 <>           '<>'                   <none> UNEQ$. 6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare

35 =            '='                    <none> EQ$.   6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare

36 >            '>'                    <none> GR$.   6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare

37 <            '<'                    <none> LT$.   6,53   IV&OP

 input on R12:
    string - first string to compare
    string - second string to compare
 output on R12:
    boolval - result of compare

38 - monadic    '-'                    <none> CHSROI 7,50   KR&MTH

 input on R12:
    numval - operand
 output on R12:
    numval - additive inverse

39 #            '#'                    <none> UNEQ.  6,51   IV&OP

 input on R12:
   numval - first number to compare
   numval - number to compare against
 output on R12:
   boolval - result of compare

3A <=           '<='                   <none> LEQ.   6,51   IV&OP

 input on R12:
   numval - first number to compare
   numval - number to compare against
```

```
  output on R12:
    boolval - result of compare


3B  >=        '>='              <none>  GEQ.     6,51     IV&OP

  input on R12:
    numval - first number to compare
    numval - number to compare against
  output on R12:
    boolval - result of compare


3C  <>        '<>'              <none>  UNEQ.    6,51     IV&OP

  input on R12:
    numval - first number to compare
    numval - number to compare against
  output on R12:
    boolval - result of compare


3D  =         '='               <none>  EQ.      6,51     IV&OP

  input on R12:
    numval - first number to compare
    numval - number to compare against
  output on R12:
    boolval - result of compare


3E  >         '>'               <none>  GR.      6,51     IV&OP

  input on R12:
    numval - first number to compare
    numval - number to compare against
  output on R12:
    boolval - result of compare


3F  <         '<'               <none>  LT.      6,51     IV&OP

  input on R12:
    numval - first number to compare
    numval - number to compare against
  output on R12:
    boolval - result of compare


40  @          '@'              <none>  ATSIGN   0,42     KR&TOK

41  ON ERROR   'ON ERROR'       ONERRO  ONERR.   0,341    KR&TOK

42  OFF ERROR  'OFF ERROR'      P1ANC!  OFFER.   0,241    KR&TOK

43  DEF KEY    'DEF KEY'        DEFKEY  DEFKY.   0,241    IV&OK

44  inv FN LET ''               FNLET   FNLET.   0,217    RH&VAR
```

```
  note: same as LET FN


  input from R10:
    FNtype - indicates the type returned by the function
    VPAptr - relative pointer to variable that contains the
  output from R12:
    if numeric then
     (nameform) - of the function variable
     (numvalptr) - pointer to the area where the value will be put
    if string then
     (strptr) - pointer to string
     (strlen) - string length
     (VPAptr) - pointer to VPA entry for the string


45  AUTO          'AUTO'            GO12N   AUTO.   0,241    KR&TOK

46  CAT ALL       'CAT ALL'         PUSH1A  CATAL.  0,241    RH&CAT

47  LISTIO        'LIST IO'         PUSH1A  LSTIO.  0,241    MJ&DIO

48  CAT$          'CAT$'            ERRORX  CAT$.   20,56    RH&CT$

49  DISPLAY IS    'DISPLAY IS'      G1$OR*  DSPIS.  0,241    MJ&DIO

4A  CAT           'CAT'             PUSH1F  CAT.    0,241    RH&CAT

4B  LIST          'LIST'            G$012N  LIST.   0,241    RH&CMD

4C  NAME          'NAME'            GET1$   NAME.   0,241    RH&RUN

4D  DELAY         'DELAY'           GET1N   DELAY.  0,241    KR&TOK

4E  MERGE         'MERGE'           G$'012  MERGE.  0,241    RH&RUN

4F  CALL          'CALL'            GET1$   CALL.   0,241    RH&RUN

50  READ#         'READ #'          READN   READ#.  0,241    MJ&TXT

  input:
    optional record number on r12
    file number on r12
  then does a read(NUM), read (STR), or read(ARRAY) for each item


51  FETCH a key   'FETCH KEY'       GET1$   FETK.   0,241    IV&OK

52  display width'WIDTH'            GET1N   WIDTH.  0,241    KR&TOK

53  POP return    'POP'             P1ANC!  POP.    0,241    RH&FOR

54  RUN           'RUN'             GO1N$   RUN.    0,241    RH&RUN

55  REAL          'REAL'            TYPSTM  SKIPR   0,321    IV&DCL

56  DISP          'DISP'            PRINT   DISP.   0,241    KR&TOK

57  FETCH         'FETCH'           GO1N$   FETCH.  0,241    RH&FET
```

```
58  printer width'PWIDTH'          GET1N   PWIDT.  0,241    KR&TOK


59  DEFAULTON/OFF'DEFAULT'         ON/OFF  DEFAL.  0,241    KR&TOK

5A  GOTO         'GOTO'            GOTOPR  JMPLN# 0,210     RH&FOR

  input from R10:
    reljmpaddr - relative address to jump to


5B  GOSUB        'GOSUB'           GOTOSU  JMPSUB 0,210     RH&FOR

5C  PRINT #      'PRINT #'         PRINT#  PRIN#.  0,241    MJ&TXT

  input from R12:
    optional record number on r12
    filenumber on r12


5D  MARGIN n     'MARGIN'          GET1N   MARGN.  0,241    IV&ED

5E  RESTORE #    'RESTORE #'       G1OR2N  REST#.  0,241    MJ&TXT

  input from R12:
    optional record number on r12
    filenumber on r12

5F  INPUT        'INPUT'           INPUT   INPUT.  0,237    IV&INP

60  ASSIGN#      'ASSIGN #'        ASSIGN  ASSIN.  0,241    MJ&TXT

  input from r12:
    lenght and address of filename
    file number (integer constant)
    optional 2byte filetype


61  LET FN       'LET FN'          FNLET   FNLET.  0,217    RH&VAR

  FNtype (1 byte) - 1 for numeric, 2 for array, 3 for string
  input from R10:
    FNtype - indicates the type returned by the function
    VPAptr - relative pointer to variable that contains the
             value to be returned.
  output from R12:
    if numeric then
     (nameform) - of the function variable
     (numvalptr) - pointer to the area where the value will be put

    if string then
     (strptr) - pointer to string
     (strlen) - string length
     (VPAptr) - pointer to VPA entry for the string
62  LET          'LET'             LET     NOP.    0,241    KR&TOK

63  STANDBY      'STANDBY'         ON/OFF  STAND.  0,241    MJ&PIL

64  ON TIMER#    'ON TIMER #'      ONTMR   TMRON.  0.341    RY&TIM
```

```
65  OFF TIMER#    'OFF TIMER #'     OFFTMR  TMROF.  0,241    RY&TIM


66  ON            'ON'              ON      ON.     0,230    RH&FOR

67  BYE           'BYE'             PUSH1A  BYE.    0,241    IV&ZZZ

68  WAIT          'WAIT'            GET1N   WAIT.   0,241    KR&TOK

69  PROTECT card  'PROTECT'         PUSH1A  CRDWPR  0,241    RY&CRD

6A  PRINTER IS    'PRINTER IS'      G1$OR*  PRINS.  0,241    MJ&DIO

6B  PRINT         'PRINT'           PRINT   PRINT.  0,241    KR&TOK

6C  printlist     'PLIST'           G$012N  PLIST.  0,241    RH&CMD

6D  RANDOMIZE     'RANDOMIZE'       TRY1N   RNDIZ.  0,241    MJ&RIZ

6E  READ          'READ'            READ    READ.   0,241    MJ&TXT

6F  RESTOREIO     'RESTORE IO'      PUSH1A  REST.   0,241    MJ&DIO

70  RESTORE       'RESTORE'         RESTOR  RESTO.  0,241    MJ&TXT

71  RETURN        'RETURN'          P1ANC!  RETRN.  0,241    RH&FOR

72  UNPROTECT     'UNPROTECT'       PUSH1A  CRDUPR  0,241    RY&CRD

73  EDIT filenam  'EDIT'            EDIT    EDIT.   0,241    RH&RUN

74  OFFIO         'OFF IO'          PUSH1A  OFFIO.  0,241    MJ&DIO

75  STOP          'STOP'            P1ANC!  STOP.   0,241    RH&RUN

76  PUT <key>     'PUT'             GET1$   PUT.    0,241    IV&IO

77  TRACE FLOW    'TRACE FLOW'      PUSH1A  TRFLO.  0,241    IV&TR

78  TRACE OFF     'TRACE OFF'       PUSH1A  TROFF.  0,241    IV&TR

79  TRACE VAR     'TRACE VARS'      PUSH1A  TRVAR.  0,241    IV&TR

7A  ENDLINE       'ENDLINE'         GO1$    EOL.    0,241    MJ&PIL

7B  CLEAR VARS    'CLEAR VARS'      PUSH1A  CLRVA.  0,241    KR&EXE

7C  COPY          'COPY'            FLTOFL  COPY.   0,241    RH&MEM

7D  PURGE filnm   'PURGE'           PUSH1F  PURGE.  0,241    RH&CMD

7E  RENAME fTOf   'RENAME'          FLTOFL  RENAM.  0,241    RH&CMD

7F  INTEGER       'INTEGER'         TYPSTM  SKIPI   0,323    IV&DCL

80  SHORT         'SHORT'           TYPSTM  SKIPS   0,322    IV&DCL

81  DELETE        'DELETE'          GO12N   DELET.  0,241    RH&CMD
```

```
82  ROM missing  'ROM missing'       ERRORX  ERRORX 0,44    KR&TOK

83  REMARK       'REM'               REM     SKIP!  0,241   IV&DCL

84  OPTION BASE  'OPTION BASE'       OPTION  OPTIO. 0,315   KR&TOK

85  END FN       'END DEF'           FNEND   FNRTN. 0,313   RH&VAR

 input from R10:
    VPAptr - pointer to VPA for function variable
 input from R12:
 output to R12:
   if numeric:
   (numval) - numeric value returned
   if string:
   (strptr) - pointer to the string (in RESMEM)
   (strlen) - length of string

86  DATA         'DATA'              DATA    SKIPEN 0,320   IV&DCL

87  DEF FN       'DEF FN'            DEF     SKPDEF 0,312   KR&TOK

 input R10:
    reljmpaddr - relative jump address to eol on END DEF line

88  DIM          'DIM'               DIM     SKIPD  0,321   IV&DCL

 input for allocation:
           array name:
                  row <integer const>
                  [col] <integer const>
        and/or
           string name:
                  max len <integer const>
 output:
     none

89  RENUMBER     'RENUMBER'          GOTO4N  RENUM. 0,241   RH&CMD

8A  END          'END'               P1RNC!  END.   0,241   RH&RUN

8B  remark       '!'                 REM     SKIP!  0,241   IV&DCL

8C  FOR          'FOR'               FOR     FOR.   0,341   RH&FOR

8D  IF           'IF'                IF      ERRORT 0,344   KR&TOK

8E  IMAGE        'IMAGE'             REM     SKIPIT 0,341   IV&DCL

8F  NEXT         'NEXT'              NEXT    NEXT.  0,341   RH&FOR

 input from R12:
    nameform - name form of the variable
    numvalptr - pointer to the value
```

```
90  BEEP         'BEEP'              BEEP    BEEP.  0,241   RY&BEE

91  LET (IMPLY)  ''                  ILET    ERRORT 0,344   KR&TOK

92  ASSIGN IO    'ASSIGN IO'         GO1$/*  CONFIG 0,241   MJ&DIO

93  clear loop   'CLEAR LOOP'        PUSH1A  CLOOP. 0,241   MJ&DIO

94  CONTINUE     'CONT'              GOIN    CONTI. 0,241   RH&RUN

95  CLEAR <dev>  'CLEAR'             GET1$   CLDEV. 0,241   MJ&DIO

96  'ed string   '"'                 <none>  SCONST 0,5     RH&VAR

97  TEXT         'TEXT'              <none>  TEXT.  0,46    RH&RUN

98  BASIC        'BASIC'             <none>  BASIC. 0,46    RH&RUN

99  LIF1         'LIF1'              <none>  LIF1.  0,46    RH&RUN

9A  RESULT       'RES'               PUSH1A  RESUL. 0,55    KR&FUN

9B  INTO         'INTO'              <none>  NOP2.  1,51    KR&TOK

 input/output
    none

9C  inv POP      ''                  <none>  INVPOP 0,44    RH&FOR

9D  clr tmrflg   ''                  <none>  TMRCLR 0,44    RY&TIM

9E  or           'OR'                <none>  OR.    2,51    IV&OP

9F  for/next to  'TO'                <none>  TO.    0,41    RH&FOR

 input from R12:
    numval - value for the upper bound of the loop

A0  using line   'USING'             <none>  ULIN#. 0,327   KR&TOK

A1  READ string  'READ'              <none>  READS. 0,44    MJ&TXT

A2  print end    ''                  <none>  PRLINE 0,35    KR&TOK

A3  string ;     ''                  <none>  PRSTR. 0,36    IV&PUN

A4  string ,     ''                  <none>  PRSTR. 0,36    IV&PUN

A5  print#;      ''                  <none>  SEMI#. 0,36    MJ&TXT

A6  print#,      ''                  <none>  COMA#. 0,36    MJ&TXT

A7  print#end    ''                  <none>  PRHEND 0,35    MJ&TXT

A8  ON of on/off 'ON'                <none>  ONTOK. 0,46    IV&ON

A9  OFF of on/off'OFF'               <none>  OFTOK. 0,46    IV&ON
```

```
AA  IP           'IP'                <none>  IP5    20,55   KR&MTH

 input on R12:
     numval - argument for the function IP

 output on R12:
     numval - integer portion <integer val> (flagged integer)

AB  EPSILON      'EPS'               <none>  EPS10  0,55    KR&MTH

 input:
    none

 output on R12:
    numval - lower bound of machine precision

AC  frac part    'FP'                <none>  FP5    20,55   KR&MTH

 input on R12:
     numval - argument for the function IP

 output on R12:
     numval - integer portion <integer val> (flagged integer)

AD  CEIL         'CEIL'              <none>  CEIL10 20,55   KR&MTH

 input on R12:
     numval - operand

 output on R12:
     numval - smallest integer >= operand <integer val> (flagged integer

AE  MAX          'MAX'               <none>  MAX10  40,55   KR&MTH

AF  inv FN END   ''                  <none>  FNRET. 0,16    RH&VAR

 input from R10:
    VPAptr - pointer to VPA for function variable

 input from R12:
   if numeric stuff returned
   (numval) - numeric value returned
   if string stuff returned
   (strptr) - pointer to the string (in RESMEM)
   if none returned due to failure

 output to R12:
   if numeric:
   (numval) - numeric value returned
   if string:
   (strptr) - pointer to the string (in RESMEM)
   (strlen) - length of string

    note: same as END DEF
```

```
B0  SQR          'SQR'               <none>  SQR5   20,55   KR&MTH

 input on R12:
     numval - operand

 output
     numval - principal square root of operand

B1  MIN          'MIN'               <none>  MIN10  40,55   KR&MTH
 input on R12:
     numval - operand
     numval - operand

 output on R12:
     numval - minimum value

B2  AVAIL MEMRY  'MEM'               <none>  MEM.   0,55    KR&FUN

B3  ABS          'ABS'               <none>  ABS5   20,55   KR&MTH
 input on R12:
     numval - operand

 output on R12:
     numval - absolute value of operand

B4  external rom ''                  <none>  ROM:GO 0,214   IV&RSW

B5  1 dim array  ''                  <none>  SVADCK 0,1     RH&VAR

B6  2 dim array  ''                  <none>  SVADCK 0,1     RH&VAR

B7  SGN          'SGN'               <none>  SGN5   20,55   KR&MTH

 input on R12:
     numval - operand

 output on R12:
     signum value of operand <integer const> (flagged int)

B8  KEY$         'KEY$'              <none>  KEY$.  0,56    KR&FUN

B9  COT          'COT'               <none>  COT10  20,55   KR&MTH

 input on R12:
     numval - argument

 output on R12:
     numval - cotangent of argument

BA  CSC          'CSC'               <none>  CSEC10 20,55   KR&MTH

 input on R12:
     numval - argument

 output on R12:
```

```
    numval - cosecant of argument

BB  APPT filename'APPT'          <none> APPT.  0,47    RH&FIL

BC  EXP         'EXP'            <none> EXP5   20,55   KR&MTH

 input on R12:
    numval - argument

 output on R12
    numval - anti-natural logarithm of argument

BD  INT         'INT'            <none> INT5   20,55   KR&MTH

 input on R12:
    numval - operand

 output on R12:
    numval - greatest integer <= operand

BE  LOG10       'LOG10'          <none> LOGT5  20,55   KR&MTH

 input on R12:
    numval - operand

 output on R12
    numval - logarithm of operand (base 10)

BF  LOG (E)     'LOG'            <none> LN5    20,55   KR&MTH

 input on R12:
    numval - operand

 output on R12:
    numval - natural logarithm of operand

C0  VER$        'VER$'          <none> VER.   0,56    KR&FUN

C1  SEC         'SEC'           <none> SEC10  20,55   KR&MTH

 input on R12:
    numval - argument

 output on R12:
    numval - secant of argument

C2  CHR$        'CHR$'           <none> CHR$.  20,56   KR&FUN

C3  STR$        'STR$'           <none> VAL$.  20,56   KR&FUN

C4  LEN         'LEN'            <none> LEN.   30,55   KR&FUN

C5  NUM         'NUM'            <none> NUM.   30,55   KR&FUN
```

```
C6  VAL         'VAL'            <none> VAL.   30,55   KR&FUN

C7  INF         'INF'            <none> INF10  0,55    KR&MTH
 input:
    none

 output on R12:
    numval - upper bound of machine precision

C8  read number ''               <none> READN. 0,44    MJ&TXT

C9  PI          'PI'             <none> PI10   0,55    KR&MTH
 input:
    none

 output on R12:
    numval - 3.14159265359

CA  UPC$        'UPRC$'          <none> UPC$.  30,56   KR&FUN

CB  USING       'USING'          <none> USING. 0,341   KR&TOK

CC  THEN        'THEN'           <none> ERRORX 0,44    KR&TOK

CD  TAB         'TAB'            <none> TAB.   20,45   KR&FUN

CE  STEP        'STEP'           <none> STEP.  0,41    RH&FOR

  input from R10:
    numval - value of increment

CF  EXOR        ' EXOR '         <none> EXOR.  2,51    IV&OP

D0  NOT         'NOT '           <none> NOT.   7,50    IV&OP

D1  DIV (\)     ' DIV '          <none> INTDIV 12,51   KR&MTH

  input on R12:
    numval - dividend
    numval - divisor

  output on R12:
    numval - integer quotient (real flagged as integer)

D2  ERRN        'ERRN'           <none> ERNUM. 0,55    KR&FUN

D3  ERRL        'ERRL'           <none> ERRL.  0,55    KR&FUN

D4  CARD filename'CARD'          <none> CARD.  0,47    RH&FIL

D5  AND         ' AND '          <none> AND.   4,51    IV&OP

D6  KEYS filename'KEYS'          <none> KEYS.  0,47    RH&FIL

D7  ELSE        'ELSE'           <none> ERRORX 0,44    KR&TOK
```

```
D8  SIN         'SIN'            <none> SIN10  20,55   KR&MTH

  input on R12:
    numval - argument

  output on R12:
    numval - sine of argument

D9  COS         'COS'            <none> COS10  20,55   KR&MTH

  input on R12:
    numval - argument

  output on R12:
    numval - cosine of argument

DA  TAN         'TAN'            <none> TAN10  20,55   KR&MTH

  input on R12:
    numval - argument

  output on R12:
    numval - tangent of argument

DB  TO binary op ' TO '          <none> NOP2.  1,51    KR&TOK

DC  RESTORE to x ''              <none> RESTN. 0,227   MJ&TXT

  input:
    address of line# in r10 stream

DD  input #     ''               <none> INPUN. 0,44    IV&INP

 input from R12
    numval - value to be stored
    nameform - name of target variable
    (dimflag) - if tracing, tracing information
    (col) - if tracing, tracing information
    (row) - if tracing, tracing information
    numvalptr - pointer to target value

 output to R12 stack
    num

DE  [           '['              <none> ERRORX 0,44    KR&TOK

DF  ]           ']'              <none> ERRORX 0,44    KR&TOK

E0  \           '\'              <none> INTDIV 12,51   KR&MTH

E1  POS         'POS'            <none> POS.   52,55   KR&FUN

E2  RTD         'DEG'            <none> DEG10  20,55   KR&MTH
```

```
E3  DTR         'RAD'            <none> RAD10. 20,55   KR&MTH

E4  FLOOR       'FLOOR'          <none> INT5   20,55   KR&MTH

 input on R12:
    numval - operand

 output on R12:
    numval - greatest integer <= operand

E5  INPUT $     ''               <none> INPU$. 0,44    IV&INP

 input from R12 stack
    strptr - source string address
    strlen - source string length
    strptr - target string address
    strlen - target string length
    VPAptr - pointer to beginning of target string

 output to R12 stack
    none

E6  ERROR       ''               <none> ERRORX 0,44    KR&TOK

E7  numeric ;   ''               <none> PRNUM. 0,236   IV&PUN
  note: it should be 36 but the attribute was left out (sigh)

E8  numeric ,   ''               <none> PRNUM. 0,236   IV&PUN
  note: it should be 36 but the attribute was left out (sigh)

From ALTROM

 00             ''               ERRORX ERRORX 0,44    KR&TOK

 01  ALARM ON/OFF 'ALARM'        ON/OFF ALARM. 0,241   KR&PS5

 02  LOCK       'LOCK'           GET1$  LOCK.  0,241   RY&LOK

 03  DEG        'OPTION ANGLE DEGREES' PUSH1A DEG. 0,241 IV&ANG

 04  RAD        'OPTION ANGLE RADIANS' PUSH1A RAD. 0,241 IV&ANG

From MELROM

 00             ''               RETURN ERRORX 0,44    KR&TOK

 01  TRANSFORM  'TRANSFORM'      TRNSLP TRFRM. 0,241   GC&TFM

   input on R12:
     [file name] <string>
     file type, 2-byte attribute
   output:
     none

 02  PACK       'PACK'           ASPACK PACK.  0,241   AS&PAK
```

```
   input on R12:
      device name <string>

   output:
      none

03  INITIALIZE   'INITIALIZE'      ASINIT  INIT.   0,241   AS&INI

   input on R12:
      device name <string>
      numval - [# of directory entries]
   output:
      none

04  TIME    'TIME$'    <none>  TIME$.  0,56    RY&T&D

05  DATE    'DATE$'    <none>  DATE$.  0,56    RY&T&D

06  TIME    'TIME'     <none>  TIME.   0,55    RY&T&D

07  DATE    'DATE'     <none>  DATE.   0,55    RY&T&D

08  ANGLE   'ANGLE'    <none>  ATN2.   20,55   KR&TRG

09  ACOS    'ACOS'     <none>  ICOS.   20,55   KR&TRG

0A  ATN     'ATN'      <none>  ITAN.   20,55   KR&TRG

0B  ASIN    'ASIN'     <none>  ISIN.   40,55   KR&TRG

0C  RMD     'RMD'      <none>  REM10.  40,50   MJ&MOD

0D  MOD     'MOD'      <none>  MOD10.  40,55   MJ&MOD

0E  RND     'RND'      <none>  RND10.  0,55    MJ&RND
```

ON ERROR comments

Raan Young
07/09/82

The ON ERROR code consists of three parts:

* the ON ERROR setup, invoked by the ON ERROR statement;
* the ON ERROR termination, invoked by the OFF ERROR statement;
* the actual invokation of the ON ERROR user code when an error happens.

The ON ERROR parsetime calls PARSE+ to parse the rest of the line, and then adds an invisble RETURN token at the end of the line. If there are any GOTO or ON...GOTO tokens in the line, an invisible POP token is placed in front of them.

The ON ERROR runtime saves the relativized address of the first token (keyword following ON ERROR) to be executed when an error is encountered, in E.EREX; and the relativized PCR in E.ERPC. It then skips to the next line.

The OFF ERROR runtime 0's out E.EREX and E.ERPC.

The runtime code executed when an error is detected is the heart of the ON ERROR function. This is invoked by REPRT+ or one of its derivatives. When REPRT+ is called to report any errors, it tests to see if E.EREX is positive and non-zero. If it is then an ON ERROR is active and we set up a GOSUB to the first token in the ON ERROR statement. If it is 0, ON ERROR has either been OFFed or was never declared to start with. If it is negative, then we are in an ON ERROR statement and do not want to call it again (read infinite loop). The call to the ON ERROR code is setup by using E.EREX for the relative R10, and E.ERPC for the relative PCR of the GOSUB address. The address of the next line is used for a return address and PCR. SUBSTF is called to do the actual GOSUB setup, and then CLRERR clears the error so the program will not stall. The GOSUB is traced, and finally RTSIGN is called to clean up any garbage left on R12 by the error. The value of E.EREX is made negative by setting the top bit. This flags us as being in an ON ERROR. When the invisible POP or invisible RETURN is executed, This value is restored to its original positive value (usually), and execution returns to the line after the one containing the error.

In addition to the PCR and R10, a flag is saved on the subroutine return stack. This flag is used by ON ERROR and ON TIMER to clean up things that need cleaning up at the end of the statement. The flag = 8000H for normal GOSUBS, is >= 0 and < 8000H for ON TIMER (see XR"TIM for more), and is > 8000H for ON ERROR. This flag contains the E.EREX value + 8000H (the E.EREX value is relative address of code, and is assumed to never be >7FFFH). This serves to flag the return info as being for an ON ERROR, and also saves the relative address. This address can not be saved in E.EREX because the ON ERROR code might contain either an OFF ERROR or another ON ERROR. Either of these will change the contents of E.EREX. If the value of E.EREX is 0 when the return is executed, then an OFF ERROR was done and the value is not restored. Otherwise the old E.EREX is restored. This means that an ON ERROR declared in an ON ERROR only is effective during that ON ERROR.

Quick Reference Documentation of Kangaroo Subroutines
File Grouping

This document is meant as an aide to people working with the Kangaroo system. Each file has a line describing the type of subroutines that it has. Beneath each file is a list of subroutines in that file along with a description of what it does and what registers it will certainly trash. (Note: there is no guarantee that it will not trash other registers. All registers which are in the subroutine are listed: however, the subroutine may call another subroutine which would trash different registers.) If there are any mistakes or untruths in this document, readers are asked to please make the changes themselves. Make the changes to KR'FIL and then run runof to obtain KR"FIL (this file). Thank-you.

9:27 AM WED., 16 SEPT, 1981

Sub-routine	Description
IV/HND - ROM initialization (HANDIO) routines	
HANDI	ROM initialization: invokes each keyword file one by one. R20=error number, E=1 if ERROR was called, E=0 if handled by file. Uses: R0-1, R2, R6 stack
HANDIO	'HANDI' with no error: passes control to all keyword files. Uses: R0-1, R2, R6 stack
ROMINI	'HANDI' with no error: passes control to all keyword files. Uses: R0-1, R2, R6 stack
IV/IO - I/O routines for the Kangaroo.	
ATTN?	Checks to see if there is any key waiting. E=1 if key is waiting & it's ATTN, E=0 otherwise. Uses: R2
CURSE-	Turns the cursor off. No error. Uses: R0-1, R2-3, R30-37, R44-47, R53-57, R0 stack, R6 stack
CURSE+	Turns the cursor on. No error. Uses: R0-1, R2-3, R30-37, R44-47, R53-57, R0 stack, R6 stack
DEQUE	Kills any queued up keyboard interrupts Uses: R3
EOLND	'OUTEOL' with No Delay Uses: R0-1, R2-3, R30-37, R44-47, R53-57, R0 stack, R6 stack
GETCHR	Gets a single character in KEYHIT and R2 Uses: R0-1, R2-3, R6 stack
GETLN	Reads a line with the initial template in INPBUF. Returned terminating character is in R25. Uses: R20-25, R6 stack
HLFLIN	Sends a string to LCD. Called with a multi-byte count in R36 and the address of the string in R26. Uses: R6 stack
HLFOUT	A cheap way to call HLFLIN Uses: R6 stack, R26 stack
KEY?	Checks to see if any key (even ATTN and mode switchers) has been pressed. E=1 if a key was hit, E=0 if no key was hit. Uses: R2
LETGO	Waits until the key is released Uses: R2-3
MSGOUT	A cheap way to call OUTSTR Uses: R6 stack
OUT1CH	Writes a character Uses: R6 stack
OUTC40	'OUTCHR' with the character in R40 Uses: R6 stack
OUTCHR	Outputs a character to the display device: ALL output to the display device filters through this routine.

F 9:27 AM WED., 16 SEPT, 1981

	Uses: R6 stack
OUTEOL	Does the end of line sequence for display devices (CRLF) Uses: R6 stack
OUTESC	Writes out an escape + 'OUT1CH' Uses:
OUTSTR	Writes out the device to the current display device Uses:
PUTKEY	Makes the character in R2 the current character Uses: R2-3, R6 stack
SETLIN	Sets up a line for input Uses: R24-25, R6 stack
IV/RSW - ROM manipulation routines	
CYCLE	Cycles through all the ROMs and ROM files. Call the subroutine each lexfile enabled: R0 is the ROM number. Uses: R0-1, R2-3, R6 stack
GETROM	Enables a ROM whose 2 byte identifier is in R0. If successful, ZR is returned, otherwise NZ is returned. Uses: R6 stack
ROM:GO	This is a token to transfer to a given address in a ROM Uses: R0-1, R2-3, R20-21, R30-31, R6 stack
ROMJSB	Switches to a given ROM, calls the address, and switches back to the current ROM Uses: R6 stack
ROMRTN	Returns to regular system ROM. System ROM is enabled, ROMOF=0, ROMPTR=24K, and NZ Uses: R2-3
IV/UT1 - Utility routines, file 1.	
BASEND	The last line in a program Uses:
BLANKS	Eight blanks for public consumption Uses:
PAR1	Get one byte parameter after call. R2=1 byte that R6 pointed to when you called PAR1. R6 := R6 + 1. Changes PROTEM. Uses: R2, R6 stack, R44 stack
RETURN	The address of a RTN instruction Uses:
IV/UT2 - Utility routines, file 2.	
DAYSEC	Seconds in day data Uses:
DTABO	Days in month data Uses:
KOPY	Byte copy routine. Source address is R74-75, destination address is R76-77, and the byte count address is R73. Uses: R72-73, R74 stack, R76 stack
NUMCHK	Numeric check and encode. The first byte (high byte) is in

F 9:27 AM WED., 16 SEPT, 1981

	R20, the second byte (low byte) is in R21. Uses: R20-21
TOASC2	Converts two digit BCD to ASCII. The two digit BCD value is in R20, the two byte result is in R20-21. Uses: R20-21
TOBCD2	Converts two byte binary to BCD. The eight bit binary value that is to be converted to 3 digits of BCD is in R20, the BCD result is in R20-21. Uses: R2-3, R20-21
TOBCD8	Converts eight byte binary to BCD. Values to be converted to BCD are in R40-47, the eight BCD results are in R40-47. Uses: R0-1, R20-21
TOBIN2	Converts two digit BCD to binary. The BCD value is in R20, the binary result is in R20. Uses: R20-21
TOBIN8	Converts eight byte BCD to binary. The eight bit BCD value is in R40-47, the eight bit binary result is in R40-47. Uses: R20-21
IV/EXE - Executions routines for the Kangaroo	
CONBIN	Converts binary to floating point. Binary number is in R36-37, converted floating point number in R40-47. Uses: R32-35, R40-47
DOCMD	Handles terminating key for command intput Uses:
EXEC	Execution loop for the Kangaroo Uses: R2-3, R20-24, R32, R76-77
GETALN	Gets a BASIC input line from the keyboard Uses: R20-24
HANG	Lets the user see the output on the LCD Uses: R2
INITGL	Initializes the GETLN parameters Uses: R2-3
OTHER	Handles other than terminating keys for command input Uses: R2-3, R10-11, R16-17, R40-47, R76-77
OUTLIN	Prints out the line pointed to by R36/37 unless it is the special last line flag EOLIN. Uses: R20, R24-25, R26-27, R30-31, R36-37, R45-47, R67
REPORP	Reports parsing errors by re-displaying the input line with the cursor over the point at which the parsing failed. R10 points one past the error. Uses: R2-3, R10-11
START	Coldstart Uses: R2-3, R6-7, R25
UNAUTO	Clears out AUTOI if there is an overflow Uses: R2-3
KR/MTH	contains mathematical functions. Also has a few parameter getting routines, random number generating routines, unpacking routines.

F 9:27 AM WED., 16 SEPT, 1981

```
        ----------------------------------------------------------------
        ABS5   ! Calculates the absolute value of a real on the R12 stack.
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R40-47, R60-67
        ADD9   ! Calculates the sum of two reals on the R12 stack. Output
               ! is on the R12 stack with a copy in R40.
               ! Uses: R32-37, R40-47, R50-57, R60-67
        ADDROI ! Calculates the sum of two reals or integers on the R12
               ! stack. Ouput is on the R12 stack with a copy on R40.
               ! Uses: R32-37, R40-47, R50-57, R60-67
        CIEL10 ! Locates the smallest integer >= the real or integer on
               ! the R12 stack. Output is on R12 with copy in R60 or R70.
               ! Uses: R2-3, R32-37, R40-47, R50-57, R60-67, R70-77
        CHSROI ! Generates the negative of the real or integer on the R12
               ! stack. Output is on the R12 stack with a copy in R40.
               ! Uses: R40-47
        COS10  ! Calculates the cosine of a real on the R12 stack. Output
               ! is on the R12 stack with a copy in R40.
               ! Uses: R20-27, R30-37, R40-47, R50-57, R60-67, R70-77
        COT10  ! Calculates the cotangent of a real on the R12 stack.
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R20-27, R30-37, R40-47, R50-57, R60-67, R70-77
        CSEC10 ! Calculates the cosecant of a real on the R12 stack.
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R20-27, R30-37, R40-47, R50-57, R60-67, R70-77
        DCON1  ! Tables and constants
               ! Uses:
        DEG10  ! Converts a real on the R12 stack from radian to degree
               ! measure. Output on the R12 stack, copy in R40.
               ! Uses: R32-37, R40-47, R50-57, R70-77
        DIV2   ! Calculates the quotient of two reals on the R12 stack
               ! Uses: R32-37, R40-47, R50-57, R60-67
        EXP5   ! Calculates the exponential function at its argument
               ! Uses: R32-37, R40-47, R50-57, R60-67
        FP5    ! Function return
               ! Uses: R32-37, R40-47, R50-57
        FTR53  ! Main froward trig subroutine
               ! Uses: R30-37, R40-47, R50-57, R60-67
        INT5   ! Calculates the greatest integer <= the real or integer on
               ! the R12 stack. Output on R12, copy in R60.
               ! Uses: R32-37, R40-47, R50-57, R60-67, R70-77
        INTMUL ! Integer multiply. R66 is the multiplier, R76 is the multi-
               ! plicand, the result is four byte starting in R54.
               ! Uses: R54-57, R60-67, R74-77
        IP5    ! Yields the integer part of a a real or integer on the R12
               ! stack. Output is on R12 stack with copy in R60 or R70.
               ! Uses: R32-37, R40-47, R50-57, R60-67, R70-77
        LN5    ! Calculates the natural log of a real on the R12 stack.
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R0, R26-27, R32-37, R40-47, R50-57, R60-67, R70-77
        LOGT5  ! Calculates the log base 10 of a real on the R12 stack.
```



```
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R0, R26-27, R32-37, R40-47, R50-57, R60-67, R70-77
        MAX10  ! Locates the maximum of two reals on the R12 stack. Output
               ! is on the R12 stack with a copy in R50.
               ! Uses: R32-37, R40-47, R50-57, R60-67
        MIN10  ! Locates the minimum of two reals on the R12 stack. Output
               ! is on the R12 stack with a copy in R50.
               ! Uses: R32-37, R40-47, R50-57, R60-67
        MPY30  ! Calculates the product of two real values (a*b). m(b) R40;
               ! exp(b) R36; sgn(b) R32; m(a) R50; exp(a) R34; sgn(a) R33;
               ! Output is in R40.
               ! Uses: R32-33, R36-37, R40-47, R50-57, R60-67
        MPYROI ! Calculates the product of two reals or integers on the R12
               ! stack. Output is on the R12 stack with a copy in R40.
               ! Uses: R0, R32-37, R40-47, R50-57, R60-67, R70-77
        ONEB   ! Gets one binary integer in R40. 32767 if too big, E<>0 if
               ! overflow or underflow.
               ! Uses: R46-47, R60-67, R70-77
        ONEI   ! Gets one BCD integer in R40. 99999 if too big, E<>0 if over-
               ! flow or underflow.
               ! Uses: R40-47, R60-67
        ONEROI ! Gets one real or integer in R40. 32767 if too big, E<> 0 if
               ! if overflow or underflow.
               ! Uses: R40-47
        RAD10  ! Converts a real or integer on the R12 stack from degrees
               ! to radians. Output on the R12 stack, with a copy in R40.
               ! Uses: R30-37, R40-47, R50-57, R60-67
        RND10  ! Generates a random number an places it on the R12 stack,
               ! with a copy in R40.
               ! Uses: R36-37, R40-47, R50-57, R60-67, R70-77
        RNDINI ! Generates a new seed for the random number generator
               ! Uses: R40-47
        SEC10  ! Calculates the secant of a real on the R12 stack. Output
               ! is on the R12 stack with a copy in R40.
               ! Uses: R20-27, R30-37, R40-47, R50-57, R60-67, R70-77
        SEP10  ! Unpacks a real in R40. Output: man R40, exp R36, sgn R32.
               ! Uses: R32, R36-37, R40-47
        SEP15  ! Unpacks two reals in R40 and R50 (a and b). Output:
               ! man(b) R36, exp(b) R36, sgn(b) R32, man(a) R50, exp(a) R34,
               ! sgn(a) R33
               ! Uses: R32-37, R40-47, R50-57
        SGN5   ! Signum function: sgn(x)=-1, x<0; sgn(x)=0, x=0; sgn(x)=1,
               ! x>0. Input is a real on the R12 stack, output is on the
               ! R12 stack with a copy in R40.
               ! Uses: R40-47
        SHF10  ! Shift leading zeros off of R40
               ! Uses: R36-37, R40-47
        SHRONF ! Shift leading zeros off of R40, round and pack the result.
               ! Uses: R32, R36-37, R40-47
        SIN10  ! Calculates the sine of a real on the R12 stack. Output
               ! is on the R12 stack with a copy in R40.
```



```
               ! Uses: R20-27, R30-37, R40-47, R50-57, R60-67, R70-77
        SQR5   ! Calculates the square root of a real on the R12 stack.
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R32-37, R40-47, R50-57, R60-67, R70-77
        SUBROI ! Calculates the difference of two reals or integers on the
               ! R12 stack. Output is on the R12 stack with a copy in R40.
               ! Uses: R32-37, R40-47, R50-57, R60-67
        TAN10  ! Calculates the tangent of a real on the R12 stack.  Output
               ! is on the R12 stack with a copy in R40.
               ! Uses: R20-27, R30-37, R40-47, R50-57, R60-67, R70-77
        TBL3B  ! Real/integer data fetch
               ! Uses:
        YTX5   ! Calculates powers of two reals on the R12 stack (x^y).
               ! Output is on the R12 stack with a copy in R40.
               ! Uses: R22-27, R30-37, R40-47, R50-57, R60-67, R70-77
        ----------------------------------------------------------------
        KR/PAR - The parsing routines for Kangaroo.
        ----------------------------------------------------------------
        ALFA   ! DRP=20 + 'ALFA '
        ALFA   ! Converts a letter to upper case, if necessary
        ASSIGN ! Parses an ASSIGN statement
        BACK10 ! Pulls R10 back if R20 <> CR
        DATA   ! Parses a DATA statement
        DEF    ! Parses a DEFINE function
        DIGIT  ! Checks to see if R20 contains a digit
        DIM    ! Parses a dimension statement
        DMNDCR ! Demands a CR or ! after a statement
        ERR89  ! Prints out bad parameters message
        FNEND  ! Parses the end of a function statement
        FNLET  ! Parses the function LET statement
        FOR    ! Parses a FOR statement
        FORMAR ! Parses a formal array
        FUN1   ! Gets one standartd function parameter
        G$O12N ! 'GO12N' + gets a filename
        G$!012 ! 'G4012N' with an error if no file
        GO1N   ! Gets 0 or 1 line numbers
        GO1N$  ! Gets a string and/or a line number
        GO12N  ! Gets 0, 1, or 2 line number
        GOTO4N ! Gets 0 to 4 line numbers
        G1OR2N ! Looks for optional parameters
        GCHAR  ! Gets the next non-blank character
        GCHAR  ! 'GCHAR' + DRP=20 and ARP=10
        GET1$  ! Gets one string
        GET1N  ! Gets one parameter
        GETCMA ! Gets a comma
        GETLIN ! Gets a sequence number
        GETPA? ! Gets all the parameters
        GETPAR ! Gets a specifiable number of parameters
        GOTOPR ! Parses a GOTO command
        GOTOSU ! Parses a GOSUB command
        IF     ! Parses an IF statement
```



```
        ILET   ! Parses a LET statement
        INPUT  ! Parses an INPUT statement
        ISCMA  ! Sees if R14 is a comma
        LET    ! Parses a LET statement
        NEXT   ! Stores the next token
        NOCLC! ! Checks for not calculator
        NUMBER ! Gets a floating point number
        NUMREP ! Chews up leading + or= and produces a signed constant
        NUMVA+ ! 'NUMVAL' + 'SCAN'
        NUMVAL ! Parses a number or a value
        ON     ! Parses the ON token
        ONERRD ! Handles any error after 'THEN'
        OPTION ! Parses an option base
        PARSE! ! Main parse inner loop
        PARSER ! Is this a program, statement, or expression
        PARSIT ! 'PARSE!' + 'SCAN'
        PU36SC ! 'PUSH32' + DRP=36
        PUSH1A ! Pushes out the token + 'SCANE1'
        PUSH32 ! Pushes an integer onto the stack
        READ   ! 'READ ' + checks for calculator mode
        READ   ! Reads one or two parameters
        REM    ! Parses a remark
        RESTOR ! Restores the parsed line
        RSTREG ! Restores R21-37, r60-67
        SCAN   ! Scanner
        SCAN+  ! 'SCAN' + 'GCHAR'
        SCANE1 ! 'SCAN' + E=1
        STRCN+ ! Remark and string constant
        STREX! ! 'STREX+' with an error if no string
        STREXP ! Get a string expression
        TRY1N  ! Looks for optional parameters
        TYPSTM ! Scans type statements
        UNQUOT ! Scans a string stopping at commas
        ----------------------------------------------------------------
        MJ/DIO - Basic routines.  Includes some parsing routines and does
                 basic manipulation with the loop.
        ----------------------------------------------------------------
        ANY.IS ! Gets either a string or a *
        CHKEND ! Checks a line number to see if it is A999
        CLDEV. ! Clears ":dev" (Token A7)
        CLOOP. ! Clears loop (Token A7)
        CLOSE+ ! Processes PRINTER IS * and DISP IS *
        CONFIG ! Token 140 assignio
        DSPIS. ! Display is runtime
        EOL.   ! End of line routine
        GO1$/* ! Gets a string, a *, or no parameters
        G1$OR* ! Gets either a string or a *
        GETADR ! Gets a loop address for a specified device
        LPOFF  ! Clears bit   0 (loop on bit) in PLSTAT
        LPON   ! Sets bit   0 (loop on bit) in PLSTAT
        LSTIO. ! Lists devices configured on loop
```

```
MOVEIT ! Moves things into the error buffer
OFFIO. ! Processes offio
PRINS. ! Printer is runtime
REST.  ! Restores I/O routine
SKPCHK ! Skips a line and checks for the end
STAR?  ! Looks for a * or null string
TAB    ! String for a tab
THERE? ! Sees if device table is there; returns the address if so
------------------------------------------------------------------
MJ/PIL - Basic PIL manipulating routines. Some parsing routines.
         Also sends commands and frames.
------------------------------------------------------------------
ACTREP ! Checks if active controller
ALARM. ! Parses ALARM command
C.INIT ! Coldstart initialization
CL.ACT ! Clears active bits in PLSTAT
CMDREP ! Sends commands with error reporting
CMDSND ! Sends SEND frame
DATREP ! Sends data with error reporting
LADSND ! Sends listen addresses
PILOF! ! Turns PIL chip off
PILON! ! Turns PIL chip on
RDYSND ! Sends READY frame
SENDID ! Verifies if loop is intact
SNDAUT ! Sends auto unconfigure and auto address
SNDFRM ! Sends a frame
STAND. ! Parses the STANDBY command
STAND- ! 'STAND.' with an arbitrary DRP
UNLREP ! Sends UNL, reports errors if any, and falls into CL.ACT
UNLSND ! Sends the UNL frame
------------------------------------------------------------------
RM/ERZ - Error and warning routines
------------------------------------------------------------------
ERR1   ! 'ERROR' with E=1
ERR1+  ! 'ERR1' with return address trashed
ERROR  ! Error parsing routine
ERROR+ ! 'ERROR' with return address trashed
ERRORR ! Takes the error number in a register
ERRREP ! Calls 'ERROR' and then 'REPORT's
WARN   ! Prints a warning message
WARN.R ! Entry for 'WARN' with number in register R36
------------------------------------------------------------------
RM/FET -  Line manipulating subroutines
------------------------------------------------------------------
CREOL? ! Tests for end of line tokens
EDLIN  ! Fetches a line number
FETCH. ! Finds a given string or line number
LINEDR ! Inserts and deletes lines in files
------------------------------------------------------------------
RM/FIL - File manipulation routines and some parsing routines.
------------------------------------------------------------------
```



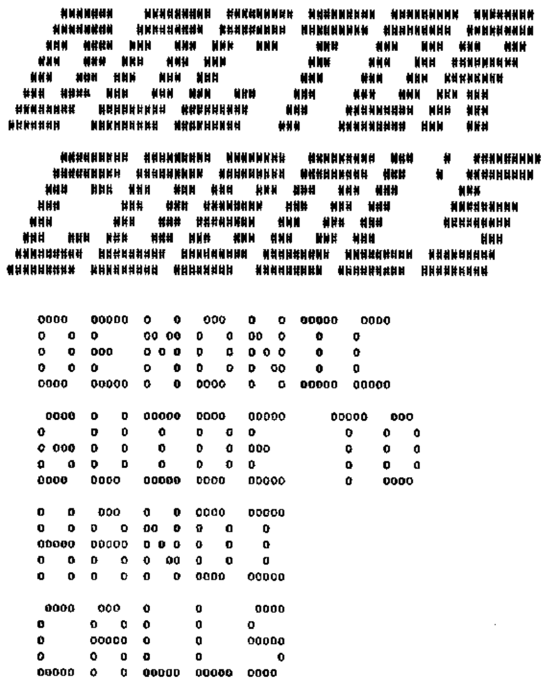
```
APPT.  ! Parses APPT command
CARD.  ! Parses CARD command
EDIT   ! Parses EDIT command
FILNM+ ! 'FILNM?' with prescan
FILNM! ! 'FILNM?' with an error
FLCHR? ! Gets a filename or a special file token
FLORD  ! 'FLORDT' with no type check
FLORDT ! Gets a filename from the R12 stack as in GETFL.
FLTOFL ! Parses COPY and RENAME
GETNAM ! Gets a filename from the R12 stack
KEYS.  ! Checks to see if any key has been pressed
PUSHIF ! Gets a filename parameter or null
------------------------------------------------------------------
RM/MEM - The memory routines. Work with memory and files.
------------------------------------------------------------------
ADJUST ! Updates the location of all files
ALCALL ! Allocates all remaining memory
ALLOC  ! Adds free space to the given location
DELETE ! Deletes data at a given location
DELLIN ! Deletes a line in a file
FCOPY  ! Copies a file
FCRALO ! Creates a file (name in R20, type in R40)
FCREAT ! Creates a file (name in R40, type in R20)
FCRNUL ! Creates a file with only a header and an endline
FEMPT? ! Tests to see if a file is empty
FLINIT ! Initializes file system variables
FNDLED ! 'FSEEK' with R36 set to EDFILE
FNDLIN ! 'FSEEK' with R36 set to EDFILE
FNDLPR ! 'FSEEK' with R36 set to PRFILE
FNDLRN ! 'FSEED' with R36 set to RNFILE
FOPEN  ! Opens a file with a given name
FPURGE ! Purges a file
FRENAM ! Renames a file
FSEEK  ! Finds a given line by number in given areas of memory
FSREPL ! 'FSEEK' + 'REPLIN'
INSERT ! Inserts a given block of data at a given location
LINLEN ! Returns the length of a line
ONR12  ! Tests to see if anything is on the R12 stack
PFNDPR ! 'PREFND' for PRFILE
PREFND ! Finds the first line before a fiven line in the edit file
REPLIN ! Replaces a line in a file
ROOM!  ! Sets an error to be reported if no room
ROOM?  ! Tests to see if the required memory is available
RSETEN ! Sets variable area
SETPR  ! Sets up the file parameters PRNAME and PRFILE
SETRN  ! Sets up the run file parameters
SKPLN  ! Finds the address of the next line
SKPLN  ! 'SKPLN' with an arbitrary DRP
------------------------------------------------------------------
DF/GPJ - The support routines for the Advance I/O ROM.
------------------------------------------------------------------
```



```
FINDME ! Locates your position in the RB
FRELSH ! Cleans up loop after halt or abort of I/O
GETVAR ! Gets the I/O RAM variable
GOODLP ! Checks to see if our RB is the same as reality
I/OCHK ! Determines whether there is I/O to do
INRST  ! Restores the current input file and status
INSAVE ! Saves the current input file and status
LSTN1  ! Listen addresses a device
LSTNR? ! Checks to see if there are listeners ready for a talker
NXTDEV ! Goes to the next device on the loop
OTLINE ! Decompiles a line into the output buffer
OTSTRT ! Output start routines
PUTSYS ! Checks to see if system I/O files should be updated
RSTORE ! Restores the PIL registers; transmits PILINT characters
STATUS ! Checks the SRQ (status) and processes it in the RB
TALK1  ! Talk addresses a particular device
------------------------------------------------------------------
DF/INT - Advance I/O ROM interrupt routines.
------------------------------------------------------------------
CMD    ! Interprets command frames as a non=controller (CA=0)
CNTRL  ! Transmits & error checks non=SRQ commands on the loop
PILER  ! Handles errors for PIL routines
PILINT ! PIL interrupt service routine (I/O engine)
RDY    ! Interprets ready frames as a non-controller (CA=0)
RECIVE ! Receives data over the loop from a file (LA=1)
SEND   ! Sends data over the loop from a file (TA=1)
SRQ    ! Services a request from a peripheral
XLATE  ! Stores collected parpheral status bytes
------------------------------------------------------------------
DF/NIT - Advance I/O ROM initialization routines.
------------------------------------------------------------------
MYLOOP ! Attempts to get control of the loop
NEWRB  ! Makes a new current resource block
PILNIT ! Initialization routine for the ROM; processes HANDI call
REQEST ! Requests service as a peripheral from the current CA
```


Table of Contents

Quick Reference Documentation of Kangaroo Subroutines
File Grouping



HANDI CALL DOCUMENTATION

Seth D. Alford
1/27/82

Event: V.LFTY Translate a strange file type.

HANDI error: 68 Invalid File type.

Where invoked: AS&VF2(509)

Under what conditions is this called?

Kangaroo File types must be translated into LIF file types for copying files to mass memory. VFTRHL does this translation. VFTRHL recognizes the file types currently existent in kangaroo: lif1, text, lex, basic, system and appt. Inevitably someone will create a new file type. VFTRHL will not find it in its table and so will issue a HANDI call.

What should the handler do?

The programmer who creates the new file type should obtain a new LIF file type number from Frank Hall, the current PL21 dispenser of LIF file type numbers.

The handler should intercept the HANDI call and look at R21 to determine the file type being translated. If this file type matches one the handler knows about he should set the HANDLD flag and return the LIF file type in R46/47. The handler should check the file type to determine if indeed it is one he knows about, and not just assume that the case because he intercepted the HANDI call. After all, we may have different ROMS creating new file types.

Entry registers and RAM parameters:
R21:   kangaroo file type
R24/25: FWB pointer
R36/37: devfile offset (points to RCB)

Exit registers and RAM parameters:
R46/47: LIF file type, if known

What registers can be changed:
R40/47
R50/57
(And more if necessary. These should be sufficient for a table lookup subroutine. See me or examine the code if this is insufficient.)

Should HANDLD be set? Only if you can translate the file name.

Notes:

Do not trash R24/25 or R36/37!

HANDI CALL DOCUMENTATION

Seth D. Alford
1/27/82

Event: V.RFTY Translate file types coming into kangaroo.

HANDI error: NONE, HANDIO call.

Where invoked: AS&VF2(82)

Under what conditions is this called?

See the V.LFTY documentation.

Now that you have read the V.LFTY documentation you will understand what is going on. (So go and read it if you have not already.)

Suppose that we want to bring back one of these new file types. VFROO? is the routine which translates LIF file types to kangaroo file types. Similar to VFTRHL, if the file type is not in its table VFROO? will generate a HANDIO call. If it can, the handler is to provide a kangaroo file type.

What should the handler do?

The handler should examine R20/21, which contains the LIF file type. If it knows of this type the handler should return the kangaroo file type in R21 and clear the HANDLD flag.

Entry registers and RAM parameters:
R20/21: LIF file type
R24/25: FWB pointer
R36/37: devfile offset

Exit registers and RAM parameters:
R21   : kangaroo file type

What registers can be changed:
R40/47
R50/57
(And maybe some others, see me or examine the code if necessary.)

Should HANDLD be set? Only if you can translate the file type.

Notes:

Do not trash R24/25 or R36/37!

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
1/27/82

Event:
V.ASSI   (not used at this time)

HANDI error:

Where invoked:

Under what conditions is this called?

What should the handler do?

Entry registers and RAM parameters:

Exit registers and RAM parameters:

What registers can be changed:

Should HANDLD be set? yes/no/other

Notes:

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
6/15/82

Event:
V.ASSN Devfile has been changed

HANDI error:
none

Where invoked:
MJ&DIO

Under what conditions is this called?
Called after assignio, printer is, or display is runtime tokens to let ioron know that the devfile has changed

What should the handler do?
Update whatever depends on the devfile

Entry registers and RAM parameters:
none

Exit registers and RAM parameters:
none

What registers can be changed:
any

Should HANDLD be set?
don't care

Notes:

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
1/28/82

Event:
V.ADDR Pil address needed for unrecognized name

HANDI error:
63D (illegal filespec)

Where invoked:
MJ&DIO

Under what conditions is this called?
Called when Getpad is called for a device name that we don't recognize (either >2 characters or not in Devfile)

What should the handler do?
Return us a pil address if it recognizes the name

Entry registers and RAM parameters:
r66m =name, if <3 characters
r32 =number of characters in name
r34m =address of name if > 2 characters

Exit registers and RAM parameters:
r20 =pil address for that name
r36m =address for that device's entry in DevFile

What registers can be changed:
r20m, r30m, r36m, r40m

Should HANDLD be set?
yes

Notes:

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
6/15/82

Event:
V.LOOP Ask-permission bit is set in plstat (we may not be controller)

HANDI error:
none

Where invoked:
MJ&PLL

Under what conditions is this called?
Called when bit#7 of plstat is =1
This bit is set and cleared by the ioron

What should the handler do?
Try to get control of the loop. If the handler can't get the loop, it needs to issue an appropriate error

Decide if it wants to send the frame itself, or if we should send it
Clear plstat bit#7 if we don't need to ask permission any more

Entry registers and RAM parameters:
r55m =pil regs 0,1,2 for frame we want to send

Exit registers and RAM parameters:
handled =set if it got control of the loop

r77 =cleared if handler is going to send frame
=unchanged if handler wants sndfrm to send the frame

plstat bit#7=0 if we don't need to ask permission next time
bit#7=1 if we still need to ask permission

What registers can be changed:
r77(if handler is going to send frame), r0m,r2m

Should HANDLD be set?
only if handler gets control of the loop. If it doesn't, we expect the handler to issue an error

Notes: If r77=0, sndfrm will finish up as if the frame was sent with no errors, returning e=0. If handled is not set, sndfrm will return with e=3 and the 'not sent' flag set. IT IS UP TO THE HANDLER TO ISSUE AN ERROR IN THIS CASE.

Mary Jo Hornberger
6/15/82

Event:
V.SRQR Intercept for service requests received

HANDI error:
none

Where invoked:
MJ&PLL

Under what conditions is this called?
Called whenever we get a service request from the loop

What should the handler do?
Whatever they want

Entry registers and RAM parameters:
r56 pil register 1 received
r57 pil register 2 received

Exit registers and RAM parameters:
none

What registers can be changed:
r0m, r2m

Should HANDLD be set?
don't care

Notes:

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
1/27/82

Event:
V.ASN# PrintN, Read# or Assign# of non-Kangaroo base machine file

HANDI error:
63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called whenever the iofile type byte for the file we're going to use is not 0 (except for assign# to * which we handle)

What should the handler do?
Everything. The handi call means we gave up.
If read#, read#. If print#, print#. Etc...
See the subevent lists following.

Entry registers and RAM parameters:

token — token that we're trying to do (may be read, read#, read# array, print#, print# item, print# array, print# end, read number, read string, assign#, restore, or restore#.)

if assign#, r20 = number to assign file to
other params from getnam

for all other operations
r14 =access needed for this operation
r76 =file number
r36 =iofile entry for that file

access	access needed for this operation (tyxxx? bits)
filnum	file number we're working with
prnt#?	=1 if print#, else cleared
prtptr	=1 if first print# item not printed yet, else cleared

if read#, print#, restore, read, or restore#, will also have

r24	=0EH if want to go to front of file
	0FH if serial
	BCD line# if random
a#io	address of iofile entry for this file

if read number or read string, will also have

r20	=09H if string
	=00H if number

Exit registers and RAM parameters:
iofile entry should be updated to reflect new status
if assign#, entry for file should be inserted
if others, current data item and current line# fields should be updated

text? =1 if working with text file (may not be necessary to update,
=0 if basic file but it can't hurt...)

What registers can be changed:
any, except usual registers under r20

Should HANDLD be set?
yes, if it was handled

Notes:

Instead of writing a book on everything that needs to be done for all 12 cases, I recommend that the user look in rh"ice or rh"eis to see what tokens come in what order for the different commands, and then check mj/txt to see what the base machine does. There is currently no clearing-house for deciding who can use what iofile type bytes. If there are 2 roms out there that both have strange files and they pick the same type number, things could get interesting. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
6/15/82

Event:
V.ASN# Data file manipulation of non-Kangaroo base machine file

Subevent:
Assign# Assign# statement that we can't handle

HANDI error:
63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called whenever a device name is given in an assign# command
Example: Assign#1 to 'joe:ca'

What should the handler do?
Here is what Kangaroo does with a Kangaroo file:
Check to see if the file exists, creating it if it doesn't. If the user specified a type, that type is checked/used. If no type was specified, the type defaults to Basic.

Build an iofile record for the file, of the following form:

byte	type	contains	example
1	BCD	least byte of file number	01
2	BCD	other byte of file number	00
3	binary	length of this iofile record -3	0C
4	ascii	first character of filename (blank filled)	J
5	"	next character of filename	O
6	"	next character of filename	E
7	"	next character of filename	
8	"	next character of filename	
9	"	next character of filename	
10	"	next character of filename	
11	"	last character of filename	
12	binary	current data item, initially zero	00
13	BCD	least byte of current line number, initially 00	00
14	"	other byte of current line number, initially 00	00
15	binary	type byte (0=regular Kangaroo mainframe file)	00
16+	?	whatever you want (K/R records stop at byte 15)	?

The record is then inserted into the iofile by the Fsrepl, writing over the old assignment for that file number if there was one.

What should the handler do?
Do what Kangaroo would have done. Some steps may need to be added,changed, or deleted, depending on what makes sense for your application.
You should probably check to see if the desired file exists on the given media. You may or may not be able to create it if it doesn't, depending on whether you are implementing a variable length or fixed length record format. If you are implementing a fixed length format, you may want to add bytes to the iofile record indicating number of records per file, number of bytes per record, etc.

Entry registers and RAM parameters:

token	60H	assign# token
r20n	xxxx	BCD file number
r40n	xxxx	upper case file name, blank filled
r50	xxxx	getnam special file type bytes
r54n	xxxx	device name
r65	xxxx	number of bytes from R12 (if >12D, type was specified)
r66n	xxxx	type (basic or text)
r74n	xxxx	password, blank filled

Exit registers and RAM parameters:
An iofile record should be inserted into the iofile for the filenumber wanted, writing over any old assignment of that filenumber. Notice that the filenumber asked for is the iofile linenumber for that record.
The first record in the iofile (linenumber 0000) is reserved for use with read and restore statements in a running program.

What registers can be changed:
any, except usual registers under r20

Should HANDLD be set?
yes

Notes:
The 'Assign#x to *' command ignores the type byte. The iofile entry for that number will be deleted (regardless or whether or not that entry is for a non-Kangaroo type file), and a Handi call will NOT be issued.

See Rh"ice or Rh"eis to find out what tokens come in what order for the different commands. Mj/txt may be useful as a reference to see what the base machine does with the different tokens. Check with the keeper of assign# non-Kangaroo file type numbers if you want to add or use a particular type. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

....

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
8/4/82

Event:
V.ASNH   Data file manipulation of non-Kangaroo base machine file

Subevent:
Print#, Read#, Restore#, Read, Restore   Major tokens for data file manipulation

HANDI error:
63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called whenever the iofile type byte for the file we're going to access is not 0.

What should the handler do?

The purpose of this handi call for these tokens is to get an accurate update of current linenumber and current datanumber in the Iofile entry for the data file, and to make sure they are really there!

Here is what Kangaroo would do with regular files (after the point where the Handi call is issued):

clear text? flag, check file for correct access bits, allowing a text file if all access bits except the runnable bit match. If the file access is wrong, we issue a V.acc# Handi call and quit. If the file is a text file, we set the text? flag.

if we're doing a print#, we check for the print# to a running file error, do a call to safe!. If safe! returns e#0, quit.

next we set up the current line# and current data# as follows:

r24	function	used by
0EH	set current line# and data# to 0, make r24 =serial flag (0FH)	Restore Restore#n
0FH	(serial flag) if print# and current data# <> 0: increment current line#, set current data# to 0. else do nothing	Print#n Read Read#n
legal BCD	(random flag) set current line# to r24n, set current data# to 0	Restore l Restore#n,l Print#n,l Read#n,l

(if the current line# would increment past 9999, we error.)

now we see if that line is really there, and if it is a valid line to access (either a DATA statement in a Basic file, or any line in a Text file).

if the line is not there:
if this is a serial print#, we try to create that line (if we're print#ing to an empty file, we create line#1)
if this is a serial access other than print#, we find the next valid line in the data file
if this is a random access, we error

Finally, we look in the data file to verify the number of the current data item. (We've already set up which item we would LIKE to be at, however the line may or may not contain that many data items!)

Entry registers and RAM parameters:

```
  token   5CH   Print# token
          50H   Read# token
          5EH   Restore# token
          6EH   Read token
          70H   Restore token
          0CH   Restore <to line#> token

  r14   =access needed for this operation
        =tyrun?|tylin?|tyedt?|tyran?        for print#
        =tyrun?|tylin?                      for read,restore,restore#
        =tyrun?|tylin?|tylst?|tycop?        for read#

  r24   =0EH if want to go to front of file (as in restore)
        =0FH if serial                      (as in read#1;a$)
  r24n  =BCD line number if random          (as in read#1,30;a$)

  r76   =file number (this will be linenumber of iofile record)
  r36   =addr of iofile entry for that file

  a#io     addr of iofile entry for this file  (=r36n)
  access   access needed for this operation  (=r14n)
  filnum   file number we're working with  (=r76n)
  prnt#?   =1 if print#, else cleared
  prnt#1   =1 if first print# item not printed yet, else cleared
```

Exit registers and RAM parameters:
iofile record should be updated to reflect new current data item and current line number

```
  text?    =1 if working with text file    (may not be necessary to update,
           =0 if Basic file                  but it can't hurt...)
```

What registers can be changed:
any except usual registers under r20

Should HANDLD be set?
yes, if you handle it

Notes:
See Rh"ice or Rh"eis to find out what tokens come in what order for the different commands. Mj/txt may be useful as a reference to see what the base machine does with the different tokens. Check with the keeper of assign# non-Kangaroo file type numbers if you want to add or use a particular type. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
8/4/82

Event:
V.ASNH   Data file manipulation of non-Kangaroo base machine file

Subevents:
Readn. Reads.   (Read<num> and Read<string> tokens)

HANDI error:
63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?

Called whenever the iofile type byte for the file we want to read a string or number from is not 0

What should the handler do?

Read one number (for Readn.) or one string (Reads.) from the data file.

The tokenized form of Basic lines in Kangaroo files are as follows:

```
  line number  length  data            data items                 end
                       token                                      token
 |_____|_____|_____|_____|__ ... ... ... ... ... __|_____|
                      |<---------------max 255 bytes ------------->|
```

The data items are of the form (no bytes are used for delimiters):

```
       'quoted strings:      96H  stringlength  string
       "quoted strings:      05H  stringlength  string
      unquoted strings:      06H  stringlength  string
          real numbers  :    04H  eight bytes of number
       integer numbers:      1AH  three bytes of number
```

The lines in Text files are of the following form:

```
  line number  length            characters in line
 |_____|_____|_____|__ ... ... ... ... ... ... ... ... ... ...__|
                      |<---------------max 255 bytes ------------->|
```

When reading a string from a text file, the whole line is read into one string.

Entry registers and RAM parameters:

```
  token =A1H if read<string>
        =C8H if read<number>
  r14   =access needed for this operation
  r20   =00H if read<num>
        =09H if read<string>
  r24   =0FH if serial
  r24n  =BCD line number if random

  r76n  =file number (#1,#2,etc.)
  r36n  =addr of iofile entry for that file

  a#io     addr of iofile entry for this file
  access   access needed for this operation (tyxxx? bits)
  filnum   file number we're working with
  prnt#?   =1 if print#, else cleared
  prtptr   =1 if first print# item not printed yet, else cleared
```

Exit registers and RAM parameters:

Current data item and line# should be correct.

What registers can be changed:
any, except the usual registers under R20

Should HANDLD be set?
yes

Notes:
See Rh"nem for information on where to put the number or string that you read. The information that you need from the variable name is already set up on R12 before you enter these tokens.

See Rh"ice or Rh"eis to find out what tokens come in what order for the different commands. Mj/txt may be useful as a reference to see what the base machine does with the different tokens. Check with the keeper of assign# non-Kangaroo file type numbers if you want to add or use a particular type. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
8/4/82

Event:
V.ASNH   Data file manipulation of non-Kangaroo base machine file

Subevent:
Semi#. Coma#. Pr#val   (Print one item tokens)

HANDI error:

63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called whenever the iofile type byte for the file we're going to print# to is not 0

What should the handler do?
Print# the number or string given to the data file at the current line number and data number. See the Print# subevent for information on how to locate the current line number and data number. It is a good idea to recalculate everything with each token, since if there is a defined function in the print list, when we hit that function it may assign# us to a different file, purge the data file, etc., all of which would be disastrous if we went ahead and wrote to where we thought we should in Ram, as now we don't know what we may be writing over.

At the first print#ed item in each parameter list, Kangaroo clears the line we're going to use.

If a item won't fit on a partially full line and the user is doing a serial print#, Kangaroo moves to the next valid line, building one if there are none left. If we're doing a random print#, we error.

If a string is too large to fit on an empty line, Kangaroo truncates it and issues a warning.

If the item is a number, Kangaroo does a V.ACC# handi call if we're in a text file.

Entry registers and RAM parameters:

string address&length OR 8 bytes of number will be on R12

```
token A5H =print# semicolon
      A6H =print# comma

r14   =access needed for this operation
r24   =OFH if serial print#
      =BCD line number if random

r76   =file number (#1,#2,etc.)
r36   =addr of iofile entry for that file

a#io     addr of iofile entry for this file
access   access needed for this operation (tyxxx? bits)
filnum   file number we're working with
prnt#?   =1 if print#, else cleared
prnt#1   =1 if first print# item not printed yet, else cleared
```

Exit registers and RAM parameters:
The current linenumber and current datanumber should be updated in the Iofile

What registers can be changed:
any, except the usual registers under r20

Should HANDLD be set?
yes

Notes:
See the subevents Readn. and Reads. for the structure of the data items in the data files.

See Rh"ice or Rh"eis to find out what tokens come in what order for the different commands. Mj/txt may be useful as a reference to see what the base machine does with the different tokens. Check with the keeper of assign# non-Kangaroo file type numbers if you want to add or use a particular type. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
8/4/82

Event:
V.ASN#   Data file manipulation of non-Kangaroo base machine file

Subevent:
P#aray   R#aray   (print# or read# array tokens)

HANDI error:
63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called whenever the iofile type byte for the file we're going to use is not 0

What should the handler do?
Either Print# or Read# the given array to/from the current data file. In Kangaroo, this is done by calling Pr#val or Readn. for each item.

Kangaroo does a V.ACC# handi call if the user if trying to print# or read# an array to or from a text file

Kangaroo also modifies the trace flag after the first item is read, so only the first item is traced. The trace flag is restored when the array read is completed.

Entry registers and RAM parameters:
information from the 1 or 2 dimensional array tokens will be on R12

```
token =22H if array print#
      =24H if array read#
r14   =access needed for this operation
r24   =OFH if serial
      =BCD line number if random
r76   =file number (#1,#2,etc.)
r36   =addr of iofile entry for that file

a#io     addr of iofile entry for this file
access   access needed for this operation (tyxxx? bits)
filnum   file number we're working with
prnt#?   =1 if print#, else cleared
prnt#1   =1 if first print# item not printed yet, else cleared
```

Exit registers and RAM parameters:
The current linenumber and current datanumber should be updated in the Iofile.

What registers can be changed:
any, except usual registers under r20

Should HANDLD be set?
yes

Notes:
See Pr#val and Readn. subevents for information about what to do with each item. See Rh"nem for information about array name forms, etc.

See Rh"ice or Rh"eis to find out what tokens come in what order for the different commands. Mj/txt may be useful as a reference to see what the base machine does with the different tokens. Check with the keeper of assign# non-Kangaroo file type numbers if you want to add or use a particular type. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
8/4/82

Event:
V.ASN#   Data file manipulation of non-Kangaroo base machine file

Subevent:
Pr#end (Print# end token)

HANDI error:
63D (Illegal filespec)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called whenever the iofile type byte for the file we're going to use is not 0, and we've reached the end of our Print#.

What should the handler do?
If no items were print#ed to the line, the line should be deleted. The Prnt#1 flag should be cleared.

Entry registers and RAM parameters:

```
token =A7 =Print#end token
r14   =access needed for this operation
r24   =OCH if want to go to front of file
      =OFH if serial
      =BCD line number if random
r76   =file number (#1,#2,etc.)
r36   =addr of iofile entry for that file

a#io     addr of iofile entry for this file
access   access needed for this operation (tyxxx? bits)
filnum   file number we're working with
prnt#?   =1 if print#, else cleared
prnt#1   =1 if first print# item not printed yet, else cleared
```

Exit registers and RAM parameters:
The current linenumber and current datanumber in the Iofile should be updated.

What registers can be changed:
any, except usual registers under r20

Should HANDLD be set?
yes

Notes:
See Rh"ice or Rh"eis to find out what tokens come in what order for the different commands. Mj/txt may be useful as a reference to see what the base machine does with the different tokens. Check with the keeper of assign# non-Kangaroo file type numbers if you want to add or use a particular type. You can add as many bytes to your iofile entry as you like (until we get to 255), as long as you keep the line length updated. Just be sure to add them AFTER the type byte! Also, NEVER assume that memory hasn't moved between one token and the next.

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
1/27/82

Event:
V.ACC#   Access bits for data file didn't match our needs

HANDI error:
65D   (Illegal access)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called when the access bits for our data file don't match what we're trying to do (i.e., trying to print# to a rom file, or trying to read# a non-copyable, non-listable file), or when we don't know how to do what the user asks (like print# numbers to a text file)

What should the handler do?
If we call handi, we've given up, and are quitting
If it knows how to do the operation with the given access bits, it should go ahead and finish the operation, being sure to update the iofile entry (See V.ASN#)

Entry registers and RAM parameters:
token   what we're trying to do
r14   =access we need for this operation
r76   =file number we're working with
r34   =address of iofile entry for that filenumber

Exit registers and RAM parameters:
iofile updated to new status

What registers can be changed:
any

Should HANDLD be set?
yes

Notes:
Note that r14 (access we need for this operation) is negotiable. (i.e., if the handler can do it anyway, do it!)

HANDI CALL DOCUMENTATION

Mary Jo Hornberger
1/27/82

Event:
V.UNKD   Unknown data type encountered in data file

HANDI error:
33D   (data type)

Where invoked:
MJ&TXT

Under what conditions is this called?
Called when token in data file is not an END, integer, real, "ed string, 'ed string, or un'ed string

What should the handler do?
if restore#, print#, read#, restore, or read, just make sure current line#, current data#, address of current line, and address of current data are set up correctly in iofile.

for the other tokens, do the above, then finish out the token. (i.e., if read number, do it.) (See V.ASN# subevents for list of tokens)

Entry registers and RAM parameters:
token   token we're executing
r20   =data# of current item -1
r34   =address of entry in iofile
r32   =unknown token
r30   =address unknown token was popped from

Exit registers and RAM parameters:
none

What registers can be changed:
any

Should HANDLD be set?
yes

Notes:
See V.ASN# subevents for more specific information

HANDI CALL DOCUMENTATION

Raan Young
6/09/82

Event: V.SPY (Allow rom to do processing at end of program line, safely)

HANDI error: 18 (ROM MISSING)

Where invoked: IV&SER

Under what conditions is this called?
If BIT#1 of SVCWRD is set, SPY will issue a V.SPY handi call whenever SPY is called (EOL, GOTO, GOSUB, NEXT, etc).

What should the handler do?
Do whatever it wants to do at that point.

Entry registers and RAM parameters:
TOKEN: The token which preceded the SPY call.

Exit registers and RAM parameters:
None.

What registers can be changed:
All normally safe ones.

Should HANDLD be set?
Yes, to prevent error report.

Notes:
BIT#1 in SVCWRD is cleared by SPY after the HANDI call returns. Therefore, the rom doesn't need to clear it, must set it for each desired occurance, and can't set it during the V.SPY call.

HANDI CALL DOCUMENTATION

Raan Young
1/28/82

Event: V.ETRG (Allow extension of comparator machine for plugins)

HANDI error: 18 (ROM MISSING)

Where invoked: RY&CMP

Under what conditions is this called?
When Comparator receives trigger for external device number 1,2, or 3. This trigger is setup by call to CMPENT. When CMPCHK processes the comparator machine, it does the HANDI call for external devices.

What should the handler do?
Determine which external device needs service, and do it.

Entry registers and RAM parameters:
R20/21: device table entry [(R20/21 - DVCTBL)/2 = device number]

Exit registers and RAM parameters:
None

What registers can be changed:
R20-77 all protected

Should HANDLD be set?
Yes, if device recognized and serviced.

HANDI CALL DOCUMENTATION

Raan Young
1/28/82

Event: V.CARD (Allow roms a chance after each track of a card operation)

HANDI error: NONE

Where invoked: RY&CRD

Under what conditions is this called?
After each track, just before the track message is displayed, this HANDIO call lets roms do whatever they want before we start the next track.

What should the handler do?
Whatever he wants.

Entry registers and RAM parameters:
E: 0 if doing write, 1 if doing read
FLHEAD: first byte is track #, second byte is # of tracks
See RY"CRD for more details about card reader function and register usage.

Exit registers and RAM parameters:
Whatever is needed to achieve desired results.

What registers can be changed:
R14/15 and normal scratch registers, all else will have some effect.

Should HANDLD be set?
Does not matter, it is ignored.

Notes:
See card reader code and documentation for more information about possible changes which can be made with this HANDIO call.

HANDI CALL DOCUMENTATION

Raan Young
1/28/82

Event: V.TMCX (Extend time mode commands)

HANDI error: 78 (invalid command)

Where invoked: RY&TMC

Under what conditions is this called?
When command typed in is not recognized by Time mode.

What should the handler do?
Handle command, if possible.

Entry registers and RAM parameters:
R43/47: uppercased comand.

Exit registers and RAM parameters:
None

What registers can be changed:
Subject to normal restrictions, anything.

Should HANDLD be set?
Yes, if command handled.

HANDI CALL DOCUMENTATION

Gary Cutler
29-Jan-82

Event: V.PAR

HANDI error: none

Invoked: KR&PAR

Conditions: This call is initialized at every call to routine PARSER, including appointments.

What should the handler do?

This call provides the handler an opportunity to parse the the command string in the input buffer.

Entry registers and RAM pointers:

INPBUF: location of the command string <INPUT BUFFER>

Exit registers and RAM parameters:

On return to the Parser, the contents of the INPUT BUFFER will be parsed. The handler may or may not supply a new string in the INPUT BUFFER.

What registers can be changed?

any

HANDLED set? not applicable

HANDI CALL DOCUMENTATION

Event: V.STRA (sub a)

HANDI error: none

Invoked: KR&PAR

Under what conditions is this called?

There are two possible situations which can generate the V.STRA HANDI call. The first (sub a) is initiated when a parameter of a function is of unknown type (boffo). Some useful information follows.

What should the handler do?

The handler should be able to recognize the current parameter and it's type and complete the parsing of this parameter.

Entry registers and RAM pointers:

```
                    R12 STACK
                    -----------------------------------
R6 PTR ----->
                    -----------------------------------
                                  .
                                  .
                    Overhead from the HANDI call
                                  .
                                  .
                    -----------------------------------
                    8 bytes of the functions
                    attributes. The two MSBits
                    of the top byte describe the
                    type of the current parameter
                    in this case STRANGE' type
                    -----------------------------------
                    R40--R47  This is the information
                    from SCAN on the current parameter
                    -----------------------------------
                    R34--R35
                    -----------------------------------
                    R32  The function's token
                    -----------------------------------
```

At the time of the call the two MSBits of R57 will also contain the type of the current parameter.

Exit registers and RAM parameters:

The contents of the R6 stack (other than the HANDI overhead) must remain intack.

What registers may be changed:

any but R33

HANDLED set? not applicable

HANDI CALL DOCUMENTATION

Event: V.STRA (sub b)

HANDI error: none

Invoked: KR&PAR

Under what conditions is this called?

This event (sub b) will be initiated if the PARSER sees a string variable and the succeeding character is not '['.

What should the handler do?

Thsi call is specifically for parsing String Arrays. The handler should therefore parse the remainder of the String expression iff it is a String Array.

Entry registers and RAM parameters:

R14: current token (other than '[')
R20: next character
R10: ptr to INPUT BUFFER
R12: ptr to parsing stack

Exit registers and RAM pointers:

Parsed string array on the R12 stack
Scan must be called to obtain the necessary information on the next token

What registers can be changed?

not R10
not R20

HANDLED set? not applicable

HANDI CALL DOCUMENTATION

Event: V.DEC

HANDI error: none

Invoked: KP&DEC

Under what conditions is this called

This event is generated when the Decompiler sees a token with class > 56.

What should the handler do?

The handler should decompile the token contained in R23 and depending upon the attributes of the token place the ASCII string in the appropriate place on the R12 stack or the Decompilation Buffer

Entry registers and RAM parameters:

R23: contains the current token
R24: points to the next token in line
R45/46: contains the BCD line number
R30: points to the input buffer one space after the line number
PRECNT: contains the # of binary and unary operators in the current line which have already been decompiled
LAVAIL: contains the pointer to the location (on the R12 stack) and precedence of each of the operators mentioned above.

Exit registers and RAM parameters:

R24: pointer to next token
R30: pointer to 2nd byte after line # in buffer
PRECNT: updated
LAVAIL: updated

What registers can be changed?

R40-47
R50-57
R60-67
R70-77
R32-37
R20-22

HANDLED set? not applicable

HANDI CALL DOCUMENTATION

Event: V.IMER

HANDI error: none

Invoked: KR&RTS

Under what conditions is this called?

This event is initiated by the appearence of a strange RADIX while attempting to format output.(other than 0 , or .)

What should the handler do?

Format the number in R40 and output to the display

Entry registers and RAM parameters

```
R40: number to format
R54: contains the character count
R70: contains the sign type
     =    -1   ---> - if neg; blank for non-neg
     =     0   ---> - as a digit if neg; blank non-neg
     =     1   ---> - if neg; + if non-neg
     =     2   ---> - if neg; blank non-neg
R71: contains the type of fill
     -1 --> default
      0 --> DZ : digit position, z in units position
      1 --> D  : digit position
      2 --> *Z : left filled with *, z in units position
      3 --> *  : left filled with *
      4 --> Z  : left filled with 0
R72: radix (other than 0 , .)
R73: E/A format
     -1 --->  A format
      0 --->  none
      1 --->  E format (IMAGE)
      2 --->  E format (suppress +,-,lead zeros)
```

Exit registers and RAM parameters:

none

What registers can be changed?

any

HANDLED set? not applicable

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.TEST (The Test event for XYZZY to use)

HANDI error: NONE

Where invoked: IV&XYZ

Under what conditions is this called?
When XYZZY wants to time HANDIO calls, it calls HANDIO 10,000 times with the V.TEST event.

What should the handler do?
Nobody should ever handle this event.

Entry registers and RAM parameters:
None, you shouldn't handle this event.

Exit registers and RAM parameters:
None, you shouldn't handle this event.

What registers can be changed:
None, you shouldn't handle this event.

Should HANDLD be set?
No, cause you shouldn't handle this event.

Notes:
Don't handle this event.

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.CHED (The Character Editor event)

HANDI error: NONE

Where invoked: IV&ED

Under what conditions is this called?
When the character editor (CHEDIT) is called.

What should the handler do?
Perform whatever action it deems fit depending on R40. If it changes R40, the new R40 will be acted upon by the character editor. Or, it could perform some action of its own, and set R40=NOPKEY, which will do nothing.

Entry registers and RAM parameters:
R40: The key that is being passed to the editor
INPBUF: The input line
INPTR: Pointer to current position in INPBUF
LASTCH: Pointer to current last character of input.
I/RFLG: Insert/Replace mode flag.

Exit registers and RAM parameters:
R40: The key that will be passed to the editor, possibly changed by the handler
INPBUF: The input line
INPTR: Pointer to current position in INPBUF
LASTCH: Pointer to current last character of input.
I/RFLG: Insert/Replace mode flag.

What registers can be changed:
R41-47, R0-3

Should HANDLD be set? NO

Notes:
It is not possible to make a non-terminating key into a terminator with V.CHED. You can perform the function yourself and make R40=CR, though. See also INPCHK, which is called just before HANDIO, which can make a non-terminator into a terminator.

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.COLD (Machine is Coldstarting)

HANDI error: NONE

Where invoked: IV&INI

Under what conditions is this called?
Coldstart occurs when the machine is just turned on, after CNTL-SHIFT-CLR, when RAM is removed, or when we get a RAM checksum error.
During coldstart, just after initializing the RESMEM area and setting up a skeleton file system.
All devices have been initialized.

What should the handler do?
Anything that should only occur once, such as stealing memory from Kangaroo, clearing your printer, etc.

Entry registers and RAM parameters:
Practically everything.

Exit registers and RAM parameters:
Practically everything.

What registers can be changed:
All of them.
R6/7,R10/13,R16,R17 are used for other stuff, other.

Should HANDLD be set? NO

Notes:

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.WAIT (The WAITKY event, waiting for a key)

HANDI error: NONE

Where invoked: IV&IO

Under what conditions is this called?
At WAITKY, when the system wants to wait for a key.

What should the handler do?
Supply a key from another keyboard into KEYHIT, or perhaps convert the key that might be in KEYHIT.

Entry registers and RAM parameters:
KEYHIT: The current pending key, if any
SVCWRD: The low bit is 1 if there is a pending key in KEYHIT

Exit registers and RAM parameters:
KEYHIT: The key that is now pending
SVCWRD: The low bit set to 1 if a key is in KEYHIT

What registers can be changed:
R0-3

Should HANDLD be set? NO

Notes:

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.ENDL (PRINT End-of-line with unrecognized ROUTE)

HANDI error: NONE

Where invoked: LINEND in file IV&PRN

Under what conditions is this called?
Routine LINEND is called when PRINT/DISP code wants to generate an end-of-line. Where the end-of-line is sent is determined by ROUTE, which contains a code for the printer/display devices.
If ROUTE is not 1 or 2, then HANDI is called with V.ENDL.

What should the handler do?
If the ROUTE code is theirs, perform the appropriate output.

Entry registers and RAM parameters:
LINELN indirect: Current device width

Exit registers and RAM parameters:
NONE

What registers can be changed:
R0-3 (R32 for OUT)

Should HANDLD be set?
YES, if ROUTE is your code.

Notes:
This is intended to create other statements of the same class as PRINT and DISP, i.e., OUTPUT.
See V.CHAR for a similar event.

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.CHAR (PRINT character with unrecognized ROUTE)

HANDI error: NONE

Where invoked: Routine OUT in IV&PRN

Under what conditions is this called?
Routine OUT is called when PRINT/DISP code wants to output a character. The code inspects ROUTE to determine whether to write to the printer, display, or other devices. If it's not printer or display, we call HANDIO with V.CHAR.

What should the handler do?
If the ROUTE is theirs, perform the appropriate output.

Entry registers and RAM parameters:
R32: character to be output.

Exit registers and RAM parameters:
none

What registers can be changed:
R0-3, R32

Should HANDLD be set?
If this was your ROUTEing code.

Notes:
See V.ENDL for a similar event.

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.SLEE (Machine is going to deep sleep)

HANDI error: NONE

Where invoked: IV&ZZZ

Under what conditions is this called?
When the machine is going to go to deep sleep.

What should the handler do?
Whatever it wants to. Possibly release memory, reset the printer, rewind the tape drive, etc.

Entry registers and RAM parameters:
Nothing in particular.

Exit registers and RAM parameters:

What registers can be changed:
All except R6-13, R16/17, of course.

Should HANDLD be set? NO

Notes:

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.WARM (Maching is Warmstarting from deep sleep)

HANDI error: NONE

Where invoked: IV&ZZZ

Under what conditions is this called?
The machine has just warmstarted (woken up from deep sleep) and has computed RAM bounds & checksum, and re-enabled the ROM that was active when we went to sleep.

What should the handler do?
Whatever it wishes. Perhaps re-enable devices that it powered down when we went to deep sleep (see V.SLEE).

Entry registers and RAM parameters:
None in particular.

Exit registers and RAM parameters:
None in particular.

What registers can be changed:
All except normal Roo precious registers like R6-13, R16/17.

Should HANDLD be set? NO

Notes:

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.VOLI (Destruction of volatile files)

HANDI error: NONE

Where invoked: IV&ZZZ

Under what conditions is this called?
When we've detected a volatile file (filename begins with a period) during warmstart. We're going to destroy that file unless someone defends it.

What should the handler do?
Set HANDLD if the volatile file belongs to it.

Entry registers and RAM parameters:
R40/47: volatile file name

Exit registers and RAM parameters:
R40/47: file name to be destroyed

What registers can be changed:
R40/47, I suppose. This would cause a different file do be destroyed.
All others except standard precious registers.

Should HANDLD be set?
If the volatile file is yours and you wish to protect it.

Notes:
Volatile files are designed to be used by ROMs. If the ROM is pulled out, the file will be destroyed upon the next warmstart, thereby insuring that junk files don't get left hanging around.

HANDI CALL DOCUMENTATION

Jack Applin IV
January 28, 1982

Event: V.CRUN (Entering CRUNCH, the interpreter)

HANDI error: NONE

Where invoked: Routine CRUNCH in KR&EXE

Under what conditions is this called?
In CRUNCH, the interpreter, when we're about to start interpreting.

What should the handler do?
Possibly change the top environment to go interpret something else.
Or perhaps change R16/17 to not interpret anything.

Entry registers and RAM parameters:
R16: Running status (0=idle, 1=calculator, 2=program)

Exit registers and RAM parameters:
R16

What registers can be changed:
All except standard Roo precious registers.

Should HANDLD be set? NO

Notes:

Robert Heckendorn
Jan 27, 1982 AD

Event: V.SPEC file specifier parsing

Subevent: R53=0 target file parsing

HANDI error: 63 (invalid file/device name)

Where invoked: RH&CMD

Under what conditions is this called?
When looking for a target file name in a statement of the form XXXX source TO target.
E=2 from FLORD of string for target name.
This occurs for syntax errors and access errors but access errors cause FLORD to do a premature return so HANDI is not called.

Entry registers and RAM parameters:
R53 - 0 indicates target file

Exit registers and RAM parameters:
```
R34/37 - device name (zero if none)
R52 - 0 handled but no device present
    - 1 handled device present
    - 2 not handled
R60/67 - target filename
R70/74 - password
```

Event: V.SPEC file specifier parsing

Subevent: R53=1 source file parsing

HANDI error: 63 (invalid file/device name)

Where invoked: RH&CMD

Under what conditions is this called?
When looking for a source file name in a statement of the form XXXX source TO target.
E=2 from FLORDT of string for source name.
This occurs for syntax errors and access errors but access errors cause FLORDT to do a premature return so HANDI is not called.

Entry registers and RAM parameters:
R53 - 1 indicates source file

Exit registers and RAM parameters:
```
R40/47 - source filename
R50/51 - "special file" file type bytes
         -1 if not a special filename
R52 - 0 handled but no device present
    - 1 handled device present
    - 2 not handled
R54/57 - device name (zero if none)
R74/77 - password
```

Event: V.FILE unable to handle this file with this command

Subevent: TOKEN=RENATK

HANDI error: 63 (invalid file/device name)

Where invoked: RH&CMD rename code

Under what conditions is this called?
unable to handle the filename with device.
This is indicated by FLSPEC flag being positive.

What should the handler do?

Entry registers and RAM parameters:
```
R34/37 - target device (R34 zero if not found)
R60    - target filename (blank if not found)
R70/73 - target password (blank if not found)
R24/27 - source device (R24 zero if not found)
R50    - source filename (blank if not found)
R74    - source password (blank if not found)
```

Exit registers and RAM parameters:

What registers can be changed:

Should HANDLD be set? yes

Notes:

Event: V.FILE unable to handle this file with this command

Subevent: TOKEN=COPYTK

HANDI error: 63 (invalid file/device name)

Where invoked: RH&CMD in the copy code

Under what conditions is this called?
unable to handle the filename with device.
This is indicated by FLSPEC flag being positive.

What should the handler do?
Try to copy the files given.

Entry registers and RAM parameters:
```
R17 - the stall bit may be set if the source file
      is allocated.
R34/37 - target device (R34 zero if not found)
R60    - target filename (= source name if not found)
          (pointer in R66 to directory entry
           if in RAM)
         filename is valid uppercase filename
          (is not volitile filename)
R70/73 - target password (blank if not found)
R24/27 - source device (R24 zero if not found)
R50    - source filename (= target name if not found)
          (pointer in R56 to directory entry
           if in RAM)
         filename is valid uppercase filename
          (may be volitile filename)
R74    - source password (blank if not found)
```

Exit registers and RAM parameters:

What registers can be changed:

Should HANDLD be set? yes

Notes: If the stall bit (see R17) is set and KR system routines are called, they may fail to behave correctly.
This effects those routines that check R17 to see if an error was called or the program was stalled internally but assume that the stall bit is NOT set on entry e.g. PTRALD.

```
Event: V.FILE unable to handle this file with this command

Subevent: TOKEN=PURGTK

HANDI error: 63 (invalid file/device name)

Where invoked: RH&CMD

Under what conditions is this called?
       Unable to handle the filename with device or
          there is a syntax error
       This is indicated by FLSPEC flag being positive.

What should the handler do?
       purge the given file

Entry registers and RAM parameters:
       E=1 - device name present
        =2 - syntax error in filename (ERROR SET)
       R14 - the mask for the access byte
       R15 - the type byte if file exists in mem
              or all ones if not
              (undefined for sytax errors)
       R30 - pointer to directory if exists
       R40 - filename blank filled
       R50/51 - special file type or FFFF if no
              special file mentioned
       R54/57 - device name
              (zero if no device)
       R74/77 - password right filled with blanks

Exit registers and RAM parameters:
       none

What registers can be changed:
       R20-77

Should HANDLD be set? yes

Notes:
```

```
Event: V.FILE

Subevent: TOKEN=EDITTK

HANDI error: 63 (invalid file/device name)

Where invoked: RH&RUN

Under what conditions is this called?
       Unable to handle the filename with device or
          there is a syntax error
       This is indicated by FLSPEC flag being positive.

What should the handler do?
       EDIT the given file

Entry registers and RAM parameters:
       E=1 - device name present
        =2 - syntax error in filename (ERROR SET)
       R14 - the mask for the access byte
       R15 - the type byte if file exists in mem
              or all ones if not
              (undefined for sytax errors)
       R30 - pointer to directory if exists
       R40 - filename blank filled
       R50/51 - special file type or FFFF if no
              special file mentioned
       R54/57 - device name
              (zero if no device)
       R74/77 - password right filled with blanks

Exit registers and RAM parameters:
       none

What registers can be changed:
       R20-77

Should HANDLD be set? yes

Notes:
```

```
Event: V.FILE

Subevent: TOKEN=CATTK

HANDI error: 63 (invalid file/device name)

Where invoked: RH&CAT

Under what conditions is this called?  When there is an unrecongnized
              device or there is a syntax error in the filename

What should the handler do?  give a catalog line for the given file

Entry registers and RAM parameters:
       E=0 - device name present
        =1 - syntax error in filename (ERROR SET)
       R14 - the mask for the access byte
       R15 - the type byte if file exists in mem
              or all ones if not
              (undefined for sytax errors)
       R30 - pointer to directory if exists
       R40 - filename blank filled (if E=1)
       R50/51 - special file type or FFFF if no
              special file mentioned
       R54/57 - device name
              (zero if no device)
       R60 - filename blank filled (if E=0)
       R74/77 - password right filled with blanks

Exit registers and RAM parameters:
       none

What registers can be changed:
       R20-77

Should HANDLD be set? yes

Notes:
```

```
Event: V.TYPE

Subevent:

HANDI error: none

Where invoked: RH&FIL

Under what conditions is this called?
     R14 has been returned by the parser and we are now
     looking for a file type token (e.g. TEXT, BASIC)

What should the handler do?
    look at R14 and R43 to see if the token scanned is a
    file type.  The handler must remove the HANDI return
    info from the stack and set the Z flag to indicate
    if the token was 'equal to' a filetype name.

Entry registers and RAM parameters:
    R14 - the token number
    R43 - the ROM#

Exit registers and RAM parameters:

What registers can be changed:
    R50-77
Should HANDLD be set? no (see above)

Notes:
```

Event: V.DALO

Subevent:

HANDI error: none

Where invoked: RH&DAL

Under what conditions is this called? An external ROM token is
being deallocated. The external ROM has been switched to and
the token information has been fetched. The token class of
the token had to be greater than 56 octal before this HANDI
call would be made. The other classes are assumed not to need
to be deallocated.

What should the handler do?
deallocate the token

Entry registers and RAM parameters:
R23 - the token number of the external ROM token
R24/25 - pointer to the next token or information
R36/37 - the token class (always -1 by the time you get here)

Exit registers and RAM parameters:

What registers can be changed:
R50-77

Should HANDLD be set? yes

Notes:

Event: V.ALLO

HANDI error: none

Where invoked: RH&PAL

Under what conditions is this called? An external ROM token is
being deallocated. The external ROM has been switched to and
the token had to be greater than 56 octal before this HANDI
call would be made. The other classes are assumed not to need
to be deallocated.

What should the handler do?
pointer allocate the token

Entry registers and RAM parameters:
R23 - the token number of the external ROM token
R24/25 - pointer to the next token or information
R36/37 - the token class (always -1 by the time you get here)

Exit registers and RAM parameters:

What registers can be changed:
R50-70

Should HANDLD be set? yes

Notes:

Event: V.EALO

HANDI error: none

Where invoked: RH&EAL

Under what conditions is this called? during environmental allocation
when the variable is found to have a non-zero reference mode or
is a string array.

What should the handler do?
allocate the space provided for in the environment

Entry registers and RAM parameters:
R32 - end of variable pointer area
R34 - pointer to 4 bytes after the name of
the currently processing variable
R54/55 - name form of the variable
R56/57 - next two bytes (usually a pointer)
R3 - the upper 2 bits of variable type
the access method type

Exit registers and RAM parameters:
R34 - pointer to next variable in VPA

What registers can be changed:

Should HANDLD be set? yes

Notes: space for what ever initail values you wish to
place in the environment must already be provided for
in the VPA which occured during pointer allocation.

Event: V.RUN

HANDI error: none

Where invoked: RH&RUN

Under what conditions is this called?
When we are prepared to construct the environment for a
program to run.

What should the handler do?
serving suggestion:
fiddle with parameters to CALL or avoid it altogethe:

Entry registers and RAM parameters:
R22 - ROM number of the file to run
R40 - name of the file to run
R76 - the line number to start running at

Exit registers and RAM parameters:

What registers can be changed:

Should HANDLD be set? yes

Notes:

Event: V.CALL

HANDI error: none

Where invoked: RH&RUN

Under what conditions is this called?
    after an environment has been created.

What should the handler do?
    adjust the environment to his liking, for instance.

Entry registers and RAM parameters:
    R16 - the run mode (idle, calc, run etc)
    R34 - pointer to the environment
    R40 - the filename of the mother of the environment

Exit registers and RAM parameters:

What registers can be changed:

Should HANDLD be set? yes

Notes:

Event: V.CAL.

HANDI error: none

Where invoked: RH&RUN

Under what conditions is this called?
    we are ready to establish the environment for the program

What should the handler do?
    do anything that needs to be done to prepare to establish the
    program environment the way you want.

Entry registers and RAM parameters:
    R22 - ROM number of the file to run
    R40 - name of the file to run
    R76 - line number (default is zero)

Exit registers and RAM parameters:

What registers can be changed:

Should HANDLD be set? yes

Notes:

Event: V.STAL

HANDI error: none

Where invoked: RH&RUN

Under what conditions is this called?
    when a program stalls. This includes attention and STOP
    as well as errors and END.

What should the handler do?
    clean up anything tied to a running program

Entry registers and RAM parameters:
    nothing too useful

Exit registers and RAM parameters:

What registers can be changed:

```
Should HANDLD be set? NO NEVER!! BEWARE!! POTENTIAL DISASTER!! ->-|
                      |-------<---------------<-------------<---------|
                      |-> math rom may not work if HANDLD set by
                          another rom here
                                              --Joseph Albert
```

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.APTO (Start of appointment command loop)

HANDI error: None

Where invoked: KR&PS1

Under what conditions is this called?
    At the start of the appointment command loop prior to displaying the
    template/appointment and inputing a command key.

What should the handler do?
    Perform any custom application desired at start of appointment command
    loop prior to appointment display and command input. This intercept
    would be used to replace the normal appointment mode features with a
    completly custom application.

Entry registers and RAM parameters:
    R30/31: pointer to start of appointment file
    R32/33: pointer to current appointment in appointment file
    R34/35: pointer to end of appointment file
    PS1OST: appointment input/display information

Exit registers and RAM parameters:
    Same as above

What registers can be changed:
    All except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.AKEY (Add, delete, or modify processing of APPT command keys)

HANDI error: None

Where invoked: KR&PS1

Under what conditions is this called?
After an input operation in appointment mode has been completed and the terminating key has been identified but before the corresponding processing routine has been invoked.

What should the handler do?
Test for any terminator that might require custom processing and perform any custom processing that is required.

Entry registers and RAM parameters:
- R24: size of appointment input (in INPBUF)
- R25: value of terminator key
- R30/31: pointer to start of appointment file
- R32/33: pointer to current appointment in appointment file
- R34/35: pointer to end of appointment file
- INPBUF: appointment input
- PSIOST: appointment input/display information

Exit registers and RAM parameters:
Same as above

What registers can be changed:
All except 10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.ARTN (Custom processing at exit from appointment mode)

HANDI error: None

Where invoked: KR&PS1

Under what conditions is this called?
When a mode switching key has been entered from appointment mode but prior to exiting appointment mode.

What should the handler do?
Any special clean-up that may be required before exiting appointment mode due to any custom features that may have been added.

Entry registers and RAM parameters:
- R30/31: pointer to start of appointment file
- R32/33: pointer to current appointment in appointment file
- R34/35: pointer to end of appointment file

Exit registers and RAM parameters:
- R30/31: pointer to start of appointment file
- R34/35: pointer to end of appointment file

What registers can be changed:
Any except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.AFET (Custom processing for Fetch key in appointment mode)

HANDI error: None

Where invoked: KR&PS1

Under what conditions is this called?
When a Fetch key has been entered in appointment mode.

What should the handler do?
Any custom processing required for a Fetch key. (This key has been intended to have an appointment search feature tied to it).

Entry registers and RAM parameters:
- R30/31: pointer to start of appointment file
- R32/33: pointer to current appointment in appointment file
- R34/35: pointer to end of appointment file
- INPBUF: current appointment input

Exit registers and RAM parameters:
Same as above

What registers can be changed:
Any except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.CLOK (Custom processing at clock comparator service call)

HANDI error: None

Where invoked: KR&PS1

Under what conditions is this called?
At the start of the service routine for processing a comparator interrupt for the clock.

What should the handler do?
Any special processing that may be associated with a custom application that uses the clock interrupt; such as a stopwatch feature.

Entry registers and RAM parameters:
- INPBUF: current time/date display
- PSSTAT: Bit#0=1 iff time display is disabled

Exit registers and RAM parameters:
Same as above

What registers can be changed:
All except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.AFMT (Custom appointment formatting)

HANDI error: None

Where invoked: KR&PS2

Under what conditions is this called?
When APTDSP is called to take the encoded appointment pointed at by the current appointment pointer (R32/33) and decode it into the input buffer

What should the handler do?
Any custom appointment display processing that may be required; such as translating the day of week fields into a foreign language.

Entry registers and RAM parameters:
R32/33: pointer to encoded appointment to be decoded

Exit registers and RAM parameters:
Same as above

What registers can be changed:
Any except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.AERR (Custom error handling in appointment mode)

HANDI error: None

Where invoked: KR&PS2

Under what conditions is this called?
When APTERR is called to set up the display in response to an erroneous entry in appointment mode.

What should the handler do?
Any special error handling that may be required for a custom application in appointment mode.

Entry registers and RAM parameters:
R20: error number (binary)

Exit registers and RAM parameters:
Same as above

What registers can be changed:
All except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.APRC (Custom appointment processing)

HANDI error: None

Where invoked: KR&PS5

Under what conditions is this called?
When APPROC is called to process a pending appointment.

What should the handler do?
Any special appointment processing that may be required by a custom application.

Entry registers and RAM parameters:
None

Exit registers and RAM parameters:
None

What registers can be changed:
Any except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.AACK (Custom processing during appointment acknowledge)

HANDI error: None

Where invoked: KR&PS5

Under what conditions is this called?
When APTACK is called to acknowledge the current appointment and perform the associated housekeeping chores

What should the handler do?
Any special processing or housekeeping chores that may be associated with acknowledging an appointment in some custom application.

Entry registers and RAM parameters:
R30/31: pointer to start of appointment file
R32/33: pointer to current appointment (one to be acknowledged)
R34/35: pointer to end of appointment file

Exit registers and RAM parameters:
Same as above

What registers can be changed:
Any except for R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.AIRG (Custom processing at appointment triggering)

HANDI error: None

Where invoked: KR&PS5

Under what conditions is this called?
When APIRIG is called to perform the necessary tasks associated with triggering an appointment when an appointment interrupt has been detected by CMPCHK

What should the handler do?
Any special processing that may be required to trigger an appointment in a custom application.

Entry registers and RAM parameters:
None

Exit registers and RAM parameters:
None

What registers can be changed:
Any except R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Mark Rowe
Feb. 2, 1982

Event: V.ANOT (Custom processing of appointment note field)

HANDI error: None

Where invoked: KR&PS5

Under what conditions is this called?
When PANOTE is called to set up the processing of a note or BASIC command that is in the note field of an appointment being processed.

What should the handler do?
Any special processing that may be associated with the note field in an appointment within a custom application.

Entry registers and RAM parameters:
R32/33: pointer to appointment being processed

Exit registers and RAM parameters:
Same as above

What registers can be changed:
Any except for R10/17

Should HANDLD be set? no

Notes:

HANDI CALL DOCUMENTATION

Gary Cutler
June 14,1982

Event: V.DIM

Error: none

Invoked: KR&PAR

Under what conditions is this generated?

This call is generated while parsing a dimension statement in which a string token is followed by a character other than '['

What should the handler do?

Parse the string array and do a post SCAN.

Entry registers and RAM parameters:

R14: current token
R10: input buffer pointer
R20: next character
R12: stack pointer

Exit registers and RAM parameters:

R14: next token
R10: input buffer pointer
R20: next character
R12: stack pointer

What registers may be changed?

All except R10/11 and R20/21

Should HANDLD be set?

not necessary

Notes:

Kangaroo Wake Up Procedure
Mary Jo Hornberger
July 8, 1982

Kangaroo Wakeup Procedure

This document details the wakeup procedure for Kangaroo. It is separated into three sections:

1) things always done
(at both warmstart and coldstart)

2) things done only at coldstart
(full initialization)

3) things done only at warmstart
(partial initialization)

-1- KR"HI

Kangaroo Wake Up Procedure
Mary Jo Hornberger
July 8, 1982

Things always done

A) Hardware detects comparator interrupt or ATTN key

The process of waking up from deep sleep starts with the hardware detecting either a comparator interrupt or an ATTN key. The hardware will force a jump to the address given in the power on interrupt vector, where the operating system starts executing.

B) Clear LCD and check for diagnostic ROM

The first thing the operating system does is to assert binary mode and clear the LCD. We then check to see if the diagnostic ROM is plugged in. If we find it, we will let it take over control of the machine. The only way to get back to Kangaroo from the diagnostic ROM is to coldstart us again.

C) Enable system ROM and decide on warmstart or coldstart

The system ROM (ALTROM) is enabled by writing to address FF46H.

To decide if we need to coldstart or warmstart, we read the Power Supply Status Byte (PSSB). If the POR bit (bit#2) is a zero, it means power has been lost since we were awake last, and the RAM may not be valid. In this case we need to go through the coldstart procedure again, resetting all of the global variables, and reinitialize the file system, leaving the user with a functionally empty Kangaroo.

If the POR bit is a 1, the RAM is still valid, so all we need to do is to restore the CPU registers 6,10,12, and 16 from TMPMM2, and do a return. The address for the warmstart wakeup has been set up on R6 by the going-to-sleep code, so this essentially accomplishes a jump to the warmstart code. Notice that if some routine wants to put us to sleep and have us wake up through a different warmstart routine, all it has to do is adjust the warmstart wakeup address on R6.

Things done only at coldstart

(Full machine initialization)

A) Clear POR and initialize some CPU registers

The coldstart initialization starts with clearing the POR bit in the PSSB. This guarantees that if something interrupts us and sends us to sleep before our initialization is done, we will coldstart again when we wake up. This bit will already be zero if this is the first time awake since power was lost, but if we are coldstarting because the software detected invalid RAM (either a bad checksum or RAM taken away), the software must clear POR.

The R6 subroutine return stack is set to address 8000H, the CPU registers 10 through 17 octal are cleared, (clearing the stall byte and putting Kangaroo into idle mode), and binary mode is asserted again.

B) Compute the RAM boundary

The RAM boundary is computed by reading two bytes from the lowest end of RAM (32K), complementing one of them, writing them back, and then reading them again to see if they are the same as what we just wrote. If they are the same, we move up 2K and try it again. If they are different, it means that we have reached the first non-writable byte of memory (ie, we're trying to write to RAM that isn't there). This RAM boundary value will be stored in LWAMEM.

This scheme assumes that RAM will always be added in a multiple of a 2K increment, up to a 64K maximum size. The two byte write and read is necessary because if we write just one byte to a non-existent location and then immediately read it back, we will indeed read what we just wrote, (even though there was no RAM there), due to bus capacitance. For this reason, two different bytes must be written and read. By complementing one of the bytes, we guarantee that if there is no RAM there (in which case the two bytes read would be identical), we still write two distinctly different bytes.

C) Clear RAM

After determining how much RAM we have, we clear all RAM from 960 bytes before the input buffer (RGLIM1) to the end of memory (LWAMEM). This saves individual initialization of global variables that are initially cleared.

D) Make sure buzzer is off

Bit#0 in the Comparator Status Byte is cleared, forcing the initial state of the buzzer to be off.

E) Build ROM table

A table of enable addresses for all ROMs plugged in (ROMTAB) is created. The last three entries in the table are the internal Kangaroo ROMs: BASROM, ALTROM, and MELROM.

F) Initialize Real Time Clock and Time and Appointment variables

The Real Time Clock (RTC) hardware is initialized by writing a zero to address FF81H. The Time and Appointment variables, including the clock rollover and adjust factors are initialized. The rollover value is written to the comparator, and the comparator interrupt is enabled.

G) Set up operating modes

VERIFY ON, BEEP ON, and DEFAULT ON modes are asserted by setting CRDSTS, BEEPOK, and DEFAUL to 1. The maximum number of bytes per track per card (FULTRK) is set to 650D. The display and print widths (DISPLW and PRNTLN) are set to 32D.

H) Enable the keyboard interrupt

The keyboard interrupt is enabled by setting bit#1 in the Keyboard Status Byte.

I) Initialize the intercepts to returns

There are ten intercepts in Kangaroo. These intercepts are each 8 bytes long, and give the interceptor a chance to change the function of a particular area of code by causing the intercept to jump to an address that the user has set up. The intercepts are all initialized to returns.

Six of the intercepts are executed in the mainframe:

Intercept	Location
KYIDLE	keyboard interrupt service routine
PSTRAP	low power interrupt service routine
CMPINT	comparator interrupt service routine
IMERR	image error processing
HPINTC	card reader header processing
PILTRP	HPIL frame sending routine (PILTRP is also the HPIL interrupt service routine)

There are also four spare intercepts, three of which are interrupt vectors:

Intercept	Function	Interrupt vector address
EXTRAH	unused	(not interrupt vector)
SPAR0	bar code reader	vector address 000AH
SPAR1	unused	vector address 000EH
SPAR2	unused	vector address 0010H

INPCHK, which is like a two-byte intercept vector in the character editor, is initialized to the address of a return.

J) Initialize the LCD globals

We now initialize the LCD globals. The width of the LCD (SIZSIZ) is set to 32D, the leftmost LCD address (MINMIN) to 80D, and the rightmost LCD address (MAXMAX) to 49D. The left edge of the LCD display window (LCDWIN) and the LCD cursor position (LCDPTR) are set to the far left edge of the LCD.

The right margin for input (RMARG) is set to 91D. The pointer to the last character of input (OLDLST) is set to the start of the input buffer, signifying that there hasn't been any input yet. The flag DEAD is set to nonzero, signalling that we may purge the current LCD line when we receive the next character.

K) Make sure the card reader is off

Five bytes of zeros are written to the card reader hardware (CRDRDR) to make sure the card reader is off. (Two bytes are all that are necessary, the other three are written for coding convenience.)

L) Set the delay

The machine DELAY is set to one-half second.

M) Initialize miscellaneous globals

The HPIL globals, GETIN parameters, and the random seed are initialized, and STANDBY OFF mode is asserted. (STANDBY OFF causes a HPIL timeout after 10 seconds, and a machine timeout to deep sleep after 5 minutes of inactivity.)

N) Set up low power detect and enable global interrupts

We determine if we have NiCad or alkaline batteries present (see KR"LOW), and set up the corresponding first interrupt level. The initial interrupt level is 3.55 volts if NiCad batteries, and 3.10 volts if alkalines. (Kangaroos plugged into the wall are treated as having NiCad batteries).

The global interrupts are enabled as we exit the low power initialization routine.

O) Create an empty file system

The GOSUB/RETURN stack pointer (NXTRTN) and the address of the last available byte of RAM (LAVAIL) are set to the last byte of RAM. The LEEWAY for the R12 stack (the minimum required distance from the bottom of the R12 stack to the last available byte of RAM) is set to 576 bytes.

An empty directory is created, and the following globals are set to the first byte after the directory:

FWUSER	the bottom of the environment stack
FWVARS	the current environment
NXTMEM	the top of the environment stack
TOS	the bottom of the active R12 stack
STSIZE	the pointer to the statement size

P) Do the coldstart HANDIO call

The HANDIO call V.COLD is executed to give plug-in ROMS a chance to do any extra initialization they might need.

Q) Ask user to set time

A protected field template is displayed for the user, asking the user to fill in day, date, and time.

R) Set up some initial files

The calculator program (name 'calcprog', type Basic) is created. The environment stack is initialized, creating an idle environment for the calculator variable file. This environment remains in Kangaroo until the next time we coldstart.

The initial 'workfile', (type Basic) is created, and made the current editfile and runfile. The current line number is set to 0.

The 'iofile', which is the system file for keeping track of assign# information, is created and a dummy record is inserted at line 0 for data and read statements.

S) Set POR to 1

The POR bit in the PSSB is set to 1 so next time we wake up we won't coldstart.

T) Go to the mode switcher

Having finished our initialization, we jump to the mode switcher with a TIMEKY as the current key. The mode switcher will see the TIMEKY and send us to time mode, where we'll display the time for the user.

Things done only at warmstart
(Partial initialization)

A) Enable the system ROM

The first thing done by the warmstart code is to disable all ROMs and then enable the system ROM (ALTROM). Even though this is usually done earlier by the initial wakeup code, we need to do it again, in case the initial wakeup code was never executed. This can happen if we have a comparator interrupt pending while we are trying to tell the hardware to put us to sleep. In this case, we will immediately start executing the warmstart code, without going to sleep and waking up through the wakeup vector.

B) Decide who woke us up

If the ATTN key woke us up, we want to go to EDIT mode when we finish the warmstart, but if a comparator interrupt woke us up, we want to go back to sleep. We determine which kind of wakeup this is by checking bit#4 of the Keyboard Status Byte. This bit will be a 1 if this is a comparator wakeup, and a 0 if it is an ATTN key wakeup. We set up R25 with the key that corresponds to the mode we want to end in, either a NAPKEY for sleep mode, or an EDITKY for EDIT mode.

C) Check to see if RAM was added or taken away

The RAM bound is recomputed to see if RAM was added or taken away while we were asleep. (For details of this procedure, see the coldstart section).

If we have less RAM than when we went to sleep, we report 'ERROR: RAM is invalid', and coldstart Kangaroo, since several of our necessary pointers will no longer be there.

D) Compute the checksum

The checksum is computed for the amount of RAM we had when we went to sleep, and compared with the checksum computed before we went to sleep. If the checksums are different, we report 'ERROR: RAM is invalid' and coldstart Kangaroo.

If RAM was added, we move the upper memory information (the bytes that reside between LAVAIL and LWAMEM) to the new upper memory bounds. The global pointers between LAVAIL and LWAMEM are updated to show the new addresses. (The new address = old address + amount of RAM added).

E) Make sure the card reader is off, and clear pending key interrupts

The card reader is set to the off state by writing two bytes of 0 to the card reader hardware. Any pending key interrupts are cleared by setting bit#1 of the Keyboard Status Byte.

F) Turn on HPIL chip oscillators if needed

If we are in STANDBY ON mode, the HPIL chip oscillators need to be turned on. This is determined by testing bit#0 of STAND?. If it is set, we initialize the HPIL chip and turn on the oscillators.

G) Initialize intercepts and the LCD display

The intercepts are reinitialized to returns, the LCD is cleared, and the annunciators are redisplayed.

H) Initialize low battery detect sequence and enable global interrupts

The battery type is checked, and the appropriate first interrupt level is set up: 3.55 volts if NiCad batteries, and 3.10 volts if alkalines.

The global interrupts are enabled at the end of the low battery initialization.

I) Make a ROM table

Now we set up a table of all ROMs that are plugged in, with the three system ROMs (BASROM, ALTROM, and MELROM) last. We enable the ROM that was enabled when we went to sleep, issuing 'ERROR: ROM missing' if it is no longer there.

J) If wakeup is from comparator, go to sleep machine

If the wakeup was from the comparator, we jump to the sleep machine, where we will take care of the timer or appointment. See KR"BYE or KR"CMP for more information about the comparator interrupt processing.

The rest of the initialization is done only for wakeups by the ATTN key.

K) Check for LOCK password

If the user has required a LOCK string, we check to see if it was supplied. If the first eight bytes of the supplied password do not match the expected password, we jump to the sleep machine and put ourselves back to sleep. See KR"LOK for more information on the LOCK token.

L) Check the HPIL loop

Bit#7 in PLSTAT is cleared. This bit is available for plug-in ROMs to set if they want us to do a HANDI call before sending any HPIL frame out on the loop. Clearing this bit prevents us from doing these extra HANDI calls if the ROM that set the bit was pulled out while we were asleep.

If the user has assigned a DISPLAY IS device, Kangaroo will send every character the user types around the HPIL loop. If any of the peripherals are off, the frame will never come back.

If the user tries to type on Kangaroo while the loop is dead, (with a DISPLAY IS device assigned), there will be either a 10 second wait before Kangaroo gives up and displays the 'Loop timeout' error message (in the case of STANDBY OFF), or worse, Kangaroo will sit there patiently waiting forever for the frame to come back (STANDBY ON).

To prevent this from happening, the 'test loop before using' bit in PLSTAT is set at wakeup. This will cause an IDY frame to be sent around the loop before we try to use it. Since the IDY frames are automatically retransmitted by the HPIL chip hardware, we can tell quickly if the loop is functional or not, without making the user wait the timeout period.

If there are any DISPLAY IS devices assigned, an UNListen frame is sent. This is a result of using the same HPIL subroutine for wakeup that we use when we are going to sleep.

M) Do the warmstart and volatile file HANDIO calls

The warmstart HANDIO call (V.WARM) is done to give plug-in ROMs a chance to do any warmstart initialization. After polling all the ROMs with event V.WARM, we assert binary mode again, and do the volatile file

HANDIO call (V.VOLT) for each volatile file in the directory. Each volatile file that does not set HANDLD is purged. After checking all the files in the directory, we execute a return, which puts us back in the mode switcher. The mode switcher will see the EDITKY, and send us to EDIT mode (unless a terminating mode switcher (TIMEKY, etc.) was pressed when entering the LOCK password, in which case we will go to that mode). Kangaroo is now fully awake and ready for input.

N) Globals used:

For more information on the initialization of globals not mentioned here, use the cross-reference to locate their initialization routines. Remember that all globals are set to zero when RAM is cleared at coldstart, so any globals that do not have initialization routines will still be zero until changed by the user.

Caution: some variables are initialized in 8 byte chunks, so the initialization routine for a specific location may not be mentioned by that name in the cross-reference. In this case, the routines involving the 8 bytes in front of it may also need to be checked.

I/O addresses:

CRDRDR	FF08H	Cardreader Status Byte
CMPSB	FF80H	Comparator Status Byte
GINTDS	FF01H	Global Interrupt Disable Addr
GINTEN	FF00H	Global Interrupt Enable Addr
KEYSTS	FF02H	Keyboard Status Byte
PSSB	FF82H	Power Supply Status Byte
RTC	FF81H	Real Time Clock Status Byte

Major entry points:

KILLIT	in KR&ZZZ	Reports RAM error, coldstarts
START	in KR&INI	Coldstart code
START?	in KR&INI	Decides on warm or coldstart
START+	in KR&EXE	Hardware vectors here at wakeup
WAKEUP	in KR&ZZZ	Warmstart code

Related routines:

ZZZZZZ	in KR&ZZZ	Start of the Sleep machine
BYE.	in KR&ZZZ	This will also put us to sleep

Related documents:

KR"PIL	HPIL theory and implementation
KR"LOK	Information about LOCK sequence
KR"CMP	Comparator information
KR"TMC	Time mode information
KR"HDI	Intercepting Handi calls
KR"LOW	Low power detect information
Roo Chip ERS	Information about the Keyboard Status Byte, Comparator Status Byte, Power Supply Status Byte, Real Time Clock Status Byte, and Global Enable and Disable addresses
Card Reader ERS	The card reader hardware
HPIL chip ERS	The HPIL chip hardware
Kangaroo Software ERS	How the Kangaroo operating system works
Kangaroo Owners' Manual	How Kangaroo appears to the user, including a table of coldstart and warmstart states of settable operating modes, such as DEFAULT ON/OFF and PWIDTH.
Lex Files for Kangaroo	More on intercepting warm or coldstart Handi calls

ROO INTERNAL CODE EXAMPLES
april 20, 1981 AD

The following are examples of unallocated code from the heart of the kangaroo.

1) A=12345

```
<ftadr> [name] <intcon> [integer constant] <stosv>
  11     20 41    1A         45 23 01          08
```

2) A$='fh'

```
<ftstls> [name] <'ed str> [string constant] <stost>
   13     20 41     96        02 66 68         07
```

3) B=A

```
<ftadr> [name] <ftsvl> [name] <stosv>
  11     20 42    01    20 41    08
```

4) B$=A$

```
<ftstls> [name] <ftstl> [name] <stost>
   13     20 42    03    20 41    07
```

5) A(1)=2

```
<svadr+> [name] <intcon> [integer constant] <avadr1>  ...
   12     20 41    1A         01 00 00          09

<intcon> [integer constant] <stosv>
   1A        02 00 00          08
```

6) A$[1]="a"

```
<ftstls> [name] <intcon> [integer constant] <1 dimsub>  ...
   13     20 41    1A        01 00 00            1D

<"ed str> [string constant] <stost>
   05          01 61          07
```

7) A=B(1)

```
<ftadr> [name] <svadr> [name] <intcon> [integer constant] ...
  11     20 41    02    20 42    1A        01 00 00

<avval1> <stosv>
   0B       08
```

```
8) A$=B$[1]

  <ftstls> [name] <ststl> [name] <intcon> [intger constant] ...
     13      20 41    03    20 42    1A        01 00 00

  <subst1>
     1D

9) A,B=4

  <ftadr> [name] <ftadr> [name] <intcon> [integer constant] ...
     11    20 41    11    20 42    1A         04 00 00

  <stosvm> <stosvm>
     14       14

10) A$[4,6]="u"

  <ftstls> [name] <intcon> [integer constant] <intcon> ...
     13     20 41    1A        04 00 00          1A

  [integer constant] <subst2> <"ed str> [string constant] <stost>
     06 00 00           1E        05          01 77           07

11) INTEGER S,T

  <integer> <ftsvl> [name] <ftsvl> [name]
     7F        01    20 53    01    20 54

12) DIM B(3)

  <dim> <svadr> [name] <intcon> [integer constant] <avvall>
    88     02    20 42    1A        03 00 00          0B

13) DATA 1,1.0,'fg',ui

  <data> <intcon> [integer constant] <realcon> [real constant] ...
    86      1A        01 00 00           04     00 00 00 00 00 00 00 01

  <'ed str> [string constant] <un"ed str> [string constant]
     96         02 61 73          06           02 75 69

14) GO TO 200

  <goto> [line number]
    5A      00 02

15) IF X THEN 200

  <ftsvl> [name] <then> [line number]
     01    20 58    1B       00 02
```

```
16) IF X THEN X=1

  <ftsvl> [name] <jfalsr> [rel jump] <ftadr> [name] ...
     01    20 58    1B        0A 00      11    20 58

  <intcon> [integer constant] <stosv>
     1A         01 00 00        08

17) IF X THEN X=1 ELSE 200

  <ftsvl> [name] <jfalsr> [rel jump] <ftadr> [name] ...
     01    20 58    1B        0D 00      11    20 58

  <intcon> [integer constant] <stosv> <jrel> [rel jmp] ...
     1A         01 00 00        08      1C     05 00

  <else jump> [line number]
     1F           00 02

18) IF X THEN X=1 ELSE Y=2

  <ftsvl> [name] <jfalsr> [rel jump] <ftadr> [name] ...
     01    20 58    1B        0D 00      11    20 58

  <intcon> [integer constant] <stosv> <jrel> [rel jmp] ...
     1A         01 00 00        08      1C     0A 00

  <ftadr> [name] <intcon> [integer constant] <stosv>
     11    20 59    1A        02 00 00          08

19) ON X GOTO 200,300,400

  <ftsvl> [name] <on> <goto> [line number]  <goto> [line number] ...
     01    20 58   66    5A      00 02         5A      00 03

  <goto> [line number]
    5A       00 04

20) RENAME TO 'frog'

  <'ed str> [string constant] <noop to> <rename>
     96        04 66 72 6F 67      DB        7E

21) COPY 'toad' TO 'frog'

  <'ed str> [string constant] <'ed str> [string constant] ...
     96        04 74 6F 61 64     96       04 66 72 6F 67

  <noop to> <copy>
     DB       7C

22) EDIT TEXT

  <text> <edit>
    52     7A
```

```
23) EDIT 'xenon',BASIC

  <'ed str> [string constant] <basic> <edit>
     96      05 78 65 6E 6F 6E    53      7A

24) INPUT A,B$

  <INPUT>    <ftadr> [name] <inp#> ...
    5F          11    20 41   DD

             <ftstl> [name] <inp$>           <tailtk>
                03    20 42   E5                19

25) INPUT 'A=';A

  <'ed str> [string constant] <INPUT> [name] <inp#> <tailtk>
     96         02 41 3D         5F    20 41   DD      19

26) DISP A, A$

  <disp>  <ftsvl>  [name]  <print#,> <ftstl> [name] <print$;>  <endprint>
    56       01     20 41     E8        03    20 41     A3         EC

27) LIST 10,70

  <intcon> [integer constant] <intcon> [integer constant] <list>
     1A         10 00 00         1A         70 00 00        4B

28) FOR I=2 TO 173 STEP 9

  <for> <ftadr> [name] <intcon> [integer constant] <stosv> ...
    8C     11    20 49    1A        02 00 00          08

  <intcon> [integer constant] <to> <intcon> [integer constant] ...
     1A        73 01 00         A4     1A         09 00 00

  <step>
    CE

29) NEXT I

  <ftadr> [name] <next>
     11    20 49   8F

30) ON ERROR BEEP @ BEEP

  <on error> <beep> <@> <beep> <inv rtn>
      41        90    40    90      10
```

```
31) ON ERROR GOSUB 100

  <on error> <gosub> [line number] <inv rtn>
      41        5B       00 01         10

32) ON ERROR GOTO 100

  <on error> <inv pop> <goto> [line number] <inv rtn>
      41        9C        5A       00 01        10

33) ON ERROR ON X GOSUB 100,200

  <on error> <ftsvl> [name] <on> <gosub> [line number] <gosub>...
      41        01    20 58   66    5B        00 01        5B

    [line number] <inv rtn>
       00 02         10

34) ON ERROR ON X GOTO 100,200

  <on error> <ftsvl> [name] <inv pop> <on> <goto> [line number] <goto>...
      41        01    20 58     9C      66    5A       00 01        5A

    [line number] <inv rtn>
       00 02         10

35) OFF ERROR

  <off error>
      42

36) ON TIMER#1,1 BEEP @ BEEP

  <intcon> [value] <intcon> [value] <on timer> <timer clear> ...
     1A    01 00 00   1A    01 00 00     64           9D

    <beep> <@> <beep> <inv rtn>
      90    40   90      10

37) ON TIMER#X,X GOSUB 100

  <ftsvl> [name] <ftsvl> [name] <on timer> <timer clear> ...
     01    20 58    01    20 58     64           9D

    <gosub> [line number] <inv rtn>
      5B       00 01         10

38) ON TIMER#1,1 GOTO 100

  <intcon> [value] <intcon> [value] <on timer> <timer clear> ...
     1A    01 00 00   1A    01 00 00     64           9D

    <inv pop> <goto> [line number] <inv rtn>
       9C       5A       00 01        10
```

```
39) ON TIMER#1,1 ON X GOSUB 100,200

  <intcon> [value]  <intcon> [value] <on timer> <tmrclr> <ftsvl> [name] ...
     1A    01 00 00    1A    01 00 00     64        9D       01    20 58

    <on> <gosub> [line number] <gosub> [line number] <inv rtn>
     66    5B       00 01         5B       00 02        10

40) ON TIMER#1,1 ON X GOTO 100,200

  <intcon> [value]  <intcon> [value]  <on timer> <tmr clr> <ftsvl> [name] ...
     1A    01 00 00    1A    01 00 00     64         9D       01    20 58

    <inv pop> <on> <goto> [line number] <goto> [line number] <inv rtn>
       9C      66    5A      00 01        5A      00 02         10

41) OFF TIMER#1

  <intcon> [value]  <off timer>
     1A    01 00 00     65

42) DEF FNA(A,B$[3])

    <def fn>  [fnname]  [rel jump] [type/count] [par1 name] [val pntr] ...
      87        20 41     00 00        04          01 0A       00 00

      [par2 name] [length] [val pntr] [rel PCR]
         02 8A      05 00     00 00     00 00

43) LET FNA = 1  or  FNA = 1

    <let fn> or <inv let fn>  <var type> [fnname] <intcon> [value] ...
      61            44           01       20 41      1A    01 00 00

      <store>
        08

44) DISP FNA(1,'HI')

    <disp> <intcon>  [value]  <' str> [length] [value] <#fncall> ...
      56      1A     01 00 00   96       02     48 49      16

      [fnname] [parcnt] [par1 type] [par2 type] <;> <eol>
       20 41      2         80          81       E7  A2

45) DEF FNA$(A,B$[3])

    <def fn>  [fnname]  [rel jump] [type/count] [par1 name] [val pntr] ...
      87        20 41     00 00        05          01 0A       00 00

      [par2 name] [length] [val pntr] [rel PCR]
         02 8A      05 00     00 00     00 00

46) LET FNA$ = 1  or  FNA$ = 1

    <let fn> or <inv let fn>  <var type> [fnname] <intcon> [value]  ...
      61            44           03       20 41      1A    01 00 00

      <store>
        07
```

```
47) END DEF

    <fn end> [fnname]
      85      20 41

48) DISP FNA$(1,'HI')

    <disp> <intcon>  [value]  <' str> [length] [value] <$fncall> ...
      56      1A     01 00 00   96       02     48 49      17

      [fnname] [parcnt] [par1 type] [par2 type] <;> <eol>
       20 41      2         80          81       A3  A2
```



```
49) DEF FNA(A,B$[3]) = 1

    <def fn>  [fnname]  [rel jump] [type/count] [par1 name] [val pntr] ...
      87        20 41     00 00        04          01 0A       00 00

      [par2 name] [length] [val pntr] [rel PCR] <intcon> [value] ...
         02 8A      05 00     00 00     00 00      1A    01 00 00

      <inv fn end> [fnname]
          AF        20 41

50) DEF FNA$(A,B$[3]) = 'A'

    <def fn>  [fnname]  [rel jump] [type/count] [par1 name] [val pntr] ...
      87        20 41     00 00        05          01 0A       00 00

      [par2 name] [length] [val pntr] [rel PCR] <' str> [length] ...
         02 8A      05 00     00 00     00 00     96       01

      [value] <inv fn end> [fnname]
        41         AF        20 41

51) ASSIGN#1 TO 'A'

    <intcon>  [value]  <'ed str> [length]  [value]  <TO>  <ASSIGN#>
       1A    01 00 00     96        01        61     DB      60

52) ASSIGN#1 TO 'A',TEXT

    <intcon>  [value]  <'ed str> [length]  [value]  <TO> <TEXT> <ASSIGN#>
       1A    01 00 00     96        01        61     DB    52      60

53) ASSIGN#1 TO 'A',BASIC

    <intcon>  [value]  <'ed str> [length]  [value]  <TO> <BASIC> <ASSIGN#>
       1A    01 00 00     96        01        61     DB    53       60

54) ASSIGN#1 TO *
    <intcon>  [value]  <un"ed str> [length] [value]  <TO>  <ASSIGN#>
       1A    01 00 00      06         01       2A     DB      60

55) RESTORE
    <RESTORE>
      70
?
56) RESTORE 300                    [line number]

    <RESTORE>  <RESTORE to line>  <offset address of line>
      70             DC                31      01

57) RESTORE#1

    <intcon>  [value]  <RESTORE#>
       1A    01 00 00     5E
```

```
58) RESTORE#1,10

    <intcon>    [value]  <intcon>    [value]    <RESTORE#>
       1A      01 00 00     1A      10 00 00        5E
?
59) READ A,A$

    <READ>  <ftadr>  [name]  <READ (NUM)>  <ftstls>  [name]  <READ$>
      6E      11     3F 01        E6          13     43 01     EB

60) READ#1,2

    <intcon>    [value]    <intcon>    [value]    <READ#>
       1A      01 00 00       1A      02 00 00      50
?
61) READ#1;A,A$

    <intcon>    [value]  <READ#>  <semicolon>  <ftadr> [name]  <READ NUM>
       1A      01 00 00     50         27          11    3F 01      E6

      <ftstls>  [name]  <READ$>
         13     43 01     EB
?
62) READ#1,2;A,A$,A(),A(,)

    <intcon>  [value]    <intcon>  [value]    <READ#>  <semicolon>
       1A    01 00 00       1A    02 00 00      50          27

      <ftadr>  [name]  <READ NUM>  <ftstls>  [name]  <READ$>
        11     3F 01       E6         13     43 01     EB

      <1 dim array> [name] <Read# Array>  <2 dim array> [name] <Read# array>
          F3        49 01       24             F4       49 01       24


63) PRINT#1,2

    <intcon>  [value]  <intcon>  [value]    <PRINT#>    <Print# EOL>
       1A    01 00 00     1A    02 00 00      5C            EC
?
64) PRINT#1;A,A$,7,'hi',A(),A(,)
    <intcon>  [value]  <PRINT#>  <semicolon>  <ftsvl> [name]    <Print# ,>
       1A    01 00 00     5C          27          01    3F 01        F0

    <ststl>  [name]   <Print# ,>  <intcon> [value]    <Print# ,>
      03     43 01       F0          1A    07 00 00       F0

      <'ed str> [length]  [value]  <Print# ,> <1 dim array>  [name]
         96        02      68 69      F0           F3        49 01


      <Print# array> <Print# ,>  <2 dim array>  [name]  <Print# array>
           22            F0           F4        49 01        22

      <Print# ;>  <Print# EOL>
          EF           EC

 65)  TRANSFORM 'file' INTO BASIC

      <sconst>  <len>  <string>  <basic>  <into>  <eronth> <ron#>
```

```
 96      04     file      98     9B       B4     0002

<transform>
   01
```

TOKEN LIST

```
token 1      name: <ftsvl>      runtime:FTSVL      parsetime: -
asc= -       takes: [name]
examples:
purpose:
```

DATA FORMAT LIST

name: [name]
decompiled format: 2 bytes of ascii being the name

compiled format: again 2 bytes of data but this time it is a pointer
to what is called the "name form" followed by a 2 byte address.

```
           +----------+
           | address  |--,             in program
           +----------+  |
                         |
       ,-----------------'
       |
       |   +----------+---------------+
       `->| nameform |  rel val addr |     in VPR
           +----------+-------|-------+
                              |
       ,----------------------'
       |
       |   +--------------+
       `->|    value     |     in environment
           +--------------+
```

function variable:

```
           +----------+
           | address  |--,             fn name in program
           +----------+  |
                         |
       ,-----------------'                      ,-> type/count in def
       |                                        |        (in program)
       |   +----------+-------------+----------+----------+
       `->| nameform | [length($)] | val pntr | def pntr |     fn name in VPR
           +----------+-------------+----------+----------+
                                         |
                                         `----,
                                              |
                                              v
           +-----+-----+-----+--------+-------+-------------+-------+
           | RTN | PCR | TOS | E.RMEM | CSTAT | [length($)] | value |   fn value
           +-----+-----+-----+--------+-------+-------------+-------+   in env.
```

function parameter:

```
   +-----------+
   | address   |--,         fn param name in program
   +-----------+  |
                  |
 ,----------------'
 |
 |  +----------+-------------+----------+
 `->| nameform | [length($)] | val pntr |   in FN VPR in program
    +----------+-------------+----------+
                                  |
           +-------+              |
           | value |<-------------'     in environment
           +-------+
```

RAM/ROM CONSUMPTION TABLE

In the following table N refers to the number of times a variable or constant is referenced. L refers to the dimensioned length of the string rather that its actual length. This defaults to an incredible 32 bytes.
The amount of room consumed can best be measured with the MEM function rather than using CAT. This is because only that part of the program labeled below under ROM will appear in the file being run. That which is under RAM will appear on the stack (not shown in CAT). If the program is in ROM then the following table shows how much RAM and ROM will be consumed by each variable type. If the program is in RAM then then same amount of memory will be consumed but all in RAM.

		amount of each consumed RAM	ROM	example inits for vars	
REAL	const	0	9N		
	var	8	3N+4	10 X=3.14159	(9+8)
INTEGER	const	0	4N		
	var	3	3N+4	10 X=3	(9+3)
SHORT	const	0	5N		
	var	4	3N+4	10 X=3.14	(9+4)
STRING (len=L)	const	0	(L+2)N		
	var	L+2	3N+6	10 X$='FROG'	(9+L+2)

TOKEN CLASS TABLES

octal token class	decompile routine	allocate routine	deallocate routine	class desciption
-1	INIROM	INIROM	*none*	ROM CLASS > 56
0	EOL	XALL1	XDALL1	END OF LINE
1	FETVAR	VALDC	VDALOC	FETCH VARIABLE
2	BINCON	BININT	BININT	BIN INTEGER
3	STOVAR	SVAL	SVALD	STORE VARIABLE
4	CONST	SKPCON	SKPCON	NUM FLOAT OR STR CONST
5	SCNST	SKPCON	SKPCON	STRING CONST
6	UFNCAL	FUNCAL	DALFNC	USER FUNCTION CALL
7	JMPL#	LINEAL	LINEDA	COND JMP LINE #
10	GOLINE	LINEAL	LINEDA	GOTO GOSUB
11	JMPREL	RELJMP	RELJMP	JMP REL
12	UFNDEF	DEFFN	DALFN	USER FN DEF
13	FNEND	DEFEND	DEFND1	FNEND
14	EXTROM	EROM	ROMCLA	EXT ROM
15	RESWD	OPTION	FRRET	OPTION BASE
16	FNRTN	DEFEND	FRET	USER FN RETURN
17	FNASGN	FNASN	FRRET	FN ASSIGN
20	RESWD	SKPNXT	SKPNX1	DATA
21	RESWD+	DIM	DIMD	DIM/REAL
22	RESWD+	SHORT	SHORTD	SHORT
23	RESWD+	INT	INTD	INTEGER
24	RESWD-	COMM	COMMD	COM
25	EJMP#	LINEAL	RELJMP	ELSE JMP#
26	EJMPR	RELJMP	LINEDA	ELSE JMP REL
27	ULIN#	LINEAL	RELJMP	USING LINE #
30	ON	*none*	*none*	ON
31	PU=	*none*	*none*	STORE
32	SUBSCR	*none*	*none*	SUBSCR
33	DEFKY	*none*	*none*	DEF KEY
34	DIMSUB	*none*	*none*	DIM SUBS
35	PRNEOP	*none*	*none*	PRINT EOL
36	PRINTS	*none*	*none*	PRINT STUFF
37	INPUT	*none*	*none*	INPUT w|w/o a prompt
40	MSCRTN	*none*	*none*	IMMED EXECUTE
41	RESWD	*none*	*none*	OTHER RESERVED WORDS
42	MISC	*none*	*none*	MISC OUTPUT
43	MSTOR	*none*	*none*	MULTI STORE
44	MSCRTN	*none*	*none*	MISC IGNORE
45	PRTFUN	*none*	*none*	PRINT FUNCTIONS
46	SYSFUN	*none*	*none*	numeric pseudo-function
47	MSCRTN	*none*	*none*	DUMMY
50	UNOP	*none*	*none*	NUM UNARY OP
51	BINOP	*none*	*none*	NUM BINARY OP
52	UNOP$	*none*	*none*	STR UNARY OP
53	BINOP$	*none*	*none*	STR BINARY OP
54	RWEX	*none*	*none*	DUMMY
55	SYSFUN	*none*	*none*	NUM FUNC
56	SYSFN$	*none*	*none*	STR FUNC

The Kangaroo Input Software

Jack Applin IV

11:11 July 7, 1982

*** Introduction ***

In this paper, we will discuss Kangaroo input. This will be divided into line input and character input. Line input will concern getting input a line at a time. Character input will concern single character input and keyboard routines.

Most users will simply use line input. Few users have to do character input.

Examples of line input are:

EDIT mode reading a line.
Appointment mode reading a line.
Time mode reading a line.
A BASIC INPUT statement reading a line.

Examples of character input are:

The KEY$ function.
The PUT statement.
CAT ALL reading up and down arrows.
Checking for the ATTN key.
Shift-APPT in appointment mode.
Shift-fetch.

*** The keyboard interrupt service routine ***

Relevant globals:

KEYHIT: The current key

SVCWRD: Bit zero is a validity flag for KEYHIT

Whenever a key is hit, the keyboard interrupts. The interrupt service routine translates the keycode to ASCII. Then bit zero of SVCWRD is checked. If the bit isn't set, then KEYHIT is undefined and is ready for a new key. If the bit is set, then KEYHIT already contains a pending key. In this case, the new key can only go into KEYHIT if it's the ATTN key, which has priority over all other keys. At any rate, bit zero in SVCWRD is set to indicate that we now have a pending key.

KEYHIT may be regarded as a one-character stack of pending keys. The system could have been designed so that KEYHIT was a larger stack (say it could hold 20 keys pending) but it wasn't.

Most of the character routines access KEYHIT and SVCWRD, and don't speak to the keyboard I/O address at all. Thus, the hardware interrupting I/O address has been transformed into a virtual keyboard, consisting of KEYHIT and SVCWRD.

*** Character routines ***

```
ATTN?   See if the ATTN key is pending
DEQUE   Eliminate any pending key
GETCHR  Wait for and eat/return a character
KEY?    See if there's a key pending
KEYSRV  Keyboard interrupt service routine
LETGO   Wait for the user to let go of the keyboard
PUTKEY  Make a key pending
SIGNIF  Wait for a 'significant' key, i.e., not left/right arrow
WAITKY  Wait for a key or timeout and return NAPKEY
```

ATTN?: Calls KEY? to see if a key is pending. If a key is pending, checks if it's the ATTN key.

DEQUE: Clears bit zero in SVCWRD. This "eats" the current key if any exists.

GETCHR: Call WAITKY to wait for a key or timeout and then calls DEQUE to eat the key.

KEY?: Checks bit zero in SVCWRD to see if a key is pending. Doesn't eat the key.

KEYSRV: Keyboard interrupt service routine. Called only by the hardware at interrupt time. Translates the key to ASCII and renders it pending.

LETGO: Waits for the user to let go of a key. Then calls DEQUE to eliminate the key.

PUTKEY: Renders the given key pending by putting it into KEYHIT and setting bit zero in SVCWRD.

SIGNIF: An extension of WAITKY that handles certain keys internally. Calls WAITKY to get a key. If that key is one of a set (left/right arrow, shift-fetch, etc.) it handles the key and goes to get another.

WAITKY: Wait for the user to hit a key or timeout. If KEY? returns true within the countdown period, return that key. If we count down, return the NAPKEY unless STANDBY ON. If STANDBY ON, don't ever time out, just wait for a key hit.

*** Overview of line input ***

All line input is done through GETLN. GETLN has many shell routines, such as GETLNX, GETREP, GETTER, etc. Curiously enough, GETLN performs both input and output. It both sets up the initial template (say in appointment mode) and reads the input line.

GETLN reads characters and feeds them to CHEDIT (the character editor). CHEDIT performs editting functions if the keys are special (like BS) or just echos them and puts them in INPBUF (the input buffer). CHEDIT returns a flag if the key is a "terminator" (CR,APP1,ATTN, etc.) that tells GETLN to exit.

Some versions of GETLN (GETTEA, GETTER, etc.) also set SHIELD. SHIELD is a 96-bit protection template that corresponds to INPBUF. CHEDIT considers a position in INPBUF protected if the corresponding bit in SHIELD is set.

Here's a diagram of GETLN:

```
+---------------------------+
| Set up initial template   |
+---------------------------+
             |
             |<----------<-------
             |                   |
+---------------------------+    |
|      Turn on cursor       |    |
+---------------------------+    |
             |                   |
+---------------------------+    |
|      Get a character      |    |
+---------------------------+    ^
             |                   |
+---------------------------+    |
|      Turn off cursor      |    |
+---------------------------+    |
             |                   ^
+---------------------------+    |
|        Call CHEDIT        |    |
+---------------------------+    |
             |                   |
             *                   |
           *   *                 |
         *       *               |
       *           *   yes       |
     *  Terminator?  *---->------
       *           *
         *       *
           *   *
             *  no
             |
+-----------------------------+
| Disable error annunciator   |
+-----------------------------+
             |
         +------+
         | EXIT |
         +------+
```

*** Line input routines ***

GETLN	Read a line
GET.IN	Call GETLN with default parameters
GETREP	Call GETLN, report errors
GETTER	Set up protection template, call GETLN
GETTEM	Call GETTER, return status describing terminator
SETLIN	Set up an initial line

BLIMP	Blank out the input buffer
CHEDIT	Character editor
I/ROFF	Disable Insert/Replace mode
LA	Go left one character
LIT?	Is the literalize flag set?
NULOLD	Reset the old line length
RA	Go right one character
SHELD?	Do we have a protection shield?
SHFET	Display the error buffer

GETLN: Read a line. Calls SETLIN to set up the initial template, then calls GETCHR to get keys and CHEDIT to process them. When CHEDIT signals that it got a terminator, GETLN exits.

GET.IN: Call GETLN with default parameters. These paramaters specify no initial input and normal cursor placement.

GETREP: Call REPORT to report any errors, call GETLN, call REPORT again, and clear out SHIELD.

GETTER: Sets up SHIELD according to the protection template pointed to by R2, and calls GETREP.

GETTEM: Sets INPCHK to the address of the input checking routine supplied in R44. Calls GETTER to set up SHIELD and get input (reporting errors). Clears the input check. Sets the status flags according to the input terminator, i.e., ZR is set if the terminator is the CLR key.

SETLIN: Sets up the initial line for GETLN. This line would contain, say, the APPT template or the auto line number.

BLIMP: Blanks out the input buffer.

CHEDIT: Character editor. This handles input keys and processes them. Normal keys, such as 'A', are echoed and placed into INPBUF. Terminator keys, such as CR or FETCH, are flagged and returned. Editting keys, such as BACK, are

processed each according to its function.

I/ROFF: Turns insert mode off by clearing I/RFLG and sending the appropriate escape sequences to the display devices.

LA: Go left one character. This is the special character routine for the left arrow.

LIT?: See if the literalize flag JUSTSO is set. JUSTSO gets set by shift-I/R. LIT? clears JUSTSO and returns its previous state.

NULOLD: Clear out the previous input length OLDLST. OLDLST is looked at by CNTL-FETCH to determine how long the previous input line was. BLIMP calls NULOLD.

RA: Go right one character. This is the special character routine for the right arrow.

SHELD?: See if SHIELD is set at all, that is, is any character position protected.

SHFET: The code for the shift-FETCH key. This looks at OLDLST to determine what the old input length is, and calls SETLIN to set up the old input.

*** Other documents ***

AS"KEY by Seth D. Alford & Jack Applin IV

This explains the keyboard hardware and software translation.

INTERPRETER

Gary K. Cutler

2:21 PM THU., 15 JULY, 1982

```
+-------------------------------------------+------------------+
|                                           |                  |
|  INTERPRETER                              |  CHAPTER  1      |
|                                           |                  |
+-------------------------------------------+------------------+
```

1.1 INTRODUCTION

The interpreter loop processes the next token (From the token stream pointed to by R10),and passes control to the runtime code for the particular token. When the runtime code has been executed, control returns to the interpreter loop, which either processes the next token, or, if flagged, returns control to the executive loop.

A token is a numeric value representing a position or rank in a table of addresses. The procedure enacted by the interpreter to initiate the runtime code for each token is as follows:

The token value is doubled, creating an offset into the table.
This offset is added to the base address of the table which yields the appropriate runtime address.
An indexed subroutine jump to this address begins the execution.

```
                               RUNTIME
                            ADDRESS  TABLE        RUNTIME ROUTINES
                             __________
                            |          |         +--------------+
                            |   ADDR   |         |              |
                            |__________|         |              |
+-------------------------+ |          |         |______________|
| (Token*2)+Base Address  |-------->|   ADDR   |-------->|  Start of    |
+-------------------------+ |__________|         |  Runtine     |
                            |          |         |  Routine     |
                            |   ADDR   |         +--------------+
                            |__________|
```

This procedure is for system tokens. To process external tokens (i.e. from plug-in ROMs) the interpreter first processes the system token, 84h (ext ROM token). The token, 84h, is first doubled, then

added to the base address of the System Table and the runtime address for 84h is located. This runtime routine takes the next two bytes in the token stream as an external ROM number and enables that ROM in the switching position. The following byte in the stream is interpreted as the actual token. The token, in turn, is doubled and added to the base of the External ROM Table thus obtaining the location of the runtime routine.

The high order bit, bit 7 in R17 is set to flag stall status. This implies an end of token stream, an error has been set or the attention key has been hit. After each runtime routine has been executed, R17 is tested. If R17 is positive (implying the upper bit is not set) then the loop continues. If R17 is negative then any pending errors are reported and control is passed back to the executive loop.

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

1.2 FLOW DIAGRAMS

```
+-------------+
| INTERPRETER |
+------|------+
       |
+------V------+
|  set basic  |
|  PCR  (R10) |
+------|------+
       |
+------V------+
| set R12=105 |
+------|------+
       |
+------V------+
|release temp.|
|   memory    |
+------|------+
       |<----------------------<
+------V------+                |
|  get next   |                |
|system token |                |
+------|------+                |
       |                       |
+------V------+                |
|  double it  |                |
|  for offset |                |
+------|------+                |
       |                       |
+------V------+                |
| add to base |                |
|  address of |                |
|runtime table|                |
+------|------+                |
       |                       |
+------V------+                |
| get runtime |                |
|address from |                |
|   table     |                |
+------|------+                |
       |                       |
+------V------>      ?---------|---------?     +------+
|  JSB  X20   |----->| bit 7 of R17 set |----->| stop |
+------------->      ?-------------------?     +------+
```


Table of Contents

Keyboard Translation
for Kangaroo

Seth D. Alford
Jack Applin IV
May 11, 1982

CHAPTER 1

Introduction

The HP-75 (Kangaroo) communicates with the keyboard via the Roo chip. The Roo chip reports which key the user presses on the keyboard in terms of an internal keycode. The TRanslate KEY (TRKEY) routine translates this internal keycode into ASCII for use by the rest of the Kangaroo operating system.

CHAPTER 2

Keycodes from the Roo Chip

The Roo chip produces 8 bit key codes according to the row and column of the keystroke. See the keyboard scan layout, attached, for which keys are on which rows and columns. B7-5 give the column and B4-1 give the row. B0 is always high (always 1.) The encoding is shown below.

B7	B6	B5	Column	B4	B3	B2	B1	Row
1	1	1	C0	1	1	1	1	R0
1	1	0	C1	0	1	1	1	R1
1	0	1	C2	1	1	0	0	R2
1	0	0	C3	0	1	0	0	R3
0	1	1	C4	1	0	0	0	R4
0	1	0	C5	0	0	0	0	R5
0	0	1	C6	1	1	1	0	R6
0	0	0	C7	0	1	1	0	R7
				1	0	1	0	R8
				0	0	1	0	R9

Except for control and shift, each key corresponds to a unique keycode, but all possible keycodes are not used. Whether control and/or shift are depressed is reported in the keyboard status byte. As an example, the keycode for "A" as reported by the Roo chip is 10011001 binary.

CHAPTER 3

How TRKEY Works

3.1 The RAWNUM Table

TRKEY uses the keycode reported by the Roo chip. Since B0 is always 1, the register containing the keycode is right shifted one bit. This value is used as an index into the RAWNUM table; based on the index a value from the table is obtained. The RAWNUM table yields 3 classes of keys: non-ASCII, alphabetic, and regular ASCII. Non-ASCII keys have values greater than or equal to 128, and include [ATTN], [LOCK], [TIME], and so forth. Regular ASCII contain the rest of the ASCII keys, such as *,&,-, and so forth. The RAWNUM table also contains dummy no-op entries which are denoted by FF. These entries are not used and merely exist as padding.

3.2 What is Done with the Value

Non-ASCII keys are handled at the label NONASC. Here the control and shift bits of the keycode are set.

Alphabetic keys are handled at the label ISALFA. The lowercase letter returned from RAWNUM has its shift or control bits set depending on the value in CAPLOK and whether the shift or control keys were depressed. The ISALFA routine falls into numeric pad processing.

Numeric pad processing proceeds if the value in CAPLOK is equal to KY.PAD. The translation table PADTBL contains values for lowercase letters followed by their numeric value. If the value cannot be found in the PADTBL then the BEEPKY is returned.

Regular ASCII keys are processed at the label REGASC using values from the KEYTAB table. KEYTAB contains triples. Each triple consists of the normal form of a key, the shifted form of the key, and the control form of the key. Shift and control are made equivalent to control. The index for the KEYTAB is obtained by multiplying the value from RAWNUM by 3. This value is then incremented by 1 or 2 depending on whether shift or control respectively were pressed.

CHAPTER 4

Flowchart

The following flowchart graphically shows how TRKEY works.

```
       * * * * *
      * start   *
       * * * * *
           |
           v
   * * * * * * * * *
   * Generate an   *
   * index into    *
   * RAWNUM.  Get  *
   * the number at *
   * the index.    *
   * * * * * * * * *
           |
           v
           *
         *Is *
       *  it   *   Y
     * non ASCII *-----------------------------|
       *   ?   *                               |
         *   *                                 |
           *                                   |
           | N                                 |
```



```
       |                                           |
       v                                           v
       *                                   * * * * * * * * *
     *Is *                                 * Set the       *
   *  it   *  Y                            * control and   *
 *within al- *---------------|             * shift bits.   *
   *phabet-*                 |             * * * * * * * * *
     *ics*                   |                     |
       *                     |                     v
       | N                   |                 * * * * *
       v                     v               *   end    *
 * * * * * * * * *   * * * * * * * * *         * * * * *
 * Generate an   *   * Look at CAPLOK*
 * index into    *   * and shift to  *
 * KEYTAB and    *   * capitalize or *
 * get the key.  *   * set the       *
 *               *   * control bit.  *
 * * * * * * * * *   * * * * * * * * *
       |                     |
       |                     v
       |                     *
       v                   *   *
   * * * * *             *process*  Y
 *   end    *          *  numeric  *-------------|
   * * * * *             *keypad *               v
                           * ? *         * * * * * * * * *
                             *           * Look in PADTBL*
                             | N         * for letter to *
                             |           * number trans- *
                             |           * lation.  If   *
                             |           * not there then*
                             |           * return BEEPKY.*
                             |           * * * * * * * * *
                             |                   |
                             |                   v
                             |               * * * * *
                             -------------->*   end    *
                                             * * * * *
```


CHAPTER 5

Globals Used

Global	Type	Usage
APPTKY	EQU	Value for [APPT] key
ATTNKY	EQU	Value for [ATTN] key
BEEPKY	EQU	Value for beep key
BS	EQU	Value for backspace
CAPLOK	DAD	Keyboard status: shift lock or numeric pad
CLRKEY	EQU	Value for [CLR] key
CR	EQU	Value for carriage return
DELKEY	EQU	Value for [DEL] key
DOWNKY	EQU	Value for [V] key
EDITKY	EQU	Value for [EDIT] key
ESC	EQU	Value for escape key
FTCHKY	EQU	Value for [FET] key
I/RKEY	EQU	Value for [I/R] key
KEYSTS	DAD	I/O address for keyboard
KY.PAD	EQU	Value in CAPLOK when in numeric pad mode
KYIDLE	DAD	Keyboard intercept location
LEFTKY	EQU	Value for [<--] key
LOCKKY	EQU	Value for [LOCK] key
RGHTKY	EQU	Value for [-->] key
RUNKEY	EQU	Value for [RUN] key
TABKEY	EQU	Value for tab key
TIMEKY	EQU	Value for [TIME] key
UPKEY	EQU	Value for [^] key

CHAPTER 6

Conclusions

We would reccomend that the keyboard translation be done by the hardware in future products. Keyboard translation occupies approximately 300 bytes of mainframe code. This space could be used elsewhere. If the mainframe software required additional freedom to reassign keys it could perform the additional translation.

CHAPTER 7

References

1. Roo Chip ERS
2. KR/GLO (Kangaroo global file)
3. KR/KEY (location of TRKEY)

Table of Contents

The Kangaroo LCD Driver Software

Jack Applin IV
2:24:39 July 19, 1982

CHAPTER 1

Introduction

This document will describe the LCD driver software for the HP-75 (Kangaroo). It will describe its character set, how it scrolls, how it responds to escape sequences, and what special characters it responds to. It will describe how the annunciators work.

It will also describe the behavior of the physical LCD, as much as is necessary.

CHAPTER 2

Description of the physical LCD

2.1 General layout of the physical LCD

```
position 80                    position 49
 |                              |
 V                              V
+--------------------------------+
|ABCDEFGHIJKLMNOPQRSTUVWXYZ789012|
+--------------------------------+

   BATT    ERROR    PRGM    APPT
```

The LCD consists of 81 addressable positions.

Addresses 80-49 are the LCD window. This contains characters.

Address 48 is the left annunciator pair. It contains the BATT and ERROR annunciator pair.

Address 47 is the right annunciator pair. It contains the PRGM and APPT annunciator pair.

Address 1 is a dummy position. If the cursor is written to this (or any other position 1<X<46) then the cursor is effectively off, it does not appear.

Address 0, if written to, copies its data to all positions. Writing a blank to location 0 is a good way to quickly clear the LCD. Of course, this data is also copied to the annunciator positions. A blank happens to clear the annunciators. Other characters might set or clear the annunciators, based on the shape of those characters. See 'The Annunciators' below.

2.2 Reading status

When reading from the LCD I/O address (FFFC), only the low-order bit of the status is used. It is 1 when the LCD is busy writing the previous data, 0 when the LCD is ready to accept data. The routine LCDRDY waits for the LCD to be ready to accept data by waiting for that bit to clear.

2.3 Sending position only

When a single byte is written to the LCD I/O address (FFFC), the value of the byte indicates where the cursor is to go. Writing 80 decimal, for instance, will send the cursor to the leftmost position of the display. Writing 1 will send the cursor to position 1, which will not be displayed and hence the cursor is effectively off. Actually, any position from 1-46 will do for this purpose, but position 1 has the least chance of being used if the LCD is widened to more than 32 characters.

WARNING

> Never write to any position greater than 80 (the leftmost position) as this will cause the LCD scanning hardware to latch up.

The high bit of the address is used to indicate the cursor type. If the high bit is off, the insert (arrow) cursor is used. If the high bit is on, the replace (box) cursor is used.

2.4 Sending position and data

When two bytes are written to the LCD I/O address (FFFC), this indicates both position and data. The first byte indicates position and cursor type, as described in the previous section. The second byte indicates the data to place at that position. If the high bit of the data is set, the character will be underlined.

2.5 The annunciators

There are two annunciator pairs, the BATT/ERROR pair and the PRGM/APPT pair. The BATT/ERROR is in position 48, and the PRGM/APPT pair is in position 47. The fact that the annunciators are paired makes them difficult to manipulate. The left annunciator of the pair is keyed to the lower-left segment. The right annunciator of the pair is keyed to the lower-right segment. Hence, the following procedure must be used:

```
To activate only the left annunciator:   write a 'p'
To activate only the right annunciator:  write a 'q'
To activate neither annunciator:         write a ' '
To activate both annunciators:           write a '_'
```

Other characters can be used, of course. For instance, a dash could be used instead of a blank since it activates neither of the lower corners. However, the characters used seemed to have the best chance of surviving changes in the character set. For example, a 'g' was formerly used instead of a 'q', but then the descender on the 'g' was changed so it didn't activate the lower-right segment.

2.6 The character set

For the most part, the character set corresponds to the ASCII character set. The characters 0-31, the control characters, display a bunch of strange characters. Character 126 displays the insert cursor. Character 127 displays the replace cursor. Any character from 128-255 displays the corresponding character from 0-127 except underlined.

To get the correct ASCII set for characters 126 and 127, the software has to do translation of characters on output. See the section on "Character translation" that follows.

CHAPTER 3

Description of the virtual LCD

3.1 General description

The virtual LCD is a thing of both software and hardware. When I speak of sending something to the virtual LCD, I mean it is sent to the LCD driver software which manipulates it and then sends stuff to the LCD hardware.

THE VIRTUAL LCD

```
<--------------- 96 character wide buffer ---------------->

        +--------------------------------+
+-------|--------------------------------|-----------------+
|       |                                |                 |
+-------|--------------------------------|-----------------+
        +--------------------------------+

         <-- 32 character wide window -->
```

The virtual LCD consists of a 32 character window into a 96 character wide buffer. The window is always within the buffer. The cursor is always within the window. (Except, of course, when the cursor's off.) (Well, not really. See 'The Earthquake problem' below.)

The window moves whenever the cursor "bumps" against it. For instance, if the cursor is at the 32nd position of the window, and a normal character is sent, the window will be forced to move forward one position to keep up with the cursor. Of course, if the cursor (and hence the window) was already at the far right end of the buffer, then the character will be rejected. Similarly, BS characters move the window back if they bump against it.

3.2 Special characters

The following are the only special characters the virtual LCD recognizes. All other characters (even if control characters) are simply displayed, perhaps as Greek or other strange symbols.

The special characters are:

1. BS: backspace
2. CR: carriage return
3. LF: line feed
4. ESC: escape

1. BS causes the cursor to go left one space. If this would cause us to go too far left, the BS is ignored. If this would move the cursor outside the window, then the window is moved.
2. CR causes the cursor to go to the far left position. This will move the window to the far left.
3. LF causes the current line to be "dead" by setting the DEAD flag. This causes the line to be cleared when the next character is sent to the LCD. Also, LF waits the current DELAY rate.
4. ESC marks the start of an escape sequence. It sets the flag LCDESC to indicate this so the code will treat the next character specially.

NOTE

There is no "newline" character in Kangaroo. The CR/LF sequence serves that purpose.

3.3 Escape sequences

Except for ESC %, all recognized escape sequences are two characters. The second character must be upper case. Unrecognized escape sequences (ESC A, say) are ignored.

1. ESC C: go forward
2. ESC D: backward
3. ESC E: home, clear
4. ESC G: go to left of display
5. ESC H: home cursor
6. ESC J: clear to end of screen
7. ESC K: clear to end of line
8. ESC O: delete with wraparound
9. ESC P: delete without wraparound
10. ESC <: cursor off
11. ESC >: cursor on
12. ESC %: cursor addressing

1. ESC C: go forward. If this would cause the cursor to move outside the buffer, then it is ignored. If this would cause the cursor to move outside the current window, then the window is moved forward.
2. ESC D: go backward. Just like BS.
3. ESC E: home and clear. This resets the LCD. It resets the cursor and window positions to the far left.
4. ESC G: go to far left. Just like CR.
5. ESC H: home cursor. Just like CR.

6. ESC J: clear to end of screen. Clears the display

and buffer from the current position to the end.

7. ESC K: clear to end of line. Just like ESC J

8. ESC O: delete with wraparound. Delete the current character, moving in characters from the right over it.

9. ESC P: delete without wraparound. Just like ESC O.

10. ESC <: turn the cursor off.

11. ESC >: turn the cursor on.

12. ESC % <column> <row>. Go to the given <column>. The <row> is ignored, and is included only for compatability with other devices. The <column> and <row> are one-byte binary quantities with only the first five bits (possible value 0-31) regarded. The window is placed at the beginning of the buffer, even if this is not absolutely required by the column given.

CHAPTER 4

Implementation

4.1 Trip through the LCD software

In this section, we will describe the path of a typical character as it travels through the LCD driver software.

4.1.1 Check CREST for ESC % <column> <row>

CREST is the flag that says if we're currently in an ESC % <column> <row> sequence, the only escape sequence that's over two characters.

If CREST='C', then the <column> is expected next. The incoming character is taken, reduced mod 32, and placed as the current cursor position. The window is forced to the leftmost position.

If CREST='R', then the <row> is expected next. The incoming character is ignored, and CREST is set to zero to indicate the end of this sequence.

4.1.2 Check for a DEAD line

If DEAD is set, then the current line is dead. If so, we clear DEAD and clear the LCD. DEAD was set when the previous LF came by.

4.1.3 Are we part of an escape sequence?

If LCDESC is set, then the previous character was an escape. If so, we take the incoming character and look it up in the table of escape sequences. If it's found, we transfer to the appropriate routine and exit. If it's not found, the character is ignored.

4.1.4 Are we a special character?

The incoming character is searched for in the table of special characters. If it's found, we transfer to the appropriate routine and exit.

4.1.5 Assure that we're within the window

INWIND is called to assure that the cursor (pointed to by LCDPTR) is within the window (pointed to by LCDWIN). See the section on 'The Earthquake problem' that follows.

4.1.6 Apply character translation

The incoming character is searched for in the translation table TRANS1. If it is found, we translate it to the parallel character in TRANS2.

4.1.7 Set LETSEE

Since we know that the incoming character is a normal character, we call DOSEE to set the LETSEE flag. This flag is interrogated by the HANG routine so that BASIC can pause to allow the user to view previous output. HANG clears the LETSEE flag by calling UNSEE.

4.1.8 Write character in insert or replace mode

Based on I/RFLG, we either call CH2LCD to display the character in replace mode, or INSCHR to display the character in insert mode.

4.1.8.1 Write character in insert mode

All the characters to the right of the current position are moved over one to the right. The incoming charcter is inserted at the current position, and the LCDPTR is moved up by one. Note that if this would cause LCDPTR to be outside the current window, the window is NOT moved now. The current window is then written out to the LCD with PUTWIN. See

the section on 'The Earthquake problem' that follows.

4.1.8.2 Write character in replace mode

The incoming character is placed into the buffer, LCDBUF, and LCDPTR is updated. The incoming character is written to the LCD. Note that if this would cause LCDPTR to be outside the current window, the window is NOT moved now. See the section on 'The Earthquake problem' that follows.

4.1.9 Display the cursor

CURSE? is called to display the cursor. If CURSOR is not set, then no cursor is displayed. If CURSOR is set, then the cursor is displayed according to I/RFLG, which says whether we're in insert or replace mode.

4.2 Flowchart

```
               +-------+
               | START |
               +-------+
                   |
                   |
                   *
                 *   *
               *       *
             *           *   yes
           *  CREST='C'?  *---->---+
             *           *         |
               *       *           |
                 *   *             |
                   *               |
                   | no    +----------------+
                   |       | Get <column>,  |
                   |       |  set cursor    |
                   |       |  and window    |
                   |       +----------------+
                   |               |
                   |               |
                   |             +---+
                   |             | E |
                   *             +---+
                 *   *
               *       *
             *           *   yes
           *  CREST='R'?  *---->---+
             *           *         |
               *       *           |
                 *   *             |
                   *               |
                   | no    +---------------------+
                   |       | Ignore incoming <row> |
                   |       +---------------------+
                   |               |
                   |               |
                   |             +---+
                   |             | E |
                   *             +---+
                 *   *
               *       *
             *   DEAD    *   yes
           *     line?    *---->---+
             *           *         |
               *       *           |
                 *   *             |
                   *               |
```

```
        | no     +----------------+
        |        | Clear the line |
```



```
        |        +----------------+
        |                |
        |                |
        |------<---------+
        |
        *
      *   *
    *       *
  *  LCDESC   *   yes
 *    set?     *---->---+
  *           *         |
    *       *           |
      *   *             |
        *               |
        | no     +----------------+
        |        | Look up routine |
        |        +----------------+
        |                |
        |                |
        |        +----------------+
        |        |  Call routine  |
        |        +----------------+
        |                |
        |                |
        |              +---+
        |              | E |
        *              +---+
      *   *
    *       *
  *  special  *   yes
 *  character? *---->---+
  *           *         |
    *       *           |
      *   *             |
        *               |
        | no     +----------------+
        |        | Look up routine |
        |        +----------------+
        |                |
        |                |
        |        +----------------+
        |        |  Call routine  |
        |        +----------------+
        |                |
        |                |
        |              +---+
        |              | E |
        |              +---+
        |
+---------------------------+
| Get us within the window  |
+---------------------------+
```

```
                   |
```



```
          +---------------------------+
          |Apply character translation|
          +---------------------------+
                       |
                       |
          +---------------------------+
          |     Set LETSEE flag       |
          +---------------------------+
                       |
                       |
                       *
                     *   *
                   *       *
       Replace   *   What    *   insert
      +----<----*    mode?    *---->----+
      |          *           *          |
      |            *       *            |
      |              *   *              |
      |                *                |
+------------------+            +------------------+
| Write character  |            | Write character  |
| in replace mode  |            | in insert mode   |
+------------------+            +------------------+
      |                                 |
      |                                 |
      +-------->--------------<---------+
                       |
          +---+        |
          | E |------> |
          +---+        |
                       |
          +---------------------------+
          |    Display the cursor     |
          +---------------------------+
                       |
                   +------+
                   | EXIT |
                   +------+
```


4.3 The Earthquake problem

The earthquake problem used to occur when a 32 character line was written to the LCD. When the 32nd character was written, the window was moved to the right and re-transmitted. Then CR/LF was sent. The CR caused the window to be at the far left and re-transmitted. Visually, the display had quivered. Hence, the earthquake.

The solution to this problem was to not move the window until the next "normal" character comes along. Hence, when the 32nd character was written, the window was NOT re-transmitted because the window wasn't moved. Then, when the CR/LF was sent, the window was set to the far left. Since the window never moved, no earthquake problem.

4.4 Quick list of globals

Here is a list of the LCD globals:

```
ANNFLG : Annunciator status
CREST  : Status of ESC % Column Row
CURSOR : Whether cursor is on/off
DEAD   : Whether current line is to be cleared
DELAY  : Current DELAY count
I/RFLG : Insert/Replace mode
INPTR  : Pointer into input buffer
LASTCH : Pointer to last character in input buffer
LCDBUF : LCD buffer
LCDESC : Flag if ESC has been seen
LCDOFF : LCD output disabled
LCDPTR : Current cursor position in LCDBUF
LCDWIN : Current window position in LCDBUF
LETSEE : We have displayed a normal character
MAXMAX : Address of far right LCD position
MINMIN : Address of far left LCD position
SIZSIZ : Width of LCD
```


4.5 Detailed list of globals

--- ANNFLG ---

Status of annunciators.

```
ANNFLG bit 3 : battery annunciator
ANNFLG bit 2 : error annunciator
ANNFLG bit 1 : program annunciator
ANNFLG bit 0 : error annunciator
```

--- CREST ---

Status of ESC % Column Row escape sequence.
If CREST=0, then we are not in the sequence.
If CREST='C', then the Column is expected next.
If CREST='R', then the Row is expected next.
When we've just seen the ESC, we don't know what kind of escape sequence we're in, so just LCDESC is set.

--- CURSOR ---

Whether cursor is on/off. If CURSOR is nonzero, then the cursor should be displayed.

--- DEAD ---

Whether the current line is to be cleared at the next character. If DEAD is nonzero, then the LCD shall be cleared when the next character comes through. DEAD is set when a LF comes in. If the LCD was cleared right when the LF came in, the line would appear and disappear quickly. This forces the line to hang around.

--- DELAY ---

The current DELAY length. This is a five-byte quantity of timer "ticks" (2^-14 seconds). Please refer to the comparator document for more information on how such timers work. We wait this long after each LF character.

--- I/RFLG ---

Input Insert/Replace mode. If I/RFLG is nonzero, then we're in insert mode. This is used for deciding what kind of cursor to emit.

--- INPTR ---

Input pointer. INPTR points to the current location in the input buffer. It is used (here he hangs his head in shame) to determine how long the buffer is for insertions and deletions.

--- LASTCH ---

Pointer to last character in the input buffer. It is used (gasp choke wheeze) to determine how long the buffer is for insertions and deletions.

--- LCDBUF ---

The 96-character LCD buffer. It contains the actual data that is displayed in the hardware.

--- LCDESC ---

Flag if escape character has been seen. If LCDESC is nonzero, then the previous character was an escape and an escape sequence is now in progress.

--- LCDOFF ---

Flag LCD output enabled/disabled. If LCDOFF is non-zero, then LCD output is disabled. Certain character editting operations temporarily set LCDOFF so that the operation will be faster.

--- LCDPTR ---

Current cursor position in LCDBUF. LCDPTR points to the current cursor position in LCDBUF.

--- LCDWIN ---

Current window position in LCDBUF. LCDWIN points to the current window position in LCDBUF.

--- LETSEE ---

Flag that is set when a normal character (not special character) is sent to the LCD. The routine HANG looks at LETSEE and waits if it's set. The routine DOSEE sets LETSEE.

--- MAXMAX ---

Address of far right LCD position. Normally contains 49.

--- MINMIN ---

Address of far left LCD position. Normally contains 80.

--- SIZSIZ ---

Width of the LCD. Normally contains 32.

4.6 Annunciator implementation

To turn on or off any annunciator, call one of the following routines:

ANN.A-	turn appointment annunciator off
ANN.A+	turn appointment annunciator on
ANN.B-	turn battery annunciator off
ANN.B+	turn battery annunciator on
ANN.E-	turn error annunciator off
ANN.E+	turn error annunciator on
ANN.P-	turn program annunciator off
ANN.P+	turn program annunciator on

The annunciator information is kept in ANNFLG. ANNFLG contains four bits, one for each annunciator. The format of ANNFLG is:

ANNFLG bit 3 : battery annunciator
ANNFLG bit 2 : error annunciator
ANNFLG bit 1 : program annunciator
ANNFLG bit 0 : error annunciator

When any of the annunciator manipulation routines is called, all four annunciators are re-written according to ANNFLG. Given more space, the routine could have only re-written the appropriate pair of annunciators, but writing both pairs saved code and seemed adequately quick.

4.7 Character translation

The LCD hardware (fixed forevermore) has the following configuration:

08 tilde character	88 underlined tilde
0A lazy-T character	8A underlined lazy-T
7E insert cursor	FE underlined insert cursor
7F replace cursor	FF underlined replace cursor

The locations 7E and 7F are fixed because the display generator displays 7E or 7F when it wants an insert or replace cursor. However, ASCII dictates thet 7E is the tilde, and we want 7F to be the lazy T. How do we solve this dilemma?

We have the SOFTWARE device driver translate 7E->08 and 7F->0A. Nothing knows about this but the lowest level of software (and us). If the user sends a 08 or 0A, these are BS and LF and get processed specially anyway.

A parallel mapping goes on with FE->88 and FF->8A, for these pairs are the previous characters with their high bits set.

CHAPTER 5

Doing strange things

5.1 Modifying window parameters

By modifying MINMIN, MAXMAX, and SIZSIZ, the implementor can alter what is written to the LCD. He can, for instance, say: "Protect the first 10 and the last 10 characters of the display" by setting MINMIN, MAXMAX, and SIZSIZ. Then characters will only appear in positions 11-22 and will scroll in those locations only. However, if any LF's or ESC-E's are sent, CLRLCD is called, which ignores the MINMIN, MAXMAX, and SIZSIZ parameters and clears the ENTIRE LCD.

5.2 Setting LCDOFF

By setting LCDOFF, the implementor can disable LCD output. This makes many operations quicker if the implementor knows that PUTWIN will be called (possibly by him) after LCDOFF is cleared again.

5.3 Putting LCDPTR and LCDWIN outside LCDBUF

I have no idea what this would do, but it sounds interesting.

CHAPTER 6

Other documents

IC Display Controller Detailed Description (A-1LF1-0301-3)
by Tim Myers

This describes the LCD hardware.

IV"COD by Jack Applin IV.

This describes the character set and character translation.

LCD Driver Soft

Table of Contents

LOCK description

Raan Young
07/09/82

NOMAS

NOt MAnufacturer Supported
...cipient agrees NOT to contact manufacturer

LOCK lives in the file KR-LOK. LOCK sets a password which the user will be asked to supply each time the machine is awakend by the ATTN key (see KR"HI for more about wakeup activites). If the user does not supply the correct password, the machine will go back to sleep immediately. If the password is null, then the LOCK option is effectively turned off. Only the first 8 characters of the LOCK string are used, although the user may type in any length. The string may contain any possible characters.

LOCK is parsed as a simple one-string keyword. The runtime code simply stores the first 8 characters of that string in LOKPSW, and clears the input buffer to prevent any chance of looking at the user input. Strings of shorter characters are end-filled with 0's, so a null string ends up equalling zero.

When Kangaroo wakes up, it calls the LOCK? routine just before returning to user control. The LOCK? routine checks the value in LOKPSW, if it is 0, it returns immediately. Otherwise, it prompts the user for the password, and reads his input. The routine compares the first 8 characters of the input to the value in LOKPSW, and returns with the results of the compare. If LOCK? returns with the zero flag set, then the machine continues to wakeup (either no password, or password right). If the zero flag is clear, then the machine goes back to sleep (wrong password). This only happens when the machine is awakend by the ATTN key. Comparator wakeups bypass LOCK?, and are handled differently (see KR"CMP for more info on this).

An interesting side effect of the LOCK? code (undocumented in the user manual) is that if the password entry is terminated with a mode key instead of the RTN key, the machine will wakeup in that mode. All other terminators will cause an invalid password result.

Mary Jo Hornberger

5/13/82

Battery Detect in Kangaroo

Following is a description of the battery detect sequence in Kangaroo, as seen from both the user's viewpoint, and the internal Kangaroo viewpoint.

From the user's viewpoint:

Fresh batteries to level 1:	Kangaroo as usual
From level 1 to level 2:	Battery annunciator is on, otherwise Kangaroo as usual
At level 2:	If Kangaroo is idle (waiting for input) when the power drops below level 2, Kangaroo will output the 'Low batteries' message and put itself to sleep.
	If we are in a BEEP or WAIT command, using the card reader or HPIL, LISTing, FETCHing or waiting for a key to be hit, these will abort. Other commands should terminate normally. As soon as the current command is finished, Kangaroo will output the 'Low batteries' message and put itself to sleep.
Somewhere below below 2.5 volts:	Hardware reset

Bringing Kangaroo back to life:

If the user tries to wake up Kangaroo and the power level is below level 2, Kangaroo will output the 'Low batteries' message again and put itself back to sleep.

If the power level is between level 2 and level 1, Kangaroo should operate normally with the battery annunciator on.

If the power level is above level 1, we're back to Kangaroo as usual.

Note: the battery annunciator is turned OFF only when we wake up from deep sleep, even if the user plugs in the recharger as soon as the annunciator is turned on.

KR"LOW page 1

From an internal viewpoint:

We use two lines from the Roo Chip to monitor the batteries. The BDR line tells us if we have alkaline batteries, and BDK monitors the actual battery voltage.

At each coldstart or warmstart:

We turn off the battery annunciator, and clear the low power flag (PWRFLG) and the Alkaline battery flag (ALKFLG).

We decide which batteries we have by the value of BDR. If BDR is >.630V we assume we have nicad batteries, otherwise we assume alkalines.

To do this, we select BDR by setting bit#1 in the Power Supply Status Byte (PSSB) to 0. We get around the fact that BDR cannot be read directly by writing a value to the PSSB that corresponds to .630V. If the value we write is less than the current value of BDR, we will get a low power interrupt, and go through the interrupt service routine, setting ALKFLG.

We wait 300 microseconds, then check ALKFLG. If ALKFLG is nonzero (we DO have alkaline batteries), we load BATTRY with ALKLN1 and ALKLN2, which are the first and second interrupt levels for alkaline batteries. If ALKFLG is still 0 (nicads), we load up BATTRY with NICAD1 and NICAD2.

We then select BDK (by setting Bit#1 in PSSB to 1) and set up our first interrupt level by writing the first value in BATTRY to the PSSB with the battery interrupt disabled, then enabling the interrupt. The re-enabling of the interrupt causes the hardware to restart its voltage comparing cycle.

Finally, we clear SVCWRD bit#6 (the 'turn on the battery annunciator' bit) and enable the global interrupts.

At each battery interrupt:

We get an interrupt whenever the BDK line drops to the level corresponding to a value we have written into the power supply comparator. We have two interrupt levels. The initial level is set up by B.INIT every time we wake up from deep sleep (including coldstart). The second level is set up when we reach the first interrupt level.

At the first interrupt, we set bit#6 in SVCWRD so the next time we pass through SPY the battery annunciator will be turned on. At the second interrupt, we set another flag(PWRFLG), so the next time we are waiting for input (WAITKY) or pass through the Mode switcher (MODEKY), those routines will send us to sleep with the 'Low batteries' message.

We keep track of which state we are in by the PWRFLG. PWRFLG is 0 until we drop below the 2nd interrupt level.

The hardware will reset Kangaroo if the power drops below the level needed to keep the system alive.

We use the following equation to find the reference numbers:

```
{(256)(voltage to interrupt)}
{ ------------------------- } -1 = reference number
{        V+ (=5.4V)         }
```

This comes out to be the following:

Battery levels	Alkalines voltage	Alkalines ref#	Nicads voltage	Nicads ref#
Battery annunciator on	3.10V	146D	3.55V	167D
'Low batteries' message	2.70V	127D	3.30V	155D
Hardware reset	-----somewhere below 2.5v-----			

Things used:

```
Globals:
     ALKFLG  1byte  =0 if nicads,1 if alkalines
     PWRFLG  1byte  =0 if BDK above 2nd level,1 if below
     BATTRY  2bytes byte1 =1st interrupt level(ALKLN1 or NICAD1)
                    byte2 =2nd interrupt level(ALKLN2 or NICAD2)
     SVCWRD  bit#6  =1 if need to turn battery annunciator on

Equates:
     ALKLN1  equ  146D  =3.10v =1st alkaline interrupt level
     ALKLN2  equ  127D  =2.70v =2nd alkaline interrupt level
     NICAD1  equ  167D  =3.55v =1st nicad interrupt level
     NICAD2  equ  155D  =3.30v =2nd nicad interrupt level

I/O addresses:
     GINTEN  dad  FF00H  =global interrupt enable address
     PSSB    dad  FF82H  =power supply status byte
     POWVEC  dad  000CH  =power supply interrupt vector

Major routines
     B.INIT   in KR&LOW   coldstart and warmstart battery init
     PWRSRV   in KR&LOW   battery interrupt service routine

Related routines
     SPY      in KR&SER   turns on batt annun if SVCWRD bit#6 set
     MODEKY   in KR&MOD   sends us to nap code if PRWFLG nonzero
     WAITKY   in KR&IO    sends us to nap code if PWRFLG nonzero
     ZZZZZZ   in KR&ZZZ   sends us to sleep with 'Low batteries'
                             message if PWRFLG #0
Other useful routines
     PWROK?   in KR&LOW   returns E nonzero if low power
     STOP?    in KR&LOW   returns E nonzero if low power or ATTN

Related documents:
     See Roo Chip ERS for more information about the PSSB.
```


Routine: ASINIT

File: KR&IN2

Author: AS, GC

Description: Parses "INITIALIZE ':DV'[,X]" and leaves the parameters on the R12 stack.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R12 stack   Parameters pushed

Routines Called:
ERROR+, GETSTP, NUMVA+, SYSJSB.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates a missing parameter error if the user inputs a comma and no number.

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+--------+   +-------------+-------------+
|   |0123|4567|     |    |in |out| Legend |   |  Entry      |  Exit       |
+---+----+----+     +----+---+---+--------+   +-------------+-------------+
|R0 |    |    |     |Mode|   | x | d-BCD  |   |             | Parameters  |
|R10|    |    |     |E   |   | x | b-BIN  |   |             | for ini-    |
|R20|    |    |     |DRP |   | x | i-input|   |             | tialize.    |
|R30|    |    |     |ARP |   | x |        |   |             |             |
|R40|    |    |     +-----------------------+   |             |             |
|R50|    |xxxx|     | MELJSB Needed:   x    |   |             |             |
|R60|    |    |     +-----------------------+   |             |             |
|R70|    |    |     | HANDI Called:         |   |             |             |
+---+----+----+     +-----------------------+   +-------------+-------------+
```

Routine: ASPACK

File: KR&PAK

Author: AS, GC

Description: Parses "PACK ':DV'" and leaves the parameters on the R12 stack.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R12 stack   Parameters pushed

Routines Called:
GETSTP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates a missing parameter error if the user inputs a comma and no number.

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+--------+   +-------------+-------------+
|   |0123|4567|     |    |in |out| Legend |   |  Entry      |  Exit       |
+---+----+----+     +----+---+---+--------+   +-------------+-------------+
|R0 |    |    |     |Mode|   | x | d-BCD  |   |             | Parameters  |
|R10|    |    |     |E   |   | x | b-BIN  |   |             | for pack.   |
|R20|    |    |     |DRP |   | 54| i-input|   |             |             |
|R30|    |    |     |ARP |   | 12|        |   |             |             |
|R40|    |    |     +-----------------------+   |             |             |
|R50|    |xxxx|     | MELJSB Needed:   x    |   |             |             |
|R60|    |    |     +-----------------------+   |             |             |
|R70|    |    |     | HANDI Called:         |   |             |             |
+---+----+----+     +-----------------------+   +-------------+-------------+
```

Routine: BADDEV

File: KR&VFO

Author: AS

Description: Generates error 57, bad transmission, and untalks and unlistens the loop.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg # 0

Routines Called:
ERROR, VFBYE

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates error 57.

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|R0 |    |    |     |Mode|   | x | d-BCD   |   |             |             |
|R10|    |    |     |E   |   | o | b-BIN   |   |             |             |
|R20|    |    |     |DRP |   | x | i-input |   |             |             |
|R30|    |    |     |ARP |   | x |         |   |             |             |
|R40|    |    |     +----------------------+   |             |             |
|R50|    |    |     | MELJSB Needed:  x    |   |             |             |
|R60|    |    |     +----------------------+   |             |             |
|R70|    |    |     | HANDI Called:        |   |             |             |
+---+----+----+     +----------------------+   +-------------+-------------+
Routine:  CLRCOD
```

File: KR&VFO

Author: AS

Description: Clears the VF.COD byte in the devfile line.

Input:
R36/37 pointer to devfile line

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|R0 |  xx|    |     |Mode|   |   | d-BCD   |   |             |             |
|R10|    |    |     |E   |   |   | b-BIN   |   |             |             |
|R20|x   |    |     |DRP |   | 20| i-input |   |             |             |
|R30|    |  ii|     |ARP |   | 36|         |   |             |             |
|R40|    |    |     +----------------------+   |             |             |
|R50|    |    |     | MELJSB Needed:  x    |   |             |             |
|R60|    |    |     +----------------------+   |             |             |
|R70|    |    |     | HANDI Called:        |   |             |             |
+---+----+----+     +----------------------+   +-------------+-------------+
Routine:  DATRP+
```

File: KR&XIT

Author: AS

Description: Calls DATREP and pops the return stack if it returns with E#0.

Input: See DATREP

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
See DATREP

Routines Called:
DATREP

Stack depth R6 (max): 2 + DATREP

Calls to Error routines (include error number and reason):

Notes: See DATREP

```
Reg: x = volatile   Status:                    Ri2 stack:
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|R0 |  xx|    |     |Mode|   | x | d-BCD   |   |             |             |
|R10|    |    |     |E   |   | o | b-BIN   |   |             |             |
|R20|    |    |     |DRP |   | x | i-input |   |             |             |
|R30|    |    |     |ARP |   | x |         |   |             |             |
|R40|    |    |     +----------------------+   |             |             |
|R50|    |    |     | MELJSB Needed:  x    |   |             |             |
|R60|    |    |     +----------------------+   |             |             |
|R70|    |    |     | HANDI Called:        |   |             |             |
+---+----+----+     +----------------------+   +-------------+-------------+
Routine:  DATSND
```

File: KR&PIN

Author: AS

Description: Sends a data byte on HPIL with HP-75 in listener mode.

Input: R57 Data byte to send

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Same as SNDFRM except that

R56 Frame recieved

Routines Called:
SNDFRM, PILREP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: See SNDFRM

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+     +----+---+---+---------+   +-------------+-------------+
|R0 |    |    |     |Mode|   | x | d-BCD   |   |             |             |
|R10|    |    |     |E   |   | o | b-BIN   |   |             |             |
|R20|    |    |     |DRP |   | x | i-input |   |             |             |
|R30|    |    |     |ARP |   | x |         |   |             |             |
|R40|    |    |     +----------------------+   |             |             |
|R50|    |    |     | MELJSB Needed:  x    |   |             |             |
|R60|    |    |     +----------------------+   |             |             |
|R70|    |    |     | HANDI Called:        |   |             |             |
+---+----+----+     +----------------------+   +-------------+-------------+
Routine:  DDLREP
```

File: KR&PIN

Author: AS

Description: Issues a DDLX, where X is the value in R57, onto HPIL.

Input:
    R57   Contains X for DDLX

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    Same as SNDFRM.

Routines Called:
    CMDREP.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                    R12 stack:
+---+----+----+    +----+---+---+--------+    +------------+------------+
|   |0123|4567|    |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+    +------------+------------+
|R0 |    |    |    |Mode|   | x | d-BCD  |    |            |            |
|R10|    |    |    |E   |   | o | b-BIN  |    |            |            |
|R20|    |    |    |DRP |   | x | i-input|    |            |            |
|R30|    |    |    |ARP |   | x |        |    |            |            |
|R40|    |    |    +---------------------+    |            |            |
|R50|    |  ob|    | MELJSB Needed:  x   |    |            |            |
|R60|    |    |    +---------------------+    |            |            |
|R70|    |    |    | HANDI Called:       |    |            |            |
+---+----+----+    +---------------------+    +------------+------------+
Routine:  DDLRP+
```

File: KR&PIN

Author: AS

Description: Calls DDLREP and does an extra return if EHO.

Input:
    Same as DDLREP.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    Same as DDLREP.

Routines Called:
    DDLREP.

Stack depth R6 (max): 2 + DDLREP.

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                    R12 stack:
+---+----+----+    +----+---+---+--------+    +------------+------------+
|   |0123|4567|    |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+    +------------+------------+
|R0 |    |    |    |Mode|   | x | d-BCD  |    |            |            |
|R10|    |    |    |E   |   | o | b-BIN  |    |            |            |
|R20|    |    |    |DRP |   | x | i-input|    |            |            |
|R30|    |    |    |ARP |   | x |        |    |            |            |
|R40|    |    |    +---------------------+    |            |            |
|R50|    |  ob|    | MELJSB Needed:  x   |    |            |            |
|R60|    |    |    +---------------------+    |            |            |
|R70|    |    |    | HANDI Called:       |    |            |            |
+---+----+----+    +---------------------+    +------------+------------+
Routine:  DDTREP
```

File: KR&PIN

Author: AS

Description: Issues a DDTX, where X is the value in R57, onto HPIL.

Input:
    R57   Contains X for DDTX

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    Same as SNDFRM.

Routines Called:
    CMDREP.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                    R12 stack:
+---+----+----+    +----+---+---+--------+    +------------+------------+
|   |0123|4567|    |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+    +------------+------------+
|R0 |    |    |    |Mode|   | x | d-BCD  |    |            |            |
|R10|    |    |    |E   |   | o | b-BIN  |    |            |            |
|R20|    |    |    |DRP |   | x | i-input|    |            |            |
|R30|    |    |    |ARP |   | x |        |    |            |            |
|R40|    |    |    +---------------------+    |            |            |
|R50|    |  ob|    | MELJSB Needed:  x   |    |            |            |
|R60|    |    |    +---------------------+    |            |            |
|R70|    |    |    | HANDI Called:       |    |            |            |
+---+----+----+    +---------------------+    +------------+------------+
Routine:  FILNIT
```

File: KR&HDR

Author: AS

Description: Initialization routine for the MELROM. Intercepts the V.FILE HANDI event and performs mass storage functions.

Input:
    R0   HANDI event

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    HANDLD   Set if FILNIT recognized the event and was able to handle it.

Routines Called:
    EVIL, FLCAT, FLCOPY, FLPURG, FLRENA, FLSBON,
    LOOKUP, RESCON.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                    R12 stack:
+---+----+----+    +----+---+---+--------+    +------------+------------+
|   |0123|4567|    |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+    +------------+------------+
|R0 |ixxx|    |    |Mode| b |   | d-BCD  |    |            |            |
|R10|    |    |    |E   |   | x | b-BIN  |    |            |            |
|R20|xx  |    |    |DRP |   | x | i-input|    |            |            |
|R30|    |    |    |ARP |   | x |        |    |            |            |
|R40|    |    |    +---------------------+    |            |            |
|R50|    |    |    | MELJSB Needed:  x   |    |            |            |
|R60|    |    |    +---------------------+    |            |            |
|R70|    |    |    | HANDI Called:       |    |            |            |
+---+----+----+    +---------------------+    +------------+------------+
Routine:  FLCAF+
```

```
File:     KR&FLO

Author:   AS

Description:   Tries to find the input file name and
     generates an error if it is not found.


Input:
     R36/37     devfile line pointer
     R40/47     name of file to find


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E-Reg      #0 if error occurred


Routines Called:
     ERR1+, FLFIN+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
     Generates an error 62, file not found, if it cannot find
     the file.


Notes:


Reg: x * volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+--------+    +------------+------------+
|   |0123|4567|      |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+    +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |    |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |    |            |            |
|R20|    |    |      |DRP |   | x | i-input|    |            |            |
|R30|    |  ii|      |ARP |   | x |        |    |            |            |
|R40|iiii|iiii|      +---------------------+    |            |            |
|R50|    |    |      | MELJSB Needed:  x   |    |            |            |
|R60|    |    |      +---------------------+    |            |            |
|R70|    |    |      | HANDI Called:       |    |            |            |
+---+----+----+      +---------------------+    +------------+------------+
Routine:  FLCAT
```

```
File:     KR&CA2

Author:   AS

Description:   Catalogs 1 or all files on a mass storage medium.
     Uses [^], [V], [SHIFT]-[^], [SHIFT]-[V] to step through the
     directory.


Input:
     R24/25     FNB Pointer


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E          # 0 if error occurs

Routines Called:
     CATBUF, DEQUE, FLCAF+, LOOKUP, SIGNTF, VF1TO2,
     VFBYE, VFDIR, VFEOD?, VFGLOC, VFHI+, VFMFP?, VFMSG,
     VFNXO-, VFNXE+, VFROE, VFRVDE, VFRWSB, VFRWSK, VFUTL+.
Stack depth R6 (max):

Calls to Error routines (include error number and reason):


Notes:


Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+--------+    +------------+------------+
|   |0123|4567|      |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+    +------------+------------+
|R0 |xxxx|    |      |Mode|   | x | d-BCD  |    |            |            |
|R10|    |    |      |E   |   | x | b-BIN  |    |            |            |
|R20|xxxx|iixx|      |DRP |   | x | i-input|    |            |            |
|R30|xx  |  xx|      |ARP |   | x |        |    |            |            |
|R40|xxxx|xxxx|      +---------------------+    |            |            |
|R50|    |  xx|      | MELJSB Needed:  x   |    |            |            |
|R60|    |xxxx|      +---------------------+    |            |            |
|R70|    |    |      | HANDI Called:       |    |            |            |
+---+----+----+      +---------------------+    +------------+------------+
Routine:  FLCOPY
```

```
File:     KR&COP

Author:   AS

Description:   Dispatches to an appropriate copy routine.
     Parameters are found in the FNB.


Input:
     R24/25     FNB Pointer


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E          #0 if error occurs


Routines Called:
     FLSTOR, FLLOAD, FLTTOT, FLTTOF.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):


Notes:    FLSTOR and FLLOAD are not entry points, but
     are documented here anyway.

Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+--------+    +------------+------------+
|   |0123|4567|      |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+    +------------+------------+
|R0 |  xx|    |      |Mode|   | x | d-BCD  |    |            |            |
|R10|    |    |      |E   |   | x | b-BIN  |    |            |            |
|R20|xx  |ii  |      |DRP |   | x | i-input|    |            |            |
|R30|x   |    |      |ARP |   | x |        |    |            |            |
|R40|xxxx|xxxx|      +---------------------+    |            |            |
|R50|    |xxxx|      | MELJSB Needed:  x   |    |            |            |
|R60|    |xxxx|      +---------------------+    |            |            |
|R70|xxxx|xxxx|      | HANDI Called:       |    |            |            |
+---+----+----+      +---------------------+    +------------+------------+
Routine:  FLSTOR
```

```
File:     KR&COP

Author:   AS

Description:   Handles copying a file from memory to the mass storage
     medium.  Parameters are passed using the FNB.

Input:
     R24/25          FNB pointer
     R54/57          mass storage device name
     DRP             64

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E               #0 if error
     &lt;medium&gt;        new file if no error


Routines Called:
     VFHI+, FLGET1, FLFIN+, FLPUR-, FLNEW, VFRWSK, VFLAD< VFWREC,
     VFRLF?, VFWR, VFRDCL, VFBSY, FLPUR', VFTERM.
Stack depth R6 (max):

Calls to Error routines (include error number and reason):


Notes:


Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+--------+    +------------+------------+
|   |0123|4567|      |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+    +------------+------------+
|R0 |  xx|    |      |Mode| b | x | d-BCD  |    |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |    |            |            |
|R20|x   |ii  |      |DRP |   | x | i-input|    |            |            |
|R30|    |  xx|      |ARP |   | x |        |    |            |            |
|R40|    |xxxx|      +---------------------+    |            |            |
|R50|    |iiii|      | MELJSB Needed:  x   |    |            |            |
|R60|    |xxxx|      +---------------------+    |            |            |
|R70|    |    |      | HANDI Called:       |    |            |            |
+---+----+----+      +---------------------+    +------------+------------+
Routine:  FLLOAD     (Not an entry point)
```

File: KR&COP

Author: AS

Description: Loads a file from medium into RAM. Parameters are found in the FNB. Purges the file in RAM if the user hits [ATTN] or an error occurs.

Input:
- R24/25 FNB pointer
- R64/67 mass storage device name

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E ≠0 if error
- <memory> contains new file if no error.

Routines Called:
VFNI, FLGOFE, FLVFO?, VFRDE, VFRDB?, ERR1, VFSKFL, VFLIF?, VFRWRD, FLSAM?, JSBCRT, ALLOC, FPURGE, VF1ERM.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an error 68, invalid file type, if it cannot recognize the LIF file type of the file on the medium. Generates an error 16, insufficient memory, if the file to be copied will not fit in Kangaroo.

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|   |0123|4567|     |    |in |out| Legend  |  | Entry      | Exit      |
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|R0 |  xx|    |     |Mode| b | x | d-BCD   |  |            |           |
|R10|    |    |     |E   |   | o | b-BIN   |  |            |           |
|R20|xx  |ii  |     |DRP |   | x | i-input |  |            |           |
|R30|xxxx|  xx|     |ARP |   | x |         |  |            |           |
|R40|xxxx|xxxx|     +----------------------+  |            |           |
|R50|    |    |     | MELJSB Needed:  x    |  |            |           |
|R60|    |iiii|     +----------------------+  |            |           |
|R70|    |xxxx|     | HANDI Called:        |  |            |           |
+---+----+----+     +----------------------+  +------------+-----------+
Routine:  FLFIN+
```

File: KR&FLO

Author: AS

Description: Sets up a call to FLFIND. FLFIND takes R70 as a parameter; FLFIN+ moves R40 to R70 and calls FLFIND.

Input:
- R36/37 Devfile line pointer
- R40/47 Name of file to find

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- Z Set if file found
- E ≠ 0 if error occurs

Routines Called:
FLFIND

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|   |0123|4567|     |    |in |out| Legend  |  | Entry      | Exit      |
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|R0 |  xx|    |     |Mode| b |   | d-BCD   |  |            |           |
|R10|    |    |     |E   |   | o | b-BIN   |  |            |           |
|R20|    |    |     |DRP |   | x | i-input |  |            |           |
|R30|    |    |     |ARP |   | x |         |  |            |           |
|R40|iiii|iiii|     +----------------------+  |            |           |
|R50|xxxx|xxxx|     | MELJSB Needed:  x    |  |            |           |
|R60|    |    |     +----------------------+  |            |           |
|R70|xxxx|xxxx|     | HANDI Called:        |  |            |           |
+---+----+----+     +----------------------+  +------------+-----------+
Routine:  FLFIND
```

File: KR&FLO

Author: AS

Description: Finds a directory entry on the mass storage medium whose name matches the 1st 8 characters of the input name. Ignores purged directory entries.

Input:
- R70/77 Name of file to find
- R36/37 devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- Z Set if file found
- <devfile line> CDE field has directory entry if found
- E ≠ 0 if error occurs

Routines Called:
VFDIR+, VFEOD, VFMFP?, VFNXE+.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|   |0123|4567|     |    |in |out| Legend  |  | Entry      | Exit      |
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|R0 |  xx|    |     |Mode|   | x | d-BCD   |  |            |           |
|R10|    |    |     |E   |   | o | b-BIN   |  |            |           |
|R20|    |    |     |DRP |   | x | i-input |  |            |           |
|R30|    |  ii|     |ARP |   | x |         |  |            |           |
|R40|    |    |     +----------------------+  |            |           |
|R50|xxxx|xxxx|     | MELJSB Needed:  x    |  |            |           |
|R60|    |    |     +----------------------+  |            |           |
|R70|iiii|iiii|     | HANDI Called:        |  |            |           |
+---+----+----+     +----------------------+  +------------+-----------+
Routine:  FLFTOF
```

File: KR&CO2

Author: AS

Description: Copies a file from HPIL mass storage device to another. Parameters are passed using the FNB.

Input:
- R24/25 FNB pointer
- R64/67 Device name of the source mass storage device

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E ≠ 0 if error occurs

Routines Called:
VFNI+, FLSWCH, VFNI, FLGTFN, FLGOF+, FLVFO?, VFCDEP, COPY, FLNEW, VFRWSK, VFSKFL, VFRREC, VFLAD, VFDDL2, FLR36, VF1RD, VFWACH, VFADCL, VF1ERM, FLPUR!, VFBYE.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|   |0123|4567|     |    |in |out| Legend  |  | Entry      | Exit      |
+---+----+----+     +----+---+---+---------+  +------------+-----------+
|R0 |  xx|    |     |Mode| b | x | d-BCD   |  |            |           |
|R10|    |    |     |E   |   | o | b-BIN   |  |            |           |
|R20|x   |ii  |     |DRP |   | x | i-input |  |            |           |
|R30|xxxx|xx  |     |ARP |   | x |         |  |            |           |
|R40|    |  xx|     +----------------------+  |            |           |
|R50|    |    |     | MELJSB Needed:  x    |  |            |           |
|R60|    |iiii|     +----------------------+  |            |           |
|R70|    |    |     | HANDI Called:        |  |            |           |
+---+----+----+     +----------------------+  +------------+-----------+
Routine:  FLGOF+
```

File: KR&FLO

Author: AS

Description: Tries to find the input file name and
generates an error if it is not found.

Input:
R36/37 devfile line pointer
R40/47 name of file to find

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg #0 if error occurred

Routines Called:
ERR1+, FLFIN+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an error 62, file not found, if it cannot find
the file.

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|iiii|iiii|     +---------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x   |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | NANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  FLCAT
```

File: KR&CA2

Author: AS

Description: Catalogs 1 or all files on a mass storage medium.
Uses [^], [V], [SHIFT]-[^], [SHIFT]-[V] to step through the
directory.

Input:
R24/25 FNB Pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs

Routines Called:
CATBUF, DEQUE, FLCAF+, LOOKUP, SIGNIF, VF1TO2,
VFBYE, VFDIR, VFEOD?, VFGLOC, VFHI+, VFMFP?, VFMSG,
VFNXD-, VFNXE+, VFRDE, VFRVDE, VFRHSB, VFRWSK, VFUTL+.
Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |xxxx|    |     |Mode|   | x | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | x | b-BIN  |  |            |            |
|R20|xxxx|iixx|     |DRP |   | x | i-input|  |            |            |
|R30|xx  |  xx|     |ARP |   | x |        |  |            |            |
|R40|xxxx|xxxx|     +---------------------+  |            |            |
|R50|    |  xx|     | MELJSB Needed:  x   |  |            |            |
|R60|    |xxxx|     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  FLCOPY
```

File: KR&COP

Author: AS

Description: Dispatches to an appropriate copy routine.
Parameters are found in the FNB.

Input:
R24/25 FNB Pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E #0 if error occurs

Routines Called:
FLSTOR, FLLOAD, FLTTOT, FLFTOF.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: FLSTOR and FLLOAD are not entry points, but
are documented here anyway.

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |     |Mode|   | x | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | x | b-BIN  |  |            |            |
|R20|xx  |ii  |     |DRP |   | x | i-input|  |            |            |
|R30|x   |    |     |ARP |   | x |        |  |            |            |
|R40|xxxx|xxxx|     +---------------------+  |            |            |
|R50|    |xxxx|     | MELJSB Needed:  x   |  |            |            |
|R60|    |xxxx|     +---------------------+  |            |            |
|R70|xxxx|xxxx|     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  FLSTOR
```

File: KR&COP

Author: AS

Description: Handles copying a file from memory to the mass storage
medium. Parameters are passed using the FNB.

Input:
R24/25 FNB pointer
R54/57 mass storage device name
DRP 64

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E #0 if error
<medium> new file if no error

Routines Called:
VFHI+, FLGET1, FLFIN+, FLPUR-, FLNEW, VFRWSK, VFLAD< VFWREC,
VFRLF?, VFWR, VFADCL, VFBSY, FLPUR', VFTERM.
Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |     |Mode| b | x | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|x   |ii  |     |DRP |   | x | i-input|  |            |            |
|R30|    |  xx|     |ARP |   | x |        |  |            |            |
|R40|    |xxxx|     +---------------------+  |            |            |
|R50|    |iiii|     | MELJSB Needed:  x   |  |            |            |
|R60|    |xxxx|     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  FLLOAD    (Not an entry point)
```

File: KR&FL1

Author: AS

Description: Finds space for and creates a new directory entry on the medium. The new entry points to a space for the new file. Uses a kangaroo directory entry or the VF.20E area of the devfile line, depending on the TTT flag.

Input:
- R20 — TTT flag, determines VF.20E or HP-75 directory entry
- R30/31 — HP-75 directory entry pointer (when needed)
- R36/37 — devfile line pointer (always needed)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- R66/67 — sector address of where the file will go
- <medium> — new directory entry
- E — #0 if error occured (new entry probably not present on medium.)

Routines Called:
ERR1+, FLGET1, JSBCRT, VFCDCD, VFCLCH, VFDIR+, VFDUDE, VFEOD?, VFLIF?, VFMFP?, VFNXE+, VFPED?, VFRENA, VFROO?, VFRVDE, VFSECT, VFTIME, VFTRNL, VFUTL+, VFURD1, WARN.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an error 95, medium full, if the medium is full.

Notes: Local subroutine references are omitted from the Routines Called list.

```
Reg: x = volatile   Status:                     R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |  xx|    |     |Mode| b | x | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | o | b-BIN   |   |            |            |
|R20|ix  |    |     |DRP |   | x | i-input |   |            |            |
|R30|iixx|  ii|     |ARP |   | x |         |   |            |            |
|R40|  xx|xxxx|     +----------------------+   |            |            |
|R50|    |xxxx|     | MELJSB Needed:  x    |   |            |            |
|R60|x  x|xxoo|     +----------------------+   |            |            |
|R70|    |xxxx|     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
Routine:  FLPUR-
```

File: KR&PRG

Author: AS

Description: Marks the directory entry in the VF.CDE (current directory entry) area of the devfile line as being purged. The directory entry is then rewritten to the medium.

Input:
- R36/37 — Devfile line pointer

Output (include E-Reg, Z-Flag, C-flag if pertinent):
- E — # 0 if error occurs

Routines Called:
VFDECL

Stack depth R6 (max): 2 + VFDECL

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                     R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |  xx|    |     |Mode|   |   | d-BCD   |   |            |            |
|R10|    |    |     |E   |   |   | b-BIN   |   |            |            |
|R20|    |    |     |DRP |   |   | i-input |   |            |            |
|R30|    |    |     |ARP |   |   |         |   |            |            |
|R40|    |  xx|     +----------------------+   |            |            |
|R50|    |    |     | MELJSB Needed:  x    |   |            |            |
|R60|    |    |     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
Routine:  FLPUR!
```


File: KR&COP

Author: AS

Description: Tries to find a purge the target filename from the mass storage medium during a copy. Will attempt to purge the file even if [ATTN] has been pressed.

Input:
- R24/25 — FNB Pointer
- R36/37 — devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:
FLGET1, FLFIN+, FLPUR-

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                     R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |  xx|    |     |Mode| b | x | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | x | b-BIN   |   |            |            |
|R20|    |    |     |DRP |   | x | i-input |   |            |            |
|R30|    |    |     |ARP |   | x |         |   |            |            |
|R40|    |    |     +----------------------+   |            |            |
|R50|    |   x|     | MELJSB Needed:  x    |   |            |            |
|R60|    |    |     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
Routine:  FLPURG
```

File: KR&PRG

Author: AS

Description: Finds and marks purged a directory entry on the mass storage medium. Parameters are found in the FNB.

Input:
- R24/25 — FNB pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E — # 0 if error occurs

Routines Called:
VFHI+, FLCAF+, FLPUR-, VFBYE

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                     R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |  xx|    |     |Mode| b | x | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | o | b-BIN   |   |            |            |
|R20|    |    |     |DRP |   | x | i-input |   |            |            |
|R30|    |    |     |ARP |   | x |         |   |            |            |
|R40|xxxx|xxxx|     +----------------------+   |            |            |
|R50|    |    |     | MELJSB Needed:  x    |   |            |            |
|R60|    |xxxx|     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
Routine:  FLR36
```

File: KR&C02

Author: AS

Description: Switches R36/37 with the value in the FNBSW in the FNB.

Input:
R24/25 FNB Pointer
R36/37 Value to be put in FNBSW

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R36/37 Value from FNBSW

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +----------+----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+  +----------+----------+
|R0 |xxxx|    |      |Mode|   |   | d-BCD  |  |          |          |
|R10|    |    |      |E   |   |   | b-BIN  |  |          |          |
|R20|    |ii  |      |DRP |   | 36| i-input|  |          |          |
|R30|    |  bb|      |ARP |   | 0 |        |  |          |          |
|R40|    |    |      +---------------------+  |          |          |
|R50|    |    |      | MELJSB Needed:  x   |  |          |          |
|R60|    |    |      +---------------------+  |          |          |
|R70|    |    |      | HANDI Called:       |  |          |          |
+---+----+----+      +---------------------+  +----------+----------+
Routine:  FLRENA
```

File: KR&REN

Author: AS

Description: Finds and renames a directory entry on the mass storage medium. Parameters are obtained from the FNB.

Input:
R24/25 FNB Pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error

Routines Called:
VFBYE, VFDECL, FLGOFE, FLGET1, FLGETFN, VFHI+, VFRENA

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +----------+----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+  +----------+----------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |  |          |          |
|R10|    |    |      |E   |   | o | b-BIN  |  |          |          |
|R20|    |ii  |      |DRP |   | x | i-input|  |          |          |
|R30|    |    |      |ARP |   | x |        |  |          |          |
|R40|    |    |      +---------------------+  |          |          |
|R50|    |    |      | MELJSB Needed:  x   |  |          |          |
|R60|    |xxxx|      +---------------------+  |          |          |
|R70|    |    |      | HANDI Called:       |  |          |          |
+---+----+----+      +---------------------+  +----------+----------+
Routine:  FLSAM?
```

File: KR&FL1

Author: AS

Description: Sets the Z flag based on FLSFLG.

Input:
R24/25 Pointer to FNB

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Z Set if FLSFLG is 0
R20 Value of FLSFLG

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +----------+----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+  +----------+----------+
|R0 |  xx|    |      |Mode|   |   | d-BCD  |  |          |          |
|R10|    |    |      |E   |   |   | b-BIN  |  |          |          |
|R20|o   |ii  |      |DRP |   | 20| i-input|  |          |          |
|R30|    |    |      |ARP |   | 24|        |  |          |          |
|R40|    |    |      +---------------------+  |          |          |
|R50|    |    |      | MELJSB Needed:  x   |  |          |          |
|R60|    |    |      +---------------------+  |          |          |
|R70|    |    |      | HANDI Called:       |  |          |          |
+---+----+----+      +---------------------+  +----------+----------+
Routine:  FLSBON
```

File: KR&FL0

Author: AS

Description: Puts Kangaroo into standby on and adjusts the stack so that the calling routine will return to FLSBOH and HP-75 can be put into previous standby mode.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R6 stack appropriately adjusted. On R6 is a return address to FLSBOF (an internal label in FLSBOH) and the old standby value.

Routines Called:
STAND-

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: The calling routine should NOT do anything with the return stack.

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +----------+----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+  +----------+----------+
|R0 |    |    |      |Mode|   | b | d-BCD  |  |          |          |
|R10|    |    |      |E   |   |   | b-BIN  |  |          |          |
|R20|x xx|    |      |DRP |   | 22| i-input|  |          |          |
|R30|    |    |      |ARP |   | 6 |        |  |          |          |
|R40|    |    |      +---------------------+  |          |          |
|R50|    |    |      | MELJSB Needed:  x   |  |          |          |
|R60|    |    |      +---------------------+  |          |          |
|R70|xxxx|xxxx|      | HANDI Called:       |  |          |          |
+---+----+----+      +---------------------+  +----------+----------+
Routine:  FLSTACK
```

File: KR&FLO

Author: AS, GC

Description: Obtains a device name from the R12 stack, in the form ":DV". The device name is converted to capitals and returned in R64/67. Checks to determine that the name is correct.

Input:
R12 stack Device name

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R64/67 Capitalized name
E 0 iff the name is correct

Routines Called:
GETADH, GETFST

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode|   | b | d-BCD  |   | Device name|            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20|x   |  xx|      |DRP |   | 64| i-input|   |            |            |
|R30| x  |xxxx|      |ARP |   | 34|        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |    |      | MELJSB Needed:  x   |   |            |            |
|R60|    |oooo|      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
Routine: FLSWCH
```

File: KR&CO2

Author: AS

Description:
Does a VFBYE and falls into FLR36.

Input:
R36/37 Devfile line pointer
R24/25 FNB pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R36/37 DevFile line pointer from FLR36

Routines Called:
FLR36

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See FLR36

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |xxxx|    |      |Mode|   | b | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|    |ii  |      |DRP |   |36 | i-input|   |            |            |
|R30|    |  bb|      |ARP |   | 0 |        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |    |      | MELJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
Routine: FLTIOT
```

File: KR&CO2

Author: AS

Description: Handles copying of a file from a mass storage device to the same mass storage device.

Input:
R24/25 FNB Pointer (parameters for copy are in FNB.)
R64/67 Device name of the mass storage device

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg # 0 if error

Routines Called:
VFHI+, FLGIFN, FLGOFE, FLVFO?, VF1102, FLNEW, VFMOVE, FLPUR', VFTERM.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20|x   |ii  |      |DRP |   | x | i-input|   |            |            |
|R30|    |  xx|      |ARP |   | x |        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |xxxx|      | MELJSB Needed:  x   |   |            |            |
|R60|  xx|iiii|      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
Routine: FLVFO?
```

File: KR&FLO

Author: AS

Description: If the VF.CDE field of the extended devfile line refers to a HP-75 readable, non LIF 1 file, then FLVFO? checks that the user has supplied a correct password. Generates an error if the password is incorrect.

Input:
R24/25 FNB Pointer
R36/37 devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if invalid password

Routines Called:
ERR1, VFRDO?, VFLIF?, VFRVDE.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates error 66, invalid password, if the user has supplied an invalid password.

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20|    |ii  |      |DRP |   | x | i-input|   |            |            |
|R30|    |  ii|      |ARP |   | x |        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |xxxx|      | MELJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |xxxx|      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
Routine: INIT.
```

File: KR&IN2

Author: AS

Description: Determines the size of and initialize the mass storage medium. Puts Kangaroo into standby on. If the user-specified size for the number of directory entries is incorrect then an invalid argument error is generated. Uses DDT6 to obtain physical attributes and DDT7 to obtain the size of the medium.

Input:
- R12 Input parameters generated by ASINIT

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E # 0 if error
- <medium> formatted according to LIF 1 if no error.

Routines Called:
FLSBON, ONR12?, ONEB, FLSTAK, VFGET, VFMM? ERR1+, CLRCOD, UNTUNL, VFSTAT, INISIZ, INICHK, VFLAD< DDLREP, VFBSY, VFRUSK, VFDUDE< VFRUSK, VFCDEP, COPY, REVBYT, DDT67, JSBCRT, VFTIME, VFCD46, VFRUUR, VFCLCH, VFBYE.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates error 63, invalid filespec, if communication with the mass storage device is not established.

Notes:
INICHK, DDT67 are local subroutines

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+--------+      +------------+------------+
|   |0123|4567|      |    |in |out| Legend |      | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+      +------------+------------+
|R0 |  xx|  xx|      |Mode|   | b | d-BCD  |      | Parameters |            |
|R10|    |    |      |E   |   | o | b-BIN  |      | from ASINIT|            |
|R20|xx  |    |      |DRP |   | x | i-input|      |            |            |
|R30|xxxx|xx  |      |ARP |   | x |        |      |            |            |
|R40|    |xxxx|      +---------------------+      |            |            |
|R50|    |   x|      | MELJSB Needed:  x   |      |            |            |
|R60|    |    |      +---------------------+      |            |            |
|R70|    |  xx|      | HANDI Called:       |      |            |            |
+---+----+----+      +---------------------+      +------------+------------+
Routine:  DDT67      (Not an entry point)
```

File: KR&IN2

Author: AS

Description: Handles the DDT6 and 7 commands under the extended Filbert protocal. Issues a DDTX, where X is an input parameter, and sends out an SDA. If an ETO is recieved then returns, otherwise sets up a call to VFURC2.

Input:
- R57 Value for DDTX command
- R44/45 How many bytes to recieve for DDTX response
- R46/47 Where to put those bytes.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E # 0 if error occurs
- <memory> Modified by any bytes recieved.

Routines Called:
DDTREP, RDYSD+, VFURC2.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+--------+      +------------+------------+
|   |0123|4567|      |    |in |out| Legend |      | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+      +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |      |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |      |            |            |
|R20|x   |    |      |DRP |   | x | i-input|      |            |            |
|R30|    |    |      |ARP |   | x |        |      |            |            |
|R40|    |iiii|      +---------------------+      |            |            |
|R50|    |  xi|      | MELJSB Needed:  x   |      |            |            |
|R60|    |    |      +---------------------+      |            |            |
|R70|    |    |      | HANDI Called:       |      |            |            |
+---+----+----+      +---------------------+      +------------+------------+
Routine:  INISIZ
```

File: KR&IN2

Author: AS

Description: Determines the size of a mass storage medium. Loads VF.MED with 1FF hex (511 decimal), the default size of the medium. Issues a DDT7 using DDT67. Then reverses the bytes of VF.MED so that Kangaroo can use them. This reversal will reverse either the default value (which was loaded in reverse) or the bytes recieved (which are recieved in reverse.)

Input:
- R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E # 0 if error
- VF.MED Highest addressable sector on the medium

Routines Called:
DDT67, REVBYT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
DDT67 is not an entry point

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+--------+      +------------+------------+
|   |0123|4567|      |    |in |out| Legend |      | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+      +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |      |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |      |            |            |
|R20|xx  |    |      |DRP |   | x | i-input|      |            |            |
|R30|    |  ii|      |ARP |   | x |        |      |            |            |
|R40|    |xxxx|      +---------------------+      |            |            |
|R50|    |   x|      | MELJSB Needed:  x   |      |            |            |
|R60|    |    |      +---------------------+      |            |            |
|R70|    |    |      | HANDI Called:       |      |            |            |
+---+----+----+      +---------------------+      +------------+------------+
Routine:  INICHK     (Not an entry point)
```

File: KR&IN2

Author: AS

Description: Determines if the medium is large enough for the number of directory entries that the user specified. The formula used is

Y := [(X + 7)/8] + 2 + X

where [] are greatest integer, X is the user-input number of directory entries, and Y is the minimum size that the medium must be to support X directory entries.

Input:
- R76/77 Value for X
- R36/37 Devfile Line Pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- R76/77 Value for Y
- E # 0 if illegal size

Routines Called:
ERR1+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an invalid argument, error 11, if the medium is too small to support X directory entries, or if X is 0 or negative.

Notes:
ARP and DRP output are only valid for E # 0.

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+--------+      +------------+------------+
|   |0123|4567|      |    |in |out| Legend |      | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+      +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |      |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |      |            |            |
|R20|    |    |      |DRP |   | 30| i-input|      |            |            |
|R30|xxxx|    |      |ARP |   | 76|        |      |            |            |
|R40|    |    |      +---------------------+      |            |            |
|R50|    |    |      | MELJSB Needed:  x   |      |            |            |
|R60|    |    |      +---------------------+      |            |            |
|R70|    |  bb|      | HANDI Called:       |      |            |            |
+---+----+----+      +---------------------+      +------------+------------+
Routine:  JSBCRT
```

File: KR&FUT

Author: AS

Description: Issues a SYSJSB to CRTMDT.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R44/47 Current time in internal format

Routines Called:
SYSJSB, CRTMDT.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+  +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |      |DRP |   | x | i-input|  |            |            |
|R30|    |    |      |ARP |   | x |        |  |            |            |
|R40|    |oooo|      +---------------------+  |            |            |
|R50|    |    |      | MELJSB Needed:  x   |  |            |            |
|R60|    |    |      +---------------------+  |            |            |
|R70|    |    |      | HANDI Called:       |  |            |            |
+---+----+----+      +---------------------+  +------------+------------+
Routine:  PACK.
```

File: KR&PAK

Author: AS

Description: Packs the medium in the device specified. Generates an error if the pack is interrupted or if the device is not a mass storage device. HP-75 is put in standby on for duration of the pack. A 4 pass fast pack algorithm is used.

Input:
R12 Parameters specifying the device to pack

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error occurs
<medium> packed.

Routines Called:
FLSBON, FLSTAK, VFHI, ERR1+, PAK0, PAK1, PAK2, PAK3, VFBSY, ERR1, VFTERM.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an error 63, invalid filespec, if VFHI fails and the user did not hit [ATTN]. Generates an error 97, invalid pack, if one of the pack passes fails.

Notes:
PAK0, PAK1, PAK2, PAK3 are local subroutines.

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+  +------------+------------+
|R0 | x  |    |      |Mode|   | b | d-BCD  |  |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |      |DRP |   | x | i-input|  |            |            |
|R30|    |    |      |ARP |   | x |        |  |            |            |
|R40|    |    |      +---------------------+  |            |            |
|R50|    |    |      | MELJSB Needed:  x   |  |            |            |
|R60|    |    |      +---------------------+  |            |            |
|R70|    |    |      | HANDI Called:       |  |            |            |
+---+----+----+      +---------------------+  +------------+------------+
Routine:  PAK0       (Not an entry point)
```

File: KR&PAK

Author: AS

Description: Packs the directory of the mass storage medium. Purged file entries are removed and holes are collapsed.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error occurs
<medium> directory packed

Routines Called:
VFDIR+, RSMEM-, VFCDEP, VFEOD?, VFRWO+, COPY, VFWOOP, VFMFP?, COPY, VFMXDE.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |  |            |            |
|R20|    |  xx|      |DRP |   | x | i-input|  |            |            |
|R30|xxxx|xxii|      |ARP |   | x |        |  |            |            |
|R40|xxxx|xxxx|      +---------------------+  |            |            |
|R50|    |  xx|      | MELJSB Needed:  x   |  |            |            |
|R60|    | xxx|      +---------------------+  |            |            |
|R70|    |    |      | HANDI Called:       |  |            |            |
+---+----+----+      +---------------------+  +------------+------------+
Routine:  PAK1       (Not an entry point)
```

File: KR&PAK

Author: AS

Description: Generates a list of triples in the temporary (RESCON) memory area. Each triple is 6 bytes long, and one triple is generates for each directory entry (which is assumed to be packed.) Each triple consists of where a file is, how long it is, and where it will go. Each triple element is in sectors.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error
R24/25 Pointer to the triple list

Routines Called:
VFDIR+, RESCON, VFEOD?, VFRVDE, VFNXE+.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |  |            |            |
|R20|xx  |oox |      |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|      |ARP |   | x |        |  |            |            |
|R40|  xx|xxxx|      +---------------------+  |            |            |
|R50|    |    |      | MELJSB Needed:  x   |  |            |            |
|R60|  xx|xxxx|      +---------------------+  |            |            |
|R70|    |    |      | HANDI Called:       |  |            |            |
+---+----+----+      +---------------------+  +------------+------------+
Routine:  PAK2       (Not an entry point)
```

File: KR&PAK

Author: AS

Description:
     Updates directory entries on the mass storage medium to
     reflect the new locations of the files. Uses the triple
     list to obtain the new values.

Input:
     R24/25   Triple list pointer
     R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E   # 0 if error occurs on HPIL

Routines Called:
     VFDIR+, VFRWSB, VFLAD+, PAK2A, DDLRP+, DATRP+.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
     PAK2A is not an entry point

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |iixx|     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    | xxx|     +---------------------+  |            |            |
|R50|    |   x|     | MELJSB Needed:  x   |  |            |            |
|R60|  xx|xxxx|     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  PAK2A     (Not an entry point)
```


File: KR&PAK

Author: AS

Description: Puts away a buffer 0 which has been updated
     with new file locations. Also reads in the next sector
     from the medium.

Input:
     R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E   # 0 if error occurs

Routines Called:
     VFRWUO, VFRREC

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    | xxx|     +---------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x   |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  PAK3
```

File: KR&PAK

Author: AS

Description: Steps through the triple list, moving each file
     to its new location appropriately.

Input:
     R24/25   Triple list pointer
     R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E-Reg   # 0 if error occurs

Routines Called:
     VFBSY+, VFMOVE

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |iixx|     |DRP |   | x | i-input|  |            |            |
|R30|    |    |     |ARP |   | x |        |  |            |            |
|R40|    |    |     +---------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x   |  |            |            |
|R60|  xx|xxxx|     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  REVBYT
```

File: KR&FUT

Author: AS

Description: Exchanges 2 consecutive bytes in memory.
     The address of the lower byte is in R20/21.
     This routine is needed since many values obtained
     from LIF media have bytes arranged in reverse order
     as far as the Capricorn CPU is concerned.
Input:
     R20/21   Pointer to bytes to exchange

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     <memory> 2 bytes exchanged with each other.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |     |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|ii  |    |     |DRP |   | 20| i-input|  |            |            |
|R30|    |    |     |ARP |   | 20|        |  |            |            |
|R40|    |    |     +---------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x   |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
Routine:  REVPSH
```

File: KR&FUT

Author: AS

Description: Pushes R44/47 in reverse order onto a stack pointed to by R32/33 (R47, R46, R45, R44 in that order.) This needs to be done since many LIF values are in reverse order as far as the Capricorn CPU is concerned.

Input:
    R44/47 Values to push
    R32/33 Where to push them

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    R32/33 Points just after data pushed

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | 44| i-input|  |            |            |
|R30|  bb|    |     |ARP |   | 32|        |  |            |            |
|R40|    |iiii|     +-----------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x     |  |            |            |
|R60|    |    |     +-----------------------+  |            |            |
|R70|    |    |     | HANDI Called:         |  |            |            |
+---+----+----+     +-----------------------+  +------------+------------+
Routine:  RDYSD+
```

File: KR&XIT

Author: AS

Description: Calls RDYSND and pops return stack if it returns with E # 0.

Input:
    See RDYSND

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    See RDYSND

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
    See RDYSND for register usage. RDYSD+ is a so-called plus routine. Do not call from outside MELROM.

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   |   | i-input|  |            |            |
|R30|    |    |     |ARP |   |   |        |  |            |            |
|R40|    |    |     +-----------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x     |  |            |            |
|R60|    |    |     +-----------------------+  |            |            |
|R70|    |    |     | HANDI Called:         |  |            |            |
+---+----+----+     +-----------------------+  +------------+------------+
Routine:  TENRIT
```

File: KR&FUT

Author: AS

Description: Divides a 3 byte number by 1024, with rounding.

Input:
    R45/47 Number to divide

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    R45/47 Original number divided by 1024.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |     |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | 2 | i-input|  |            |            |
|R30|    |    |     |ARP |   |   |        |  |            |            |
|R40|    | bbb|     +-----------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x     |  |            |            |
|R60|    |    |     +-----------------------+  |            |            |
|R70|    |    |     | HANDI Called:         |  |            |            |
+---+----+----+     +-----------------------+  +------------+------------+
Routine:  UNTUNL
```

File: KR&PIN

Author: AS

Description: Untalks, unlistens the loop.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    E # 0 if error
    See SNDFRM also

Routines Called:
    UNTREP, UNLREP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: All this routine does is call UNTREP and UNLREP

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |    |     |ARP |   | x |        |  |            |            |
|R40|    |    |     +-----------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x     |  |            |            |
|R60|    |    |     +-----------------------+  |            |            |
|R70|    |    |     | HANDI Called:         |  |            |            |
+---+----+----+     +-----------------------+  +------------+------------+
Routine:  VF1TD2
```

File: KR&VF4

Author: AS

Description: Moves the bytes in VF.CDE to VF.2DE in a devfile line.

Input:
    R36/37 Pointer to the devfile

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:
    VFCDCO

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+-----------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit      |
+---+----+----+     +----+---+---+--------+  +------------+-----------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |           |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |           |
|R20|    |    |     |DRP |   | x | i-input|  |            |           |
|R30|xx  |xxii|     |ARP |   | x |        |  |            |           |
|R40|    |    |     +-----------------------+  |            |           |
|R50|    |    |     | MELJSB Needed:  x    |  |            |           |
|R60|    |    |     +-----------------------+  |            |           |
|R70|    |    |     | HANDI Called:        |  |            |           |
+---+----+----+     +-----------------------+  +------------+-----------+
```

File: KR&VFW

Author: AS

Description: Does a VFADDR and drops into VFCLCH. Gets the current address of the medium and conditionally sends a close record to the mass storage device.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error

Routines Called:
VFADDR, VFCLCH

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+   +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD   |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |   |            |            |
|R20|    |    |      |DRP |   | x | i-input |   |            |            |
|R30|    |  ii|      |ARP |   | x |         |   |            |            |
|R40|    |    |      +----------------------+   |            |            |
|R50|    |    |      | MELJSB Needed:  x    |   |            |            |
|R60|    |    |      +----------------------+   |            |            |
|R70|    |    |      | HANDI Called:        |   |            |            |
+---+----+----+      +----------------------+   +------------+------------+
```

Routine: VFADDR

File: KR&VFO

Author: AS

Description: Gets the current position of the medium from the mass storage device, updating the VF.LOC field.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs
<VF.LOC> Updated position

Routines Called:
DDTREP, VFTAD+, RDYSD+, DATSND

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+   +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD   |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |   |            |            |
|R20|xx  |    |      |DRP |   | 56| i-input |   |            |            |
|R30|    |  ii|      |ARP |   | x |         |   |            |            |
|R40|    |    |      +----------------------+   |            |            |
|R50|    |  xx|      | MELJSB Needed:  x    |   |            |            |
|R60|    |    |      +----------------------+   |            |            |
|R70|    |    |      | HANDI Called:        |   |            |            |
+---+----+----+      +----------------------+   +------------+------------+
```

Routine: AS&VFB

File: KR&VFB

Author: AS

Description: Obtains the status from the mass storage device and possibly generates an error.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R57 Status byte, invalid if E # 0
E # 0 if error

Routines Called:
VFSTAT, VFERR

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+   +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD   |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |   |            |            |
|R20|    |    |      |DRP |   | x | i-input |   |            |            |
|R30|    |    |      |ARP |   | x |         |   |            |            |
|R40|    |    |      +----------------------+   |            |            |
|R50|    |    |      | MELJSB Needed:  x    |   |            |            |
|R60|    |    |      +----------------------+   |            |            |
|R70|    |    |      | HANDI Called:        |   |            |            |
+---+----+----+      +----------------------+   +------------+------------+
```

Routine: VFBSY+

File: KR&XIT

Author: AS

Description: Sets up a call to VFBSY and does an extra pop from the R6 stack if VFBSY returns with E # 0.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See VFBSY for input and output parameters and register usage. VFBSY+ is a plus routine. Do not call from outside MELROM.

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |    |    |      |Mode|   |   | d-BCD   |  |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |  |            |            |
|R20|    |    |      |DRP |   |   | i-input |  |            |            |
|R30|    |    |      |ARP |   |   |         |  |            |            |
|R40|    |    |      +----------------------+  |            |            |
|R50|    |    |      | MELJSB Needed:  x    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|    |    |      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+
```

Routine: VFBYE

File: KR&VF1

Author: AS

Description: Calls UNTUNL after setting the NORTTN flag.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
UNTUNL

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 | x  |    |      |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |  |            |            |
|R20|    |    |      |DRP |   | 2 | i-input |  |            |            |
|R30|    |    |      |ARP |   | x |         |  |            |            |
>|ur|Has'ent
|R50|    |    |      | MELJSB Needed:  x    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|    |    |      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+
```

Routine: VFCD96

File: KR&VFU

Author: AS

Description: Sets up R44/47 are set with the values described below.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R44/45 32 decimal
R46/47 points to the VF.CDE field in the devfile line

Routines Called:
VFCDEP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |  |            |            |
|P20|    |    |      |DRP |   | x | i-input |  |            |            |
|P30|    |  ii|      |ARP |   | x |         |  |            |            |
|R40|    |oooo|      +----------------------+  |            |            |
|R50|    |    |      | MELJSB Needed:  x    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|    |    |      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+
```

Routine: VFCDCO

File: KR&VFU

Author: AS

Description: DRP is set the VF.CDE in the devfile line. R32/33 is set to 32 decimal, and COPY is called.

Input:
R36/37 Devfile line pointer
DRP set to register by caller

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:
COPY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |    |    |      |Mode|   |   | d-BCD   |  |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |  |            |            |
|R20|    |    |      |DRP | i | x | i-input |  |            |            |
|R30|  xx|  ii|      |ARP |   | x |         |  |            |            |
|R40|    |    |      +----------------------+  |            |            |
|R50|    |    |      | MELJSB Needed:  x    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|    |    |      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+
```

Routine: VFCDEP

File: KR&FUT

Author: AS

Description: DRP is set to VF.CDE field in the devfile
     line.

Input:
     DRP   Set by calling routine
     R36/37   devfile pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     DRP   = R36/37 + VF.CDE

Routines Called:

Stack depth R6 (max):   0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry       | Exit        |
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |   |             |             |
|R10|    |    |      |E   |   |   | b-BIN  |   |             |             |
|R20|    |    |      |DRP | i |   | i-input|   |             |             |
|R30|    |  ii|      |ARP |   | 36|        |   |             |             |
|R40|    |    |      +---------------------+   |             |             |
|R50|    |    |      | MELJSB Needed:  x   |   |             |             |
|R60|    |    |      +---------------------+   |             |             |
|R70|    |    |      | HANDI Called:       |   |             |             |
+---+----+----+      +---------------------+   +-------------+-------------+
```

Routine: VFLCCH

File: KR&VFW

Author: AS

Description: Closes a sector if the byte pointer of VF.LOC is
     nonzero.

Input:
     R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E-Reg   # 0 if HPIL error occurs

Routines Called:
     VFWRCL, VFLRD+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry       | Exit        |
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |             |             |
|R10|    |    |      |E   |   | o | b-BIN  |   |             |             |
|R20|    |    |      |DRP |   | x | i-input|   |             |             |
|R30|    |  ii|      |ARP |   | x |        |   |             |             |
|R40|    |    |      +---------------------+   |             |             |
|R50|    | xxx|      | MELJSB Needed:  x   |   |             |             |
|R60|    |    |      +---------------------+   |             |             |
|R70|    |    |      | HANDI Called:       |   |             |             |
+---+----+----+      +---------------------+   +-------------+-------------+
```

Routine: VFDDL2

File: KR&VFW

Author: AS

Description: Puts filbert into write mode by issuing a DDL2.

Input:
     R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E-Reg   # 0 if HPIL error occurs

Routines Called:
     DDLREP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry       | Exit        |
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |   |             |             |
|R10|    |    |      |E   |   | o | b-BIN  |   |             |             |
|R20|    |    |      |DRP |   | x | i-input|   |             |             |
|R30|    |  ii|      |ARP |   | x |        |   |             |             |
|R40|    |    |      +---------------------+   |             |             |
|R50|    |   x|      | MELJSB Needed:  x   |   |             |             |
|R60|    |    |      +---------------------+   |             |             |
|R70|    |    |      | HANDI Called:       |   |             |             |
+---+----+----+      +---------------------+   +-------------+-------------+
```

Routine: VFDECL

File: KR&VF1

Author: AS

Description: Rewrites the directory entry from VF.CDE and closes
     the record.

Input:
     R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E-Reg   # 0 if error occurs

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry       | Exit        |
+---+----+----+      +----+---+---+--------+   +-------------+-------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |   |             |             |
|R10|    |    |      |E   |   | o | b-BIN  |   |             |             |
|R20|    |    |      |DRP |   | x | i-input|   |             |             |
|R30|    |  ii|      |ARP |   | x |        |   |             |             |
|R40|    |    |      +---------------------+   |             |             |
|R50|    |    |      | MELJSB Needed:  x   |   |             |             |
|R60|    |    |      +---------------------+   |             |             |
|R70|    |    |      | HANDI Called:       |   |             |             |
+---+----+----+      +---------------------+   +-------------+-------------+
```

Routine: VFDIR

File: KR&VF1

Author: AS

Description: Seeks the mass storage medium to the first directory entry.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFRUSK, VFNXD-

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		ii
R40		xxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFDIR+

File: KR&XIT

Author: AS

Description: Calls VFDIR and pops R6 stack on E # 0.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See VFDIR for input and output parameters and register usage. VFDIR+ is a plus routine. Do not call from outside MELROM.

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode			d-BCD
E			b-BIN
DRP			i-input
ARP			
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFDUDE

File: KR&VF2

Author: AS

Description: Writes dummy directory entries to the medium according to the input parameters.

Input:
R20 Dummy byte
R21 Repeat count. Write this many directory entries to the medium, each filled with the dummy byte.
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg # 0 if error occurs

Routines Called:
VFCDEP, VFWRD-, VFCLCK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20	ii	xx
R30		ii
R40	xxxx	xxxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFEOD?

File: KR&VF1

Author: AS

Description: Determines whether the meidum is at the end of the directory

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Z Set if end of edirectory

Routines Called:
VFPED?, VFLED?

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		
R40		xx
R50		xx
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		46	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFERR

File: KR&VF8

Author: AS

Description: Tests the status input and generates a possible error message. Assumes that the status byte has been generated under the filbert protocal.

Input:
R57 Status to test
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error was generated

Routines Called:
ERRORR

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates 93, mass memory, 94, no medium, or 96, invalid medium errors.

Notes:
Does not generate an error for a new medium status.

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+    +----+---+---+---------+  +------------+------------+
|   |0123|4567|    |    |in |out| legend  |  | Entry      | Exit       |
+---+----+----+    +----+---+---+---------+  +------------+------------+
|R0 |    |    |    |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |    |E   |   | o | b-BIN   |  |            |            |
|R20|xx  |    |    |DRP |   | x | i-input |  |            |            |
|R30|xx  |    |    |ARP |   | x |         |  |            |            |
|R40|    |    |    +----------------------+  |            |            |
|R50|    |   i|    | MELJSB Needed:  x    |  |            |            |
|R60|    |    |    +----------------------+  |            |            |
|R70|    |    |    | HANDI Called:        |  |            |            |
+---+----+----+    +----------------------+  +------------+------------+
```

Routine: VFEXCH

File: KR&VF1

Author: AS

Description: Causes mass storage device to exchange its buffers 0 and 1.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error occurs

Routines Called:
VFLAD+, DDLREP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+    +----+---+---+---------+  +------------+------------+
|   |0123|4567|    |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+    +----+---+---+---------+  +------------+------------+
|R0 |    |    |    |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |    |E   |   | o | b-BIN   |  |            |            |
|R20|    |    |    |DRP |   | x | i-input |  |            |            |
|R30|    |  ii|    |ARP |   | x |         |  |            |            |
|R40|    |    |    +----------------------+  |            |            |
|R50|    |    |    | MELJSB Needed:  x    |  |            |            |
|R60|    |    |    +----------------------+  |            |            |
|R70|    |    |    | HANDI Called:        |  |            |            |
+---+----+----+    +----------------------+  +------------+------------+
```

Routine: VFGET

File: KR&VFO

Author: AS

Description: Sets up and calls GETPAD. If successful, expands the devfile line to accomodate additional information stored there by the mass storage code. Clears HANDLD if GETPAD call successful.

Input:
R64/67 Device mnemonic

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R36/37 Devfile line pointer if E = 0
E ≠ 0 if error occurs

Routines Called:
SYSJSB, GETPAD, ALLOC, CLRCOD, SKPLNN

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+    +----+---+---+---------+  +------------+------------+
|   |0123|4567|    |    |in |out| legend  |  | Entry      | Exit       |
+---+----+----+    +----+---+---+---------+  +------------+------------+
|R0 |  xx|    |    |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |    |E   |   | o | b-BIN   |  |            |            |
|R20|x   |    |    |DRP |   | x | i-input |  |            |            |
|R30|xxxx|xxoo|    |ARP |   | x |         |  |            |            |
|R40|    |    |    +----------------------+  |            |            |
|R50|    |    |    | MELJSB Needed:  x    |  |            |            |
|R60|    |iiii|    +----------------------+  |            |            |
|R70|    |    |    | HANDI Called:        |  |            |            |
+---+----+----+    +----------------------+  +------------+------------+
```

Routine: VFGLOC

File: KR&VF1

Author: AS

Description: Get the directory starting sector (VF.DL) and the current directory location (VF.CDL)

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R45/47 VF.CDL
R56/57 VF.DL

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+    +----+---+---+---------+  +------------+------------+
|   |0123|4567|    |    |in |out| legend  |  | Entry      | Exit       |
+---+----+----+    +----+---+---+---------+  +------------+------------+
|R0 |  xx|    |    |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |    |E   |   |   | b-BIN   |  |            |            |
|R20|    |    |    |DRP |   | 46| i-input |  |            |            |
|R30|    |  ii|    |ARP |   | 56|         |  |            |            |
|R40|    | ooo|    +----------------------+  |            |            |
|R50|    |  oo|    | MELJSB Needed:  x    |  |            |            |
|R60|    |    |    +----------------------+  |            |            |
|R70|    |    |    | HANDI Called:        |  |            |            |
+---+----+----+    +----------------------+  +------------+------------+
```

Routine: VFHI

File: KR&VFO

Author: AS

Description: Establish communication with the mass storage device. Ensures that the device is a filbert (SAI response 10 hex), gets its status and medium address, and expands the devfile line.

Input:
R64/67 Device name

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R36/37 Devfile line pointer
E # 0 if error occurs

Routines Called:
VFGET, VFMM?, VFBSY, BLEBUF, VADDR, VFBYE, UNTUNL, LADREP, VFRUSK, VFRWRD, INISIZ, ERR1

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an invalid medium error if it is not formatted according to LIF. Invalid medium error = 96.

Notes:
Calls the internal subroutine VFFBHI. I have included the entry points which VFFBHI calls in the above list.

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD   |   |             |             |
|R10|    |    |      |E   |   | o | b-BIN   |   |             |             |
|R20|x   |    |      |DRP |   | x | i-input |   |             |             |
|R30|  xx|  oo|      |ARP |   | x |         |   |             |             |
|R40|    |xxxx|      +----------------------+   |             |             |
|R50|    |    |      | MELJSB Needed:  x    |   |             |             |
|R60|    |    |      +----------------------+   |             |             |
|R70|    |    |      | HANDI Called:        |   |             |             |
+---+----+----+      +----------------------+   +-------------+-------------+
```

Routine: VFHI+

File: KR&XIT

Author: AS

Description: Sets up a call to VFHI. Does an extra pop of the R6 stack if VFHI returns with E # 0.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
VFHI+ is a plus routine. See VFHI for a input and output parameters and register usage. Do not call from outside MELROM.

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|R0 |    |    |      |Mode|   |   | d-BCD   |   |             |             |
|R10|    |    |      |E   |   |   | b-BIN   |   |             |             |
|R20|    |    |      |DRP |   |   | i-input |   |             |             |
|R30|    |    |      |ARP |   |   |         |   |             |             |
|R40|    |    |      +----------------------+   |             |             |
|R50|    |    |      | MELJSB Needed:       |   |             |             |
|R60|    |    |      +----------------------+   |             |             |
|R70|    |    |      | HANDI Called:        |   |             |             |
+---+----+----+      +----------------------+   +-------------+-------------+
```

Routine: VFLAD+

File: KR&XIT

Author: AS

Description: Sets up a call to VFLAD. If VFLAD returns with E # 0 then VFLAD+ does an extra pop of the R6 stack.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: VFLAD+ is a plus routine. See VFLAD for input and output parameters and register usage. Do not call from outside MELROM.

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|R0 |    |    |      |Mode|   |   | d-BCD   |   |             |             |
|R10|    |    |      |E   |   |   | b-BIN   |   |             |             |
|R20|    |    |      |DRP |   |   | i-input |   |             |             |
|R30|    |    |      |ARP |   |   |         |   |             |             |
|R40|    |    |      +----------------------+   |             |             |
|R50|    |    |      | MELJSB Needed:       |   |             |             |
|R60|    |    |      +----------------------+   |             |             |
|R70|    |    |      | HANDI Called:        |   |             |             |
+---+----+----+      +----------------------+   +-------------+-------------+
```

Routine: VFLED?

File: KR&VF1

Author: AS

Description: Determines if the current directory entry is the logical end of the directory. Z is set if at the logical end of directory. If at the physical end of the directory then Z is not set.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Z Set if at logical end of directory

Routines Called:
VFPED?

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|   |0123|4567|      |    |in |out| Legend  |   | Entry       | Exit        |
+---+----+----+      +----+---+---+---------+   +-------------+-------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD   |   |             |             |
|R10|    |    |      |E   |   |   | b-BIN   |   |             |             |
|R20|    |    |      |DRP |   | 46| i-input |   |             |             |
|R30|    |  ii|      |ARP |   | 56|         |   |             |             |
|R40|    |  xx|      +----------------------+   |             |             |
|R50|    |  xx|      | MELJSB Needed:  x    |   |             |             |
|R60|    |    |      +----------------------+   |             |             |
|R70|    |    |      | HANDI Called:        |   |             |             |
+---+----+----+      +----------------------+   +-------------+-------------+
```

Routine: VFLIF?

File: KR&VF2

Author: AS

Description: Examines the directory entry in the VF.CDE field of the devfile line. If the directory entry refers to an LIF 1 file then returns with Z set.

Input:
    R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    Z   Set if LIF 1 file
    R20/21   File type decremented once

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend  |    | Entry     | Exit      |
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|R0 |  xx|    |    |Mode| b |   | d-BCD   |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN   |    |           |           |
|R20|oo  |    |    |DRP |   | 20| i-input |    |           |           |
|R30|    |  ii|    |ARP |   | 36|         |    |           |           |
|R40|    |    |    +----------------------+    |           |           |
|R50|    |    |    | MELJSB Needed:  x    |    |           |           |
|R60|    |    |    +----------------------+    |           |           |
|R70|    |    |    | HANDI Called:        |    |           |           |
+---+----+----+    +----------------------+    +-----------+-----------+
```

Routine: VFLTBY

File: KR&VFB

Author: AS

Description: Does a VFBSY+ and listens the mass storage device if VFBSY+ found no errors.

Input:
    R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    E   # 0 if error occurs

Routines Called:
    VFBSY+, VFLAD

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend  |    | Entry     | Exit      |
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|R0 |    |    |    |Mode| b |   | d-BCD   |    |           |           |
|R10|    |    |    |E   |   | o | b-BIN   |    |           |           |
|R20|    |    |    |DRP |   | x | i-input |    |           |           |
|R30|    |    |    |ARP |   | x |         |    |           |           |
|R40|    |    |    +----------------------+    |           |           |
|R50|    |    |    | MELJSB Needed:  x    |    |           |           |
|R60|    |    |    +----------------------+    |           |           |
|R70|    |    |    | HANDI Called:        |    |           |           |
+---+----+----+    +----------------------+    +-----------+-----------+
```

Routine: VFLTY+

File: KR&XIT

Author: AS

Description: Calls VFLTBY and pops the R6 stack on E # 0. A plus routine.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: For input and output parameters and register usage see VFLTBY. Do not call from outside MELROM.

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend  |    | Entry     | Exit      |
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|R0 |    |    |    |Mode|   |   | d-BCD   |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN   |    |           |           |
|R20|    |    |    |DRP |   |   | i-input |    |           |           |
|R30|    |    |    |ARP |   |   |         |    |           |           |
|R40|    |    |    +----------------------+    |           |           |
|R50|    |    |    | MELJSB Needed:       |    |           |           |
|R60|    |    |    +----------------------+    |           |           |
|R70|    |    |    | HANDI Called:        |    |           |           |
+---+----+----+    +----------------------+    +-----------+-----------+
```

Routine: VFMFP?

File: KR&VF1

Author: AS

Description: Determines if the directory entry in the VF.CDE field of the devfile line is marked for purging. Returns Z set if it is.

Input:
    R36/37   Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
    Z   Set if file is marked for purging
    R20/21   File type of directory entry in VF.CDE

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend  |    | Entry     | Exit      |
+---+----+----+    +----+---+---+---------+    +-----------+-----------+
|R0 |  xx|    |    |Mode|   |   | d-BCD   |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN   |    |           |           |
|R20|xx  |    |    |DRP |   | 20| i-input |    |           |           |
|R30|    |  ii|    |ARP |   | 36|         |    |           |           |
|R40|    |    |    +----------------------+    |           |           |
|R50|    |    |    | MELJSB Needed:  x    |    |           |           |
|R60|    |    |    +----------------------+    |           |           |
|R70|    |    |    | HANDI Called:        |    |           |           |
+---+----+----+    +----------------------+    +-----------+-----------+
```

Routine: VFMM?

File: KR&VF0

Author: AS

Description: Determines if the specified device is a mass memory which responds to the filbert protocal. Mass memories of this type respond with 10 hex to the SAI command.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Z Set if a mass storage device operating under the filbert protocal
E # 0 if an error occurs on HPIL

Routines Called:
UNTUNL, VF1AD+, RDYSD+, ERROR, DATSND, ERR1, VFBYE

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates an error 56, no loop response, if the SAI frame returns to Kangaroo. Generates an error 57, invalid transmission, if the device does not source an ETO within 10 frames after the SAI. Generates an error 92, not a mass memory, if the device does not respond with a 10 hex to an SAI command.

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+--------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+   +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |   |            |            |
|R20|    |    |     |DRP |   | x | i-input|   |            |            |
|R30|    |  ii|     |ARP |   | x |        |   |            |            |
|R40|    | xxx|     +---------------------+   |            |            |
|R50|    |  xx|     | MELJSB Needed:  x   |   |            |            |
|R60|    |    |     +---------------------+   |            |            |
|R70|    |    |     | HANDI Called:       |   |            |            |
+---+----+----+     +---------------------+   +------------+------------+
```

Routine: VFMOVE

File: KR&VF1

Author: AS

Description: Moves a file on a mass storage device to a new location.

Input:
R36/37 Devfile line pointer
R62/63 Where the file is \
R64/65 How long the file is >- in sectors
R66/67 Where the file should go /

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs on HPIL

Routines Called:
VFRWK+, VFRCEX, VFRWUO, VFEXCH, VFWBUO

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+--------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |   |            |            |
|R20|    |    |     |DRP |   | x | i-input|   |            |            |
|R30|    |  ii|     |ARP |   | x |        |   |            |            |
|R40|    | xxx|     +---------------------+   |            |            |
|R50|    |    |     | MELJSB Needed:  x   |   |            |            |
|R60|  ii|iiii|     +---------------------+   |            |            |
|R70|    |    |     | HANDI Called:       |   |            |            |
+---+----+----+     +---------------------+   +------------+------------+
```

Routine: VFMSG

File: KR&VF2

Author: AS

Description: Untalks, unlistens the loop. Displays a message using MSGOUT. Untalks, unlistens the loop again.

Input:
R26/27 Pointer to message for MSGOUT
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs on HPIL

Routines Called:
UNTUNL, MSGOUT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+--------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+   +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |   |            |            |
|R20|    |  ii|     |DRP |   | x | i-input|   |            |            |
|R30|    |  ii|     |ARP |   | x |        |   |            |            |
|R40|    |    |     +---------------------+   |            |            |
|R50|    |    |     | MELJSB Needed:  x   |   |            |            |
|R60|    |    |     +---------------------+   |            |            |
|R70|    |    |     | HANDI Called:       |   |            |            |
+---+----+----+     +---------------------+   +------------+------------+
```

Routine: VFNXD-

File: KR&VF1

Author: AS

Description: Reads in 32 bytes of into the current directory entry, VF.CDE, in the devfile line.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error on HPIL occurs

Routines Called:
VFCD46, VFRWRD

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                   R12 stack:
+---+----+----+     +----+---+---+--------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+   +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |   |            |            |
|R20|    |    |     |DRP |   | x | i-input|   |            |            |
|R30|    |    |     |ARP |   | x |        |   |            |            |
|R40|    |    |     +---------------------+   |            |            |
|R50|    |    |     | MELJSB Needed:  x   |   |            |            |
|R60|    |    |     +---------------------+   |            |            |
|R70|    |    |     | HANDI Called:       |   |            |            |
+---+----+----+     +---------------------+   +------------+------------+
```

Routine: VFNXDE

File: KR&VF1

Author: AS

Description: Updates the current directory location (VF.DL) and reads in the next directory entry using VFNXD-.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error occurs on HPIL

Routines Called:
VFNXD-

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		ii
R40		xxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFNXE+

File: KR&XIT

Author: AS

Description: Sets up a call to VFNXDE and pops R6 stack if it returns with E ≠ 0. A plus routine.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
This is a plus routine. For a description of register usage and input and output parameters see VFNXDE. Do not call from outside MELROM.

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode			d-BCD
E			b-BIN
DRP			i-input
ARP			
MELJSB Needed:			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFPED?

File: KR&VF1

Author: AS

Description: Determines if the current directory entry in VF.CDE is the physical end of the directory. That is, if VF.CDE has the first 32 bytes of file space.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Z Set if physical end of directory

Routines Called:
VFGLOC

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		ii
R40		xxx
R50		xx
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E			b-BIN
DRP		46	i-input
ARP		56	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFRCEX

File: KR&VFU

Author: AS

Description: Reads a record and exchanges the buffers in the mass storage device.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E ≠ 0 if error occurs

Routines Called:
VFRREC, VFEXCH

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		ii
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFRDE

```
File:    KR&VF4

Author:  AS

Description:  Translates the LIF directory entry in VF.CDE to
     a Kangaroo directory entry in VF.RDE.  Uses only the first
     8 characters of the name and translates the size into bytes,
     kilobytes, or megabytes.  Depending on the size of the file,
     R20 will have a ' ', 'K', or a 'M' respectively.
Input:
     R36/37     Devfile line pointer


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     E          # 0 if error
     R20        Size character


Routines Called:
     SYSJSB, EMCLOK, VFROO?, TENRIT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):



Notes:


Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20|xx  |    |      |DRP |   | 56| i-input|   |            |            |
|R30|    |  ii|      |ARP |   | 36|        |   |            |            |
|R40|xxxx|xxxx|      +---------------------+   |            |            |
|R50|    |xxxx|      | MELJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+

Routine:  VFRENA
```

```
File:    KR&VF1

Author:  AS

Description:  Changes the name in the VF.CDE field of the devfile
     line.  The last 2 characters of the name field are filled
     with blanks.


Input:
     R40/47     New name to put into VF.CDE
     R36/37     Devfile line pointer


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     <VF.CDE>  New name field


Routines Called:


Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):



Notes:


Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|    |    |      |DRP |   | 46| i-input|   |            |            |
|R30|    |  ii|      |ARP |   | 36|        |   |            |            |
|R40|iiii|iiii|      +---------------------+   |            |            |
|R50|    |    |      | MELJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+

Routine:  VFRLF?
```

```
File:    KR&VF2

Author:  GC

Description:  Determines if z Kangaroo directory entry
     in the HP-75 refers to an LIF 1 file.

Input:
     R30/31     Pointer to Kangaroo directory entry

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     Z          Set if LIF 1 file
     R22/23     Type/access bytes of the Kangaroo directory entry

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):


Notes:


Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|  oo|    |      |DRP |   | 22| i-input|   |            |            |
|R30|ii  |    |      |ARP |   | 30|        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |    |      | MELJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+

Routine:  VFROO?
```

```
File:    KR&VF2

Author:  AS

Description:  Determines if the VF.CDE contains a directory entry
     which Kangaroo can read.  If it is not a readable directory
     entry then a HANDIO call is issued.  Only if someone inter-
     cepts the call or if the entry is readable is Z set on return.

Input:
     R36/37     Devfile line pointer


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
     R21        Kangaroo file name type (valid if Z set)
     Z          Set if Kangaroo readable file type


Routines Called:
     HANDIO

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
     Generates a HANDIO call with the event V.RFTY.
     See the HANDI call documentation for a description


Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | x | b-BIN  |   |            |            |
|R20|    |    |      |DRP |   | x | i-input|   |            |            |
|R30|    |    |      |ARP |   | x |        |   |            |            |
|R40|    | xxx|      +---------------------+   |            |            |
|R50|    |  xx|      | MELJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+

Routine:  VFRREC
```

File: KR&VFR

Author: RS

Description: Instructs the mass storage device to read a sector.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs

Routines Called:
VFTAD+, DDTREP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry     | Exit      |
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|R0 |    |    |      |Mode| b |   | d-BCD  |  |           |           |
|R10|    |    |      |E   |   | o | b-BIN  |  |           |           |
|R20|    |    |      |DRP |   | x | i-input|  |           |           |
|R30|    |    |      |ARP |   | x |        |  |           |           |
|R40|    |    |      +---------------------+  |           |           |
|R50|    |   x|      | MELJSB Needed:  x   |  |           |           |
|R60|    |    |      +---------------------+  |           |           |
|R70|    |    |      | HANDI Called:       |  |           |           |
+---+----+----+      +---------------------+  +-----------+-----------+
```

Routine: VFRVDE

File: KX&VF2

Author: RS

Description: Reverses the length and start fields of the LIF directory entry in VF.CDE. This is done so that the Capricorn CPU can manipulate these values.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<VF.CDE> Length and starting address fields reversed

Routines Called:
REVBYT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry     | Exit      |
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|R0 |    |    |      |Mode| b |   | d-BCD  |  |           |           |
|R10|    |    |      |E   |   |   | b-BIN  |  |           |           |
|R20|xx  |    |      |DRP |   | 20| i-input|  |           |           |
|R30|    |  ii|      |ARP |   | 20|        |  |           |           |
|R40|    |    |      +---------------------+  |           |           |
|R50|    |    |      | MELJSB Needed:  x   |  |           |           |
|R60|    |    |      +---------------------+  |           |           |
|R70|    |    |      | HANDI Called:       |  |           |           |
+---+----+----+      +---------------------+  +-----------+-----------+
```

Routine: VFRWO+

File: KR&XIT

Author: RS

Description: A plus routine for VFRWSO.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: A plus routine. See VFRWSO for input/output parameters and register usage. Do not call from outside MELROM.

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry     | Exit      |
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|R0 |    |    |      |Mode|   |   | d-BCD  |  |           |           |
|R10|    |    |      |E   |   |   | b-BIN  |  |           |           |
|R20|    |    |      |DRP |   |   | i-input|  |           |           |
|R30|    |    |      |ARP |   |   |        |  |           |           |
|R40|    |    |      +---------------------+  |           |           |
|R50|    |    |      | MELJSB Needed:      |  |           |           |
|R60|    |    |      +---------------------+  |           |           |
|R70|    |    |      | HANDI Called:       |  |           |           |
+---+----+----+      +---------------------+  +-----------+-----------+
```

Routine: VFRWK+

File: KR&XIT

Author: RS

Description: A plus routine for VFRWSK

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: A plus routine for VFRWSK. See VFRWSK for input/output parameters and register usage. Do not call from outside MELROM.

```
Reg: x = volatile    Status:                  R12 stack:
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|   |0123|4567|      |    |in |out| Legend |  | Entry     | Exit      |
+---+----+----+      +----+---+---+--------+  +-----------+-----------+
|R0 |    |    |      |Mode|   |   | d-BCD  |  |           |           |
|R10|    |    |      |E   |   |   | b-BIN  |  |           |           |
|R20|    |    |      |DRP |   |   | i-input|  |           |           |
|R30|    |    |      |ARP |   |   |        |  |           |           |
|R40|    |    |      +---------------------+  |           |           |
|R50|    |    |      | MELJSB needed       |  |           |           |
|R60|    |    |      +---------------------+  |           |           |
|R70|    |    |      | HANDI Called:       |  |           |           |
+---+----+----+      +---------------------+  +-----------+-----------+
```

Routine: VFRWRD

File: KR&VFR

Author: AS

Description: Read bytes from the mass storage device into memory. Updates VF.LOC when finished.

Input:
R36/37 Devfile line pointer
R44/45 Number of bytes to read
R46/47 Where to put the bytes read

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs
R46/47 Points to the location just after the bytes read in.

Routines Called:
VFBSY+, VFRREC, VFLAD+, DDLREP+, DATRP+, VFTAD+, VFWACH, VFADDR

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|x   |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    |iiii|     +-------------------+  |            |            |
|R50|x   |   x|     | MELJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|    |    |     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: VFRWSO

File: KR&VFS

Author: AS

Description: Seeks the mass storage device to a sector. Checks VF.RSW and VF.FLG flags to detrmine whether the seek should be done.

Input:
R36/37 Devfile line pointer
R46/47 Address of sector to seek to

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFLAD+, DDLAP+, DATRP+, DATREP.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    |  ii|     +-------------------+  |            |            |
|R50|    |  xx|     | MELJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|    |    |     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: VFRWSB

File: KR&VFS

Author: AS

Description: Sets the byte pointer on the mass storage device.

Input:
R36/37 Devfile line pointer
R45 Value to set byte pointer to

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFLAD+, DDLREP+, DATRP+, VFADDR

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    | i  |     +-------------------+  |            |            |
|R50|    |   x|     | MELJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|    |    |     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: VFRWSK

File: KR&VFS

Author: AS

Description: Seeks the mass storage device to a specified sector and byte address.

Input:
R36/37 Devfile line pointer
R45 Byte in sector
R46/47 Sector

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs

Routines Called:
VFRWO+, VFLTY+, VFRWSB

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |    |     |ARP |   | x |        |  |            |            |
|R40|    |    |     +-------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|    |    |     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: VFRWUO

File: KR&VFU

Author: AS

Description: Seeks to a sector and writes buffer 0 on the mass storage device.

Input:
- R36/37 Devfile line pointer
- R45/47 Byte and sector address to which to seek

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E # 0 if HPIL error occurs

Routines Called:
VFRWK+, VFWBUO

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |     | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|R0 |    |    |     |Mode| b |   | d-BCD  |     |           |           |
|R10|    |    |     |E   |   | o | b-BIN  |     |           |           |
|R20|    |    |     |DRP |   | x | i-input|     |           |           |
|R30|    |    |     |ARP |   | x |        |     |           |           |
|R40|    |    |     +---------------------+     |           |           |
|R50|    |    |     | MELJSB Needed:  x   |     |           |           |
|R60|    |    |     +---------------------+     |           |           |
|R70|    |    |     | HANDI Called:       |     |           |           |
+---+----+----+     +---------------------+     +-----------+-----------+
```

Routine: VFRWWR

File: KR&VFW

Author: AS

Description: Writes data to the mass storage medium. Puts the mass storage device into partial write mode before sending bytes.

Input:
- R36/37 Devfile line pointer
- R44/45 Number of bytes to write
- R46/47 Where to obtain bytes to write

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E # 0 if HPIL error occurs
- <mass storage> Information written out

Routines Called:
VFLTY+, VFWRBK, VFWR, VFADDR, VFLTBY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |     | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|R0 |    |    |     |Mode| b |   | d-BCD  |     |           |           |
|R10|    |    |     |E   |   | o | b-BIN  |     |           |           |
|R20|    |    |     |DRP |   | x | i-input|     |           |           |
|R30|    |  ii|     |ARP |   | x |        |     |           |           |
|R40|    |iiii|     +---------------------+     |           |           |
|R50|    |    |     | MELJSB Needed:  x   |     |           |           |
|R60|    |    |     +---------------------+     |           |           |
|R70|    |    |     | HANDI Called:       |     |           |           |
+---+----+----+     +---------------------+     +-----------+-----------+
```

Routine: VFSECT

File: KR&VF4

Author: AS

Description: Determines how large a file will be in sectors from how large the file is in Kangaroo.

Input:
- R30/31 Kangaroo directory entry pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- R20/21 Size of the file in sectors

Routines Called:
VFRLF?

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |     | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |     |           |           |
|R10|    |    |     |E   |   | o | b-BIN  |     |           |           |
|R20|oo  |    |     |DRP |   | x | i-input|     |           |           |
|R30|ii  |    |     |ARP |   | x |        |     |           |           |
|R40|    |    |     +---------------------+     |           |           |
|R50|    |    |     | MELJSB Needed:  x   |     |           |           |
|R60|    |    |     +---------------------+     |           |           |
|R70|    |    |     | HANDI Called:       |     |           |           |
+---+----+----+     +---------------------+     +-----------+-----------+
```

Routine: VFSKFL

File: KR&VF2

Author: AS

Description: Seeks the mass storage medium to the start of the file referred to by the directory entry contained in the VF.CDE field of the devfile line.

Input:
- R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E # 0 if error occurs on HPIL
- <medium> At start of file

Routines Called:
VFRWSK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                      R12 stack:
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |     | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+     +-----------+-----------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |     |           |           |
|R10|    |    |     |E   |   | o | b-BIN  |     |           |           |
|R20|    |    |     |DRP |   | x | i-input|     |           |           |
|R30|    |  ii|     |ARP |   | x |        |     |           |           |
|R40|    | xxx|     +---------------------+     |           |           |
|R50|    |    |     | MELJSB Needed:  x   |     |           |           |
|R60|    |    |     +---------------------+     |           |           |
|R70|    |    |     | HANDI Called:       |     |           |           |
+---+----+----+     +---------------------+     +-----------+-----------+
```

Routine: VFSTAT

File: KR&VFB

Author: AS

Description: Obtains the status of the mass storage medium by performing a serial poll (SST frame). If the busy bit is set then VFSTAT obtains the status again.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R57 Status byte from mass storage device
E # 0 if HPIL error occurs

Routines Called:
RDYSD+, VFIAD+, ERR1+, DATSND

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                R12 stack:
+---+----+----+    +----+---+---+--------+  +------------+------------+
|   |0123|4567|    |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+  +------------+------------+
|R0 |    |    |    |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |    |E   |   | o | b-BIN  |  |            |            |
|R20|    |    |    |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|    |ARP |   | x |        |  |            |            |
|R40|    |    |    +---------------------+  |            |            |
|R50|x   |   o|    | MELJSB Needed:  x   |  |            |            |
|R60|    |    |    +---------------------+  |            |            |
|R70|    |    |    | HANDI Called:       |  |            |            |
+---+----+----+    +---------------------+  +------------+------------+
```

Routine: VFIAD+

File: KR&XIT

Author: AS

Description:
A plus routine for VFIAD.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: A plus routine for VFIAD. For input/output parameters and register usage see VFIAD. Do not call from outside MELROM.

```
Reg: x = volatile   Status:                R12 stack:
+---+----+----+    +----+---+---+--------+  +------------+------------+
|   |0123|4567|    |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+  +------------+------------+
|R0 |    |    |    |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |    |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |    |DRP |   |   | i-input|  |            |            |
|R30|    |    |    |ARP |   |   |        |  |            |            |
|R40|    |    |    +---------------------+  |            |            |
|R50|    |    |    | MELJSB Needed:      |  |            |            |
|R60|    |    |    +---------------------+  |            |            |
|R70|    |    |    | HANDI Called:       |  |            |            |
+---+----+----+    +---------------------+  +------------+------------+
```

Routine: VFTERM

File: KR&VF1

Author: AS

Description: Rewinds the mass storage medium (issues a DDL7) and untalks and unlistens the mass storage device. If a previous error has occurred and E # 0 on input then the device is only untalked and unlistened.

Input:
R36/37 Devfile line pointer
E = 0 then device is rewound

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:
DDLREP, VFLAD+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                R12 stack:
+---+----+----+    +----+---+---+--------+  +------------+------------+
|   |0123|4567|    |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+  +------------+------------+
|R0 |    |    |    |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |    |E   | i |   | b-BIN  |  |            |            |
|R20|    |    |    |DRP |   | 2 | i-input|  |            |            |
|R30|    |    |    |ARP |   | x |        |  |            |            |
|R40|    |    |    +---------------------+  |            |            |
|R50|    |   x|    | MELJSB Needed:  x   |  |            |            |
|R60|    |    |    +---------------------+  |            |            |
|R70|    |    |    | HANDI Called:       |  |            |            |
+---+----+----+    +---------------------+  +------------+------------+
```

Routine: VFTIME

File: KR&VF3

Author: AS

Description: Converts internal Kangaroo time into LIF format and puts it in memory.

Input:
R32/33 Where to put converted time information
R44/47 Internal time to convert

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R32/33 Points just after converted time information

Routines Called:
SYSJSB, DCCLOK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                R12 stack:
+---+----+----+    +----+---+---+--------+  +------------+------------+
|   |0123|4567|    |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+    +----+---+---+--------+  +------------+------------+
|R0 |    |    |    |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |    |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |    |DRP |   | 40| i-input|  |            |            |
|R30|  bb|    |    |ARP |   | 32|        |  |            |            |
|R40|   x|iiii|    +---------------------+  |            |            |
|R50|xxxx|xxxx|    | MELJSB Needed:  x   |  |            |            |
|R60|    |    |    +---------------------+  |            |            |
|R70|    |    |    | HANDI Called:       |  |            |            |
+---+----+----+    +---------------------+  +------------+------------+
```

Routine: VFIAHL

File: KR&VF2

Author: AS

Description: Translates a Kangaroo file type into an LIF file
type. Obtains the Kangaroo file type from a Kangaroo directory
entry. The LIF file type is put into VF.FTY in the devfile
line. If the Kangaroo file type is not mainframe defined
then VFTRNL assumes that a LEX file created it, and therefore
issues a HANDI call for the LEX file to intercept and
translate the file type.

Input:
R30/31 Kangaroo directory entry pointer
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if VFTRNL cannot translate the file type (no
one intercepts the HANDI call.)
<VF.FTY> LIF file type

Routines Called:
HANDI

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: Issues a HANDI call with the event V.LFTY

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20| x  |    |     |DRP |   | x | i-input|  |            |            |
|R30|ii  |  ii|     |ARP |   | x |        |  |            |            |
|R40|    | xxx|     +---------------------+  |            |            |
|R50|    |  xx|     | MELJSB Needed:  x   |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:   x   |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
```

Routine: VFUTL+

File: KR&XIT

Author: AS

Description: A plus routine for VFUTL.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: See VFUTL for input/output parameters and register usage.
Do not VFUTL+ from outside the MELROM

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   |   | i-input|  |            |            |
|R30|    |    |     |ARP |   |   |        |  |            |            |
|R40|    |    |     +---------------------+  |            |            |
|R50|    |    |     | MELJSB Needed:      |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
```

Routine: VFWACH

File: KR&VFR

Author: AS

Description:
Reads or monitors bytes from a mass storage device. If reading
stores the bytes in RAM. If monitoring, decrements a counter
for each data frame which traverses the loop. When the counter
zeroes then a NRD is sent.

Input:
R36/37 Devfile line pointer
R20 Read/monitor flag, if 0 then read mode
R44/47 Read mode: same as VFRWRD
Monitor mode: R45/47 is the number of bytes
which will traverse the loop.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
DDTREP, RDYSD+, VFWAC2.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|i   |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    |iiii|     +---------------------+  |            |            |
|R50|    |  xx|     | MELJSB Needed:  x   |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
```

Routine: VFWAC2

File: KR&VFR
Author: AS

Description: Monitors or reads bytes as they travel around
the loop. If reading then bytes are stored in memory.
If monitoring then a counter is decremented. When the
counter zeroes then a NRD frame is sourced. VFWAC2
assumes that the caller has sent a DDTX and an SDA, and that
a recieved data frame is in R56/57. If in monitor mode
VFWAC2 will put away R57 in memory.

Input:
R20 Monitor/read flag; if 0 then read
R37/37 Devfile line pointer
R44/47 Read mode: same as VFRWRD
Monitor mode: R45/47 is a counter telling
how many bytes should
traverse the loop.
R56/57 Frame recieved by caller

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs
<memory> modified if in read mode.

Routines Called:
DATSND, UNTUNL, VFBSY+, RDYSND, ERR1+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Generates error 57, bad transmission, if a premature
ETO is recieved.

Notes: VFWAC2 was added to fix bug #963, the LIF physical
attributes bug. In the mainframe VFWAC2 is only called
in read mode to recieve bytes sent in response to a
DDT6 or 7 command.

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |  |            |            |
|R20|i   |    |     |DRP |   | x | i-input|  |            |            |
|R30|    |  ii|     |ARP |   | x |        |  |            |            |
|R40|    |iiii|     +---------------------+  |            |            |
|R50|    |  ii|     | MELJSB Needed:  x   |  |            |            |
|R60|    |    |     +---------------------+  |            |            |
|R70|    |    |     | HANDI Called:       |  |            |            |
+---+----+----+     +---------------------+  +------------+------------+
```

Routine: VFWBUO

File: KR&VF1

Author: AS

Description: Issues a DDL2 and closes the sector on the mass storage device.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFLAD+, VFDDL2, VFWRCL

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+---------+      +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |      | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+      +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD   |      |            |            |
|R10|    |    |     |E   |   | o | b-BIN   |      |            |            |
|R20|    |    |     |DRP |   | x | i-input |      |            |            |
|R30|    |  ii|     |ARP |   | x |         |      |            |            |
|R40|    |    |     +----------------------+      |            |            |
|R50|    |    |     | MELJSB Needed:  x    |      |            |            |
|R60|    |    |     +----------------------+      |            |            |
|R70|    |    |     | HANDI Called:        |      |            |            |
+---+----+----+     +----------------------+      +------------+------------+
```

Routine: VFWOOP

File: KR&VFW

Author: AS

Description: Sets up the VF software and the mass storage device in write only mode. Writes bytes to the mass storage medium and then updates the medium position in VF.LOC.

Input:
R36/37 Devfile line pointer
R44/45 Number of bytes to write
R46/47 Where to obtain bytes from

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if error occurs
<medium> Bytes written (valid if E = 0)

Routines Called:
VFLTY+, VFWREC, VFWR, VFADDR, VFLTBY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+---------+      +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |      | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+      +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD   |      |            |            |
|R10|    |    |     |E   |   | o | b-BIN   |      |            |            |
|R20|    |    |     |DRP |   | x | i-input |      |            |            |
|R30|    |  ii|     |ARP |   | x |         |      |            |            |
|R40|    |iiii|     +----------------------+      |            |            |
|R50|    |   x|     | MELJSB Needed:  x    |      |            |            |
|R60|    |    |     +----------------------+      |            |            |
|R70|    |    |     | HANDI Called:        |      |            |            |
+---+----+----+     +----------------------+      +------------+------------+
```

Routine: VFWR

File: KR&VFW

Author: AS

Description: Sends out data bytes on the loop. Assumes that the caller has done any other necessary setup.

Input:
R44/45 Number of bytes to send out
R46/47 Where to obtain those bytes

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
DRTRP+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
ARP/DRP output are only valid if E = 0

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+---------+      +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |      | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+      +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD   |      |            |            |
|R10|    |    |     |E   |   | o | b-BIN   |      |            |            |
|R20|    |    |     |DRP |   | 2 | i-input |      |            |            |
|R30|    |    |     |ARP |   | 44|         |      |            |            |
|R40|    |iiii|     +----------------------+      |            |            |
|R50|    |   x|     | MELJSB Needed:  x    |      |            |            |
|R60|    |    |     +----------------------+      |            |            |
|R70|    |    |     | HANDI Called:        |      |            |            |
+---+----+----+     +----------------------+      +------------+------------+
```

Routine: VFWRBK

File: KR&VFW

Author: AS

Description: Puts the VF machine in partial write mode by examining the VF.RSW flag and possibly backing the medium up 1 sector. The mass storage device is put into partial write mode with a DDL6.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFRWSK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+---------+      +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |      | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+      +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD   |      |            |            |
|R10|    |    |     |E   |   | o | b-BIN   |      |            |            |
|R20|    |    |     |DRP |   | x | i-input |      |            |            |
|R30|    |  ii|     |ARP |   | x |         |      |            |            |
|R40|    | xxx|     +----------------------+      |            |            |
|R50|    |   x|     | MELJSB Needed:  x    |      |            |            |
|R60|    |    |     +----------------------+      |            |            |
|R70|    |    |     | HANDI Called:        |      |            |            |
+---+----+----+     +----------------------+      +------------+------------+
```

Routine: VFWRCL

File: KR&VFW

Author: AS

Description: Closes the sector on the mass storage device. Issues a DDL8.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
DDLREP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		x
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFWRD1

File: KR&VF1

Author: AS

Description: Seeks to the input address and writes the VF.CDE area (32 bytes) of the devfile line to the mass storage medium.

Input:
R36/37 Devfile line pointer
R45/47 Address to write VF.CDE

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFRWK+, VFWRD-

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		ii
R40		iii
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFWRD-

File: KR&VF1

Author: AS

Description: Writes out 32 bytes from the VF.CDE field of the devfile line.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFCD46, VFRWWR

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		ii
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFWRDE

File: KR&VF1

Author: AS

Description: Calls VFWRD1 using the current directory location, VF.CDL, as the address to seek filbert.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E # 0 if HPIL error occurs

Routines Called:
VFWRD1

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		ii
R40		xxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: VFWREC

File: KR&VFW

Author: AS

Description: Puts the VF machine into write mode and falls into VFDDL2.

Input:
R36/37 Devfile line pointer

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E N 0 if HPIL error occurs

Routines Called:
VFDDL2

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		ii
R40		
R50		x
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
MELJSB Needed: x			
HANDI Called:			

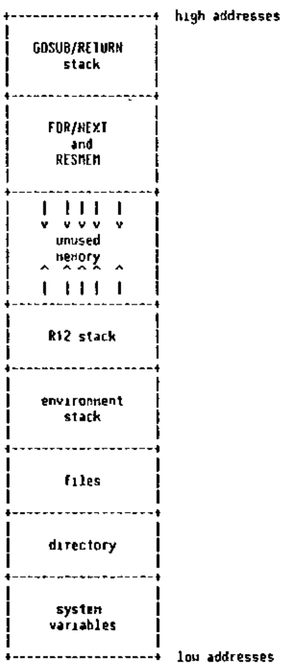
R12 stack:

Entry	Exit

"Memories, light the corners of my mind"
-from the movie "The Way We Were"

The Overall Layout of Memory

Memory is divided into the eight regions. Each of these regions will be discussed in the paper to follow. But first to give you an idea of how it is all laid out, here is a small map of memory.



Basically memory consists of two areas of memory growing toward each other with the unallocated region in between. The allocation and

release of memory in each area is under the control of specific memory routines. I would caution the user not to try to circumvent the use of these routines since the slightest nudge to the precarious balance of Kangaroo's memory could be fatal.

For the purposes of this paper up, higher and above mean larger addresses and down, lower, and below mean smaller addresses. The convention of always drawing memory with the highest addresses at the top of the paper will be adhered to throughout this document. Top and bottom usually refer to active and nonactive ends respectively of stack-like structures and do not mean to imply higher or lower addresses. The term upper memory refers to all the reserved memory above the unused area of memory. The term lower memory refers to all the reserved memory below the unused area of memory. The implementor will be distinguished from the user or BASIC user by being the individual who programs in assembler and interacts with the system at that level.

The GOSUB/RETURN Stack

For every pending RETURN of a GOSUB there is an entry on the GOSUB/RETURN stack. The stack grows down to lower addresses pushing the RESMEM area before it. A GOSUB entry is six bytes long and is in the following format:

```
offset  size      function
  0      2    relative return address (R10 rel. to RNFILE)
  2      2    relative PCR (PCR rel. to RNFILE)
  4      2    GOSUB type flag which has one of these values:
                  if  highest bit 0 then it contains ON
                      TIMER number.
                  if  only highest bit is set then this
                      GOSUB  entry  is   for  a  regular
                      GOSUB.
                  if highest bit is set and the rest is
                      nonzero  then  the number ignoring
                      the  high  bit is the relative ad-
                      dress of the ON ERROR statement.
```

The following are some important pointers in this region of memory.

LWAMEM - the highest usable location in RAM and the bottom of the GOSUB/RETURN stack.
NXTRTN - the top of the GOSUB/RETURN stack. This is where the next GOSUB entry would go.
E.GCNT - offset into the environment to a one byte quantity for the number of GOSUB's pending in the current running program.

The routine that both allocates and releases memory in this area is GETMEM.

GETMEM - A positive size allocates memory in the GOSUB/STACK area. Negative size deallocates the space.

Roo Memory Document hp pirate 7/29/1982 page 2

RESMEM Area

This area is the most error prone of all the memory regions. The important variables in this region are:

NXTRTN - the pointer to the bottom byte in the RESMEM area.

LAVAIL - the pointer to the top of the RESMEM area. This is where the most recent FOR entry is expected to be found AND where to find the first byte of the most recently allocated scratch space.
E.FCNT - offset into the environment to a one byte quantity for the number of FOR's pending in the current running program.

In this area lives simultaneously the FOR/NEXT stack and the scratch memory heap known as RESMEM (reserved memory). Even though these two functions of memory coexist in the same area, let us treat them separately for a moment and then resolve the conflicts that will arise. The FOR entries are placed in stack like fashion in the RESMEM area growing downward into unused memory.

FOR entries are 22 byte quantities that contain:

```
offset  size      function
  0      2    relative PCR (relative to RNFILE)
  2      2    relative R10 (relative to RNFILE)
  4      2    relative pointer  to  VPA entry  for index variable
                        (relative to RNFILE)
  6      8    terminating value for loop (stack number format)
 14      8    increment value for loop (stack number format)
```

FOR entries are created by a call to RSMEM= in the FOR token. If the FOR code attempts to place a FOR entry for a variable whose loop index variable is already used by a pending FOR then the old FOR entry is deleted with the help of COPY before the new one is created. The top entry in the FOR stack is destroyed by the routine EXNXT.

The RESMEM area is also where scratch memory can be had. A call to an allocation routine pushes down the LAVAIL pointer and returns it as a pointer to the newly acquired memory. When RESMEM area is appropriated the amount taken is usually added to a running total kept in the current environment. A flag in the aquisition routine determines if the size is to be added. When memory is released with the RELMEM statement, the amount of memory that was summed into the special location in the current environment will be released.
The scratch memory is used for intermediate results such as string values returned by functions like CHR$ and the string concatenation operator. Scratch memory is also used for formatting numbers for output.
The question that now arises is how can the RESMEM area contain

Roo Memory Document hp pirate 7/29/1982 page 3

both scratch memory and FOR stack? The answer is: not easily! Whenever the most recent FOR entry needs to be found it is assumed that it is at the boundary of the unused memory and the RESMEM area. This implies that there is no scratch memory on the top of the FOR stack to impede the fetching. This is done by calling RELMEM appoximately one zillion times throughout Kangaroo's interpretation loop! For instance, RELMEM's can be found at the end of each statement, including at-signs, and before each expression in a FOR statement. The effect that this has on the implementor is he can't, for example, assume that any information placed in the RESMEM area will survive over statement boundaries. This was necessary however to allow NEXT, FOR, TO and STEP to get at the values in the FOR entry which can only be found by discovering where the top of the FOR stack is. RELMEM's are found:

1. in the beginning of POPENV. This routine is called for execution of the END token.
2. at GORTN, which is called at the end of lines, NEXT, GOTO, and GOSUB.
3. at the ATSIGN code.
4. in GTFOR which is called at the beginning of the TO and STEP tokens.
5. in the BASIC to TEXT and TEXT to BASIC transform code.

To further complicate things, as multiple CALLs are made new layers of FOR stack and RESMEM area are laid down. The size of the new FOR stack and the size of the scratch memory consunned are recorded in each new environment. After several calls the RESMEM area may look like this:

```
                  |                  |
                  |      GOSUB       |
                  |      stack       |
       NXTRTN ->  +------------------+  from earlier cal-
ling pgm
                  |     FOR/NEXT     |  (sizes in environment)
                  |      stack       |
                  +------------------+
                  |     reserved     |
                  |     scratch      |
                  |      memory      |
                  +------------------+
                  |     FOR/NEXT     |  from current program
                  |      stack       |  (sizes in environment)
                  +------------------+
                  |     reserved     |
                  |     scratch      |
                  |      memory      |
       LAVAIL ->  +------------------+
                  |      unused      |
                  |      memory      |
                  |                  |
```

Roo Memory Document hp pirate 7/29/1982 page 4

The important routines for this region are:

EXNXT - deletes the top entry of the FOR stack
GTFOR - does a RELMEM and then fetches the index variable from the top entry in the FOR stack.
NOMORE (local label in RN&FOR) - creates the FOR entry.
RSMEM- - reserves R56 bytes in the RESMEM area recording the amount stored in the environment.
RSMEM= - reserves R56 bytes in the RESMEM area recording the amount stored in the environment if the E flag is set.

RESCON - reserves the amount of memory specified in the byte following the JSB to RESCON.

Roo Memory Document hp pirate 7/29/1982 page 5

The Unused Memory

The 'unused memory' is the part of memory that falls between the downward growing RESMEM area and the upward growing R12 and file area. For the machine to continue to function there is a minimal amount of unused memory that must exist. It is from this space that memory will be allocated for the runtime stack (R12) and for storage of the calculator program tokens. To do this, as memory is allocated the distance between LAVAIL and TOS must and will be kept larger than the value stored in location LEEWAY (see the section on memory overflow).

Relevant variables and pointers in this part of memory are:

LAVAIL - the lower bound of the RESMEM area.
R12 - the register that points to the top of the Kangaroo operand stack
TOS - (short for Top Of Stack -old capricorn terminology [misguided terminology -editorial]) the pointer to the BOTTOM of the R12 stack.
LEEWAY - contains the required amount of separation between TOS and LAVAIL to prevent memory lockup.

The R12 Stack

The R12 stack is used during Kangaroo execution as the operand stack for the token based machine. The R12 stack is also used by the decompiler as a stack to disentangle the operators and operands from their RPN internal form. Finally the R12 stack is used during parsing as the place where the token stream is constructed before being placed in the desired file.

Since the R12 stack is the working stack of Kangaroo, it is important to know when Kangaroo clears it. This is done by setting R12 to TOS (see SETR12). The R12 stack is cleared in the following places:

1. at ENDIT when the interpreter loop is finished.
2. at SETINI to init the parser for parsing a line.
3. at PAREX when we exit the parser.
4. at the end of pointer allocation (PTRALO) if there was a error.
5. at the end of decompile.
6. if there was an error or attn was hit in the INPUT statement.
7. if FLORD or FLORDT fails the type test.
8. at end of statement in ATSIGN and GORTN.
9. at the end of the TRANSFORM command.
10. even though this is already handled in some cases, the ATSIGN code is called in the ERROR code thereby clearing the R12 stack.

During the running of the interpreter the operands for the tokens are placed on the R12 stack. The token that removes the operands must know what type of operands are there in order to remove them; however, the type is not stored explicitly on the stack. To determine the type, the token usually either assumes that the operands are of a fixed type in a fixed order or by looking at how many bytes are stored on the stack determines what is on the stack. For example numbers go on the stack as 8 bytes regardless of whether they were generated by a real, integer or short variable. ON the other hand, strings go on the stack as 2 bytes of address and 2 bytes of length.

Important pointers in the R12 stack are:

R12 - the top of the R12 stack.
TOS - the bottom of the R12 stack. This is good for protecting earlier R12 stacks from current work. NOTE: if the user leaves something on the R12 stack below TOS no one will clean up it for him.
NXTMEM - the top of the environment stack.
STSIZE - an archaic but still used pointer to the line length of the statement being currently parsed onto the R12 stack. This is used by the parser and line editor.

There are times when the user may want to protect the contents of the R12 stack against being cleared or misused by by some intermediate routine. This can be accomplished by remembering the current value of TOS in a private location and moving TOS up to R12. Since TOS is the bottom of the stack the stack is now effectively empty and the inter-

mediate routine can be called. Be sure that TOS gets restored! If the intermediate routine moves memory around then an absolute pointer to the old TOS can't be saved but we might suggest that the implementor save the size of the R12 stack.

```
                  |                    |
                  |       unused       |
         R12-->   |       space        |
                / +--------------------+
```

```
   current | |                    |
     stack | |        R12         |
           | |       stack        |
     TOS-->  |                    |
             +--------------------+
           / |                    |
     saved | |        old         |    R12
     stack | |       stack        |
           | |                    |
           \ +--------------------+
```

It used to be the case that string pointers and l-values where placed on the stack as absolute addresses. This is not a safe. Witness the case in which a statement makes a function call. The function will save the contents of the R12 stack. The function may then perform as much memory manipulation and execute as many statements as it chooses before returning to use the data that was saved there. Indeed at the end of each statement the R12 stack was cleared but not completely because the old R12 stack still presists below TOS. Upon return to the caller the addresses in the old R12 stack will be expected to be valid. Not so. Our function moved all the data that we were pointing to out from under us.

In order to alleviate this problem addresses that would point down into environments that is areas that might move during memory allocation will be made relative to the value of R12 at the time they are placed on the stack. The assumption here is that the distance between the place where the address is placed and where the address points will not change for addresses pointing into the environment. Note that this scheme is of no help to those who want to point into a file. We recommend that these kind of pointers be relative to the beginning of the file and a filename and offset take the place of the address. For pointers to other areas of memory absolute pointers will work fine.

There is a convention for placing relative addresses on the R12 stack. The convention is, if the value is positive then it is relative, if it is negative then it is absolute. Note, this prevents one from having absolute pointers to the lower 32K bytes.

When an address is plucked from the R12 stack it is necessary to decide if it is a relative address or not. If it is, then it must be converted to an absolute address. PUTREL, PUTADR, and PUTRLM are routines in RM&UTY to help make the addresses for the stack relative. GETADR, GETADM, and GETAD+ are helpful little routines to pull addresses from the stack and make them absolute if they need be. Routines that might be useful in the R12 stack are:

SETR12 - a small routine to set R12 to TOS.
EXPAND - (used in the decompiler) a disgusting routine to allow non-stack like insertions in the R12 stack.
GETADR, GETADM, GETAD+, PUTREL, PUTRLM, PUTADR, - routines for getting addresses from the R12 stack.

Environment Stack

The environments contain all the data necessary to run a given instance of a program and are created one for each instance of a running program. In the environment you will find all the things about a program that change from one execution to the next such as the variable values, GOSUB and FOR counts, DATA pointers, and ON ERROR status.

The layout of an environment is:

offset	size	equate	mod	
0	2	E.LEN	C	- length of environment including the ECB and divider byte.
2	2	E.PREV	C	- length of previous block. This gives us a way back to the previous environment.
4	2	E.RMEM	S	- amount of reserved memory (RESMEM) allocated for this program
6	1	E.FCNT	S	- FOR/NEXT count
7	1	E.GCNT	S	- GOSUB count
8	2	E.EREX	A	- the address of code to be executed upon an ON ERROR
10	2	E.ERPC	A	- the Kangaroo PC after an ON ERROR
12	2	E.ROMN	C	- ROM number of mother program
14	8	E.MOM	C	- name of the mother program
22	2	E.RTN	S	- relative address for R10 for next startup of program
24	2	E.PCR	S	- relative address of PCR for next startup of program
26	1	E.STAT	S	- the current value of R16 (the run mode of the program)
27	1	E.DATA	S	- location in current DATA line
28	2	E.DATL	S	- pointer to current DATA line

mod(ify) legend:
- A - always kept up to date
- C - established at creation and constant (exception is calc program)
- S - saved in SAVENV

Note: ECBLEN is equated to the length of the environment control block (ECB): which is 30 bytes.

Relevant pointers are:

NXTMEM - the top of the environment stack. This is one byte above where the next environment will be inserted.
FWVARS - a pointer to the ECB of the current environment to be executed.
FWUSER - a pointer to the bottom of the environment stack. This points to the ECB of the idle environment.

Environments are created by the routine CALL (not to be confused with the runtime routine CALL.) When environments are created they appear between the two divider bytes (Here is the first example of the use of what is called a divider byte. Divider bytes are aids to the memory management routines below TOS and will be discussed in detail later in the section on memory allocation.) right below NXTMEM (see figure below). The environments themselves are pushed and popped in a stack like fashion though the contents of the environments are likely to change during a running program and in the case of calculator environments even the size will change.

Since environments are variable sized objects, the push and pop operation is made more complicated. At the beginning of each environment is the size of the current environment and the size of the previous environment. This allows the environment stack to behave much like a doublely linked list.

```
                |                   |
                |                   |
      NXTMEM-> +-------------------+  <-new environment would
                |   divider byte    |       go here
                +-------------------+
                |                   |
                |      current      |
                |      running      |
                |    environment    |
                |                   |
      FWVARS-> +-------------------+
                |   divider byte    |
                +-------------------+
                |                   |
                |      calling      |
                |     program's     |
                |    environment    |
                |                   |
                +-------------------+
                |   divider byte    |
                +-------------------+
                |                   |
                |       idle        |
                |    environment    |
                |                   |
      FWUSER-> +-------------------+
                |                   |
                |                   |
```

Routines that are useful here are:

PUSHEN - called in the environmental allocator, it places an environment for a given program of a given program status (E.STAT).
ENVALO - environmentally allocates a program, that is, it examins a pointer allocated program and in turn sets up an appropriate environment for it including the setting of variables to un-

defined.
CALL - the routine generally called to pointer and environmentally allocate a program. This calls PTRALO and ENVALO. This routine should not be confused with the runtime routine "CALL."
MOVENV - moves the variable values from one environment to another. This is used to move the calculator values from calc environment to another (see theory of operation).
ADJENV - is used to change the size of the environments when the above routine is used.
POPENV - removes the top environment from the stack.
CLRENV - removes all but the bottom environment from the stack.

The bottom environment is a special environment called the idle environment and must not be removed.
INENV? - determines if an environment for a given program exists somewhere in the environment stack. This routine is important to know if the user is going to change something in that program destroying its ability to be restarted from the information in the environment.

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

The File and Directory Region

The area below FWUSER and above the system RAM is the realm of files and their directories.

Files are the largest named, manageable unit of memory. A program in Kangaroo is a file. Data, system information and text are also stored in files. If the user or implementor wants to reserve some memory for long term use they generally allocate a file.

The file area consists of a directory at the bottom which is a collection of directory entries and above it, a region of files pointed to by those entries.

When a file is created a directory entry for it is placed at the BEGINNING of the directory and the file itself is placed at the END of the file space followed by a divider byte. The end of the directory is marked by two bytes of zero. These two bytes would fall in the address field of the next directory entry but since no file will have an address of zero, this is a unique marker. (see diagram)

```
      FWUSER-> +-------------------+  high addresses
                |   divider byte    |
   next file-> +-------------------+
                |   divider byte    |
                +-------------------+
                |                   |
                |     last file     |
                |      entered      |
                |                   |  <------+
                +-------------------+         |
                |   divider byte    |         |
                +-------------------+         |
                |                   |         |
                |    second file    |         |
                |                   |  <----+ |
                +-------------------+       | |
                |   divider byte    |       | |
                +-------------------+       | |
                |                   |       | |
                |    first file     |       | |
                |                   |  <--+ | |
                +-------------------+     | | |
             /  |       00 00       |     | | |
             |  +-------------------+     | | |
             |  |    first entry    |  ---+ | |
             |  +-------------------+       | |
   directory |  |   second entry    |  -----+ |
             |  +-------------------+         |
             |  |     last file     |  -------+
             |  |       entry       |
     DIRECT-> \ +-------------------+  low addresses
```


The file may contain the information desired by the user but it is the directory that contains the information about the file. In the directory you will find the location, size, type and name of the file.

The directory entries have the following format:

offset	size	function
0	2	DR.LOC - the absolute address of the file
2	2	DR.SIZ - the (binary) size of the file
4	1	DR.TYP - the access bits for the file
5	1	DR.TNM - the (ASCII) name of the type of the file

```
6     4     DR.DAT - the date of creation
10    8     DR.NAM - the (ASCII) name of the file
```

The address of the file is absolute and therefore must change if memory is allocated below the the file. The routine ADJUST along with adjusting a small set of critical system pointers will handle this updating problem provided the system memory routines are called. Since the directory is ordered from highest address files to lowest as one preceeds 'up' through memory, the ADJUST routine proceeds in this direction also updating the absolute addresses of the files til it discovers the file in which the change was made. At this point it updates the size of the file. As a result, the file pointers and sizes remain constantly up to date. WARNING: THE SYSTEM FILE MANAGEMENT ROUTINES ARE INTERRUPTABLE AND THEREFORE FILE INFORMATION MAY NOT BE VALID AT INTERRUPT TIME. An address of zero marks the end of the directory and the remaining portion of the directory entry is not present.

The next two bytes in the directory are the size of the file EXCLUDING THE DIVIDER BYTE. File size is a binary number.

The next two bytes together are refered to as the type bytes. In the first, each bit is intended as an access permission flag or to indicate some other property of the file. The second byte is used to store the name that is to be used by the CAT program. Often options can be overridden with the 'in RAM' bit. For example, if the RAM bit is NOT set (that is the file is in ROM) and the list bit IS set then the file cannot be listed. If this file is copied into RAM, the system routines for the copy will turn on the RAM bit exposing the list bit and the file can be listed.

The following options are overridden by the RAM bit being off. Also provided is a table of useful equates for setting the access byte.

-- OPTIONS OVERRIDDEN BY THE RAM BIT BEING OFF --

edit, list, purge

-- BIT MASKS FOR ACCESS TYPE BYTE --

```
TYRAM? EQU  10000000B      in RAM (MUST be set if in RAM!)
TYROM? EQU  00000000B      in ROM
TYRUN? EQU  01000000B      test if file can be run
TYEDT? EQU  00100000B      test if file editable
TYLST? EQU  00010000B      test if file is listable
TYPUR? EQU  00001000B      test if file can be purged
TYCOP? EQU  00000100B      test if file can be copied
TYLIN? EQU  00000010B      standard lined file?
TYTOK? EQU  00000001B      test if file is token file
TYPRI? EQU  00110100B      test if private
TYDAT? EQU  01111110B      data file for PRINT#/READ#
TYLF1? EQU  00001100B      is it LIF type 1 file?
```

The second byte tells the system what type to present to the user as the type of the file when asked in a CAT command (the type name byte). For convenience this byte contains the ASCII letter that will be printed. If the file is a system file it WILL NOT BE PRINTED in the catalog listings. System files are designed to be unseen by the user.

The user can use the following equates to formulate their own type bytes.

-- JUST THE NAME BITS --

```
TYNSYS EQU  00000000B      name of system type
TYNAPP EQU  'A'            name of appointment type
TYNBAS EQU  'B'            name of basic type
TYNLI1 EQU  'I'            name of LIF type 1 file
TYNLEX EQU  'L'            name of LEX file type
TYNROM EQU  'R'            name of ROM file
TYNTEX EQU  'T'            name of text type
TYNVOL EQU  'V'            name of volatile file
TYN??? EQU  '?'            name of stranger type
```


-- SOME COMMON TWO BYTE BASIC FILE TYPES --

```
            < name >     <access>
                         MRELPCLT
TYCALC DAD  (TYNSYS*100H+11100010B)     filetype for calc
TYSYSM DAD  (TYNSYS*100H+00000000B)     filetype for syst
TYTEXT DAD  (TYNTEX*100H+10111110B)     filetype for text
TYBASC DAD  (TYNBAS*100H+11111110B)     filetype for BASIC
TYAPPT DAD  (TYNAPP*100H+10001100B)     filetype for appo
TYLEXF DAD  (TYNLEX*100H+00001001B)     filetype for LEX
TYLIF1 DAD  (TYNLI1*100H+10001100B)     filetype for LIF
TYROMF DAD  (TYNROM*100H+10001100B)     filetype for ROM
TYDIAG DAD  (TYNSYS*100H+00001000B)     filetype for diagnotic ROM
```

The time/date field contains the time and date encoded number as the offset from the beginning of the century in 2**-14 seconds. (see list of routines for time conversion subroutine)

The name is stored in an eight byte field in ASCII. Even though the name of the file may have been given in lowercase by the BASIC user it will be placed in uppercase by the system, left justified and blank filled in the name field. The assembler implementor, however; can create files with any eight characters.

If the filename begins with a period then the file is a volatile file. All volatile files are disposed of that are not accounted for by a ROM during the V.VOLT MANDI call at warm start. This gives user and implementor alike a simple mechanism for creation and distruction of scratch files. Note that since the volatility of the file is determined by the name alone, a file of any type can be volatile.

Another filenaming convention is used for system files (DR.TNM=0). This scheme adds security by having lowercase names for user inaccessible files. Since the filenames specified by the user are all converted to uppercase by runtime filename parsing routines, the filename can never be specified. This convention prevents a 'file already exists' error from occuring for a file name the user can type in but does not appear in a catalog.

-- A LIST OF INTERNAL FILES --

- calcprog - name of calculator program. This is a normal BASIC file whose only access security is through the lowercase name only. This file is created at coldstart and is destroyed and recreated during a CLEAR VARS.
- devfile - a lined system file that contains the assignio information. This file is created when the first assignio is done and is purged when an assignio to '' or * is done.
- iofile - lined system file with all the assign # information in it. This file is created at coldstart and is never destroyed.
- appt - this file is a file of type appointment. It is not a lined file and its format is a mystery to this author.
- timers - this lined system type file contains ON TIMER information. It is created when the first ON TIMER in a program is executed. It is destroyed when the program is deallocated in DALLAL (a system routine for deallocating all programs in memory).
- keys - a text file that contains the encoded DEF KEY information. This file exists only when at least one key currently has other than default assignment. This file though it is in lowercase is

user accessible though the special filename keyword "keys".
- workfile - a standard BASIC or TEXT file that is used as temporary workspace for the user. The workfile is never explicitly referred to by name but rather is created as the default filename for the EDIT command. The file may thereafter be referred to by virtue of the default name for most file commands being the current edit file. The workfile was created to help control the proliferation of scratch files in a highly memory restricted system.

System RAM

This is an area of 1354 bytes starting at 8000(hex) that contains important system variables such as the input buffer, file pointers, flags, the CPU's subroutine stack (R6 stack), and many, many more. Collect the whole set. No purchase necessary. This area does not move about. The fixed value DIRECT points to the first directory entry, one byte past the system RAM.

ROM Is Like RAM Except ...

Kangaroo ROM is like RAM in that each ROM contains a complete file directory and corresponding set of files, in the much same format as in RAM. (see Joey's Big Book of ROMs) ROM is not like RAM in that these files are not always searched and the incorperation of ROM files into the RAM system is as transparent as 17mm lead plate. For example there are routines for seaching just the RAM file directory, for searching the ROM directories and for searching both. (When searching both, the RAM files are searched first.) The implementor must be sure when writing his code that the routines he chooses search the desired directories. The lesson to be learned here is: when using system routines with the intent of them working for both ROM and RAM examin them carefully to see if they provide for ROM files. The routines you choose may only work in RAM.

Acquisition And Release Of Memory

Memory is acquired for one of two general areas of memory from the unused central region. As memory is requested, each region will expand to absorb more of the unused middle portion. (see memory layout diagrams).

```
+------------------+ large addresses
|                  |
|      UPPER       |
|      MEMORY      |
|                  |
+------------------+
|   |   |   |      |
|   |   |   |      |
|   v   v   v      |
|                  |
|     unused       |
|      space       |
|                  |
|   ^   ^   ^      |
|   |   |   |      |
|   |   |   |      |
|                  |
+------------------+
|                  |
|      LOWER       |
|      MEMORY      |
|                  |
+------------------+ lower addresses
```

Memory above the unused portion is aquired by the routines specified in the sections above concerned with this area. The R12 stack is generally pushed and popped like a stack, except in the case of the decompiler; therefore, outside of that case there is no need for special allocation and deallocation routines. Memory below the unused portion and under TOS is ultimately allocated by the ALLOC routine and released by DELETE. Both these routines rely on ADJUST and COPY.

The COPY routine is a general memory copy routine that will copy from lower to higher or higher to lower addresses without fear of a propagating copy. Most large sections of data are moved about with the COPY routine.

There is no fragmentation of memory on kangaroo and no garbage collection. The reason for this is that all of the allocations and releases of memory cause all the data contained in the memory above that altered address and below R12 to be moved up or down accordingly. This means if there are critical system pointers into these moved areas they must be adjusted. ADJUST is ultimately called by any memory

routine that may move memory.

The following pointers are adjusted by the routine ADJUST so that they are always valid even though memory may be moved about as the system runs. Remember this is guaranteed only as long as the system implementor uses the memory managment routines provided (e.g. DELETE, INSERT, ...) Below is a table which indicates what relation is tested in ADJUST to determine if the given system pointer is to be adjusted. (e.g. if location of adjustment is < EOFILE then EOFILE is updated.)

pointer name	adjustment relation
EOFILE	<
PRFILE	<
RNFILE	<
KLUDGE	<
R10	<=
PCR	<
FWVARS	<
FWLVAR	<
STSIZE	<
FWUSER	<
NXTMEM	always
TOS	always
R12	always

One of the inputs to many of these routines is the address at which to insert or delete information. The address is assumed to point to the first byte of where the information is to be placed or deleted. An ambiguity arises for ADJUST if this address is at the beginning of a file. Is the information inserted in the file or just below the beginning of the file? This dilemma is resolved with a DIVIDER BYTE. There is a byte called a divider byte after each file (they also appear in the environment stack). The divider byte belongs to no file! It is used simply as a convenience for insertion and deletion. Furthermore the divider byte has and can have no reliable value. It contains junk!

As an example of the use of the divider byte assume we have an address for insertion that is the divider byte at the end of the file then the insertion takes place at the end of the file. If on the other hand the insertion is made after the divider byte then it is in the following file.

The out of memory condition occurs when the distance between TOS and LAVAIL would be forced to be less than the value stored in LEEWAY. This is a guarantee that there is at least LEEWAY bytes for the R12 stack.

LEEWAY is initialized in the calculator initialization routine SETCAL. This routine is called on coldstart and during a CLEAR VARS command. The starting value is 320D+256D which is equated to LEWYSZ. The 320 is the worst case requirement for the size of the R12 stack. No machine function can generate more than 320 bytes of stuff on the R12 stack. The 256 is the maximum size of a calculator statement.

When a line in the calcprog file is edited, the size of the line is subtracted from LEWYSZ and placed into LEEWAY. By always adjusting LEEWAY by just enough for the calculator statement we guarantee that there is always room for it.

When the creation of an environment is attempted for the calculator program and it fails to find enough room, LEEWAY is reduced by the length of an empty environment (ECBLEN+1) and the creation is again attempted. This insures that the calcprog, even though possibly restricted to no variables, will be able to run.

The Birth and Death of Environments

An environment is a program's activation record. For every instance of a program, an environment can be found. The environments are stored in a stack in the area know, logically enough, as the environment stack. In this section we will be looking at the environment stack's relation to the execution of BASIC tokens.

When nothing is executing the environment stack contains only the 'idle' environment. The purpose of this environment will be made clear later.

When a program runs, an environment is created, placed on the environment stack and associated with the running file. The environment contains the BASIC variables and important system variables necessary for the BASIC program's. execution. When the program reaches its END statement the environment is removed from the stack and execution is resumed for the environment beneath. Here we uncover the 'idle' environment which, by design, stops further execution of tokens.

In the case that during the execution of a program, another program is called, the new programs environment is placed on the top of the stack covering the environment below. When the called program reaches its END statement, its environment is poped off the stack and the calling program is resumed.

Since the state of a BASIC program is stored in the environment, recursion is simply a matter of placing another environment on the stack associated with the recursive procedure.

Calculator execution is an unfortunate departure from this. When the user types in a calculator expression it is stored in the file 'calcprog' which is then run. A calculator environment is created and destroyed during this execution.

In the midst of this devastation of environments one might ask; how do the calculator variables survive? Calculator variable values are saved in the 'idle' environment which is guaranteed to always exist (exculding CLEAR VARS). There are two rules that govern the movement of the calculator variable values. When a calculator environment is created the calculator variable values are moved from the previous environment whose running program is 'calcprog'. (The program associated with the 'idle' environment is 'calcprog' so that when the values are saved in the 'idle' environment, they can be found.) The second rule is: when a calculator environment is popped off the stack, the calculator variables are moved to the calcultor environment that is highest in the environment stack. The effect of these two rules is that the calculator variables have a tendancy to 'float' in the topmost calculator environment.

Since, unlike a regular program, a calculator program can be changed while its environment is on the stack, a notation has been developed to prevent the continuation of environments that are no longer valid. These environments are called deactivated environments and are marked by having their status byte set to DACSTS (a 1 byte equate). When the environment to be recovered after the execution of an END statement is a deactivated environment, that environment is popped off, restored (see RESTEN) and then popped off and ignored. Control passes though deactivated environments as if they weren't there!

Even though an environment is deactivated; it may contain the calculator variable values while other environments are running. Deactivated environments are popped off under the same rule as metioned above for regular environments.

Memory at Coldstart and Warmstart

During the warmstart routine, WAKEUP, memory is checked to to see if any was added and checksummed to to see if any was damaged while we were asleep. If there is less memory or the checksum after sleep did not match the checksum before sleep then some data is assumed to be lost and the machine is coldstarted. If there is more memory than when we went to sleep, the upper memory area from LAVAIL to LWAMEM is copied to the new end of memory.

At coldstart all memory from the top of the R6 stack to the end of memory is zeroed, the file structure is rebuilt, and upper memory is initialized to empty. The calcprog, workfile and iofiles are then created in that order and the idle environment placed in the environment stack.

The Lined File Format

One of the most popular file attributes is the lined file attribute. These files contain numbered variable length records and are of the same form as BASIC, text and some system files. The advantage this attribute offers to the implementor is that he will find many system routines designed to manipulate these kinds of files.

As an example the timer file is a lined file. The timer numbers correspond to the line numbers, so that routines for insertion, deletion and searching by line number could be used to reduce implementation time and conserve code space.

Lined files have a ten byte header called the program control block (PCB). This block is used by BASIC files alone even though all lined files have this block. For a BASIC program the elements of the block have the following purposes:

EQU FOR OFFSET	BYTE NO.	DESCRIPTION
P.LEN	0/1	length of pgm and PCB
P.PLEN	2/3	length of var pointer area
P.CLEN	4/5	spare for use with COM
P.ELEN	6/7	length of environment area
P.SPAR	8/9	spare for use with CALL

Total length: 10 bytes equated to PCBLEN

The remainder of the file consists of records called lines. A line begins with a two byte line number in BCD and is followed by the one byte length of the remainder of the line. This means that lined files can contain no records longer than 255 bytes.

The lines, which may begin numbering with zero, must be in increasing order with no duplicates. There is a mandatory trailing line on all lined files called the end of file line. The end of file line is line number A999 and is a total of five bytes long including the line number and length. The DAD ENDLIN is a global set equal to the end of file line number and at the address BASEND in the system is stored the five byte EOF line. Even though the EOF line is never printed, it does serve two useful purposes. The first is to tell routines that loop though the lines of a file when to stop. The second is to act as an invisible END to BASIC programs since the remaining bytes of the EOF line are an END token and an EOL token for BASIC.

A sample file might look like this:


```
               +-------------------+
               |     EOF line      |
               |     line 7        |
               |     line 3        |
               |     line 2        |
               |     line 1        |
               |     line 0        |
               +-------------------+
               |      PCB          |
 beginning-->  +-------------------+
```

Note that an empty lined file must be at least 15 bytes long since both the EOF line and the PCB must exist.

File and Program Memory Routines

The user generally wishes to do more than just allocate some space. He may wish to create a file, insert a line in a lined file or replace an existing line. Many different flavors of acquisition and release routines are provided for by the system. The implementor is encouraged to look into what routines are available before implementing his own since the memory management routines supplied by the system often update critical "system pointers" as memory is moved. The implementor is also incouraged to use file types that already exist since there are many routines available for dealing with them. The concept of a lined file is particularly useful (see section on file layouts) since there are many routines already present in the mainframe to deal with them.

The following is a list of design assumptions used in construction of the system's memory routines.

1. Most of the memory routines assume BINARY MODE!
2. Most memory routines that take a size, take a signed two byte number for the size.
3. It is assumed that one can write to address 0 safely.
4. The acquisition and release of memory is controlled by the memory management routines provided for by the system.
5. It is assumed that there is only one file in RAM of a given name.
6. There is at least one file in the system.

LIST OF MAINFRAME MEMORY ROUTINES

--- memory initialization ---
BOUND - checks to see how much RAM is available to the machine. This routine is used in the wakeup code and at coldstart.
CHECK - checksums all the memory from a little above the R6 stack up to LWAMEM
SUMIT+ - checksums a region of memory
SETCAL - sets up the calculator program and creates the idle environment.
UPSET - initializes the upper region of memory.
FLINIT - initializes file system (that is the area below R12.)
Z36.56 - zeroes the memory from addresses R36 to R56-1
ZERO1- - fills an area of memory with copies of a bit pattern in R(R0)
ZROM1- - fills an area of memory with copies of the 8 bytes in R60
ZROMEM - zeroes or blanks an area of memory
SETED - sets up a new editfile. This routine is used when we want to change edit files.
SETRN - sets RNFILE and RNNAME system variables.
SETPR - sets PRFILE and PRNAME system variables.

--- raw memory acquisition ---
ROOM? & ROOM! - are there R32 bytes available in lower memory?
UROOM! - are there R32 (unsigned) bytes available in lower memory?

ALCALL - allocate all free file space.
ALLOC - allocates a specified amount of space in lower memory.
DELETE - deallocates a specified amount of space in lower memory.
FPURGE - purges a file.
INSERT inserts data into file space.

--- file creation ---
FCRALO - creates a file of a given size.
FCREAT - creates a file of length zero.
FCFNUL - creates a functionally empty program file.

--- renaming and copying files ---
FRENAM - renames a file.
FCOPY - makes a copy of a file under new name.
FCOPYG - does nearly the same thing

--- line oriented operations ---
FNDLED & FNDLIN - finds line with a given number in the edit (EDFILE) file.
FNDLPR - finds line with a given number in the parameter (PRFILE) file.
FNDLRN - finds line with a given number in the run (RNFILE) file.
FSEEK - finds line with a given number starting at a given location.
PREFND - finds preceding line in EDFILE.
PFNDPR - finds preceding line in PRFILE.
SKPLN, SKPLN, SKPLNN - gets address of next line.
IOOPEN - put a record in the iofile.
FSREPL - replaces lines in a file given target line number.
REPLIN - replaces lines in a file given target line location.
LINLEN - returns the length of a line.
DELLIN - deletes a line in a file given the address.
FDELLN - deletes a line from a given file at given line number.
DELLNS - deletes a collection of lines given by line number.
FREPLS & FREPLN - make an entry in a line file. If the file did not exist it is created.

--- file predicates ---
TYPOK? - tests the access bits.
FEMPT? - is the file empty?
PTALO? - asks if a file is pointer allocated.

--- data movement primatives ---
MOVE - moves bytes around in core without overwrite or deletion.
COPY - bytes from one place to another.
ADJUST - adjusts the directory entries and critical system pointers during any memory management routine for lower memory.
INSRTM - moves (as in the MOVE routine) data in lower memory.
REPLAC - replaces a block of memory with another.

--- RESMEM stuff ---
RSMEM- - acquire temp memory.
RSMEM= - acquire temp memory.
RELMEM - releases temp memory.

--- locate file stuff ---
FLOPEN - returns a pointer to the first line in a lined file.
FOPAC? - finds a file by name and tests the access bits.
FOPEN - finds a file by name in RAM only.
GOPEN - finds a file by name anywhere in RAM or ROM. This routine searches RAM first.
ROPEN - finds a file by name in ROM.
GETRID - get current ROM id number.

--- R12 stack routines ---
PUTREL, PUTADR, and PUTRLN - routines to help relativize addresses for

the stack.
GETADR, GETADN, and GETAD+ - to pull addresses from the stack and make then absolute if they need be.
DNR12? - test size of R12 stack.

--- filename parsing ---
FLTOFL - parse for "file TO file" syntax.
PUSHF - gets a filename parameter or null.
FILNM+, FILNM? & FILNM! - acquire filenames.

--- runtime retrieval of filenames ---
FLORDT - gets and typechecks filename or default from the R12 stack.
FLORD - gets filename or default from the R12 stack.
GETFL - gets filename from the R12 stack.
GETNAM - pops filename off R12 stack.
GETFST - gets a string of legal name characters.
FLCHR? - is a character a legal filename character.

Roo Memory Document hp pirate 7/29/1982 page 30

List of Important System Variables

The important system variables are as follows:
LEEWAY - The required distance between TOS and LAVAIL. This is the LEEWAY for the calculator program and the R12 stack.
LWAMEM - points to just above the highest location in RAM that the system has.

NXTRTN - the next available slot for a GOSUB/RETURN entry.
LAVAIL - The bottom of the RESMEM area which grows down from the top
R12 - points to the top of the R12 stack (see below)
TOS - remembers where the bottom of the current R12 stack is. (TOS stood for Top Of Stack but it is the opinion of the Roo Crew that the active end of a stack is the top.)
STSIZE - often points to the size byte of the statement being currently processed.
NXTMEM - points to the absolute bottom of the R12 stack and the absolute top of the environment stack.
FWVARS - points to the currently active environment block (see below). It also acts as the base pointer for referencing remote variables.
FWUSER - points to the bottom of environment stack which is the calculator environment. It also points to the top of the file area.
PCR - points to the current executing line in a program.
R10 - points to the next token to be executed.
EDFILE - points to the current edit file.
RNFILE - points to the current running file.
PRFILE - points to a file being processed and like the two pointers above is updated by the kangaroo allocation system. This pointer is used as a temporary working pointer.
KLUDGE - actually doesn't point to anything in particular. Now the R12 stack may contain absolute addresses (as in the case of strings) these addresses must now be adjusted but the adjust routine (ADJUST) DOES NOT go into the stack to change these (cause it can't find them). However, ONE kludge location is provided which will be adjusted as the other pointers. What does this mean to YOU, the man on the street? Lets take an example, the ON KEY statement during runtime will pop the address of the string to be equated with the key. If the key file does not exist, it is created possibly moving the string out from under the pointer that you just popped. When the file was created all of memory after the file was moved up, in the address space, to make room and all the pointers it knew were important were adjusted. This did not include the address you popped off the stack. If you were to place that address in the KLUDGE location it too would be adjusted. Yuchy but functional. What if I had two addresses you ask? You'll have to byte the bit as they say and invent your own horrible kludge. So there!

Roo Memory Document hp pirate 7/29/1982 page 31

The Kangaroo Mass Storage Driver

Seth D. Alford
July 27, 1982

HP Confidential

Mass Storage Driver
Seth D. Alford
July 27, 1982

CHAPTER 1

Introduction

The HP-75 (Kangaroo) mass memory driver allows the user to store, retrieve, catalog, rename, and purge files on a mass storage device such as the HP 82161A digital tape drive (filbert.) The user is also able to initialize and pack media in a mass storage device. Media are formatted according to LIF level 1 extension.

HP Confidential

4

Mass Storage Driver
Seth D. Alford
July 27, 1982

CHAPTER 2

Organization of the Driver

The subroutines that make up the mass storage driver for Kangaroo are hierarchical. Routines or classes of routines call lower level routines in order to accomplish the mass storage task. Three important classes of routines exist in the driver: the FL machine, the VF machine, and the lower level Kangaroo HP-IL routines. This document only discusses the FL and VF machines.

2.1 The FL Machine

The various features of the mass storage driver are executed by a group of subroutines referred to as the FL machine. These routines are called from the initialization routine (see the next chapter). Examples of routines which are classified in the FL group include the code which executes the various forms of copy, rename, catalog, purge, pack, and initialization, as well as code which finds files on the medium. These routines only perform the high level decision making involved in their tasks. Subroutines in the VF machine are called to actually access the mass storage device.

2.2 The VF Machine

2.2.1 Using the VF Machine

Various VF routines require R36-37 to be a pointer to a line in the devfile. Some of these routines assume that the corresponding device is a mass storage device or that the devfile line has been expanded so that the routines can store or retrieve information about the mass storage opera-

tion. This block of information is referred to as the RCB (Roofid Control Block). The RCB is stored in the devfile line starting at byte 7. Bytes 0-6 are used for storing information used by ASSIGNIO, DISPLAY IS, and other non-mass storage HPIL commands. The RCB is layed out according to the following diagram. Information obtained from the medium is stored in corresponding places in the devfile line.

```
                        ------------->|              -------------->|
                        ^             v              ^              v
+-------------------+ | +vvvvvvvvvvvvvvvvvvv+ | +vvvvvvvvvvvvvvvvvvv+
| VF.COD    7     1 | | | VF.RSW   22     1 | | | VF.CMP  +28     4 |
| Location of code  | | | Read, seek, write | | | Current imple-    |
| to indicate MELRDN| | +-------------------+ | | mentation bytes   |
| has setup devfile | | | VF.TTT   23     1 | | +-------------------+
| line.             | | | Tape to tape flag | | | VF.CDL   56     3 |
+-------------------+ | +-------------------+ | | Location of       |
| VF.DL     8     4 | | | VF.CDE   24    32 | | | current directory |
| Directory location| | | Current Directory | | | entry             |
+-------------------+ | | Entry             | | | (byte, rec, track)|
| VF.BIG   12     4 | | + - - - - - - - - - + | +-------------------+
| Directory Size    | | | VF.FTY  +10     2 | | | VF.2DE   59    32 |
+-------------------+ | | Type of the the   | | | Current directory |
| VF.LOC   16     3 | | | current direct-   | | | location #2, a    |
| Current Location  | | | ory entry         | | | spare dir entry   |
| (byte, rec, track)| | + - - - - - - - - - + | +-------------------+
+-------------------+ | | VF.CST  +14     4 | | | VF.RDE   91    18 |
| VF.MED   19     2 | | | Current starting  | | | Space for a roo   |
| Number of records/| | | address           | | | directory entry   |
| medium, starting  | | |                   | | +-------------------+
| the count with 0  | | + - - - - - - - - - + |
+-------------------+ | | VF.CLN  +18     4 | |
| VF.FLG   21     1 | | | Current length    | |
| Seek flag         | | |                   | |
+vvvvvvvvvvvvvvvvvvv+ | + - - - - - - - - - + |
         |            | | VF.CTC  +20     6 | |
         |            | | Time Of Creation, | |
         |            | | YYMMDDHHMMSS      | |
         |            | +vvvvvvvvvvvvvvvvvvv+ |
         v            |          V            |
         ------------>|          ------------>|
```


2.2.2 Entering the VF Machine

VFHI or VFGET/VFMM? are used to initiate communications with a mass storage device. VFHI determines whether named device exists, whether it is a mass storage device, obtains information concerning its directory start and length, and expands the devfile line corresponding to the device. VFGET/VFMM? are two routines which do the same as VFHI except for obtaining the information concerning the directory start. VFGET/VFMM? are only be used when communication with the filbert is desired and the tape is possibly not initialized or is not formatted according to LIF. An example of this case is with INITIALIZE. In other cases VFHI is used because other routines in the VF machine use the directory start and length.

2.2.3 Traversing a Directory

Several routines exist which allow traversal of a directory. VFDIR seeks the medium to the directory start and reads in the first entry. VFNXDE reads in a directory. VFEOD? determines whether the medium is at either the physical or logical end of the directory. VFLED? and VFPED? only determine whether the medium is at the logical or physical end of the directory, respectively.

2.2.4 Other Functions

Other routines in the VF machine seek, read, rewrite a directory entry, write information to the medium, monitor bytes travelling on the loop, translate LIF directory entries to Kangaroo entries and vice versa.

2.2.5 Terminating Communication

Communication with the device is termin[illegible], using either VFTERM or VFBYE. VFTERM issues a DDL7 (rewind the tape on a filbert) and falls into VFBYE. VFBYE will issue an UNT and UNL to the loop regardless of the value in KEYHIT. The cassette must be untalked or it may interfere with Kangaroo's communication with a display device.

2.3 The E Nonzero Return

SNDFRM, the routine which issues HPIL frames from Kangaroo returns with the E flag nonzero in the case of an error. When a communication failure occurs the mass storage operation should terminate. Hence many of the subroutine calls are followed by a REN (return on EN0.) The "plus" routines were created to solve this problem. These routines reside in the KRAXIT file. A plus routine, FRED+ for example, calls FRED. FRED returns with EN0 when an error occurs. FRED+ does an extra pop from the R6 stack in this case. FRED+ can be called instead of FRED, saving approximately 2 bytes per call. See the KR/XIT file for additional details.

3.1 Sizing a Medium

Media initialized under the LIF level 1 extension contain information in the volume header regarding physical attributes. In addition, in order to properly copy files to the medium, Kangaroo must know the size of the medium.

3.1.1 Pre-CMOSC Kangaroos

In pre-CMOSC versions of Kangaroo the size of the medium is obtained through a binary search for the highest addressable sector on the medium. This information is saved for future reference when copying files to the medium. The highest sector number is also used to incorrectly generate the LIF level 1 physical attributes for the volume header. The attributes are incorrect because the code assumes that tracks are always composed of 256 sectors. Other devices, such as floppy disks, have a different number of sectors/track. Other floppy disk drivers may require the correct physical attributes information to properly read the disk. This problem was fixed for the CMOSC release of Kangaroo.

3.1.2 CMOSC and Later Kangaroos

In CMOSC and later versions of Kangaroo, the physical attributes and size information is obtained using the DDT6 and DDT7 commands, respectively. These commands have been added to the protocal which Kangaroo uses the communicate with mass storage devices on HPIL. (This protocal is referred to as the Filbert protocal.) These commands are used similarly

to the DDT3 command: after the command an RFC and an SDA are sent, and the device responds by sending an appropriate stream of data frames and an end of transmission (ETO.)

3.1.2.1 How DDT6 and 7 Work

The mass storage device should respond to a DDT6 by sending the 12 bytes of physical attributes data as described in the LIF manual under "Extensions." The 12 bytes are sent as they would appear in the volume header of the medium, starting with the most significant byte of word 12. Kangaroo uses the 12 bytes returned by the DDT6 command while creating the volume header while initializing a medium.

The mass storage device should respond to a DDT7 by sending 2 bytes which contain the number of the highest addressable sector on the medium. The more significant byte is sent first. Kangaroo saves this number for future reference in determining whether a file will fit when copied onto the medium.

Since the mass storage device presumeably "knows" the format of the medium that it is using, information for the DDT6 and DDT7 commands should be sent by the device itself and NOT read from the medium.

3.1.2.2 Filbert and Kangaroo

Filbert (that is, the HP-82161A,) was put into production before the DDT6 and DDT7 commands were introduced, and responds to them by sending an ETO only. The Kangaroo mass storage driver assumes that any device which responds to a DDT6 or DDT7 with an ETO is a Filbert, and substitutes the appropriate physical attributes (2 tracks/surface, 1 surface/medium, and 256 sectors/track) and highest addressable sector (511.)

3.2 Packing a Medium

A fast-pack algorithm is used during PACK. This algorithm first packs the directory and then the files. (A slow-pack algorithm would pack 1 file at a time, updating the directory after each file.) The fast-pack algorithm requires four passes.

3.2.1 Pass 0

The directory is packed by reading in successive non-purged directory entries into a 256 byte buffer in Kangaroo. When this buffer fills it is written back out to the directory. This continues until the end of the directory is found.

3.2.2 Pass 1

Triples are used to determine where files are to be moved. Triples are stored in temporary memory (memory obtained with RESCON) and are each 6 bytes long. Each triple contains where a file is on the medium, how long it is in sectors, and where it should be. The triples are generated by traversing the directory.

3.2.3 Pass 2

The directory is updated by overwriting the location of the file with its present location with where it will go after the pack. The value from the corresponding triple is used. After this pass and before the next, the medium may be completely unuseable.

3.2.4 Pass 3

In this final pass the files are moved to their new locations according to the values from the triples.

CHAPTER 4

Invoking the Driver

The code for the mass memory driver lives in a switchable ROM, referred to as the MELROM. The code is entered either through a HANDI call or from executing a BASIC keyword.

4.1 The HANDI Call

The MELROM contains a ROM header and initialization code similar to other plug-in ROMS. The initialization code intercepts the V.FILE HANDI call. This HANDI call is generated from the operating system for CAT, PURGE, RENAME, and COPY when a device name is found in any of the filespecs used in those statements. The MELROM initialization code uses the event number and the currently executing token to generate a call to an appropriate subroutine which will catalog, purge, rename, or copy a file to, from, or between the mass storage device(s).

4.1.1 Parameter Passing

File and device names and passwords are passed as parameters in CPU registers with the HANDI call. To allow the registers to be re-used these parameters are stored in a temporary area referred to as the File Name Block (FNB). When needed these parameters are retrieved from the FNB and a pointer to the FNB is passed to those routines which require it. The FNB is layed out according to the following diagram.

```
+-------------------+
| FLDEVO   0      4 |
| source device for |
| COPY, RENAME      |
+-------------------+
| FLDEV1   4      4 |
```



```
| target device for |
| COPY, RENAME      |
```

```
+-------------------+
| FLNAM    8      8 |
| Filename for      |
| PURGE.            |
+-------------------+
| FLNAMO  16      8 |
| Source filename   |
| for RENAME, COPY  |
+ - - - - - - - - - +
| FLDEV   20      4 |
| Device for CAT    |
| and PURGE         |
+ - - - - - - - - - +
| FLDIRO  22      2 |
| Pointer to roo    |
| directory entry   |
| for source file   |
| for COPY          |
+-------------------+
| FNBR60  24      8 |
| FLNAM1            |
| Filename for CAT, |
| target filename   |
| for COPY, RENAME  |
| - - - - - - - - - |
| FLDIR1  30      2 |
| Pointer to roo    |
| directory entry   |
| for target file   |
| for COPY          |
+-------------------+
| FLPWD1  32      4 |
| Target password   |
| for COPY, RENAME  |
+-------------------+
| FLPWDO  36      4 |
| Source password   |
| for COPY, RENAME  |
+-------------------+
| FNBSW   40      2 |
| Space for switch- |
```



```
| ing R36/37 -- dev-|
| file pointers     |
```

```
+-------------------+
| FLSFLG  42      1 |
| Flag for the same |
| target and source |
| filename's.       |
+-------------------+
```


CHAPTER 5

Flowcharts

5.1 Flow of Control for MELROM

This flowchart describes the flow of control from the V.FILE HANDI call into the FL machine.

```
          * * * * *
         *initiali-*
         * zation  *
          * * * * *
              |
              v
              *
           *     *
         * Event *  N      * * * * *
        * V.FILE ?  *----->*  end    *
         *         *       * * * * *
           *     *
              *
              |
              v
   * * * * * * * * * * * * * *
   * Save registers, set up  *
   * FNB, dispatch to FL     *
   * routine using LOOKUP.   *
   * * * * * * * * * * * * * *
              |
```

```
      |         |         |          |
      v         v         v          v
   * * * *  * * * * *  * * * * *  * * * * * *
   * CAT *  * PURGE *  * COPY  *  * RENAME  *
   * * * *  * * * * *  * * * * *  * * * * * *
      |         |         |          |
      v         |         v          |
* * * * * * * * | * * * * * * * * *  |
* Catalog all  * | * Copy file to  *  |
* entries or a * | * or from RAM,  *  |
* specified file*| * or copy file  *  |
* * * * * * * * | * from medium to*  |
      |         | * medium or from*  |
      |         | * filbert to    *  |
      |         | * filbert       *  |
      |         | * * * * * * * * *  |
      |         |         |          |
      |         |         |          |
      ---------------------------------
                    |
                    v
                  * * * * *
                  *  end  *
                  * * * * *
```

HP Confidential

5.2 Flowchart for FLCATA

FLCATA is the routine which lists directory entries for an entire directory. It waits for the user to input up, down, shift-up, and shift-down keys to determine which directory entry it should display.

```
             * * * * *
             * start *
             * * * * *
                 |
                 v
         * * * * * * * * *
         * display cata- *
         * log header    *
         * line          *
         * * * * * * * * *
                 |
                 |
                 v
         * * * * * * * * *
         * seek to start *
         * of directory  *
         * * * * * * * * *
                 |
```

HP Confidential


```
                |
                v
         * * * * * * * * *
         * Find 1st non- *
         * purged direc- *
         * tory entry    *
         * further down  *
         * in the direc- *
         * tory.         *
         * Remember the  *
         * location of   *
         * the starting  *
         * address of the*
         * file.         *
         * * * * * * * * *
                |
                v
                *
              *   *
            * end of*  Y        * * * * *
           * directory?*----->*  stop   *<---|
            *       *           * * * * *     |
              *   *                           |
                *                             |
                | N                           |
|------->|<-----------------------------    ----
|        v                              |    |
|        *               *              |    |
|      *   *           *Is *            |    |
|N  *has a  *  Y    *it an  *  N        |    |
---* key been *--->*up or down *-----        |
     * hit?  *        * key?  *              |
       *   *            *   *                |
         *                *                  |
                          | Y                |
```

HP Confidential


```
* * * * * * * * *         |
* Dispatch to an*         |
* appropriate   *         |
* routine which *         |
* finds an entry*         |
* in the direc- *         |
* tory based on *         |
* up, down,     *         |
* shift-up,     *         |
* or shift-down.*         |
* * * * * * * * *         |
        |                 |
        v                 |
        *                 |
      *Is *               |
    * this a*  N          |
  *different  *--------->|
    * entry *             |
      * ? *               |
        *                 |
        | Y               |
        v                 |
* * * * * * * * *         |
* Display this  *         |
* entry         *         |
* * * * * * * * *         |
        |                 |
        |------------------
```

HP Confidential

5.3 Flowchart for FLNEW

FLNEW finds space for and creates new directory entries on the medium. FLNEW is called by the different routines which copy files to a mass storage medium.

```
              * * * * *
             * start   *
              * * * * *
                  |
                  v
         * * * * * * * * * * * * * * * *
         * Seek to the directory       *
         * start.  Initialize:         *
         * P--purged file count        *
         * D--first purged entry       *
         *     found                   *
         * F--free space pointer       *
         * C--current directory        *
         *     pointer                 *
         * * * * * * * * * * * * * * * *
                  |
  |-------------->|
  |               |
  |               v
  |               *
  |             *   *
  |           * end of*  Y
  |          * directory *-----------------------------|
  |           *   ?   *                                |
  |             *   *                                  |
  |               *                                    |
  |               | N                                  |
```

HP Confidential

21

Mass Storage Driver
Seth D. Alford
July 27, 1982

```
  |               |                                    |
  |               v                                    v
```

```
  |               *                          * * * * * * * * *
  |             *   *                        * File has to   *
  |          *current*   Y                   * fit between   *
  |        *file purged*---------|           * F and end of  *
  |          *   ?   *           |           * medium        *
  |             *   *            |           * * * * * * * * *
  |               *              |                   |
  |               |N             |                   |
  |               |              v                   v
  |               |              *                   *
  |               |            *   *               *   *
  |               |          *       *  N        * will  *  N
  |               |        * P = 0 ?  *---|     * it fit ? *---|
  |               |          *       *    |      *       *     |
  |               |            *   *      |        *   *       |
  |               |              *        |          *         |
  |               |              | Y      |          | Y       |
  |               |              v        |          v         |
  |               |      * * * * * * * * *|  * * * * * * * * * |
  |               |      * Re-initialize *|  * Write the     * |
  |               |      * D.            *|  * entry, and add* |
  |               |      * * * * * * * * *|  * a new end of  * |
  |               |              |        |  * directory     * |
  |               |              |<-------|  * mark.         * |
  |               |              |           * * * * * * * * * |
  |               |              v                   |         |
  |               |      * * * * * * * * *           |         v
  |               |      * increment P,  *           |  * * * * * * *
  |               |      * get next entry*           |  * generate a
  |               |      * * * * * * * * *           |  * medium full
  |               |              |                   |  * error
  |<--------------|  ------------|                   |  * * * * * * *
  |               |                                  |         |
  |               v                                  |-------->|
  |               *                                            |
  |             *   *                                          |
  |           *       *  Y                                     |
  |         * P = 0?   *------->|<---------|                   |
  |           *       *         |          |                   |
  |             *   *           |          |                   |
  |               *             |          |                   |
```

HP Confidential

22

Mass Storage Driver
Seth D. Alford
July 27, 1982

```
  |               | N           |          |                   |
                                v
```

```
  |               |        * * * * * * * * *  |              |
  |               |        * Update F, P:=0*  |              |
  |               |        * get next entry*  |              |
  |               |        * * * * * * * * *  |              |
  |               |                |          |              |
  |<--------------   --------------|          |              |
                  |                           |              |
                  |                           |              |
                  v                           |              |
          * * * * * * * * *                   |              |
          * Just skipped  *                   |              |
          * over a hole   *                   |              |
          * * * * * * * * *                   |              |
                  |                           |              |
                  v                           |              |
                  *                           |              |
                *   *                         |              |
              * will  *  N                    |              |
            *file fit in*--------------------|               |
              * hole? *                                      |
                *   *                                        |
                  *                                          |
                  |                                          |
                  v                                          |
          * * * * * * * * *                                  |
          * Seek to D and *                                  |
          * write an      *                                  |
          * entry         *                                  |
          * * * * * * * * *                                  |
                  |                                          |
                  v                                          |
              * * * * *                                      |
             *   end   *<------------------------------------|
              * * * * *
```

HP Confidential

23

Mass Storage Driver
Seth D. Alford
July 27, 1982

CHAPTER 6

New File Types

The Kangaroo mass memory driver is able to copy all file types created by the mainframe to and from mass storage. New file types created by applications programs can also be copied if the application programs respond to the V.LFTY and V.RFTY HANDI calls. See the HANDI call documentation for further information.

HP Confidential

24

Mass Storage Driver
Seth D. Alford
July 27, 1982

CHAPTER 7

Conclusions

Several improvements should be kept in mind for the mass memory driver written for future products.

7.1 Centralized Copy

A better method of copying files needs to be used. I would recomend a scheme where tables are maintained which contain names or addresses of setup, transfer, and cleanup routines for both source and target devices. This allows for copying between various source and target devices. If this scheme had been used in Kangaroo, for example, files could be copied directly from card to cassette, cassette to printer, and so forth.

7.2 Buffering Medium References

Remembering where a file is on the medium would decrease medium wear and time requirements. With this scheme the last few directory entries referenced would be maintained in a cache in RAM. Before referencing the medium directory the cache would be searched first. If a directory entry had to be updated then it would be rewritten to the medium. Otherwise only the file would be rewritten. The 41C uses a similar scheme employing buffer 1 of the filbert. This method is not available to Kangaroo since the mass memory driver uses buffer 1 for moving files during some copy operations.

Mass Storage Driver
Seth D. Alford
July 27, 1982

CHAPTER 8

References

1. LIF Standard (HP doc # A-5955-6529-1, available from Gordon Nutall at Greeley Division)
2. HP-IL manual (available from Steve Harper at CVD)
3. Kangaroo owner's manual
4. HP 82161A Digital Cassette Drive Owner's manual (HP part 82161-90002)
5. HP-IL Driver for Kangaroo document
6. Joey's book of ROMS
7. Kangaroo MEM document
8. HANDI call documentation

Table of Contents

The Kangaroo Output Software
Jack Applin IV
10:27 July 8, 1982

Overview of Kangaroo output

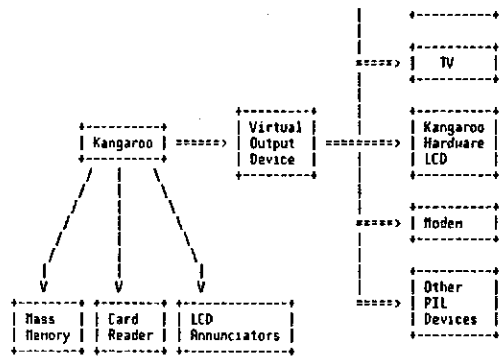
In this paper, Kangaroo output will be divided into several sections.

1) General output scheme
2) High-level output
3) Low-level output

Most users will simply use high-level output. Few users have to do low-level or device-dependent stuff.

Overall output structure

```
                +-----------+
      =====>    | Printer   |
```



The Kangaroo attempts to deal with only one sort of idealized virtual output device. It is then up to the individual device drivers to translate this idealized output to whatever the device hardware wants.

Other devices exist which bypass this scheme. The card reader is spoken to directly, as are the LCD annunciators, and mass memory devices.

General output scheme

Most top-level output is directed to print or display devices. These devices have been previously set up with DISPLAY IS or PRINTER IS statements and stored away in the device file (DEVFILE). When a string is output, it is directed to the print or display devices. The device file is scanned to find out what physical devices they are and the output is routed to them.

The output from the operating system for the most part goes to the display devices. DISP output, LIST output, ASSIGNIO prompts, INPUT prompts, TIME and APPT mode output, the EDIT mode prompt, and key echoing all go to the display devices.

The output from PRINT and PLIST statements goes to the printer devices.

High-level output overview

At the highest level in Kangaroo, all output is directed towards the PRINTER IS or DISPLAY IS devices. Here is a typical sequence to write out a string to the PRINTER IS devices:

```
      PRINT "Howdy!"

      -or-

      JSB  =PRINT.          set output stream to print devices
      LDM  R36,=STRLEN      get length of the string
      LDM  R26,=STRADR      get address of the string
      LDB  R20,=1           don't go to next print zone

      JSB  =PRNFMT          write number to current devices
       *                    (which are the print devices)
      JSB  =LINEND          write EOL sequence to current devices
      RTN

STRLEN DAD  6
STRADR ASC  'Howdy!'
```

The general sequence is:

1) Set up the output route by calling PRINT. or DISP. to send output to the PRINTER IS or DISPLAY IS devices.

2) Call PRNFMT or PRNUM. or PRSTR. to write data to the current output stream. Repeat as needed.

3) Call LINEND to write out the end-of-line sequence. Note that the current route is _s_t_i_l_l_ set and that further calls to PRNFMT and such can be made without another call to PRINT. or DISP.

High-level output routines

```
DISP.   Set output to display devices
PRINT.  Set output to print devices
LINEND  Write EOL to DISP/PRINT devices
PRNFMT  Write a string to DISP/PRINT devices
PRNUM.  Write a number to DISP/PRINT devices
PRSTR.  Write a string to DISP/PRINT devices
```

DISP.: Send future output to the DISPLAY IS devices by setting the global ROUTE.

PRINT.: Send future output to the PRINTER IS devices by setting the global ROUTE.

LINEND: Send end-of-line sequence. The end-of-line sequence for DISPLAY IS devices is CR/LF. The end-of-line sequence for PRINTER IS devices is initially CR/LF, but may be changed by the ENDLINE statement.

PRNFMT: Send a string, pointed to by R26 with length in R36. Based on R20, PRNFMT will tab to the next print zone or not. This implements the semicolon/comma terminator on PRINT/DISP statements.

PRNUM.: Send a number from the R12 stack.

PRSTR.: Send a string from the R12 stack.

Low-level display output

The low-level display output routines all write to the display devices, of which the LCD is always included. The display devices are assumed to understand several control characters and the "HP standard" escape sequences.

```
EROUT-  Write a message, put it into the error buffer
HLFLIN  Write a line with no EOL
HLFOUT  Send a string of characters (no EOL)
MSGOUT  Write a string of characters and EOL
OUTEOL  Write out CR/LF
EOLND   Send end-of-line with no delay
OUTSTR  Write a line with EOL
```

EROUT-: Send a string to the display devices and put that string into the error buffer ERRBUF. The string can then be retrieved by CNTL-FETCH.

HLFLIN: Send a string (pointed to by R26 with length in R36) to the display devices. N_o_ end-of-line sequence is generated.

HLFOUT: Send a string (R26 points to length byte followed by string) to the display devices. N_o_ end-of-line sequence is generated.

MSGOUT: Like HLFOUT, except OUTEOL is called to generate the end-of-line sequence.

OUTEOL: Send CR/LF to the display devices.

EOLND: Like OUTEOL except writes with DELAY 0.

OUTSTR: Like HLFLIN, except OUTEOL is called to generate the end-of-line sequence.

Low-level character display routines

CURSE+ Turn on the cursor

CURSE- Turn off the cursor
OUT1CH Write out a character, static parameter
OUTC40 Write out the character in R40
OUTCHR Write out the character in R32
OUTESC Write out ESC and the character in R32

CURSE+: Turn on the cursor by sending ESC > to display devices.

CURSE-: Turn off the cursor by sending ESC < to display devices.

OUT1CH: Write out a character to display devices. The character is a static parameter placed after the call to OUT1CH.

OUTC40: Write out the character in R40 to display devices.

OUTCHR: Write out the character in R32 to display devices.

OUTESC: like OUT1CH, except ESC is written before the character. This is useful for writing escape sequences.

Other documents

KR"LCD by Jack Applin IV This explains the LCD driver software

KR"IN by Jack Applin IV This explains the Kangaroo input software

PARSER

Gary K. Cutler

2:19 PM THU., 15 JULY, 1982

THE PARSER	CHAPTER 1

1.1 INTRODUCTION

As a line of program or calculator mode statement is entered, it is in ASCII code. When <RTN> is pressed, the line is parsed. The PARSER controls the conversion of ASCII code into the internal 'tokenized' form in which programs are run and stored.

As a line is parsed, it is checked for syntax errors, changed to RPN from its algebaic form, and converted into executable tokens which are then stored. Each token consists of a single byte, and can represent a single keyword, such as LET, FOR, or DISP. Token B4 (external ROM token) is used to allow extensions of the system by means of extenal ROMs and LEX files.

The system PARSER is comprised of three main functions.

1) Scanning: the process of translating ASCII code into internal tokens

2) Parsing: manipulation of tokens into an executable stream

3) Editing: inserting the token stream into the appropriate location in memory

1.2 SCANNING

SCAN: The routine SCAN has the responsibility of differentiating between numbers, variables and BASIC keywords. If SCAN makes the determination that the current collection of ASCII code is neither a number nor a variable then the routine SALT is called. SALT has the responsibility of matching the ASCII code in the input buffer with its image in the collection of ASCII tables. The search through the ASCII tables is completed by SALT, whichs polls, in order, LEX files, externals ROMs and if unsuccessful, proceeds to the system ROMs. It is important to note that SALT does not have the capability to recognize the end of a keyword that appears in the input buffer.

Ex; If ABS is a keyword in the mainframe and an external ROM creates a keyword AB, SALT will always match both ABS and AB with 'AB' in the ext ROMs ASCII table. Thus the statement

10 X=ABS(Y*Z)

will probably not parse as long as the external ROM is plugged in.

Each entry in an ASCII table has its last character flipped or negated. This signifies the end of a keyword. When the last character is reached, SALT determines if it has a match. If so SALT returns to SCAN with the appropriate token value in R14, otherwise SALT increments the token value and continues searching that particular ASCII table until the end of table is reached. At the conclusion of each ASCII table is the value FF. This informs SALT that it has reached the end of this ASCII table and searching should continue in another ROM. The final output from scanning is a token value in R14.

CAUTION: At the conclusion of SCAN, the integrity of the following registers should be maintained until the next call to SCAN.

R40:	The first character of the current keyword
R41:	Low order byte of the ROM #
R42:	High order byte of the ROM #
R43:	Token value if external ROM token
R44-R45:	Name if variable
R44-R46:	Value if integer
R46:	Secondary attributes if function
R47:	Primary attributes

1.3 PARSING

PARSIT: The first keyword (tokenized) generates the parsing scheme, through the routine PARSIT. To determine the location of the parsetime routine, PARSIT doubles the token value and adds in this offset to the PARSE table. A direct load and

indexed subroutine jump initiates the desired routine.

There are four conditional tests upon the initial keyword in PARSIT, before any parsing routine is entered. These tests involve the primary attribute of the initial token.

1) If the first token is a variable then the variable token is replaced with the implied LET token and the primary attribute is given a value of 200 (octal).

2) If the THEN flag is set (during the parsing of an IF-THEN statement the PARSER considers the hypothesis and conclusion as two seperate statements) then the primary attribute of the token is tested to ensure a legal after THEN status (< 300 octal).

3) The primary attribute is tested for value greater than or equal to 200 (octal). If so we have a programmable keyword and parsing initiates. If not:

4) The primary attribute is tested for value greater than or equal to 100 (octal).

TRUE: Then we have a BASIC system call and the machine must be in calculator mode or we error.

FALSE: PARSIT stops and sets an error (regard expression parsing for primary attributes less than 100 octal).

Once these conditionals are passed we enter the appropriate parsing routine.

1.4 EDITING

EDITIT: Each token stream, whether in program or calculator mode, is preceded by three bytes of information. A two byte BCD line number (0 if in calc mode) followed by a one byte value representing the length of the token stream. The routine EDITIT calculates the size of the token stream and uses the BCD line number to insert the line from the R12 stack into the appropriate location of the EDIT FILE.

Note: The implied length of a token stream must be less than

or equal to 255 bytes (not including line number and statement size byte).


```
***********************************************************
*                                                         *
*  Some examples of statements and how they are           *
*  interpreted by the parser can most easily              *
*  be shown by these two tables.                          *
*  NOTE: order of attmpted parse is                       *
*                                                         *
*          1. program statements                          *
*          2. calculator statements                       *
*          3. calculator expressions                      *
*          4. program expressions                         *
*                                                         *
*  when editing a BASIC file:                             *
*                                                         *
*       program              calculator                   *
*                                                         *
*  +-----------------+-----------------+                  *
*  |                 |                 |                  *
*  |   10 X          |    X            |                  *
*  |   10X+1         |    10E+1        | expresions       *
*  |               <-|--- 10+2         |                  *
*  |                 |    (X=5)   /|\  |                  *
*  |                 |             |   |                  *
*  +-----------------+-----------------+                  *
*  |                 |             |   |                  *
*  |  10 BEEP        |    X=5      |   |                  *
*  |  20<null>   ----|->  BEEP         | statements       *
*  |                 |    <null>       |                  *
*  |                 |                 |                  *
*  +-----------------+-----------------+                  *
*                                                         *
*  The <null> statement will delete the line# given.      *
*  20<null> will delete statement 20.                     *
*  Just <null> deletes line 0 in the calculator file.     *
*                                                         *
***********************************************************
```

1.5 STATEMENT vs. EXPRESSION PARSING

There exists four states of allowable basic form in the machine

1) Program statements

2) Calculator statements

3) Calculator expressions

4) Program expressions

Program and calculator statements are subject to the previous discussions on parsing procedures. Program and calculator expression parsing is nothing more than inserting or forcing the initial token to be the DISPLAY token and treating this whole entity as a program or calculator statement.

```
Ex:  user        10 x*y/z
     after parsing and decompiling line 10 is
                 10 DISP X*Y/Z
```

1.6 GLOBALS

Name	Location	Description
EDNAME	826B	name of current edit file
ERLINN	8378	line # of which error occurred
ERRR10	836C	loc of R10 at error
ERRTMP	836E	temporary location for error information
INPBUF	8180	location of ASCII code to be parsed
PERRSV	851B	save area for parser error information
PRNAME	8263	name of file to be parsed
PROTEM	8233	temporary memory
R6LIM1	8120	limit on the R6 stack depth
RM.PAR	0006	relative offset to parse table
ROMOFF	82A5	offset to make ROMPTR absolute
ROMPTR	82A3	relative pointer to current ROM enabled
RTNFLG	8365	invisible return flag
SAVR10	8368	loc last error that occurred
STSIZE	8255	location of beginning of line
THENFL	8362	then flag
TOS	8257	current top of stack (R12)

1.7 HANDI CALL EVENTS

V.PAR -- sent upon initiation of PARSER
V.STA -- sent for string arrays
V.DIM -- sent to intercept the DIM statement
V.PARA -- sent to intercept function parameters

1.8 CROSS REFERENCES

Handi Call Document	RH"HDI
Internal Code Examples Document	RH"ICE
Source File	KR&PAR
Global File	KR&GLO

1.9 FLOW DIAGRAMS

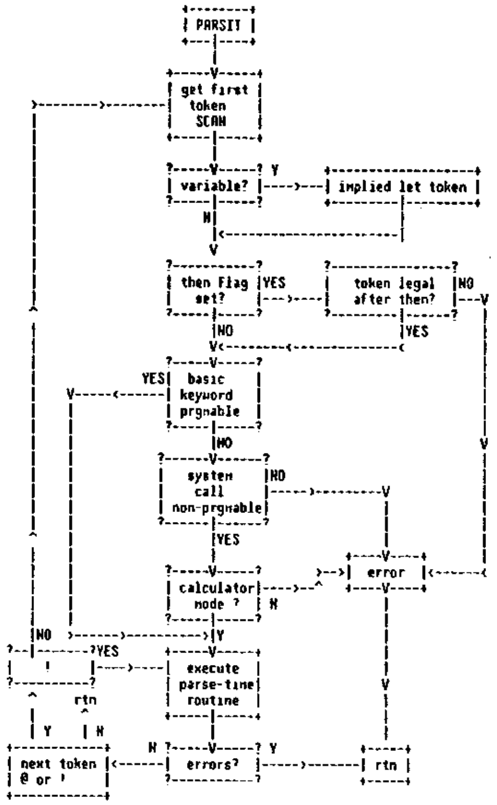
THE PARSER

```
+-------------+
|   PARSER    |
+------|------+
       |
       V
?-------------?NO              +-------------+
|is there a   |-->-->-->-------|    calc     |
|line number? |      ^         |  statement  |
?------|------?      |         |   STPARS    |
       |YES          |         +------|------+
       |             |                |
       V             |                V
+-------------+      |     NO?-------------?
| parse prgm  |      |  V---|   errors?   |
| statement   |      |  |   ?------|------?
|  STPARS     |      |  |          |YES
+------|------+      |  |          V
       |             |  |   +-------------+
       V             |  |   |    calc     |
?-------------? YES  |  |   | expression  |
|   errors?   |-->---^  |   |   EXPARS    |
?------|------?         |   +------|------+
       |NO              |          |
       |                |          V
       V                V NO?-------------?
+-------------+             |   errors?   |
|  edit in    |----<--<-----|             |
|  the line   |    |        ?------|------?
|  EDITIT     |    |               |YES
+------|------+    |               V
       |           |        +-------------+
       V           |        |    prgm     |
+-------------+    |        | expression  |
|    rtn      |    |        |   EXPARS    |
+-----^-------+    |        +------|------+
      |            |               |
      |            |               V
      |            |   NO  ?-------------?
      ^            ^-<-----|   errors?   |
+-----|-------+            ?------|------?
|   report    |<------------------<YES
|   error     |
+-------------+
```

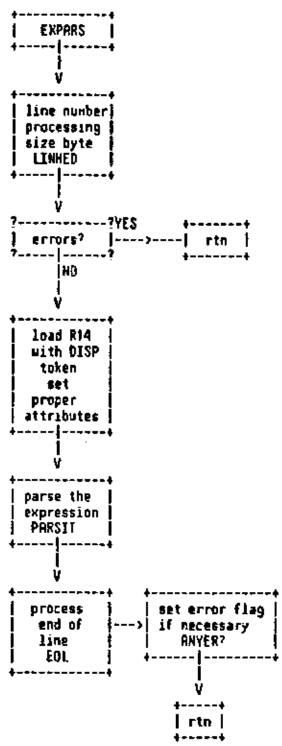

STATEMENT PARSING

```
+-------------+
|   STPARS    |
+------|------+
       |
       |
       V
+-------------+
|process line |
|number and   |
|size byte    |
| LINHED      |
+------|------+
       |
       |
       V
?-------------?YES      +--------+
|   errors?   |---->----|  rtn   |
?------|------?         +--------+
       |NO
       |
+------V------+
| parse the   |
| statement   |
|  PARSIT     |
+------|------+
       |
       |
+------V------+
|   process   |
| end of line |
|     EOL     |
+------|------+
       |
       |
       V
+-------------+
| set error   |
|  flag if    |
| necessary   |
|   ANYER?    |
+------|------+
       |
       |
+------V------+
|     rtn     |
+-------------+
```

THE PARSIT ROUTINE



EXPRESSION PARSING



EXAMPLES A	CHAPTER 2

ACTION OF THE PARSER ON THE PROGRAM STATEMENT

10 A=X*(Z+SIN(Y))^2/T

[It is implied that all entities which are examined, manipulated, and pushed onto the R12 stack are the respective tokens of the characters, operators, and functions.]

To parse the preceding program statement the parser will first attempt to find a line number. If successful, control will be passed to the statement parser, STPARS which will place the line number and a blank byte for the statement size on the R12 stack. The first character of the statement is then examined and in this case, since it is a variable, control is then passed to the IMPLIED LET routine.

LET places the let token and the token for A on the R12 stack. The next character encountered is = which initiates the following progression of subroutine jumps.

NUMVA+ --> LOGFAC --> LOGPRIM --> NUMEXP -->
TERM --> FACTOR --> PRIMARY

PRIMARY is the routine which examines the next character and directs control to the appropriate routine for parsing. In the given case the next character is X, since it is a numeric variable its token is placed on the stack and * becomes the current character. Control now proceeds from PRIMARY backwards through the progression, stopping at each routine and testing the attributes of the current character.

FACTOR: tests for ^
TERM: tests for *,/
NUMEXP: tests for +,-,monadic -
LOGPRIM: tests for binary relation operator
LOGFAC: tests for logical and
NUMVA+: tests for logical or

* passes the test at TERM so * is stored in R33 and (becomes the current character. When a test is passed, control

is directed forward through the progression to PRIMARY.

PRIMARY then directs control to the appropriate routine for (. Since (implies a nested expression, Z becomes the current character and is immediately placed on the R12 stack. + is now current, control is digressed, and + passes at NUMEXP. Before + is stored, however, the occurrence of (causes * and the remainder of R32-37 to be saved on the R6 stack. + is then stored in R34 making the next character SIN. Control passes to PRIMARY which directs control to the system functions routine, FUN1. FUN1 then pushes R32-35,R40-47 onto the R6 stack and stores SIN in R35. FUN1 then examines the next character and checks for a (. If a (is found this signals FUN1 to obtain the next character(Y). If Y is acceptable as a parameter for SIN, FUN1 places Y on the R12 stack and shifts SIN to R36. R32-35 and R40-47 are restored and SIN is pushed to the R12 stack. FUN1 then tests the next character for). If true, the) is disregarded and the next character obtained. Control is then returned to PRIMARY with the second) as the current character. Since PRIMARY sees the), + is placed on the R12 stack thus ending the nested expression and making ^ the current character. ^ attributes are checked and since they are greater than the attributes of the saved *, ^ is processed first while * remains on the R6 stack. ^ passes the test in FACTOR. The ^ is stored in R32 and the next character, 2, is examined in PRIMARY. Since 2 is acceptable, first 2 then ^ are placed on the R12 stack, and / becomes current. The attributes of / are checked and found equal to the attributes of the saved *, the / is processed first for convenience. / attributes pass the test in TERM.
is stored in R33 and T becomes current. Control goes to PRIMARY which examines T. Being acceptable, T then / are placed on the R12 stack. Finally since there are no more operable characters * is popped off the R6 stack and pushed on the R12 stack.

R12 STACK

10	low order byte of line number: 10
0	high order byte of line number: 0
18	program statement size: 18 bytes
91 (let)	let token: 91

11 (A)	simple numeric address
20	ASCII code for blank A
41	
1 (X)	fetch numeric value: X
20	ASCII code for blank X
58	
1 (Z)	fetch numeric value: Z
20	ASCII code for blank Z
5A	
1 (Y)	fetch numeric value: Y
20	ASCII code for blank Y
59	
D8 (SIN)	SIN token
2B (+)	ADD token
1A	integer constant token
2	three byte
0	integer: 2 , in

2:19 PM THU., 15 JULY, 1982

	0	BCD mode
	30 (^)	exponentiate token
	1 (T)	fetch numeric value
	20	ASCII code for blank T
	54	
	2F (/)	divide token
	2A (*)	multiply token
	8	store numeric value token
	E	statement end token
PTR -->		next byte available on R12 stack

Parsed statement: [10] A X Z Y [SIN] + 2 ^ T / * [STORE]

2:19 PM THU., 15 JULY, 1982

ACTION OF THE PARSER ON THE USER DEFINED FUNCTION

```
10  DEF FNR=SQR(X^2+Y^2)
```

The parser attempts to locate a line number. If successful, control is passed to the statement parser, STPARS (note: if DEF is located in a calculator statement an error will be generated). STPARS then places the line number and a null byte for the statement size on the R12 stack. STPARS then examines the next character (DEF) and passes control to the user definded function routine . DEF FN places the DEF token on the stack and passes control to the function name routine, FNNAME. FNNAME pushes the name 'R' on the R12 stack followed by two null bytes which reserve the position for the relative jump past the fn end. FNNAME then pushes the parameter type/count (one byte value giving the type: numeric or string, and the count: number of parameters).

Note: User defined functions cannot be longer than 65,535 bytes in definition.

Note: It is ASSUMED (given the constraints of the input buffer) that the user cannot assign more than 43 [sic] parameters. The maximum number of parameters that may be handled is 128.

```
|_______|  n...n (bit#1-bit#7) param count in binary
|nnnnnnnt|
|_______|      t (bit#0)  type:  0 for numeric
                                 1 for string
```

EXAMPLE: FNX4(a,b,c,d) = numeric expression

param type/count byte: |00001000| numeric, 4 parameters

EXAMPLE: FNX$(A$,F$,C$) = string expression

param type/count byte: |00000111| string, 3 parameters

FNR is a numeric function with no parameters, thus the

2:19 PM THU., 15 JULY, 1982

param type/count byte is zero. Control is now returned to DEF FN. DEF FN now examines the parameter count and the type. If there were one or more parameters DEF FN would push each parameter name followed by two null bytes (holders for variable value ptr.) onto the R12 stack. In the current case, however, no parameters exist. DEF FN then places two null bytes onto the stack, reserving space for the relative PCR.

DEF FN then examines the current character (=). Since this implies an in-line definition control is passed to NUMVA+ for a numeric expression, or STREX+ for a string expression. In the current example, NUMVA+ places, in order, the following on the R12 stack.

X 2 ^ Y 2 ^ + [SQR]

Control is returned to DEF FN which places the three bytes 00EA, 0, 0 (first byte: invisible fn end token; next two bytes: position holder for store numeric value ptr) and the carriage return on the stack.

R12 STACK

20	low order byte of line number: 20
0	high order byte of line number: 0
1E	statement size
87	DEF FN token
20	ASCII code for blank R
52	
0	position of relative jump past fn end
0	
0	parameter type/count
0	

2:19 PM THU., 15 JULY, 1982

----	position of relative PCR
0	
1 (X)	fetch numeric value
20	ASCII code for blank X
58	
1A	integer constant
2	three byte integer
0	constant: 2
0	in BCD mode
30 (^)	exponentiate token
1 (Y)	fetch numeric value
20	ASCII code for blank Y
59	
1A	integer constant
2	three byte integer
0	constant: 2
0	in BCD mode
30 (^)	exponentiate token

	2B (+)	add token
	B0	square root token
	00EA	invisible fn end token
	0	position of store variable ptr.
	0	
	E	statement end token
PTR --->		next available byte on R12 stack

Parsed statement: [20] [DEF FN] R X 2 ^ Y 2 ^ + [SQR]

ACTION OF THE PARSER ON THE PROGRAM STATEMENT

30 IF L$=CHR$(7*5) THEN 180 ELSE L$='='

Following action upon the line number, we start the discussion at the IF parsing routine. The IF token initiates the following subroutine jumps.

NUMVA+ (gets next token L$) --> LOGFAC --> LOGPRIM (sees string variable token 3) --> STREXP

STREXP, string expression parser, passes control to SOURCE, which differentiates between, string constants, string variables and string functions. In this case SOURCE recognizes L$ as a string variable thus sending control to STVRBL. STVRBL pushes L$ onto the R12 stack and obtains the next character (=). Control returns to LOGPRIM which checks the attributes of =. A match occurs, which causes = to be saved on the R6 stack and control passes back to STREX+ (adds scan to STREXP). The next character (CHR$) is obtained and control passes to SOURCE. SOURCE catagorizes CHR$ as a string function and thus directs control to FUN1.

FUN1 saves registers 32-35,40-47 on the R6 stack and stores CHR$ in R35. The next character is checked for (, and if so, the first (and in our case the only) parameter type is obtained. Since CHR$ is a function of one numeric parameter, control is passed to NUMVA+. NUMVA+ parses the numeric expression (see above) and places 7, 5, and * on the R12 stack. Control is returned to FUN1, which pushes CHR$ on the stack, restores registers 32-35,40-47 and checks the next character for). If) is found, control is directed back to LOGPRIM through STREXP. LOGPRIM restores the saved =, pushes it to the R12 stack and returns to IF with the next character (THEN). IF seeing the THEN passes control to DIGIT which searches for a digit. In this case DIGIT confirms that the next character is a digit (1) and returns control to IF. IF pushes the jump true token, then pushes the line number 180 and gets the next character, ELSE. The appearance of ELSE initiates a jump to DIGIT. If a digit is found control returns; if not, as in our case, control passes to PARSIT (in this case a dummy jump relative 0 token is placed on to the stack). PARSIT sees the following;

LET L$ = "="

Thus, following many of the progressions as above the expression is parsed as follows. The let token and L$ are placed on the R12 stack. The = initiates the jumps to SOURCE. SOURCE recognizes the next character STRCON saves the " in R2, pushes a byte to the R12 stack to reserve space for the string

length and gets the first character of the string (=). This is pushed on to the stack and the next character (") is recognized as the correct delimiter. The string length is calculated and this value is inserted into the reserved byte on the stack. Control returns to IF where the carriage return is pushed.

R12 STACK

30	low order byte of line number: 30
0	high order byte of line number: 0
1B	statement size
3 (L$)	fetch string value
24	ASCII code for L$
4C	
1A	integer constant
7	three byte integer
0	constant 7 in
0	BCD mode
1A	integer constant
5	three byte integer
0	constant 5 in
0	BCD mode

2A	multiply token
C2	CHR$ token
35	= token
18	jump true token
80	low order byte of line number: 80
01	high order byte of line number: 1
1C	jump relative token
2	number of bytes to jump starting
0	directly after jump rel token (dummy jump)
91	let token
13 (L$)	simple string address
24	ASCII code for L$
4C	
5	string constant
1	of length 1 byte
3D	ASCII code for =
7	store string value token

	E	statement end token
R12 PTR --->		next available byte on R12 stack

Table of Contents

Routine: APPTMD

File: KR/PS1

Author: MK

Description: Appointment mode.
1) Display current apointment or APPT template
2) Input and execute appointment commands
3) Maiuntain appointment file

Input:
PSSTAT--System variables affecting time/appointment functions

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
PSSTAT--Updated system status
<APPT file>--Updated appointment data
<comparator>--next appointment, if any
R25--terminator key
Routines Called: ACREAT, ERROR+ APSTAT, HANDIO, BLIMP, ATMPLT, APTDSP, GETTEM, KOPY, ANN.E-, LOOKUP, APTCHK, APTFND, DUPCHK, APTINS, NUMPCK, APTERR, APTACK, APTR-, APTR+, APPROC, APEXIT, APINFO, APTDEL.
Stack depth R6 (max):
Calls to Error routines (include error number and reason):
16--called if ACREAT, or appointment insert fails.
71--called if duplicated appointment input

Notes:
HANDI calls V.APTO, V.AKEY

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20	x	xo x
R30	xxxx	xxxx
R40		
R50		
R60		xx
R70	x	xxxx

Status:

	in	out	Legend
Mode	b		d-BCD
E			b-BIN
DRP			i-input
ARP			

ROMJSB Needed: x

HANDI Called: x

R12 stack:

Entry	Exit

Routine: APEXIT

File: KR/PS1

Author: MK

Description: Performs the necessary housekeeping tasks prior to exiting from appointment mode.
1.) Purge the APPT file if it is empty
2.) Blank the error buffer

Input:
R30/31--Start of Appointment file
R34/35--End of Appointment file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<File structure>--possibly altered
<Error buffer>--blanked

Routines Called:
HANDIO, BLEBUF, FPURGE.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
HANDI call to V.ARTN

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: X

HANDI Called: X

R12 stack:

Entry	Exit

Routine: TIMEMD

File: KR/PS1

Author: MK

Description: This subroutine executes the time mode features of Kangaroo.
1.) Display the current time and update it once per second
2.) Input and execute time mode commands.

Input:
<system variables>--Set up for current time
PSSTAT--Current display options set

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
PSSTAT--Updated display options
<Comparator>--Tables updated if necessary
<Ssyten Time Variables>--Updated if necessary

Routines Called: BLIMP, TCKTGL, TICK, GETTEM, TMECMD, CMPENT, CMPCHK.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
Uses background processing to update time display.

Reg: x = volatile

	0123	4567
R0	xxxx	
R10		
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: X

HANDI Called:

R12 stack:

Entry	Exit

Routine: CSTRIG

File: KR/PS1

Author: MK

Description:
Routine called from the comparator service routine to execute the V.CLOK HANDIO call.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E as set from HANDIO call.

Routines Called:
HANDIO

Stack depth R6 (max): ?

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode		x	d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: X

HANDI Called: X

R12 stack:

Entry	Exit

Routine: CKTRIG

File: KR/PS1

Author: MK

Description: This subroutine processes a clock interrupt which updates the clock display every second.
1.) Display the current time and date if time display not disabled.
2.) Retain the command field in columns 27 to 31
3.) Restore the cursor location in the command being entered.

Input: <BIN>
<input buffer>--Current time/date display.
PSSTAT--BitNO = 1 iff time display disabled.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<input buffer>--Updated time/date display.

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+---------+     +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |     | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+     +------------+------------+
|R0 |    |    |      |Mode|   |   | d-BCD   |     |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |     |            |            |
|R20|    |    |      |DRP |   |   | i-input |     |            |            |
|R30|    |    |      |ARP |   |   |         |     |            |            |
|R40|    |    |      +----------------------+     |            |            |
|R50|    |    |      | ROMJSB Needed:       |     |            |            |
|R60|    |    |      +----------------------+     |            |            |
|R70|    |    |      | HANDI Called:        |     |            |            |
+---+----+----+      +----------------------+     +------------+------------+
```

Routine: CKTRIG

File: KR/PS1

Author: MK

Description: This subroutine processes a clock interrupt which updates the clock display every second.
1.) Display the current time and date if the time display is not disabled.
2.) Retain the command field in columsn 27 to 31.
3.) Restore the cursor location in teh command being entered.

Input:
BIN
<input buffer>--Current time/date display
PSSTAT--BitNO = 1 iff time display disabled

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<input buffer>--Updated time/date display.

Routines Called: STDATE, FXTIME, TOASC2, UPDISP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+---------+     +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |     | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+     +------------+------------+
|R0 |  xx|    |      |Mode| b | x | d-BCD   |     |            |            |
|R10|    |    |      |E   |   | x | b-BIN   |     |            |            |
|R20|x   |    |      |DRP |   | x | i-input |     |            |            |
|R30|    |  xx|      |ARP |   | x |         |     |            |            |
|R40|xxxx|xxxx|      +----------------------+     |            |            |
|R50|    |    |      | ROMJSB Needed:  X    |     |            |            |
|R60|    |    |      +----------------------+     |            |            |
|R70|    |    |      | HANDI Called:        |     |            |            |
+---+----+----+      +----------------------+     +------------+------------+
```

Routine: AINCHK

File: KR/PS1

Author: MK

Description: This routine provides character checking of input duyring appointment template entry.
1.) Process special write protection of the note field prompt character.
2.) Update the IO status byte PSIOST as needed.

Input: BIN
R40--Character input to be checked
INPTR--Current address of the cursor
PSIOST--Current IO status

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

R40--Character to be processed as the input
PSIOST--Updated IO status byte
E-Reg--Set toe terminate IO on the key input.

Routines Called:

Stack depth R6 (max): 2

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+---------+     +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |     | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+     +------------+------------+
|R0 |   x|    |      |Mode| b |   | d-BCD   |     |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |     |            |            |
|R20|    |    |      |DRP |   | x | i-input |     |            |            |
|R30|    |    |      |ARP |   | x |         |     |            |            |
|R40|b   |  xx|      +----------------------+     |            |            |
|R50|    |    |      | ROMJSB Needed:  X    |     |            |            |
|R60|    |    |      +----------------------+     |            |            |
|R70|    |    |      | HANDI Called:        |     |            |            |
+---+----+----+      +----------------------+     +------------+------------+
```

Routine: RINCHK

File: KR/PS1

Author: MK

Description: Provides the input checking for some of the template IO done in TIME and APPT modes.
1.) Disable the insert/replace key to leave the IO in replace mode.

Input: (BIN)
R40--Input key to be checked

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R40--Input key or NOPKEY if input was I/R.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                      R12 stack:
+---+----+----+      +----+---+---+---------+     +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |     | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+     +------------+------------+
|R0 |    |    |      |Mode| i |   | d-BCD   |     |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |     |            |            |
|R20|    |    |      |DRP |   | 40| i-input |     |            |            |
|R30|    |    |      |ARP |   | x |         |     |            |            |
|R40|b   |    |      +----------------------+     |            |            |
|R50|    |    |      | ROMJSB Needed:  X    |     |            |            |
|R60|    |    |      +----------------------+     |            |            |
|R70|    |    |      | HANDI Called:        |     |            |            |
+---+----+----+      +----------------------+     +------------+------------+
```

Routine: TIMCHK

File: KR/PS1

Author: MK

Description: Provides the input checking for some of the template IO done in TIME and APPT modes.
1.) Disable the insert/replace key to leave the IO in replace mode.

Input: (BIN)
R40--Input key to be checked

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R40--Input key or NOPKEY if input was I/R.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|R0 |    |    |     |Mode| i |   | d-BCD  |    |           |           |
|R10|    |    |     |E   |   |   | b-BIN  |    |           |           |
|R20|    |    |     |DRP |   | 40| i-input|    |           |           |
|R30|    |    |     |ARP |   | x |        |    |           |           |
|R40|b   |    |     +-----------------------+    |           |           |
|R50|    |    |     | ROMJSB Needed:  X     |    |           |           |
|R60|    |    |     +-----------------------+    |           |           |
|R70|    |    |     | HANDI Called:         |    |           |           |
+---+----+----+     +-----------------------+    +-----------+-----------+
```

Routine: YINCHK

File: KR/PS1

Author: MK

Description: Provides the input checking for some of the template IO done in TIME and APPT modes.
1.) Disable the insert/replace key to leave the IO in replace mode.

Input: (BIN)
R40--Input key to be checked

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R40--Input key or NOPKEY if input was I/R.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|R0 |    |    |     |Mode| i |   | d-BCD  |    |           |           |
|R10|    |    |     |E   |   |   | b-BIN  |    |           |           |
|R20|    |    |     |DRP |   | 40| i-input|    |           |           |
|R30|    |    |     |ARP |   | x |        |    |           |           |
|R40|b   |    |     +-----------------------+    |           |           |
|R50|    |    |     | ROMJSB Needed:  X     |    |           |           |
|R60|    |    |     +-----------------------+    |           |           |
|R70|    |    |     | HANDI Called:         |    |           |           |
+---+----+----+     +-----------------------+    +-----------+-----------+
```

Routine: UPDISP

File: KR/PS1

Author: MK

Description: Outputs data from the input buffer to the display and turns on the cursor, without terminating the current input operation.
1.) Output a specified number of characters from the input buffer to the start of the display.
2.) Assert the cursor at a specified location

Input: (BIN)
R36/37--Numberr of characters to output
<input buffer>--Characters to output to display
INPTR--Address at which to set cursor

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<display>--New data from input buffer with cursor on at specified location

Routines Called:
CURSE-, OUTICH, HLFLIN, CURSE+

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |    |           |           |
|R10|    |    |     |E   |   | x | b-BIN  |    |           |           |
|R20|    |    |     |DRP |   | x | i-input|    |           |           |
|R30|    |  ii|     |ARP |   | x |        |    |           |           |
|R40|    |    |     +-----------------------+    |           |           |
|R50|    |    |     | ROMJSB Needed:  X     |    |           |           |
|R60|    |    |     +-----------------------+    |           |           |
|R70|    |    |     | HANDI Called:         |    |           |           |
+---+----+----+     +-----------------------+    +-----------+-----------+
```

Routine: STDATE

File: KR/PS1

Author: MK

Description: Sets up the day & date APPT fields in the input buffer for a date a specified number of days in the future. The information is stored in ASCII.

Input: (BIN)
R20--number of days in the future to display.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R50/57--Time/date information in BCD.
<input buffer>--Day & date information in ASCII.

Routines Called:
DCCLOK, DCDAY, FXDATE, FXDAY, GETCLK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|     |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+     +----+---+---+--------+    +-----------+-----------+
|R0 |xxxx|    |     |Mode| b |   | d-BCD  |    |           |           |
|R10|    |    |     |E   |   | x | b-BIN  |    |           |           |
|R20|xxxx|xxxx|     |DRP |   | x | i-input|    |           |           |
|R30|xxxx|xxxx|     |ARP |   | x |        |    |           |           |
|R40|xxxx|xxxx|     +-----------------------+    |           |           |
|R50|xxxx|xxxx|     | ROMJSB Needed:  X     |    |           |           |
|R60|xxxx|xxxx|     +-----------------------+    |           |           |
|R70|xxxx|xxxx|     | HANDI Called:         |    |           |           |
+---+----+----+     +-----------------------+    +-----------+-----------+
```

Routine: TICK

File: KR/PS1

Author: MK

Description: Sets up the clock comparator table entry to interrupt every second on the second.

Input: (BIN)
RTCSB--Current time (offset from time base)
TMBASE--Current time base

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<Clock entry in comparator table>--Set to interrupt every second on the second.

Routines Called:
CMPENT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | x | b-BIN   |   |            |            |
|R20|xx  |    |     |DRP |   | x | i-input |   |            |            |
|R30|    |    |     |ARP |   | x |         |   |            |            |
|R40|xxxx|xxxx|     +----------------------+   |            |            |
|R50|    |    |     | ROMJSB Needed:  X    |   |            |            |
|R60|    |    |     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
```

Routine: APINFO

File: KR/PS2

Author: MK

Description: Displays the 4 digit year and the repeat fields (if any) of the current appointment. The information is held in the display for as long as a key is held down.

Input: (BIN)
R32/33--Pointer to current appointment

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called: GETLNX, DCCLOK, TOASC2, FXDAY, TIMDIV, OUTSTR, LETGO

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |    |    |     |Mode| b | x | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | x | b-BIN   |   |            |            |
|R20|xx  |  xx|     |DRP |   | x | i-input |   |            |            |
|R30|    |  xx|     |ARP |   | x |         |   |            |            |
|R40|  xx|xxxx|     +----------------------+   |            |            |
|R50|    |    |     | ROMJSB Needed:  x    |   |            |            |
|R60|    |    |     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
```

Routine: APTDEL

File: KR/PS2

Author: MK

Description: Deletes an appointment from the APPT file after acknowledging the current appointment. Also enables the pending appointment.

Input: (BIN)
R32/33--Pointer to current appointment
R34/35--Pointer to end of APPT file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R34/35--Updated end of file pointer
<APPT file>--File shortened with deletion

Routines Called:
APTACK, DELETE, STALRM.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: See APDEL'

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | x | b-BIN   |   |            |            |
|R20|xx  |   x|     |DRP |   | x | i-input |   |            |            |
|R30|xxii|bb  |     |ARP |   | x |         |   |            |            |
|R40|    |    |     +----------------------+   |            |            |
|R50|    |    |     | ROMJSB Needed:  x    |   |            |            |
|R60|    |    |     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
```

Routine: APDEL'

File: KR/PS2

Author: MK

Description: Deletes an apontement from the APPT file without acknowledging the current appointment.

Input: (BIN)
R32/33--Pointer to current appointment
R34/35--Pointer to the end of the APPT file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R34/35--Updated end of file pointer
<APPT file>--Shortened due to deletion

Routines Called:
DELETE, STALRM

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: See APTDEL. APDEL' is an entry point within APTDEL.

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry      | Exit       |
+---+----+----+     +----+---+---+---------+   +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD   |   |            |            |
|R10|    |    |     |E   |   | x | b-BIN   |   |            |            |
|R20|xx  |    |     |DRP |   | x | i-input |   |            |            |
|R30|  ii|bb  |     |ARP |   | x |         |   |            |            |
|R40|    |    |     +----------------------+   |            |            |
|R50|    |    |     | ROMJSB Needed:  x    |   |            |            |
|R60|    |    |     +----------------------+   |            |            |
|R70|    |    |     | HANDI Called:        |   |            |            |
+---+----+----+     +----------------------+   +------------+------------+
```

Routine: APTDSP

File: KR/PS2

Author: MK

Description: Sets up the specified appointment in the input buffer ready for display (decoded ASCII format.) R20/24 are left set up as parameters for a call to GETTEM.

Input: (BIN)
R32/33--Pointer to current appointment (may not be at the end of the appointment list.)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<input buffer>--Decoded current appointment
R20/21--Points to the start of the input buffer
R22/23--same as R20/21
R24--Size of decoded appointment in bytes

Routines Called: HANDIO, APTGET, FXAPPT

Stack depth R6 (max): ?

Calls to Error routines (include error number and reason):

Notes: Generates a HANDIO call with event V.AFMT.

Reg: x = volatile

	0123	4567
R0	xxxx	
R10		
R20	oooo	o
R30	ii	
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: X			
HANDI Called: X			

R12 stack:

Entry	Exit

Routine: APTERR

File: KR/PS2

Author: MK

Description: Sets up the display in response to an erroneous appointment entry. Sets up the error buffer and annunciator for the specified error, restores the input buffer to the way it was entered, sets error indication in PSIOST, and sets up R20/24 for GETTEm to display the erroneous entry.

Input:
R20--Error number in binary

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Error annunciator on
<error buffer>--error message
<input buffer>--original input that caused the error
PSIOST--set to 41 hex to indicate error
R20/21--Points to INPBUF
R22/23--At start of line
R24--Set to size of APPT entry
E--set to 1

Routines Called:
ERRORR, KOPY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Calls ERRORR to set ERRBUF and annunciator.

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		x x
R20	booo	o
R30		
R40		
R50		
R60		
R70	x	xxxx

Status:

	in	out	Legend
Mode		x	d-BCD
E		1	b-BIN
DRP		20	i-input
ARP		22	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: APTFND

File: KR/PS2

Author: MK

Description: Finds the location in the apppointment file where the specified appointment would belong.

Input: (BIN)
R30/31--Pionter to start of appointment file
R34/35--Pointer to end of appointment file
<input buffer>--Encoded APPT to search for

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R32/33--POIhnter to first apointment that occurs at or past the one specified in the input buffer.
E-reg--Equal 1 iff the found appointment is at the same time as the target appointment.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes: See APFND'

Reg: x = volatile

	0123	4567
R0		
R10		
R20		xx
R30	iixx	ii
R40	x	xxxx
R50	xx	xxxx
R60		
R70		

Status:

	in	out	Legend
Mode	b	b	d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: X			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: APFND'

File: KR/PS2

Author: MK

Description: Finds the location in the appointment file where an appointment with the specified time would go. A pointer is returned which points to the first appointment whose time field is >= the input time.

Input:
R43/47--Time to search for (in encoded form.)
R30/31--Pointer to start of appointment file
R34/35--Pointer to the end of the appointment file
<input buffer>--Encoded APPT to search for

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R32/33--Pointer to the first appointment in the APPT file which occurs at or past the one specified in the input buffer.
E=1 iff the found appointment has the same time as the input time.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes: See APTFND

Reg: x = volatile

	0123	4567
R0		
R10		
R20		xx
R30	iioo	ii
R40	i	iiii
R50	xx	xxxx
R60		
R70		

Status:

	in	out	Legend
Mode	b	b	d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: APTGET

File: KR/PS2

Author: MK

Description: Move the current apointment to the input buffer from the APPT file.

Input: (BIN)
R32/33--Pointer to current appointment

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<input bufer>--Current encoded appointment

Routines Called:
KOPY

Stack depth R6 (max): 2

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +------------+------------+
|   |0123|4567|     |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+    +------------+------------+
|R0 |    |    |     |Mode| i |   | d-BCD  |    |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |    |            |            |
|R20|    |    |     |DRP |   | x | i-input|    |            |            |
|R30|  ii|    |     |ARP |   | x |        |    |            |            |
|R40|    |    |     +---------------------+    |            |            |
|R50|    |    |     | ROMJSB Needed:  X   |    |            |            |
|R60|    |    |     +---------------------+    |            |            |
|R70|   x|xxxx|     | HANDI Called:       |    |            |            |
+---+----+----+     +---------------------+    +------------+------------+
```

Routine: APTINS

File: KR/PS2

Author: MK

Description: Inserts an encoded appointment into the appointment file. The encoded appointment is in the input buffer. Sets the comparator with the pending appointment. E is nonzero iff the insertion fails.

Input: (BIN)
R32/33--Location at which to insert the APPT
R34/35--Pointer to the end of the APPT file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R34/35--Updated pointer to the end of APPT list
<APPT file> APPT inserted and file size is increased.
E-Reg--0 iff insert occurs as described.

Routines Called: INSERT, STALRM

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +------------+------------+
|   |0123|4567|     |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+    +------------+------------+
|R0 |  xx|    |     |Mode| b |   | d-BCD  |    |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |    |            |            |
|R20|    |    |     |DRP |   | x | i-input|    |            |            |
|R30|  ii|bb  |     |ARP |   | x |        |    |            |            |
|R40|  xx|xxxx|     +---------------------+    |            |            |
|R50|    |    |     | ROMJSB Needed:  X   |    |            |            |
|R60|    |    |     +---------------------+    |            |            |
|R70|    |    |     | HANDI Called:       |    |            |            |
+---+----+----+     +---------------------+    +------------+------------+
```

Routine: APTR+

File: KR/PS2

Author: MK

Description: Advances the current APPT pointer (R32/33) to the next apointment in the APPT file. E=1 iff the pointer is at the end of the file.

Input: (BIN)
R32/33--Pointer to current APPT
R34/35--Pointer to end of the APPT file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R34/33--Updated current appointment pointer
E-Reg--1 iff the pointer was at the end of the APPT file, 0 otherwise.

Routines Called: APSTAT, APTR-

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: Falls into the APTR- routine.

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +------------+------------+
|   |0123|4567|     |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+    +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |    |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |    |            |            |
|R20|xxxx|    |     |DRP |   | x | i-input|    |            |            |
|R30|  bb|ii  |     |ARP |   | x |        |    |            |            |
|R40|    |    |     +---------------------+    |            |            |
|R50|    |    |     | ROMJSB Needed:  X   |    |            |            |
|R60|    |    |     +---------------------+    |            |            |
|R70|    |    |     | HANDI Called:       |    |            |            |
+---+----+----+     +---------------------+    +------------+------------+
```

Routine: APTR-

File: KR/PS2

Author: MK

Description: Sets the current appointment pointer to the previous entry in the APPT file. Does not move the pointer prior to the first entry in the file.

Input: (BIN)
R32/33--Pointer to the current appointment
R30/31--Pointer to the start of the APPT file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R32/33--Updated current appointment pointer

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+--------+    +------------+------------+
|   |0123|4567|     |    |in |out| Legend |    | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+    +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |    |            |            |
|R10|    |    |     |E   |   | x | b-BIN  |    |            |            |
|R20|xxxx|    |     |DRP |   | x | i-input|    |            |            |
|R30|iibb|    |     |ARP |   | x |        |    |            |            |
|R40|    |    |     +---------------------+    |            |            |
|R50|    |    |     | ROMJSB Needed:  x   |    |            |            |
|R60|    |    |     +---------------------+    |            |            |
|R70|    |    |     | HANDI Called:       |    |            |            |
+---+----+----+     +---------------------+    +------------+------------+
```

Routine: GETLNX

File: KR/PS2

Author: MK

Description: Sets up the input bufer and necessary parameters for a call
to the keyboard/display routine (GETTEM) based on a given data
string.

Input:
- R20/21--Pointer to parameter string
- [R20/21]--Parameter string consisting of:
  - Bytes to place in the input buffer
  - 0 byte
  - Input check routine address
  - Template shield address
  - Line position address
  - Cursor position address
  - Byte count

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- <input buffer>--Data to be displayed in the subsequent call to GETTEM.
- R20/21--Line position for the display
- R22/23--Cursor position for the display
- R24--Number of bytes to be displayed
- R44/45--[template protect shield]
- R46/47--[input check routine]

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:
See GETTEM.

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode|   |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|bboo|o   |     |DRP |   |53 | i-input|  |            |            |
|R30|    |    |     |ARP |   |20 |        |  |            |            |
|R40|    |oooo|     +-------------------+  |            |            |
|R50|   x|xxxx|     | ROMJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|    |    |     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: RSTBUF

File: KR/PS2

Author: MK

Description: Copies 40 bytes from the pocket secretary scratch
area (PSTEMP) to the input buffer.

Input: (BIN)
<PSTEMP>--40 bytes to be copied

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<input buffer>--First 40 bytes from PSTEMP

Routines Called: KOPY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See SAVBUF, a routine which copies the first 40 bytes
of the input buffer to PSTEMP.

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   |   | i-input|  |            |            |
|R30|    |    |     |ARP |   |   |        |  |            |            |
|R40|    |    |     +-------------------+  |            |            |
|R50|    |    |     | ROMJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|  xx|xxxx|     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: SAVBUF

File: KR/PS2

Author: MK

Description: Copies 40 bytes from the input buffer to the pocket
secretary scratch area (PSTEMP.)

Input: (BIN)
<input buffer>--First 40 bytes to be copied

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<PSTEMP>--40 bytes from the input buffer

Routines Called: KOPY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See RSTBUF, a routine which copies the first 40 bytes
of PSTEMP to the input buffer.

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|    |    |     |DRP |   |   | i-input|  |            |            |
|R30|    |    |     |ARP |   |   |        |  |            |            |
|R40|    |    |     +-------------------+  |            |            |
|R50|    |    |     | ROMJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|  xx|xxxx|     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: TIMDIV

File: KR/PS2

Author: MK

Description: Extracts a particular unit of time from
a given number of seconds. Two 3 byte numbers (both in
seconds) are divided to yield a result which is returned in
ASCII. For example, 638 seconds can be divided by 60
seconds to yield 11 minutes 23 seconds

Input:
- R26/27--Pointer to a 3 byte binary divisor
- R45/47--3 byte binary dividend

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- R20/21--2 byte ASCII quotient
- R45/47--3 byte binary remainder
- Z-Flag--Set if result (R20/21) is 0.

Routines Called:
TOBCD2, TOASC2.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                  R12 stack:
+---+----+----+     +----+---+---+--------+  +------------+------------+
|   |0123|4567|     |    |in |out| Legend |  | Entry      | Exit       |
+---+----+----+     +----+---+---+--------+  +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |  |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |  |            |            |
|R20|oo  |  ii|     |DRP |   |20 | i-input|  |            |            |
|R30|    |    |     |ARP |   | x |        |  |            |            |
|R40|    | bbb|     +-------------------+  |            |            |
|R50|    |    |     | ROMJSB Needed:  x  |  |            |            |
|R60|    |    |     +-------------------+  |            |            |
|R70|    |    |     | HANDI Called:      |  |            |            |
+---+----+----+     +-------------------+  +------------+------------+
```

Routine: TTMPLT

File: KR/PS2

Author: MK

Description: Sets up a call to GETLNK which sets up a call to
GETTEM for input with the time template.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

```
<input buffer>     Template display string
R20/21             Line position in display
R22/23             Initial cursor position
R24                Byte count of template
R44/45             Pointer to input check routine
R46/47             Pointer to template protect shield
```

Routines Called:
GETLNX

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See ATMPLT

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|oooo|o  x|      |DRP |   |20 | i-input|   |            |            |
|R30|    |    |      |ARP |   |20 |        |   |            |            |
|R40|    |oooo|      +---------------------+   |            |            |
|R50|  xx|xxxx|      | ROMJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: ATMPLT

File: KR/PS2

Author: MK

Description: Sets up a call to GETLNX which sets up a call to
GETTEM for input with the appointment template.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

```
<input buffer>     Template display string
R20/21             Line position in display
R22/23             Initial cursor position
R24                Byte count of template
R44/45             Pointer to input check routine
R46/47             Pointer to template protect shield
```

Routines Called:
GETLNX

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See TTMPLT

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|oooo|o  x|      |DRP |   |20 | i-input|   |            |            |
|R30|    |    |      |ARP |   |20 |        |   |            |            |
|R40|    |oooo|      +---------------------+   |            |            |
|R50|  xx|xxxx|      | ROMJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: APMSKE

File: KR/PS2

Author: MK

Description: Protect shield for APPT template in EXTD mode.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode|   |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|    |    |      |DRP |   |   | i-input|   |            |            |
|R30|    |    |      |ARP |   |   |        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |    |      | ROMJSB Needed:      |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: APMSKY

File: KR/PS2

Author: MK

Description: Protect shield for APPT template in YEAR mode.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode|   |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   |   | b-BIN  |   |            |            |
|R20|    |    |      |DRP |   |   | i-input|   |            |            |
|R30|    |    |      |ARP |   |   |        |   |            |            |
|R40|    |    |      +---------------------+   |            |            |
|R50|    |    |      | ROMJSB Needed:      |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: STMMSK

File: KR/PS2

Author: MK

Description: Protect shield for set time template.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                      R12 stack:
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|R0 |    |    |    |Mode|   |   | d-BCD  |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN  |    |           |           |
|R20|    |    |    |DRP |   |   | i-input|    |           |           |
|R30|    |    |    |ARP |   |   |        |    |           |           |
|R40|    |    |    +--------------------+    |           |           |
|R50|    |    |    | ROMJSB Needed:     |    |           |           |
|R60|    |    |    +--------------------+    |           |           |
|R70|    |    |    | HANDI Called:      |    |           |           |
+---+----+----+    +--------------------+    +-----------+-----------+
```

Routine: TIMMSK

File: KR/PS2

Author: MK

Description: Protect shield for time command input template.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                      R12 stack:
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|R0 |    |    |    |Mode|   |   | d-BCD  |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN  |    |           |           |
|R20|    |    |    |DRP |   |   | i-input|    |           |           |
|R30|    |    |    |ARP |   |   |        |    |           |           |
|R40|    |    |    +--------------------+    |           |           |
|R50|    |    |    | ROMJSB Needed:     |    |           |           |
|R60|    |    |    +--------------------+    |           |           |
|R70|    |    |    | HANDI Called:      |    |           |           |
+---+----+----+    +--------------------+    +-----------+-----------+
```

Routine: YEARTM

File: KR/PS2

Author: MK

Description: 4 digit YEAR entry template.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                      R12 stack:
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|R0 |    |    |    |Mode|   |   | d-BCD  |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN  |    |           |           |
|R20|    |    |    |DRP |   |   | i-input|    |           |           |
|R30|    |    |    |ARP |   |   |        |    |           |           |
|R40|    |    |    +--------------------+    |           |           |
|R50|    |    |    | ROMJSB Needed:     |    |           |           |
|R60|    |    |    +--------------------+    |           |           |
|R70|    |    |    | HANDI Called:      |    |           |           |
+---+----+----+    +--------------------+    +-----------+-----------+
```

Routine: REPTTM

File: KR/PS2

Author: MK

Description: APPT repeat field entry template.

Input:

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile  Status:                      R12 stack:
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|   |0123|4567|    |    |in |out| Legend |    | Entry     | Exit      |
+---+----+----+    +----+---+---+--------+    +-----------+-----------+
|R0 |    |    |    |Mode|   |   | d-BCD  |    |           |           |
|R10|    |    |    |E   |   |   | b-BIN  |    |           |           |
|R20|    |    |    |DRP |   |   | i-input|    |           |           |
|R30|    |    |    |ARP |   |   |        |    |           |           |
|R40|    |    |    +--------------------+    |           |           |
|R50|    |    |    | ROMJSB Needed:     |    |           |           |
|R60|    |    |    +--------------------+    |           |           |
|R70|    |    |    | HANDI Called:      |    |           |           |
+---+----+----+    +--------------------+    +-----------+-----------+
```

Routine: APTCHK

```
File:     KR/PS3

Author:   MK

Description:   Checks each field of the appointment entered
      in the input buffer.  Encodes the appointment for storage
      in the appointment file.  If a field is incorrect APICHK
      returns with the appropriate error number.

Input:     (BIN)
      <input buffer>--ASCII appointment (as displayed)


Output (include E-Reg, Z-Flag, C-Flag if pertinent):
      E=0 or 1 if there was no error or an error occured,
          respectively.
      R20/24--GETTEM parameters for error display.

Routines Called:  ALMCHK, APTERR, DATCHK, DAYCHK, FINDTD,
      GETTD, KOPY, RPTIMP, TIMCHK.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):



Notes:


Reg: x = volatile     Status:                     R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |    |    |      |Mode| b | x | d-BCD   |  |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |  |            |            |
|R20|xxxx|   x|      |DRP |   | x | i-input |  |            |            |
|R30|    |    |      |ARP |   | x |         |  |            |            |
|R40|xxxx|xxxx|      +----------------------+  |            |            |
|R50|xxxx|xxxx|      | ROMJSB Needed:  x    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|   x|xxxx|      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+


Routine:  DCCLOK
```

```
File:     KR/PS3

Author:   MK

Description:   Converts a binary number of seconds from midnight
      1-JAN-0000 to BCD representing seconds, minutes, hour of
      day, day of month, month, year, and century.
Input:     (BIN)
      R43/47--Binary number of seconds from midnight 1-JAN-0000
            (40 bit value.)
Output (include E-Reg, Z-Flag, C-Flag if pertinent):
      R40  seconds (0-59)
      R41  minutes (0-59)
      R42  hours   (0-23)
      R43  days    (1-31)
      R44  month   (1-12)
      R45  year    (0-99)
      R46  century (0-99)

Routines Called:
      TOBCD8

Stack depth R6 (max):

Calls to Error routines (include error number and reason):



Notes:


Reg: x = volatile     Status:                     R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |  |            |            |
|R20|xxxx|xxxx|      |DRP |   | x | i-input |  |            |            |
|R30|    |    |      |ARP |   | x |         |  |            |            |
|R40|oonb|bbbb|      +----------------------+  |            |            |
|R50|   x|xxxx|      | ROMJSB Needed:  x    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|    |    |      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+


Routine:  ENCLOK
```

```
File:     KR/PS3

Author:   MK

Description:   Converts an array of 7 BCD time fields representing
      centuries, years, months, days, hours, minutes and seconds
      to a single 40 bit binary number representing the number of
      seconds since 1-Jan-0000.
Input:     (BIN)
      R40  Seconds into minute     \
      R41  minutes into hour        \
      R42  hours into day            \
      R43  day of month               >----BCD data
      R44  month of year             /
      R45  years into century       /
      R46  century                 /

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
      R43/47  Binary numbers of seconds from 1-Jan-0000.
Routines Called:
      TOBIN8

Stack depth R6 (max):

Calls to Error routines (include error number and reason):



Notes:  See DCCLOK


Reg: x = volatile     Status:                     R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD   |  |            |            |
|R10|    |    |      |E   |   |   | b-BIN   |  |            |            |
|R20|  xx|xxxx|      |DRP |   | 20| i-input |  |            |            |
|R30|    |    |      |ARP |   | 6 |         |  |            |            |
|R40|iiib|bbbb|      +----------------------+  |            |            |
|R50|    |    |      | ROMJSB Needed:  X    |  |            |            |
|R60|    |    |      +----------------------+  |            |            |
|R70|    |    |      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+


Routine:  FINDTD
```

```
File:     KR/PS3

Author:   MK

Description:   Computes the first occurence of a time and date that
      meets a given set of specifications that includes a base time/
      date, a time/date mask, and a day mask indicating a day of
      the week or a default.

Input:     (BIN or BCD)
      R21       Day of week mask
      R40/46    Base time and date for time/date search
      R50/56    Time and date mask
      PSSTAT    Bit#6 indicates entry mode (year/extd)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
      E-Reg  0 if match was found, 1 if no match found.
      R43/47  Matching time/date in seconds since 1-Jan-0000,
              valid only if E=0.
Routines Called:
      DAYOK, DCDAY, ENCLOK, MINDD, MINHH, MINMM, MINMN, MINYY.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):



Notes:    FNDTD' is an entry point within FINDTD.


Reg: x = volatile     Status:                     R12 stack:
+---+----+----+      +----+---+---+---------+  +------------+------------+
|   |0123|4567|      |    |in |out| Legend  |  | Entry      | Exit       |
+---+----+----+      +----+---+---+---------+  +------------+------------+
|R0 |  xx|    |      |Mode|   | b | d-BCD   |  |            |            |
|R10|    |    |      |E   |   | o | b-BIN   |  |            |            |
|R20| ixx|xxxx|      |DRP |   | x | i-input |  |            |            |
|R30|    |    |      |ARP |   | x |         |  |            |            |
|R40|iiib|bbbo|      +----------------------+  |            |            |
|R50|iiii|iii |      | ROMJSB Needed:  x    |  |            |            |
|R60|xxxx|xxxx|      +----------------------+  |            |            |
|R70|xxxx|xxxx|      | HANDI Called:        |  |            |            |
+---+----+----+      +----------------------+  +------------+------------+


Routine:  FINDTD
```

File: KR/PS3

Author: MK

Description: Operates the same as FINDTD entered with appointments in extended entry mode.

Input: (BCD)
- R21 Day of week mask
- R50/56 Time and date mask
- R60/66 Minimum time/date fields
- R70/76 Maximum time/date fields

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- E-Reg 0 if match was found, 1 if no match found.
- R43/47 Matching time/date in seconds since 1-Jan-0000, valid only if E=0.

Routines Called:
DAYOK, DCDAY, ENCLOK, MINDD, MINHH, MINMM, MINMN, MINYY.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: FNDTD' is an entry point within FINDTD.

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode|   | b | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20| ixx|xxxx|      |DRP |   | x | i-input|   |            |            |
|R30|    |    |      |ARP |   | x |        |   |            |            |
|R40|   o|oooo|      +---------------------+   |            |            |
|R50|iiii|iii |      | ROMJSB Needed:   x  |   |            |            |
|R60|iiii|iii |      +---------------------+   |            |            |
|R70|iiii|iii |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: FXAPPT

File: KR/PS3

Author: MK

Description: Decodes and formats an encoded appointment, leaving the resulting formatted appointment in the input buffer. The first 21 characters of a past due appointment are underlined.

Input: (BIN)
<input buffer> Encoded appointment

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<input buffer> Decoded formatted appointment

Routines Called:
DCCLOK, DCDAY, FXALRM, FXDATE, FXDAY, FXTIME, GETCLK, NUMPCK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |  xx|    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | x | b-BIN  |   |            |            |
|R20|xx  |x xx|      |DRP |   | x | i-input|   |            |            |
|R30|    |    |      |ARP |   | x |        |   |            |            |
|R40|xxxx|xxxx|      +---------------------+   |            |            |
|R50|   x|xxxx|      | ROMJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: ALMCHK

File: KR/PS4

Author: MK

Description: Checks the ALARM field for proper syntax and returns an encoded alarm byte.

Input: (BIN)
- R46/47 First and second bytes of the alarm field in that order

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- R47 & R21 Encoded alarm byte
- E-Reg 0 if syntax ok, 1 if not

Routines Called:
NUMCHK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |    |    |      |Mode| b |   | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20|xo  |    |      |DRP |   |21 | i-input|   |            |            |
|R30|    |    |      |ARP |   |47 |        |   |            |            |
|R40|    |  ii|      +---------------------+   |            |            |
|R50|    |    |      | ROMJSB Needed:  x   |   |            |            |
|R60|    |    |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: DATCHK

File: KR/PS4

Author: MK

Description: Checks for a valid date field and returns an encoded date field.

Input:
- R40/47 ASCII date field from appointment entry

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- R44/47 Encoded date field
- E-reg E=0 iff date valid

Routines Called:
FLDCHK, FXYEAR, LEAPYR.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See DATCK'.

```
Reg: x = volatile    Status:                   R12 stack:
+---+----+----+      +----+---+---+--------+   +------------+------------+
|   |0123|4567|      |    |in |out| Legend |   | Entry      | Exit       |
+---+----+----+      +----+---+---+--------+   +------------+------------+
|R0 |xx  |    |      |Mode| b | x | d-BCD  |   |            |            |
|R10|    |    |      |E   |   | o | b-BIN  |   |            |            |
|R20|    |    |      |DRP |   | x | i-input|   |            |            |
|R30|    |    |      |ARP |   | x |        |   |            |            |
|R40|iiii|bbbb|      +---------------------+   |            |            |
|R50|    |    |      | ROMJSB Needed:  x   |   |            |            |
|R60|    | xx |      +---------------------+   |            |            |
|R70|    |    |      | HANDI Called:       |   |            |            |
+---+----+----+      +---------------------+   +------------+------------+
```

Routine: DATCK'

File: KR/PS4

Author: MK

Description: DATCK' is an alternative entry point into the routine DATCHK. It is used during the set time command when the century field has been explicitly set.

Input:
R40/47 ASCII date field from appointment entry

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R44/47 Encoded date field
E-reg E=0 iff date valid

Routines Called:
FLDCHK, FXYEAR, LEAPYR.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
See DATCK'.

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit       |
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|R0 |xx  |    |     |Mode| b | x | d-BCD   |   |             |            |
|R10|    |    |     |E   |   | o | b-BIN   |   |             |            |
|R20|    |    |     |DRP |   | x | i-input |   |             |            |
|R30|    |    |     |ARP |   | x |         |   |             |            |
|R40|iiii|bbbb|     +----------------------+   |             |            |
|R50|    |    |     | ROMJSB Needed:  x    |   |             |            |
|R60|    | xx |     +----------------------+   |             |            |
|R70|    |    |     | HANDI Called:        |   |             |            |
+---+----+----+     +----------------------+   +-------------+------------+
```

Routine: DAYCHK

File: KR/PS4

Author: MK

Description: Checks the validity of an ASCII day of week field and returns an encoded DOW byte.

Input: (BIN)
R45/47 ASCII DOW field

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R21&r47 Encoded DOW byte
E-Reg E=0 if field OK, E=1 otherwise

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit       |
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|R0 |    |    |     |Mode| b | b | d-BCD   |   |             |            |
|R10|    |    |     |E   |   | o | b-BIN   |   |             |            |
|R20|xox |    |     |DRP |   | x | i-input |   |             |            |
|R30|    |    |     |ARP |   | x |         |   |             |            |
|R40|    | iib|     +----------------------+   |             |            |
|R50|    | xxx|     | ROMJSB Needed:  x    |   |             |            |
|R60|    |    |     +----------------------+   |             |            |
|R70|    |    |     | HANDI Called:        |   |             |            |
+---+----+----+     +----------------------+   +-------------+------------+
```

Routine: DAYOK

File: KR/PS4

Author: MK

Description: Takes a date and determines if the day is legitimate for the month.

Input: (BIN)
R63 Day of month \
R64 Month of year _____ BCD data
R65 Year of century /
R66 Century /

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-reg E=1 if day of moth is too large, otherwise 0.

Routines Called:
LPYEAR

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit       |
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|R0 |xxxx|    |     |Mode| b |   | d-BCD   |   |             |            |
|R10|    |    |     |E   |   | o | b-BIN   |   |             |            |
|R20|    |    |     |DRP |   | 0 | i-input |   |             |            |
|R30|    |    |     |ARP |   | 63|         |   |             |            |
|R40|    |    |     +----------------------+   |             |            |
|R50|    |    |     | ROMJSB Needed:  x    |   |             |            |
|R60|   i|iii |     +----------------------+   |             |            |
|R70|    |    |     | HANDI Called:        |   |             |            |
+---+----+----+     +----------------------+   +-------------+------------+
```

Routine: DCDAY

File: KR/PS4

Author: MK

Description: Determines the day of week for a given time and date.

Input:
R43/47 Time/date in binary format

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R27 Day of week number (1 for Sat .. 7 for Fri)

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                    R12 stack:
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|   |0123|4567|     |    |in |out| Legend  |   | Entry       | Exit       |
+---+----+----+     +----+---+---+---------+   +-------------+------------+
|R0 |    |    |     |Mode| b | b | d-BCD   |   |             |            |
|R10|    |    |     |E   |   |   | b-BIN   |   |             |            |
|R20|    |   o|     |DRP |   | 43| i-input |   |             |            |
|R30|    |    |     |ARP |   | 6 |         |   |             |            |
|R40|   i|iiii|     +----------------------+   |             |            |
|R50|    |    |     | ROMJSB Needed:  x    |   |             |            |
|R60|    |    |     +----------------------+   |             |            |
|R70|    |    |     | HANDI Called:        |   |             |            |
+---+----+----+     +----------------------+   +-------------+------------+
```

Routine: DUPCHK

File: KR/PS4

Author: MK

Description: Checks for an appointment in the appointment file that matches the contents of the input buffer.

Input: (BIN)
<input buffer> Test appointment
R32/33 Current appointment pointer (assumed set by call to APTFND)
R34/35 Pointer to end of appointment file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-reg E=0 if there is no duplicate
=1 if there is a duplicate
R32/33 Pointer to duplicate appointment (E=1)
Pointer to location for test appointment (E=0)

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20	xxxx	xxxx
R30	bb	ii
R40	x	xxxx
R50	x	xxxx
R60		
R70		

Status:

	in	out	Legend
Mode	b	b	d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: FLDCHK

File: KR/PS4

Author: MK

Description: Checks for default or numeric value in a 2 byte field. Returns FF hex if default value or blanks, or returns the BCD equivalent of the ASCII input. E=1 if the field does not contain numbers.

Input: (BIN)
R0/1 ASCII default value
R20/21 Field to be encoded

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R20 BCD value of numeric field (or FF if default)
R21 0
E-Reg 0 if value was ok; 1 if value not ok

Routines Called:
)NUMCHK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
E is set by NUMCHK.

Reg: x = volatile

	0123	4567
R0	ii	
R10		
R20	bb	
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: FXALRM

File: KR/PS4

Author: MK

Description: Decods the encoded alarm byte and returns a 2 byte ASCII alarm field.

Input: (BIN)
R47 Encoded alarm byte

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R46/47 Decoded ASCII alarm field

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes: Consult the Appointment mode documentation for the encoding scheme for the encoded alarm byte.

Reg: x = volatile

	0123	4567
R0	xxxx	
R10		
R20		
R30		
R40		ob
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b	b	d-BCD
E			b-BIN
DRP		47	i-input
ARP		0	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: FXDATE

File: KR/PS4

Author: MK

Description: Formats a BCD date specification into an ASCII date field. Arranges the fields according to the date mode, DMY or MDY and fills in slash separators.

Input: (BIN)
R45/47 BCD formatted date specification

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R40/47 ASCII formatted ddate specification

Routines Called:
TOASC2

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20	xxxx	x
R30		
R40	oooo	oiii
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		20	i-input
ARP		46	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: FXDAY

File: KR/PS4

Author: HK

Description: Given a day of the week number, this routine returns the ASCII day of the week abbreviation.

Input: (BIN)
R20 Day of the week number

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R45/47 ASCII day of the week abbreviation

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes: Day of the week number is 1-7: Sat=1 .. Fri=7.

```
Reg: x = volatile    Status:                     R12 stack:
+---+----+----+      +----+---+---+--------+    +----------+----------+
|   |0123|4567|      |    |in |out| Legend |    | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+    +----------+----------+
|R0 |  xx|    |      |Mode| b | b | d-BCD  |    |          |          |
|R10|    |    |      |E   |   |   | b-BIN  |    |          |          |
|R20|i   |  xx|      |DRP |   |45 | i-input|    |          |          |
|R30|    |    |      |ARP |   |26 |        |    |          |          |
|R40|    | ooo|      +---------------------+    |          |          |
|R50|    |    |      | ROMJSB Needed:  x   |    |          |          |
|R60|    |    |      +---------------------+    |          |          |
|R70|    |    |      | HANDI Called:       |    |          |          |
+---+----+----+      +---------------------+    +----------+----------+
```

Routine: FXTIME

File: KR/PS4

Author: HK

Description: Translates a time of day in BCD format to the ASCII. Numbers are arranged in the proper format and with regard to 12 or 24 hour time.

Input: (BIN)
R41/42 BCD minutes and hours, respectively

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R40/47 ASCII time of day

Routines Called:
TOASC2

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: Looks at PSSTAT to determine 12 or 24 hour mode.

```
Reg: x = volatile    Status:                     R12 stack:
+---+----+----+      +----+---+---+--------+    +----------+----------+
|   |0123|4567|      |    |in |out| Legend |    | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+    +----------+----------+
|R0 |    |    |      |Mode| b |   | d-BCD  |    |          |          |
|R10|    |    |      |E   |   | x | b-BIN  |    |          |          |
|R20|xx x|    |      |DRP |   | 42| i-input|    |          |          |
|R30|    |    |      |ARP |   | 40|        |    |          |          |
|R40|obbo|oooo|      +---------------------+    |          |          |
|R50|    |    |      | ROMJSB Needed:  x   |    |          |          |
|R60|    |    |      +---------------------+    |          |          |
|R70|    |    |      | HANDI Called:       |    |          |          |
+---+----+----+      +---------------------+    +----------+----------+
```

Routine: FXYEAR

File: KR/PS4

Author: HK

Description: Prompts and inputs a 4 digit year for an appointment specification. Saves and restores the input buffer used for input of the 4 digit year. Also adjusts the default entry so that it will parse properly in the appointment check routine.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R44/47 ASCII 4 digit year

Routines Called:
GETLNX, GETITM, RSTBUF, SAVBUF.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                     R12 stack:
+---+----+----+      +----+---+---+--------+    +----------+----------+
|   |0123|4567|      |    |in |out| Legend |    | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+    +----------+----------+
|R0 |  xx|  xx|      |Mode| b | x | d-BCD  |    |          |          |
|R10|    |    |      |E   |   | x | b-BIN  |    |          |          |
|R20|xx  |    |      |DRP |   | x | i-input|    |          |          |
|R30|    |    |      |ARP |   | x |        |    |          |          |
|R40|    |oooo|      +---------------------+    |          |          |
|R50|    |    |      | ROMJSB Needed:   x  |    |          |          |
|R60|    |    |      +---------------------+    |          |          |
|R70|    |    |      | HANDI Called:       |    |          |          |
+---+----+----+      +---------------------+    +----------+----------+
```

Routine: LPYEAR

File: KR/PS4

Author: HK

Description: Tests to determine if a given yar is a leap year or not.

Input: (BIN)
R65 Year of century (BCD digits)
R66 Century (BCD digits)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E=1 iff the year is a leap year; 0 otherwise.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile    Status:                     R12 stack:
+---+----+----+      +----+---+---+--------+    +----------+----------+
|   |0123|4567|      |    |in |out| Legend |    | Entry    | Exit     |
+---+----+----+      +----+---+---+--------+    +----------+----------+
|R0 |    |    |      |Mode| b | b | d-BCD  |    |          |          |
|R10|    |    |      |E   |   | o | b-BIN  |    |          |          |
|R20|    |   x|      |DRP |   | x | i-input|    |          |          |
|R30|    |    |      |ARP |   | x |        |    |          |          |
|R40|    |    |      +---------------------+    |          |          |
|R50|    |    |      | ROMJSB Needed:  x   |    |          |          |
|R60|    | ii |      +---------------------+    |          |          |
|R70|    |    |      | HANDI Called:       |    |          |          |
+---+----+----+      +---------------------+    +----------+----------+
```

Routine: MINDD

File: *KR/PS4*

Author: *MK*

Description: *Sets BCD day, hour, and minute and second to their lowest possible values, 1, 0, 0, and 0 respectively.*

Input: *(BIN or BCD)*

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- *R63* *1, minimum day*
- *R62* *0, minimum hour*
- *R61* *0, minimum minutes*
- *R60* *0, minimum seconds*

Routines Called:

Stack depth R6 (max): *0*

Calls to Error routines (include error number and reason):

Notes: *See MINYY, MINMM, MINHH, and MINMH also.*

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+     +----+--+---+--------+     +----------+----------+
|   |0123|4567|     |    |in|out| Legend |     | Entry    | Exit     |
+---+----+----+     +----+--+---+--------+     +----------+----------+
|R0 |    |    |     |Mode|  |   | d-BCD  |     |          |          |
|R10|    |    |     |E   |  |   | b-BIN  |     |          |          |
|R20|    |    |     |DRP |  |60 | i-input|     |          |          |
|R30|    |    |     |ARP |  |   |        |     |          |          |
|R40|    |    |     +----+--+---+--------+     |          |          |
|R50|    |    |     | ROMJSB Needed:  x  |     |          |          |
|R60|oooo|    |     +--------------------+     |          |          |
|R70|    |    |     | HANDI Called:      |     |          |          |
+---+----+----+     +--------------------+     +----------+----------+
```

Routine: *MINDD*

File: *KR/PS4*

Author: *MK*

Description: *Sets BCD day, hour, and minute and second to their lowest possible values, 1, 0, 0, and 0 respectively.*

Input: *(BIN or BCD)*

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- *R63* *1, minimum day*
- *R62* *0, minimum hour*
- *R61* *0, minimum minutes*
- *R60* *0, minimum seconds*

Routines Called:

Stack depth R6 (max): *0*

Calls to Error routines (include error number and reason):

Notes: *See MINYY, MINMM, MINHH, and MINMH also.*

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+     +----+--+---+--------+     +----------+----------+
|   |0123|4567|     |    |in|out| Legend |     | Entry    | Exit     |
+---+----+----+     +----+--+---+--------+     +----------+----------+
|R0 |    |    |     |Mode|  |   | d-BCD  |     |          |          |
|R10|    |    |     |E   |  |   | b-BIN  |     |          |          |
|R20|    |    |     |DRP |  |60 | i-input|     |          |          |
|R30|    |    |     |ARP |  |   |        |     |          |          |
|R40|    |    |     +----+--+---+--------+     |          |          |
|R50|    |    |     | ROMJSB Needed:  x  |     |          |          |
|R60|oooo|    |     +--------------------+     |          |          |
|R70|    |    |     | HANDI Called:      |     |          |          |
+---+----+----+     +--------------------+     +----------+----------+
```

Routine: *MINHH*

File: *KR/PS4*

Author: *MK*

Description: *Sets BCD hour, and minute and second to their lowest possible values, 0, 0, and 0 respectively.*

Input: *(BIN or BCD)*

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- *R62* *0, minimum hour*
- *R61* *0, minimum minutes*
- *R60* *0, minimum seconds*

Routines Called:

Stack depth R6 (max): *0*

Calls to Error routines (include error number and reason):

Notes: *See MINYY, MINMM, MINDD, and MINMH also.*

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+     +----+--+---+--------+     +----------+----------+
|   |0123|4567|     |    |in|out| Legend |     | Entry    | Exit     |
+---+----+----+     +----+--+---+--------+     +----------+----------+
|R0 |    |    |     |Mode|  |   | d-BCD  |     |          |          |
|R10|    |    |     |E   |  |   | b-BIN  |     |          |          |
|R20|    |    |     |DRP |  |60 | i-input|     |          |          |
|R30|    |    |     |ARP |  |   |        |     |          |          |
|R40|    |    |     +----+--+---+--------+     |          |          |
|R50|    |    |     | ROMJSB Needed:  x  |     |          |          |
|R60|ooo |    |     +--------------------+     |          |          |
|R70|    |    |     | HANDI Called:      |     |          |          |
+---+----+----+     +--------------------+     +----------+----------+
```

Routine: *MINHM*

File: *KR/PS4*

Author: *MK*

Description: *Sets BCD month, day, hour, minute, and second fields to their lowest possible values, 1, 1, 0, 0, 0 respectively.*

Input: *(BIN or BCD)*

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
- *R64* *1, minimum month*
- *R63* *1, minimum day*
- *R62* *0, minimum hour*
- *R61* *0, minimum minutes*
- *R60* *0, minimum seconds*

Routines Called:

Stack depth R6 (max): *0*

Calls to Error routines (include error number and reason):

Notes: *See MINYY, MINMM, MINHH, and MINDD also.*

```
Reg: x = volatile    Status:                    R12 stack:
+---+----+----+     +----+--+---+--------+     +----------+----------+
|   |0123|4567|     |    |in|out| Legend |     | Entry    | Exit     |
+---+----+----+     +----+--+---+--------+     +----------+----------+
|R0 |    |    |     |Mode|  |   | d-BCD  |     |          |          |
|R10|    |    |     |E   |  |   | b-BIN  |     |          |          |
|R20|    |    |     |DRP |  |60 | i-input|     |          |          |
|R30|    |    |     |ARP |  |   |        |     |          |          |
|R40|    |    |     +----+--+---+--------+     |          |          |
|R50|    |    |     | ROMJSB Needed:  x  |     |          |          |
|R60|oooo|o   |     +--------------------+     |          |          |
|R70|    |    |     | HANDI Called:      |     |          |          |
+---+----+----+     +--------------------+     +----------+----------+
```

Routine: *MINMH*

File: KR/PS4

Author: MK

Description: Sets BCD minute, and second fields to their lowest possible values, 0, 0 respectively.

Input: (BIN or BCD)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R61 0, minimum minutes
R60 0, minimum seconds

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes: See MINYY, MINMM, MINHH, and MINDD also.

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		
R60	oo	
R70		

Status:

	in	out	Legend
Mode			d-BCD
E			b-BIN
DRP		60	i-input
ARP			

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: MINYY

File: KR/PS4

Author: MK

Description: Sets BCD year, month, hour, minute, and second fields to their lowest possible values, 0, 1, 1, 0, 0, 0 respectively.

Input: (BIN or BCD)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R65 0, minimum year
R64 1, minimum month
R63 1, minimum day
R62 0, minimum hour
R61 0, minimum minutes
R60 0, minimum seconds

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes: See MINYY, MINMM, MINHH, and MINDD also.

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40		
R50		
R60	oooo	oo
R70		

Status:

	in	out	Legend
Mode			d-BCD
E			b-BIN
DRP		60	i-input
ARP			

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: NUNPCK

File: KR/PS4

Author: MK

Description: Moves the note field from its position in an encoded appointment to its position in a decoded appointment within the input buffer.

Input: (BIN)
‹input buffer› Encoded appointment

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
‹input buffer› Encoded appointment with the note field moved to INP+26.

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20	x	
R30		
R40		
R50		
R60		
R70	x	xxxx

Status:

	in	out	Legend
Mode	b		d-BCD
E			b-BIN
DRP		73	i-input
ARP		x	

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: RPTADJ

File: KR/PS4

Author: MK

Description: Reschedules a repeating apointment.

Input: (BIN)
R32/33 Pointer to a repeating appintment to be rescheduled

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
‹APPT file› Appointment rescheduled

Routines Called:
APFND', DCCLOK, DCDAY, ENCLOK, GETCLK, GETTD, MOVE, STALRM

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: RPTADJ will attempt to reschedule an appointment 16 times. That is, if 16 * repeat_interval added to the time of the appointment creates an appointment that is still past due, then the routine will give up and set the appointment to the current time.

Reg: x = volatile

	0123	4567
R0	xx	xx
R10		
R20	xx	x
R30	xx	
R40	x xx	xxxx
R50	x x	xxxx
R60	xxxx	xxxx
R70	xxxx	xxxx

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: RPTINP

File: KR/PS4

Author: MK

Description: Inputs and encodes the repeat specification for a repeating appointment (type R or A.)

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-reg 0 if repeat field encoded correctly
1 if repeat field encoded incorrectly
R43/47 Encoded repeat field, if E=0.

Routines Called:
DAYCHK, GETLNK, GETTEN, MULT60, NUMCHK, RSTBUF, SAVBUF, TOBIN2.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20	xxx	
R30		
R40	o	oooo
R50		xxx
R60		
R70		

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: TIMCHK

File: KR/PS4

Author: MK

Description: Checks and converts an ASCII time field of an APPT entry to BCD 24 hour mode.

Input: (BIN)
R40/47 ASCII time field from appointment entry

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R47 BCD hour or FF if default
R46 BCD minute or FF if default
R45 0 seconds or FF if default

Routines Called:
NUMCHK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20	xxxx	xxxx
R30		
R40	iiii	ibbb
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E			b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: ACREAT

File: KR/PS5

Author: MK

Description: Opens already existing or creates an appointment file with the name "APPT " and type TYAPPT. Sets up the start and end of apointment file pointers and the current appointment pointer.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg E=1 if insufficient memory to create file; 0 otherwise
R30/31 Start of appointment file pionter
R32/33 Current appointment file pointer (same as start)
R34/35 End of apointment file pointer

Routines Called:
FCREAT, FOPEN

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: AOPEN is an entry point within ACREAT.

Reg: x = volatile

	0123	4567
R0		
R10		
R20	xx	
R30	oooo	oo
R40	xxxx	xxxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: AOPEN

File: KR/PS5

Author: MK

Description: Opens already existing appointment file. Does not create a non-existent appointment file. Sets up appointment file pointers.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg E=1 if file does not exist; 0 otherwise
R30/31 Start of appointment file pionter
R32/33 Current appointment file pointer (same as start)
R34/35 End of apointment file pointer

Routines Called:
FOPEN

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: AOPEN is an entry point within ACREAT.

Reg: x = volatile

	0123	4567
R0		
R10		
R20	xx	
R30	oooo	oo
R40	xxxx	xxxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		o	b-BIN
DRP		x	i-input
ARP		x	

ROMJSB Needed: x

HANDI Called:

R12 stack:

Entry	Exit

Routine: AOPEN'

File: KR/PS5

Author: MK

Description: Opens a file and sets up pointers to the start and end of the file.

Input: (BIN)
R40/47 Name of file to open

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg E=1 if file does not exist; 0 otherwise
R30/31 Start of file pointer
R32/33 Start of file pointer
R34/35 End of file pointer

Routines Called:
FOPEN

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: AOPEN' is an entry point within ACREAT.

```
Reg: x = volatile     Status:                       R12 stack:
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|    |0123|4567|      |     |in |out| Legend |      | Entry    | Exit     |
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|R0  |    |    |      |Mode | b | x | d-BCD  |      |          |          |
|R10 |    |    |      |E    |   | o | b-BIN  |      |          |          |
|R20 |xx  |    |      |DRP  |   | x | i-input|      |          |          |
|R30 |oooo|oo  |      |ARP  |   | x |        |      |          |          |
|R40 |iiii|iiii|      +----------------------+      |          |          |
|R50 |    |    |      | ROMJSB Needed:  x    |      |          |          |
|R60 |    |    |      +----------------------+      |          |          |
|R70 |    |    |      | HANDI Called:        |      |          |          |
+----+----+----+      +----------------------+      +----------+----------+
```

Routine: ALBEEP

File: KR/PS5

Author: MK

Description: Executes tone patterns for the alarms of appointments.

Input: (BIN)
R27 Encoded alarm byte (low digit is the alarm number)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

Routines Called:
BEEPER, STCOMP, STOP?

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile     Status:                       R12 stack:
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|    |0123|4567|      |     |in |out| Legend |      | Entry    | Exit     |
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|R0  |  xx|    |      |Mode | b |   | d-BCD  |      |          |          |
|R10 |    |    |      |E    |   | x | b-BIN  |      |          |          |
|R20 |xxxx|   i|      |DRP  |   | x | i-input|      |          |          |
|R30 |    |    |      |ARP  |   | x |        |      |          |          |
|R40 |xxxx|xxxx|      +----------------------+      |          |          |
|R50 |    |    |      | ROMJSB Needed:  x    |      |          |          |
|R60 |    |    |      +----------------------+      |          |          |
|R70 |    |    |      | HANDI Called:        |      |          |          |
+----+----+----+      +----------------------+      +----------+----------+
```

Routine: APPROC

File: KR/PS5

Author: MK

Description: Processes pending appointments. Sets up the note of a pending appointment for display or execution as a BASIC statement. If a pending appointment has no note field then the file is searched for another pending appointment.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
E-Reg E=0 if processing due according to contents of R25; E=1 if no processing due.
R25 Terminator for note field processing (0 if no processing due.)

Routines Called:
AOPEN, APSTAT, APTDEL, APTRIG, HANDIO, PRNOTE

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: HANDIO called with the V.APRC event.

```
Reg: x = volatile     Status:                       R12 stack:
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|    |0123|4567|      |     |in |out| Legend |      | Entry    | Exit     |
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|R0  |  xx|    |      |Mode | b | x | d-BCD  |      |          |          |
|R10 |    |    |      |E    |   | o | b-BIN  |      |          |          |
|R20 |    | o x|      |DRP  |   | x | i-input|      |          |          |
|R30 |xxxx|xx  |      |ARP  |   | x |        |      |          |          |
|R40 |xxxx|xxxx|      +----------------------+      |          |          |
|R50 |    |    |      | ROMJSB Needed:  x    |      |          |          |
|R60 |    |    |      +----------------------+      |          |          |
|R70 |    |    |      | HANDI Called:   x    |      |          |          |
+----+----+----+      +----------------------+      +----------+----------+
```

Routine: APPTRS

File: KR/PS5

Author: MK

Description: Loads registers with pointers to the start and end of a file.

Input: (BIN)
R30/31 Pointer to directory for a file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R30/31 Start of file
R32/33 Also start of file
R34/35 End of file + 1

Routines Called:

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile     Status:                       R12 stack:
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|    |0123|4567|      |     |in |out| Legend |      | Entry    | Exit     |
+----+----+----+      +-----+---+---+--------+      +----------+----------+
|R0  |    |    |      |Mode |   |   | d-BCD  |      |          |          |
|R10 |    |    |      |E    |   |   | b-BIN  |      |          |          |
|R20 |    |    |      |DRP  |   |   | i-input|      |          |          |
|R30 |    |    |      |ARP  |   |   |        |      |          |          |
|R40 |    |    |      +----------------------+      |          |          |
|R50 |    |    |      | ROMJSB Needed:       |      |          |          |
|R60 |    |    |      +----------------------+      |          |          |
|R70 |    |    |      | HANDI Called:        |      |          |          |
+----+----+----+      +----------------------+      +----------+----------+
```

Routine: APSTAT

File: KR/PS5

Author: MK

Description: Returns status information regarding the current appointment. Returns the size, the encoded alarm byte, whether or not the appointment has been triggered, and whether or not the appointment has been acknowledged.

Input:
R32/33 Pointer to the curent appointment

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R2/3 Size of the current appointment
R27 Encoded alarm byte from current appointment
E-Reg E=1 if apointment is triggered; 1 otherwise
CY-Flag high iff appointment acknowledged
DRP 2 (This is used elsewhere in the pocket secretary code.)

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		x
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		o	b-BIN
DRP		2	i-input
ARP		32	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: APTACK

File: KR/PS5

Author: MK

Description: Acknowledges the current appointment. Returns without acknowledging if the appointment is already acknowledged or has not yet been triggered. Sets the MSB of the size byte in the appointment. Reschedules type A appointments. Sets current appointment pointer at the oldest due appointment, if any exist. Clears the appointment annuynciator and PSIOST if there are no further due appointments.

Input: (BIN)
R30/31 Pointer to start of appointment file
R32/33 Pointer to apointment to acknowledge
R34/35 Pointer to end of appointment file +1.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Assuming appointment was acknowledged:
MSB of size byte of acknowledged appointment is set
R32/33 If there are further due appointments, a pointer to another next due appointment; otherwise a pointer to the acknowledged appointment.
<appointment annunciator> Cleared
PSIOST Cleared, such that the template is displayed by the next command entry.

Routines Called:
ANN.A-, APSTAT, HANDIO, RPTADJ

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
HANDIO call is with event V.AACK.

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		x
R30	iibb	ii
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called: x			

R12 stack:

Entry	Exit

Routine: APTMRG

File: KR/PS5

Author: MK

Description: Merges the scratch appointment file into the pocket secretary's appointment file. Handles the case where either the scratch file or the APPT file does not exist and does not transfer duplicate appointments. The scratch file is deleted after the merge.

Input: (BIN)
(APFILO) Name of appointment scratch file
(APFILE) Name of appointment file

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<APPT file> Merged contents of scratch file and appointment file
<scratch file> purged

Routines Called:
AOPEN, AOPEN', APDEL', APTGET, APTEND, APTINS, ATTN?, BLIMP, DUPCHK, ERROR, FOPAC?, FPURGE, FRENAM

Stack depth R6 (max):

Calls to Error routines (include error number and reason):
Error 68 if scratch file is an invalid file type.

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10		x
R20	x	
R30		
R40	xxxx	xxxx
R50	xxxx	xxxx
R60		
R70		

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called: x			

R12 stack:

Entry	Exit

Routine: APTRIG

File: KR/PS5

Author: MK

Description:
Performs the necessary tasks associated with an appointment interrupt detected by CMPCHK.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<All past due appointments have been triggered>
<Comparator is set for next pending appointment>
<Status and annunciator set for due appointment>

Routines Called:
ALBEEP, ANN.A+, AONOI?, APTACK, GETCLK, NXTAPT, STALRM, STOP?

Stack depth R6 (max): 7

Calls to Error routines (include error number and reason):

Notes:
Generates a HANDIO call with the event V.ATRG

Reg: x = volatile

	0123	4567
R0	xxxx	
R10		
R20		xx
R30		
R40	x	
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b	x	d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called: x			

R12 stack:

Entry	Exit

Routine: STALRM

File: KR/PS5

Author: MK

Description: Sets up the appointment comparator table entry with the currently pending appointment. If OFF ALARM has been executed then returns with out setting up an appointment.

Input: (BIN)
PSSTAT Bit #2 = 1 indicates that appointments have been turned off via OFF ALARM.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Comparator set up with pending appointment

Routines Called:
CMPENT, EVIL, NXTAPT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:
EVIL saves registers 27-57.

Reg: x = volatile

	0123	4567
R0	xx	
R10		
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: ALARM.

File: KR/PS5

Author: MK

Description:
Runtime code for ALARM on/off.

Input:
R12 stack 1 byte; alarm on if value is 1, alarm off if value is 0.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
PSSTAT Bit #2 = 1 if executing ALARM OFF.

Routines Called:
OFALRM, STALRM

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	xx	
R10	ii	
R20		
R30		
R40		
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called: x			

R12 stack:

Entry	Exit

Routine: OFALRM

File: KR/PS5

Author: MK

Description: Disables the appointment entry in the comparator tables, preventing appointments from triggering regardless of the appointment file.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
<Comparator table entyr for appointments>
disabled

Routines Called: STACMP

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		
R30		
R40	xxxx	xxxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b	b	d-BCD
E			b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: CNTRIG

File: KR/PS5

Author: MK

Description: Processes an alarm pattern in response to a continuous alarm interrupt.

Input: (BIN)
[CNALRM] Coninuous alarm number (0 to disable continuous alarm)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
Continous alarm is disabled if CNALRM was 0, else the alarm pattern corresponding to the alarm number in CNALRM is sounded.

Routines Called:
ALBEEP, CMPENT

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0		
R10		
R20		x
R30		
R40	xxxx	xxxx
R50		
R60		
R70		

Status:

	in	out	Legend
Mode	b		d-BCD
E		x	b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
HANDI Called:			

R12 stack:

Entry	Exit

Routine: GETCLK

File: KR/PS5

Author: MK

Description: Gets the binary value of the current time, in seconds from 00:00:00 on 1-Jan-0000. TMBASE and the contents of the real time clock are used.

Input: (BIN)
  <TMBASE> Contains the time at which the clock was last cleared, in seconds from 00:00:00 on 1-Jan-0000.
  <RTCSB> Contains the time since the clock was last set

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
  R40/47 Current time in binary seconds since 00:00:00 on 1-Jan-0000.
  DRP 42

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+--------+       +------------+------------+
|   |0123|4567|     |    |in |out| Legend |       |  Entry     |  Exit      |
+---+----+----+     +----+---+---+--------+       +------------+------------+
|R0 |    |    |     |Mode| b |   | d-BCD  |       |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |       |            |            |
|R20|    |    |     |DRP |   | 42| i-input|       |            |            |
|R30|    |    |     |ARP |   |   |        |       |            |            |
|R40|oooo|oooo|     +-----------------------+     |            |            |
|R50|    |    |     | ROMJSB Needed:   x   |      |            |            |
|R60|    |    |     +-----------------------+     |            |            |
|R70|    |    |     | HANDI Called:         |     |            |            |
+---+----+----+     +-----------------------+     +------------+------------+
```

Routine: GETTD

File: KR/PS5

Author: MK

Description: Gets the current time and date in a standard BCD format.

Input: (BIN)
  <TMBASE> The time at which the clock was last cleared, in seconds from 00:00:00 on 1-Jan-0000.
  <RTCSB> Time since the clock was last set.

Output (include E-Reg, Z-Flag, C-Flag if pertinent):

```
R47   0        \
R46   century   \
R45   year       \
R44   month       \___ BCD data
R43   day         /
R42   hour       /
R41   minute    /
R40   second   /
```

Routines Called:
  DCCLOK, GETCLK

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes: Does nothing other than call GETCLK and DCCLOK.

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+--------+       +------------+------------+
|   |0123|4567|     |    |in |out| Legend |       |  Entry     |  Exit      |
+---+----+----+     +----+---+---+--------+       +------------+------------+
|R0 |    |    |     |Mode|   |   | d-BCD  |       |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |       |            |            |
|R20|    |    |     |DRP |   |   | i-input|       |            |            |
|R30|    |    |     |ARP |   |   |        |       |            |            |
|R40|    |    |     +-----------------------+     |            |            |
|R50|    |    |     | ROMJSB Needed:   x   |      |            |            |
|R60|    |    |     +-----------------------+     |            |            |
|R70|    |    |     | HANDI Called:         |     |            |            |
+---+----+----+     +-----------------------+     +------------+------------+
```

Routine: MULT60

File: KR/PS5

Author: MK

Description: Special purpose multiply routine. Multiplies a 3 byte binary number by 60.

Input: (BIN or BCD)
  R45/47 Number to multiply

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
  (BIN)
  R45/47 Number multiplied

Routines Called:

Stack depth R6 (max): 0

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+--------+       +------------+------------+
|   |0123|4567|     |    |in |out| Legend |       |  Entry     |  Exit      |
+---+----+----+     +----+---+---+--------+       +------------+------------+
|R0 |    |    |     |Mode|   | b | d-BCD  |       |            |            |
|R10|    |    |     |E   |   |   | b-BIN  |       |            |            |
|R20|    |    |     |DRP |   | 45| i-input|       |            |            |
|R30|    |    |     |ARP |   | 55|        |       |            |            |
|R40|    | bbb|     +-----------------------+     |            |            |
|R50|    | xxx|     | ROMJSB Needed:   x   |      |            |            |
|R60|    |    |     +-----------------------+     |            |            |
|R70|    |    |     | HANDI Called:         |     |            |            |
+---+----+----+     +-----------------------+     +------------+------------+
```

Routine: NXTAPT

File: KR/PS5

Author: MK

Description: Finds and returns status for the next appointment that will go off. Opens the appointment file if it can. Finds the earliest appointment in the file that has not triggered.

Input: (BIN)

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
  If there are no untriggered appointments:
    E > 0
  If there is an untriggered appointments:
    E = 0
    R30/31 Address of the start of the appointment file
    R34/35 Address of the end of the appointment file
    R32/33 Address of the next appointment to trigger
    R27 Encoded alarm byte for this appointment
    R2/3 Size of this appointment

Routines Called:
  AOPEN, APSTAT.

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

```
Reg: x = volatile   Status:                       R12 stack:
+---+----+----+     +----+---+---+--------+       +------------+------------+
|   |0123|4567|     |    |in |out| Legend |       |  Entry     |  Exit      |
+---+----+----+     +----+---+---+--------+       +------------+------------+
|R0 |  oo|    |     |Mode| b |   | d-BCD  |       |            |            |
|R10|    |    |     |E   |   | o | b-BIN  |       |            |            |
|R20|    |   o|     |DRP |   | x | i-input|       |            |            |
|R30|oooo|oo  |     |ARP |   | x |        |       |            |            |
|R40|    |    |     +-----------------------+     |            |            |
|R50|    |    |     | ROMJSB Needed:   x   |      |            |            |
|R60|    |    |     +-----------------------+     |            |            |
|R70|    |    |     | HANDI Called:         |     |            |            |
+---+----+----+     +-----------------------+     +------------+------------+
```

Routine: PRNOTE

File: KR/PS5

Author: MX

Description: Sets up processing of note field of a processing appointment.

Input: (BIN)
R32/33 Pointer to processing appointment

Output (include E-Reg, Z-Flag, C-Flag if pertinent):
R25 Terminator indicating what type of processing to be done: 0 for no note field, APNOTE for displaying a note, APCMND for execution of a BASIC command.
<input buffer>
Note to be processed by EXEC code

Routines Called:
APTDSP, KOPY

Stack depth R6 (max):

Calls to Error routines (include error number and reason):

Notes:

Reg: x = volatile

	0123	4567
R0	x	
R10		
R20	xxxx	xo
R30		
R40		
R50		
R60		
R70	x	xxxx

Status:

	in	out	Legend
Mode	b		d-BCD
E			b-BIN
DRP		x	i-input
ARP		x	
ROMJSB Needed: x			
BANDI Called:			

R12 stack:

Entry	Exit

Appointment Mode	CHAPTER 1

1.1 Overview

The appointment mode of Kangaroo is allows the user to enter, examine, modify, acknowledge, and delete appointments. The appointment and time management code occupies about 6000 bytes of the Kangaroo mainframe software. The software structure for the appointment mode uses several components other than the code itself. Some of these components are dedicated to the appointment management code and some are heavily used by other parts of the Kangaroo system. Some of the major software components of the appointment management code include the appointment file, the pocket secretary status byte (PSSTAT), the real time clock (RTC), the time base (TMBASE), the input buffer, and the pocket secretary IO status byte (PSIOST).

This document outlines the major components and operations of the appointment mode. It is intended to supplement the documentation provided with the appointment management code and is not intended to provide a complete description in itself, but rather to be used as an appendix for the code documentation. The appointment mode code and code documentation is listed in the following files: KR/PS1, KR/PS2, KR/PS3, KR/PS4, KR/PS5.

1.2 Time

Time in the pocket secretary can refer to either elapsed time such as 5 days or 126230400 seconds, or a particular time and date such as 11:43:22 AM on September 12, 1987. The former case is fairly common in many of the routines and most frequently occurs as some number of seconds. Another common unit of measure for elapsed time is 61.03515625 microseconds (2^{-14} seconds) which is the unit of the internal system clock. These numbers typically occur as binary values and are converted to BCD or ASCII only for display purposes.

Time and date specifications within the appointment mode are usually specified as either the number of seconds or as the number of 61.03515625 microseconds from midnight (AM) on January 1 of the year 0000. Note, unless stated otherwise, 61.03515625 microseconds will be stated as 61 microseconds throughout the remainder of this document. If the units are seconds then the value will generally occupy five bytes. If the units are 61 microseconds, then the value will usually occupy seven bytes.

The Appointment File	CHAPTER 2

2.1 Overview

The appointment file is where all the appointments are kept. The file is created, opened, purged, and initially accessed through the standard file manipulation routines. The appointments are kept in chronological order (based on trigger time) and in an encoded format (see section 2.3). When the appointment mode is executing, it will have opened the appointment file and set up three pointers identifying the file bounds and current appointment.

2.2 The Appointment File Pointers

The pointers for the appointment file are set up by AOPEN or ACREAT and are generally kept in R30/35 as follows:

```
R30/31   start of appoinment file
R32/33   current appointment in display (usually)
R34/35   next byte after appointment file
```

These pointers are used by many routines in appointment mode and most routines that do not use them will not destroy their contents.

2.3 The Encoded Appointment

The appointments are stored in the appointment file in an encoded format to conserve memory space. The encoded appointment consists of four fields (five for a repeating appointment). The first field (size byte) is one byte long and contains the appointment size and appointment acknowledged flag. The second

field (time field) is five bytes long and contains the time that the appointment is set to go off. The third field (encoded alarm byte) contains the alarm type, alarm number, appointment triggered flag and an appointment processing due flag. If the appointment is a repeating appointment (this is known via the encoded alarm byte) then the fourth field will be a five byte repeat specification field; if the appointment is not repeating then this field does not exist. The last field is the note field and contains a note or BASIC command associated with the appointment.

Encoded Appointment Format

```
       Repeating Appointment            Non-repeating Appointment
     +-----------------------+         +-----------------------+
  0  |       Size Byte       |      0  |       Size Byte       |
     |-----------------------|         |-----------------------|
  1  |                       |      1  |                       |
  2  |      Appointment      |      2  |      Appointment      |
  3  |         Time          |      3  |         Time          |
  4  |         Field         |      4  |         Field         |
  5  |                       |      5  |                       |
     |-----------------------|         |-----------------------|
  6  |  Encoded Alarm Byte   |      6  |  Encoded Alarm Byte   |
     |-----------------------|         |-----------------------|
  7  |                       |      7  |                       |
  8  |      Appointment      |      8  |         Note          |
  9  |        Repeat         |      9  |         Field         |
 10  |     Specification     |     10  |                       |
 11  |                       |     11  |                       |
     |-----------------------|     12  |                       |
 12  |                       |     13  |                       |
 13  |         Note          |     14  |                       |
 14  |         Field         |     15  |                       |
 15  |                       |     16  |                       |
 16  |                       |     17  |                       |
  .  |           .           |      .  |           .           |
  .  |           .           |      .  |           .           |
  .  |           .           |      .  |           .           |
```

2.4 The Size Byte

The first byte of the encoded appointment is the size byte which contains two pieces of information. The msb of the size byte (bit 7) is the appointment acknowledged flag. This bit is set when the appointment is acknowledged (see section 7.1). The

seven low order bits of the size byte (bits 0-6) give the size of the encoded appointment. This value varies due to the variable length note field and the existence or nonexistence of the repeat field. The minimum size for an appointment is seven bytes (no repeat field and no note). The maximum size of an appointment is 81 bytes: a repeating appointment with a 69 byte note, including the note prompt.

2.5 The Appointment Time Field

The time field occupies the 2nd through 6th bytes of the encoded appointment. This field contains a 40 bit positive integer that represents the number of seconds from midnight (AM) on January 1 of the year 0000 to the time that the appointment has been set to go off. The minimum value for the time field is 0 for an appointment 00:00 of 1/1/0000. The maximum value for the time field is 10000 years minus 1 second which equals 315,569,519,999 seconds (497968BD7F hex) for an appointment at 23:59:59 of 12/31/9999.

2.6 The Encoded Alarm Byte

The seventh byte of the encoded appointment contains encoded information giving the alarm type and alarm number as well as status flags describing the current state of the appointment. The two most significant bits of the byte contain the alarm type: 00 for normal appointments (type N), 10 for an immediately repeating appointment (type R), and 11 for a repeat-at-acknowledge appointment (type A) (see section 4.2). Type 01 is unused.

Bit 5 of the encoded alarm byte is the alarm triggered flag. This bit is cleared when the appointment is set up and is set when the appointment is triggered (see chapter 7). Bit 4 of the alarm byte is the processing due flag. This bit is initially cleared when the appointment is set up and is set when the appointment triggers. It indicates that the appointment needs to be processed (see section 7.2), and is cleared when the appointment is processed.

The four low order bits of the encoded alarm byte contain the alarm number. This value is a binary number 0 to 9 corresponding to the number entered along with the alarm type

when the appointment was entered (see section 4.3). The values from 10 to 15 have not been implemented in the base machine but could be used by an enhancement ROM for some special purpose applications.

2.7 The Appointment Repeat Specification

If the appointment is of type R or A (bit 7 of encoded alarm byte set), then the 8th through 12th bytes of the encoded appointment contain a repeat specification consisting of three distinct parameters. They determine when a repeating appointment is to be rescheduled.

The first byte of the repeat specification contains the day-of-the-week parameter. If this byte is zero then the day-of-the-week defaults to a don't care condition and the appointment will repeat irrespective of the day-of-the-week. If this byte is not zero then the left digit contains the day-of-the-week number:

1 for Saturday
2 for Sunday
3 for Monday
4 for Tuesday
5 for Wednesday
6 for Thursday
7 for Friday

The right digit specifies any modifiers to the day-of-the-week as follows:

0 for a normal day-of-week specification (i.e., Mon)
1-5 for a particular week (i.e., Mo3)
8 for a forward relative specification (i.e., Mo+)
9 for a backward relative specification (i.e., Mo-)

All other possibilities are unused (note that in the repeat specification, the normal and forward relative specifications are treated the same).

The second byte of the repeat specification contains the month parameter. This byte specifies how many months are to be added when the appointment is rescheduled. The amount of time that is actually added depends on the current month; thus adding one month to a date in January will add 31 days, but adding a month to a date in April will add 30 days. Thus adding one month to March 31 will result in May 1. The parameter can be as small as 0 and as large as 99.

The third through fifth bytes of the repeat specification contain a number of seconds specifying the days, hours, and minutes field. The minimum value for this field is zero (for 0 days, 0 hours, and 0 minutes). The maximum value for this field is 8,915,940 seconds for 99 days, 99 hours, and 99 minutes (880BD4 hex).

2.8 The Note Field

In non-repeating appointments the note field starts at the eighth byte. In repeating appointments the field starts at the thirteenth byte. Due to the 96 character limit to the input, the note field is limited to a maximum length of 69 bytes. The minimum length of the note field is zero; thus the note field does not exist if no note is specified. If a note is specified then the first character of the note field is the prompt character and is either a "!" if the note field contains a note to be displayed or is a ">" if the note field contains a BASIC command to be executed (see section 7.2).

The Status Bytes	CHAPTER 3

3.1 The Pocket Secretary Status Byte (PSSTAT)

The pocket secretary status byte (PSSTAT) is a variable stored in system memory. The 8 bits of this byte contain eight flags that identify the status information pertinent to the pocket secretary. The msb of this byte (bit 7) is the appointment due flag. This flag is set when an appointment triggers and is reset when all appointments have been acknowledged.

Bit 6 of PSSTAT contains the appointment entry mode flag. If this flag is clear then the appointments are entered in YEAR mode. In YEAR mode the year field of the date is not used and the appointment will be set for some time in the next 12 months. If this flag is set, then the appointments are entered in EXTENDED mode. In EXTENDED mode the year of the appointment can be set to any time in the current century or, if ** is entered, the year can be set to any value from 0000 thorugh 9999.

Bit 5 of PSSTAT contains the date format flag. If this flag is clear, then the dates are entered and displayed in month/day/year format (with / delimiters). If this flag is set, then dates are entered and displayed in day\month\year format (with \ delimiters). In a similar manner, bit 4 of PSSTAT contains time format flag. If this flag is clear, then time-of-day information is displayed in 12 hour mode with an AM or PM indicator. If this flag is set, then time-of-day is displayed in 24 hour mode with the AM or PM indicator being replaced with **.

Bit 3 of the status byte contains the time adjust decrement flag. If this flag is clear, then all internal corrections to the time will be 1/4 second increments. If this flag is set, then all internal corrections to the time will be 1/4 second decrements. This flag is used and maintained by the TIME mode software.

Bit 2 of PSSTAT is the appointment disable flag. If this flag is clear, then the appointments are enabled and operate

normally; otherwise, if the flag is set, the appointments have been disabled. When the appointments are disabled, the appointment mode will operate normally except the comparator entries will be disabled and thus no appointment interrupts will be allowed to occur. In this state the appointments will not go off even though the user can still create, acknowledge, set, delete, and edit appointments in the usual manner. This flag is controlled by the ALARM ON and ALARM OFF commands in EDIT mode.

Bit 1 of PSSTAT contains the time exact flag. If the EXACT time command has not been executed, then this flag is clear and time correction is disabled. Executing an EXACT command in TIME mode will set this flag. Executing a RESET time command will clear this flag. This flag is used and maintained by the TIME mode software.

The lsb PSSTAT is the time enabled flag. If this flag is set, then the time display is enabled and the clock interrupts in time mode will result in the display being updated. If this flag is clear, then clock interrupts in time mode are ignored This flag allows time commands to disable the time display while they use the display to display information or input values.

3.2 The Pocket Secretary IO Status Byte (PSIOST)

The Pocket Secretary IO Status Byte (PSIOST) contains temporary flags which are set up and used in appointment mode. This byte is set up in temporary system memory since it need not be maintained outside of the appointment mode. PSIOST contains three flags with five unused bits.

The msb of PSIOST (bit 7) contains the display appointment flag. This flag is set up by the appointment commands and is passed to the command entry software. If this flag is set, then the command entry software will display the current appointment (pointed at by R32/33); otherwise, the flag is clear and the appointment template is displayed. The flag is also used by some of the commands that need to know whether or not an appointment was in the display. At the initial entry into appointment mode, the flag is set if there is a due appointment to display; otherwise, it is cleared and the template is displayed.

Bit 6 of PSIOST contains the template changed flag. This flag is cleared prior to calling the IO routine to input an appointment command. The flag is set by the appointment input check intercept routine whenever a character is entered in the

display. This flag is checked by those commands that need to know if anything was actually entered in the display, such as the appointment entry command which will ignore the input if no entry was made. This flag is necessary since with the template IO it is not possible to distinguish what was in the display when the input routine was called from what was actually input.

The lsb of PSIOST (bit 0) contains the erroneous input flag. This flag is set by a command that detects an error. In addition to setting the flag, the command sets up the input buffer as it was entered and sets the desired cursor address in R22/23. When the command entry software detects that the erroneous input flag is set, it reports the error (error annunciator and beep) and outputs the erroneous input buffer contents with the cursor at the specified position. The command entry will clear this flag.

Alarm Field Specifications	CHAPTER 4

4.1 Introduction

The alarm field in the appointment specifies some of the actions that are to be taken when the appointment is triggered or processed. The field consists of two characters. The first character specifies the appointment type, and the second character specifies the alarm number. The appointment type specifies the general disposition of the appointment after is has triggered (see section 6.1). The alarm number specifies the tone pattern (if any) and whether or not the pattern will sound continuously.

4.2 Appointment Type

The appointment type is specified in the alarm field by one of the characters N, A, or R corresponding to three different types of appointments with type N being the default. Type N appointments are normal appointments, type R appointments reschedule themselves immediately when they go off, and type A appointments are repeating appointments that wait until they are acknowledged before they will reschedule themselves.

4.2.1 Type N Appointments

Type N appointments are normal appointments which go through the normal process of triggering, processing the note field, and waiting to be acknowledged. After being acknowledged, type N appointments remain in the appointment file where they can be examined, editted, or deleted in the normal manner.

4.2.2 Type R Appointments

Type R appointments are repeating appointments and as such have a repeat interval specified when they are entered. When a type R appointment is triggered it will reschedule itself immediately by invoking the appointment acknowledge software which reschedules repeating appointments. Thus this appointment will not wait to be acknowledged and as such it may be missed if the user is not present when the appointment triggers. This appointment type is nonetheless important for those applications where a process needs to be scheduled at regular intervals without user interaction or in applications like a snooze alarm where an alarm will sound at regular intervals until the appointment is removed. Processing of the appointment takes place even if the appointment has been rescheduled.

4.2.3 Type A Appointments

Type A appointments are also repeating appointments with a repeat interval specified. Type A appointments trigger and are processed in the normal manner; however, when the appointment is acknowledged the appointment will reschedule itself according to the specified interval. Processing will take place normally whether the appointment has been rescheduled or not. This type of appointment is important for those applications where a recurring event must be brought to the attention of the user, such as a reminder to the user to take medicine at four hour intervals or of meetings that occur at 9:00 AM every Monday.

4.3 Alarm Number

The alarm number is the second part of the alarm field. This field contains a number from 0 to 9 and specifies what (if any) tone pattern to sound and whether or not to sound it continuously.

4.3.1 Alarm Tone Patterns

The purpose of the alarm number is to specify which tone pattern to execute. There are six distinct tone patterns associated with the number 1 through 6. These six patterns proceed from a innocuous little beep at 1 to an attention grabbing string of rising and falling tones at 6. Specifying 0 as the alarm number will result in a silent alarm (annunciator but no tones). The alarm numbers of 7, 8 and 9 are the same as the tone patterns of alarm numbers 2, 4 and 6 respectively but these patterns sound continuously (see section 6.2.2). Using the system command BEEP OFF to disable the beeper will override all of the tone patterns except for number 6 (and 9) which will always sound unless the appointments have been disabled; all other alarms will be silenced (but all other processing will still occur).

4.3.2 Continuous Alarms

Alarm numbers 7 through 9 will set up continuous alarms. When a continuous alarm goes off, the appropriate tone pattern is executed and the alarm software sets up a special interrupt that goes off every 15 seconds and sounds the specified tone pattern. This continues until a key has been hit (any key will do). Hitting a key clears the continuous alarm interrupt.

Appointment Mode Commands	CHAPTER 5

5.1 Overview

In appointment mode command entry the display will contain either an appointment template or one of the appointments in the file. A command is entered by entering data from the keyboard (if necessary) and pressing one of the command terminator keys. The terminator keys determine which command is being executed. The clear key and shift delete change their function in appointment mode; the clear restores the template and the shift delete key deletes or edits appointments.

5.2 Exiting Appointment Mode

Pressing any of the mode switch keys will cause the appointment mode to exit. If the [APPT] key is pressed, then the appointment mode will be restarted (the mode switching software will send control back to appointment mode). When appointment mode is exited, the appointment file length is checked and if the length is zero then the file is purged since it contains no appointments. Note that the [ATTN] key is not a mode switching key and is used to acknowledge appointments.

5.3 Examining Appointments

The appointment mode provides commands that scroll up and down through the appointment file allowing the user to view appointments in sequential order. The up arrow key will cause the appointment immediately preceding the current appointment to be displayed. The down arrow key in a similiar manner will cause the appointment that immediately follows the current appointment to be displayed. Both of these keys will stop when they encounter the top or bottom of the file. The shift up arrow key will move the display to the first appointment in the file, and the shift down arrow key will move the display to the last appointment in the file.

Provision has been made for providing an appointment search on the fetch key, but this has not been implemented on the base system. An intercept has been provided that will allow the feature to be added in a plug-in ROM. See the Handi call documentation.

5.4 Clearing the Display

Since appointment information is input on a template, a clear display function would not be useful since it would clear the template information that is necessary to enter any appointment data; therefore, the function of the clear key in appointment mode has been changed so that it will display an appointment template.

5.5 Appointment Acknowledge

The [ATTN] key is used to acknowledge an appointment. When the [ATTN] key is detected, the software checks the display. If an appointment was not in the display or if the appointment that is in the display had been editted, then the software returns to the appointment command entry without acknowledging an appointment; otherwise, the appointment in the display is acknowledged. The software processes involved in acknowledging an appointment are covered in chapter 7.

5.6 Adding Appointments

An appointment is added to the appointment file by entering the necessary information into the appointment template and pressing the [RTN] key. The internal processes that are involved in entering an appointment are covered in chapter 6.

5.7 Editing and Deleting Appointments

The appointment editting and delete features are implemented via the shift delete key. To delete an appointment, the appointment is displayed and [SHIFT] [DEL] pressed. To edit an appointment, the appointment is displayed, any required changes are made, and the [SHIFT] [DEL] is pressed. The software then checks to see if the display has been changed. If the display has not been changed, then the appointment is deleted; otherwise, the appointment is deleted and the new entry is added as though [RTN] had been pressed.

5.8 Processing Appointments

Appointments can, under normal circumstances, be processed only when the software is about to go to deep sleep in order to prevent a conflict between the processing required by the appointment and a running BASIC program, editting, or user calculations; however, in appointment mode the user is not calculating, editting or running a BASIC program. Therefore, appointments with processing pending can be processed from appointment mode through an explicit command. The [RUN] key implements such a command. Pressing [RUN] will in appointment mode will invoke appointment processing.

5.9 Viewing Extended Appointment Information

In appointments there are two pieces of information that do not occur as part of the standard appointment display: the century of an appointment, and the repeat specification of a repeating appointment. A command is provided in the appointment mode that allows the user to examine these missing pieces of information. This command is implemented via [SHIFT] [APPT] key. With an appointment in the display, if the [SHIFT] [APPT] key is pressed, the software will display the four digit year of the appointment and, if it is a repeating appointment, then repeat specification is also displayed. This information is held the display as long as [SHIFT] [APPT] is held down. When the key is released the display returns to the normal appointment display.

Entering Appointments — CHAPTER 6

6.1 Setting Up an Appointment

Appointments are set up by entering information on an appointment template and pressing [RTN] (template IO is covered in section 6.2). When [RTN] is pressed, the software parses the contents of the input buffer (which contains all of the template information) and (if the contents are valid) produces an encoded appointment which is subsequently stored in the appointment file. The method of entering and computing the appointment date varies depending on whether extended mode or year mode is active (this is covered in section 6.1). Section 6.4 covers some of the enhancements in setting up the day and date fields.

6.2 Extended Mode Verses Year Mode

Appointments are set up in either extended mode or year mode. The active mode can be examined or changed via the STATS command in time mode. The default (at coldstart) is year mode. In year mode the year field of the appointment template is protected and the user is only allowed to enter a month and day in the date field. In year mode all appointments are set for some time up to the end of the following year. If the information entered is not compatible with some time before the end of the following year, then an error is reported.

In extended mode the appointments are not required to be in the coming year (and are in fact not required to be in the future or in the current century). The year field is not protected in extended mode and the user may enter a year number, may enter a ** or may let it default. If the field is allowed to default, the appointment is parsed in the same manner as year mode except that the software will continue to search forward in time until a match is found. Only if the day and date fields cannot be matched within the following 400 years will an error occur (the Gregorian calendar repeats every 400 years). If a numeric value

is specified in the year field then it specifies a year in the current century and the software attempts to set up an appointment within this year (else error). A ** in the year field is interpreted as a request to input a 4 digit year; the user will be prompted with "Year? YYYY" and allowed to entered a 4 digit year which fixes the year in which the appointment is set.

6.3 Template IO

The appointment template is a string of characters in the display that identify the various fields of an appointment that the user can specify. The blanks (or other characters) between the fields and the delimiters within the fields (such as the : in the time field) are protected and cannot be typed over by the user; in fact, the cursor will skip over the protected fields stopping only on those spaces that can be typed into. The template's appearance varies slightly depending on the current date mode and time mode, but the position of the fields never change.

6.3.1 Appointment Template Fields

The first two fields are the day and date fields. These fields determine the date of the appointment. The characteristics of these fields and the way in which they interact is covered in section 6.4. The third field is the time field which appears as Hr:Mn ** is the time is in 12 hour mode or as Hr:Mn AM if the time is in 24 hour mode. The hours and minutes can be entered as desired and can be specified as AM, PM, or ** (** for 24 hour mode) regardless of what the current time mode is (the time mode affects the display only). The fourth field is the alarm field which appears as 1N which is the default alarm number and appointment type. The alarm number can be any digit from 0-9 and the type can be N, A, or R.

The last field is the note field which appears as !Note. The ! character at the start of the field is semi-protected. The cursor can be positioned on this space only by backspacing from the position immediately in front of it. Once the cursor is positioned there, the only characters that are allowed to be typed are a ! and a >. The remainder of the field is unprotected and can be filled as desired. If the first character of the note field is a ! then the software interprets the note field as DISP "message" and displays it when the appointment is processed; if the first character is a › then the note field is interpreted as a BASIC command and executed when the appointment is processed.

6.3.2 Input Check Intercepts

In appointment mode input, after every key is hit an intercept routine (AINCHK) is called which does some IO processing on the fly. In this routine the special semi-protection of the first character of the note field is implemented. This routine also flags the clear key and shift delete key as terminators which will bypass the normal processing of these keys allowing the special appointment mode functions to be implemented.

6.3.3 Editting on Protected Fields

When editting a protected field, the insert character and delete character function only within the current field. This prohibits the fields from shifting around while one of the fields is being editted. The [TAB] key and [SHIFT] [TAB] key can be used to skip from field to field. The [TAB] key skips to the start of the next field and the [SHIFT] [TAB] skips back to the start of the previous field (or current field if not already at the start).

6.4 Day/Date Interactions

In specifying an appointment, the day field, date field, and the current time interact to determine the date of the appointment. Unless the year field specifies otherwise, the software starts at the current time and searches forward in time until a match is found with the date field or the search fails. If the day field is not compatible then the match fails and the search continues. If the day field is compatible then the date is accepted and possibly modified depending on the contents of the day field.

A default day field is compatible with any date. A day field may specify a particular day of the week (i.e. Mon) in

which case it is compatible only with dates that are on that particular day of the week. Another possibility for the day field is a specification of a particular day of the week and week of the month; for example: Mo3 will specify the third Monday of the month and this day field would only be compatible with a date that occured on the third Monday of a month. None of the above specifications will modify the specified date.

A final possible format for the day field is relative to the specified date. This format is compatible with any date, but will modify the date to match the specified day. The day can be specified as forward relative with a + or backward relative with a -. For example, Mo+ would be compatible with any date but would change the date to the subsequent Monday (even if the date is on a Monday); while Mo- would likewise be compatible with any date but would change the date to the previous Monday (even if the date is already a Monday). If the date field defaults, then a relative day specification will move forwards or backwards from the current date.

6.5 Repeat Specification Template

When a repeating appointment is specified for the appointment (types R & A), a repeat specification template is displayed to allow the repeat specification to be input. The repeat template has five fields that combine to specify the repeat interval. The first four fields specify the number of months, days, hours, and minutes to add to the old appointment time and date to get the new appointment time and date. The fifth field is the day of the week field (DOW) which differs slightly from the day field in the appointment template. In the repeat specification's DOW, the date is not changed if the day of the week matches a relative specification (i.e. a date on Monday will not be advanced a week by a Mo+ specification). If the day of the week does not match the date than the date is advanced in 24 hour increments until the date and DOW field match; unless the DOW field contains a backward relative specification (i.e. Mo-) in which case the date is moved back in 24 hour decrements until the date and day match. A repeat specification is never allowed to move backward in time; hence, if a backward relative DOW specification would set the new appointment time earlier than the old appointment time then the minus sign is ignored. Thus if a Mo- in the DOW field causes a repeating appointment to back up to before the old appointment time then the appointment is scheduled for the following Monday.

Triggering an Appointment	CHAPTER 7

An appointment is triggered when the comparator interrupts on that appointment's entry. When this happens the comparator service routine sets a flag that is examined fairly frequently and causes the appointment trigger routine to be called (APTRIG). This routine will search the appointment file for an appointment that is past due but has not yet been triggered. If no such appointment is found then the routine will exit without triggering an appointment. If such an appointment is found then the routine will set two bits in the appointment's encoded alarm byte. Bit 5 is set to indicate that the appointment has triggered and bit 4 is set to indicate that the appointment needs processing. The routine also sets bit 7 of the pocket secretary status byte (PSSTAT) indicating that an appointment is due. The routine then turns on the appointment annunciator. The appointment is then checked to see if it is an immediatly repeating appointment (type R). If so then the appointment is acknowledged (by calling the appointment acknowledging routine) which reschedules the appointment. The last step in triggering the appointment is to sound the tone pattern (if any) associated with the appointment.

After the appointment has been triggered, the trigger routine loops back to its start and searches for another appointment that needs processing. This continues until all past due appointments in the appointment file have been triggered. When the seach of the appointment file fails to turn up an appointment that needs to be triggered the routine calls STALRM which sets up the comparator with the next appointment to trigger and the routine returns.

After this routine returns, all past due appointments will have been triggered and the next appointment will be in the comparator. This is true whether or not an appointment was actually triggered.

7.1 Acknowledging an Appointment

After an appointment has triggered it can be acknowledged.

This is done automatically in repeat immediately appointments (type R). In the other two cases the appointment will be either acknowledged or deleted by the user. Deleting an appointment will cause the appointment to be acknowledged before it is deleted. To acknowledge an appointment explicitly the user positions the appointment in the display and presses [ATTN]. When the [ATTN] key is hit the software checks for an unchanged appointment in the display. If the display contains anything other than an unchanged appointment then the software will perform no operation in response to the [ATTN] key; otherwise, the software calls the routine that acknowledges appointments (APTACK).

The first action of the routine is to check the appointment to see that it has triggered and has not already been acknowledged. If the appointment has not triggered or has already been acknowledged then the routine exits without acknowledging; otherwise, the routine acknowledges the appointment by setting bit 7 of the size byte which is the first byte of the encoded appointment. If the appointment is repeating appointment, then the routine invokes another routine which reschedules the appointment according to its repeat specification (RPTADJ).

After the appointment has been acknowledged the routine searches the appointment file for another appointment that has been triggered but has not been acknowledged. If such an appointment is found then the current appointment pointer is set to that appointment and the routine returns; otherwise, the appointment annunciator is cleared and the IO status is cleared to cause the template to be displayed.

7.2 Processing an Appointment

An appointment can cause several different activities to occur. Among these are lighting the appointment annunciator, sounding a tone pattern, rescheduling itself to a future time, displaying a note, deleting itself, and executing a BASIC statement. The first of these activities take place when the appointment triggers and has already been described. These three activities can occur at essentially any time since the appointment triggering occurs at the first comparator check following the interrupt for the appointment in the comparator regardless of what the machine was doing at the time. The other activities cannot be allowed this much freedom; deleting an appointment changes the file structure, and displaying a note or executing a BASIC command may interfer with the display that the user might be using. For this reason these activities have been designated as "processing an appointment," which is allowed to occur only when it will not upset any other activities that the machine may be doing. There are only two times at which an appointment can be processed; one is on explicit request from the user. This is provided in the appointment mode with the [RUN] key.

The other time at which appointments can be processed is when the machine is going to go to sleep. Before the machine is allowed to go to sleep the routine which processes appointments is invoked (APPROC) and any appointments needing processing are found and processed. If there is a note field then it is set up in the input buffer. A special parameter is provided which informs the mode switching software whether the buffer contains a note to be displayed or a BASIC command to be executed. If there is not a note field, then the routine searches for another appointment that needs processing. If the routine finds no appointments that contain a note field to process, then the routine exits to the mode switching software with a status flag set to tell it to go to sleep. In any case the appointments contain a bit in the encoded alarm byte which indicates whether or not the appointment is due. If the flag is set then the software will clear the flag and process the appointment.

Table of Contents

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

LEX Files for Kangaroo:
Joey's Big Book of ROMS

Seth D. Alford
David A. Barrett
June 14, 1982

CHAPTER 1

Introduction

Language EXtension (LEX) files allow additional features to be added to the HP-75 (Kangaroo), such as new BASIC keywords, enhancement of existing statements and interception of system events. LEX files allow Kangaroo's BASIC to access programs written in assembly language, and can greatly enhance Kangaroo's capabilities.

1.1 The Purpose of this Document

This document describes general concepts of, the structure of, how to write, and how the system accesses LEX files.

The intended reader is the programmer who is interested in extending Kangaroo's BASIC with LEX files. We assume that the reader is already familiar with the Kangaroo assembly language.

CHAPTER 2

Concepts--When the System Uses LEX Files

The Kangaroo operating system accesses LEX files at parse time, run time, from intercepts, and with subroutine calls.

2.1 Parse Time

In BASIC, when a user types in a line to Kangaroo and presses [RTN], the Kangaroo operating system translates the line from an ASCII string to an internal representation. This internal representation is referred to as tokenized BASIC or simply as tokens. The translation phase is referred to as parse time.

LEX files can add new keywords to BASIC. The tokenized form of a LEX file created keyword indicates which LEX file created the keyword. Since the operating system may not know how to properly parse these new keywords, the LEX file may have to provide appropriate parsing routines.

2.2 Run Time

Run time refers to the execution of a tokenized BASIC program. When tokens referring to the keyword created by a LEX file are encountered, the operating system transfers control to an appropriate assembly language run time routine contained in the LEX file.

2.3 Intercepts

The operating system allows LEX files to intercept an interesting event, such as a coldstart, HPIL operation, or file command. The operating system uses two methods of intercepts which we refer to as polled and direct. Polled intercepts are also known as HANDI calls (each of the LEX files is polled), and direct intercepts refer to the RAM-based intercept vectors which are called by the interrupt service routines.

2.4 Subroutine Calls

Subroutines within LEX files can also be referred to by other LEX files through subroutine calls. Usually these subroutine calls will be through ROMJSBs. A ROMJSB allows transfer of control in and out of switching ROMs and LEX files. See the System Hooks and Handles document for a complete description of how to use ROMJSBs.

CHAPTER 3

A Standard Structure for LEX Files

3.1 Introduction

This chapter presents a standard structure for LEX files. This standard is more rigid than the structure imposed by the operating system. LEX file handlers, such as Faline, expect LEX files to follow this standard. Other LEX file handlers which already exist or could be written will use the standard to allow LEX files to be moveable and mergeable. In this way libraries of LEX files could be constructed. Because each LEX file follows the standard, different modular LEX files could be joined together to form a larger LEX file which is suited for a particular application.

If you are reading this document in looseleaf form we recommend that you separate the Appendix containing the example LEX file and refer to it as you read the detailed description.

3.2 Specific Structure

3.2.1 General Overview

Shown below, and on page 2 of the example LEX file, is the general structure of a LEX file:

```
+-------------------------------------+
|  HEADER  (order mandatory)          |
+-------------------------------------+
|  identification number              |
|  runtime table address              |
|  keyword table address              |
|  parse table address                |
|  error table address                |
|  intercept routine address          |
+-------------------------------------+
|  TABLES  (order mandatory)          |
+-------------------------------------+
|  runtime table                      |
|  parse table                        |
|  relocation marker                  |
|  keyword table                      |
|  error table                        |
+-------------------------------------+
|  EXECUTABLE CODE                    |
+-------------------------------------+
|  code attributes byte               |
|                                     |
|  intercept routine  \               |
|  parse routines      > any order    |
|  runtime routines   /               |
|                                     |
+-------------------------------------+
            LEX FILE STRUCTURE
```

Note that the order in the header is different from the order in which the tables themselves occur. This is a historical accident. However, both orders are mandatory under the standard.

3.2.2 Identification Number

The identification number is used to distinguish LEX files and ROMS from each other for purposes of ROMJSB and errors. Chapter 7 discusses identifier and other numbers used in LEX files.

3.2.3 Relationship Between the Tables

Entries in the runtime, parse, and keyword tables correspond to each other. The runtime table maps one-to-one onto the keyword table. The first n entries of the parse table map onto the first n entries in both the keyword and runtime tables.

3.2.4 The Runtime Table

Under the standard, the runtime table immediately follows the LEX file header. The runtime table lists addresses of all the runtime routines in a LEX file. The first two bytes of the table are not accessed or used by the operating system, so the address given in the header is offset by 2.

By convention, names of runtime routines are terminated with a period. The runtime routine "SPEAK." in the example illustrates this.

3.2.5 Parse Table

Kangaroo parses keywords created by LEX files according to the attributes associated with the keyword. Attributes indicate whether the keyword is a function or a statement. Attributes for a keyword immediately precede its runtime, NOT parsetime, routine. (see item 267 right before the label "speak." in the example.) The operating system is able to parse functions, but requires LEX files to provide their own parse routines for statements.

The parse table contains addresses of parse routines. In the standard the parse table immediately follows the runtime table. The first two bytes of the parse table are never accessed or used, and as with the runtime table the address of the parse table in the header is offset by 2. The entries in this table correspond to keywords in the ASCII table whose attributes indicate that the keyword requires parsing. The Parser document gives additional information on assigning attributes and parsing routines to call. The HP-85 Assembler ROM manual (page 5-19 to 5-22) also provides useful information.

In the above example "SPEAK#" is a parse routine.

3.2.6 Relocation Marker

Under the standard the relocation marker immediately follows the parse table. This relocation marker is a two byte quantity equal to RELMAR, a global address. Everything before the relocation marker in the LEX file is altered it moves to or from ROM or RAM. In ROM these addresses are absolute whereas in RAM the are relative to the starting address of the LEX file.

3.2.7 Keyword Table

The keyword table, following the relocation marker under the standare, lists the new keywords created by the LEX file. The last character in each keyword has its most significant bit set by the KARMA ASP instruction, as shown in the example LEX file. The end of the table is indicated by FF hex. The keyword table is searched by the scanner/parser. Each string in the keyword table corresponds to an address in the runtime table, and may correspond to an address in the parse table.

In the above example the last character of the "SPEAK" keyword has its high bit set. The runtime routine "SPEAK." and the parse routine "SPEAK#" correspond to the "SPEAK" keyword (each item appears first in its respective table.)

3.2.7.1 Keyword Searching

When the system attempts to process a keyword, the parser searches each keyword table in order until a match is found. Table searching proceeds from RAM LEX files, to ROM lex files and lastly to the operating system keyword tables. The parser treats all LEX file keyword tables as if they were concatenated and followed by the operating system tables.

3.2.7.2 The "Lockout" Problem

Let X and XY be two keywords present within Kangaroo. If X appears in a keyword table in a LEX file before XY, the parser will match on input string containing XY to X, subsequent parsing will attempt to process Y (which will probably fail). The keyword XY has been "locked out"; the user cannot access it. This particular problem can be corrected by placing XY preceding X.

Since the parser views all keywords in a single list, lockouts can occur between LEX files or between a LEX file and the operating system.

An example of this problem occurs with the keyword "ONE", defined in a LEX file. ONE locks out ON ERROR; ON ERROR would not parse.

When choosing keywords the programmer must be aware of this problem.

3.2.8 Error Table

The error table allows LEX files to create and use their one error messages. The error table follows the keyword table under the standard. This table consists of a starting error number and strings which define error messages. Each string corresponds to a successive error number. In the LEX file example, the "malfunction malfunction!" error message is generated by calling ERROR with errnum+1. The most significant bit of the last character of each of the error message strings is set using ASP instructions. The table is terminated with FF. Chapter 7 discusses how to which errornumbers to use. A LEX file cannot access error messages in other LEX files.

3.2.9 Code Attributes

Code attributes immediately precede the intercept routine under the LEX file standard. The code attributes give information about the LEX file. The operating system has no knowledge of the code attributes; these will only be used by the LEX file handlers. The following table describes the usage of the bits of the attributes; if a bit is set then the code has the corresponding attribute.

Bit	Description
0	RAMable (bit 2 must also be set). The code will run correctly in RAM.
1	ROMable The code will run correctly in ROM.
2	Position independent code
3	Mergeable (bits 2 & 4 must also be set). This LEX file can be merged with another LEX file. This attribute is not completely defined.
4	LEX identifier number independent. The LEX id can be changed by a LEX file handler and the code will still run correctly. This implies that the LEX id occurs ONLY in the header and nowhere else.
5-7	Reserved for future use (set to 0).

CODE ATTRIBUTES BYTE

3.2.10 Executable Code

Under the standard, the intercept routine, parse routines, and runtime routines follow the attribute byte in that order. Intercepts are described in the next chapter. Parse and runtime routines are discussed here; additional information can be found in the Parser, Interpreter, and Entry Point documentation.

3.2.10.1 Parse Routines

Parse routines are provided for each keyword which is a statement. Parse routines are called at parse time to translate a user input into tokens. Note that keywords defined by their attributes to be functions do not require user written parse routines.

3.2.10.2 Runtime Routines

Runtime routines are provided for every keyword. Based upon the attributes for the keyword and the result of a possible parse routine, runtime routines pop parameters from the R12 stack. Runtime routines then perform appropriate manipulations on the parameters and push any results onto the R12 stack.

CHAPTER 4

Intercepts

The operating system uses two methods of intercepts: direct and polled.

4.1 Direct Intercepts

The Kangaroo hardware is capable of generating interrupts for the keyboard, the comparator, the power supply, and HPIL. When one of these interrupts occur, control is transferred to an address contained in an interrupt vector. The interrupt vectors are located at the start of ROM. The addresses in the interrupt vectors refer to interrupt routines, which are located somewhere in ROM or RAM. Since the interrupt vectors are in ROM there is no way of changing the addresses of the interrupt routines.

Direct intercepts are one method provided to allow for future expansion. At or near the start of each of the interrupt routines is a subroutine call to an intercept vector. Each intercept vector is 8 bytes long, and is located in the global area in RAM. The system initializes the intercept vectors with RTNs. To modify interrupt handling a LEX file can change the intercept vector.

4.1.1 Problems with this Method

A major drawback with this technique is that only one LEX file may use an intercept at any given time. Coordination in using an intercept between two different LEX files is difficult.

Another major drawback is that ROMJSB cannot be used at interrupt time. The ROMJSB code is not re-entrant; successive intercepts can occur while a ROMJSB is executing. Fixing this problem is possible by using RAM-based code. Appendix E, which contains code from RY/JSP, presents a possible approach.

Direct intercepts should only be used when the problem at hand requires it and when speed is absolutely necessary. In many cases a polled intercept could be used instead.

4.2 Polled Intercepts--HANDI Calls

HANDI calls are generated at many interesting events. Each of the LEX files present is polled to determine if it can handle or is interested in the event. The LEX file can perform special parsing, modify input, and so forth. Because each LEX file is polled, we refer to HANDI calls as polled intercepts. By referring to the header in each LEX file, HANDI is able to find and call each LEX file's intercept routine.

The interesting events upon which HANDI calls are generated are numbered. A list of HANDI events can be found in KR/GLO, and is reproduced below. Additional information can be found in the HANDI call and the System Hooks and Handles documentation.

event	value	error	raison d'etre
V.AACK	42	none	Acknowledge an appointment
V.ACC#	46	none	ASSIGN#: wrong type file
V.ADDR	45	none	PIL: get address for unknown name
V.AFMT	39	none	Format display of appointment
V.AKEY	36	none	Process input terminator in APPT
V.ALLO	8	none	Allocate token with class > 56 octal
V.APRC	41	none	Process an appointment
V.APTO	35	none	Start of APPT command loop
V.ARTN	38	none	Exit APPT mode
V.ASN#	25	none	PIL: assign# with device name
V.ASSN	24	none	PIL: assignio hook for I/O rom
V.ATRG	43	none	Trigger an appointment
V.CARD	50	none	Doing card copy, E=1 (read), E=0 (write)
V.CHAR	56	none	Character output with undefined ROUTE
V.CHED	20	none	CHEDIT: character editor
V.CLOK	34	none	Clock/Stopwatch Interrupt
V.COLD	0	none	Coldstart (power on)
V.CRUN	18	none	CRUNCH: start of interpreter loop
V.DALO	9	none	De-allocate token with class > 56 octal
V.DEC	10	none	Decompile token with class > 56 octal
V.DIM	11	none	Dimension for string arrays
V.EALO	7	none	Environment allo (undef access method)
V.ENDL	57	none	EOL output with undefined ROUTE
V.ERR	63	none	ERROR routine called
V.ETRG	29	15	External comparator trigger #1,#2,#3
V.FILE	49	60	A file command (use TOKEN to identify)
V.LFTY	59	none	Translate internal name to LIF format id
V.LOOP	19	none	PIL: get control of loop
V.PAR	21	none	Start of PARSER
V.PARA	12	none	Strange function parameter
V.RFTY	60	none	Translate LIF format id to internal name
V.RSTN	4	none	RESTEN (Restore environment)
V.SLEE	3	none	Goto deep sleep
V.SPEC	48	60	Filespec syntax error
V.SPY	5	15	Interruptable point in program (SVCWRD)
V.SRQR	23	none	PIL: service request received
V.STAL	6	none	Interpreter stops (opposite of V.CRUN)
V.STRA	22	none	PARSER: strange data type
V.TMCX	51	92	Time mode command extensions
V.TYPE	16	none	Typename extension in EDIT and ASSIGN
V.UNKD	26	none	ASSIGN#: unknown data type
V.VOLT	62	none	Volatile file purge at warmstart
V.WAIT	32	none	WAITKY: We're waiting for a key
V.WARM	1	none	Warmstart (up from deep sleep)

HANDI EVENTS

4.2.1 Receiving a Polled Intercept

NOTE

> Historically the intercept routine was referred to as the initialization routine. Initialization is a misnomer. This name may still persist in some parts of the documentation.

HANDI calls the intercept routine with the event number contained in R0 (1 byte). To intercept an event, the LEX file should examine R0. If R0 does not have the event number, the intercept routine should immediately return. HANDI is called quite often; failure to return quickly upon uninteresting events will affect the speed of the entire system.

As can be seen in the LEX file example, the value of R0 is compared with the event V.WARM (warmstart), and the intercept routine immediately returns if R0 is not not equal to V.WARM. If this intercept is called at V.WARM it displays the message 'Kangaroo at your service!' and returns.

The HANDLD flag is used to indicate to the operating system that an event was successfully handled by a LEX file. Putting 0 into HANDLD indicates that the event was handled and terminates further polling. Clearing HANDLD may also cause or prevent system action, depending on the HANDI call. Writing 0 to HANDLD after a V.VOLT HANDI call, for example, will prevent a volatile file from being purged.

In the example above, the intercept routine does not set the HANDLD flag. After it returns to the operating system polling of LEX files with the event V.WARM continues. The HANDLD flag should NOT be set for many of the intercepts such as coldstart or warmstart since it would prevent other LEX files a chance at the event. The HANDI documentation should be consulted for HANDLD information for a specific event.

4.2.2 Other Parameters for Intercepts

The intercept routine may alter R0-3, the ARP, DRP, status, and the E register. The following summarizes input and output conditions for intercept routines.

Input:

```
BIN mode set
DRP 0
R0          = Event number
ROMFL       = Event number
ROMPTR      = Base address of this LEX file
TOKEN       = A pointer to the last token called by the
              interpreter.  If TOKEN points to EROMTK,
              then it points to 4 bytes as shown:

              +--------+--------+--------+--------+
              | EROMTK |   ROM NUMBER    | Token  |
              +--------+--------+--------+--------+
  TOKEN   +       0        1        2        3
```

Output:

```
BIN mode set
HANDLD      = Clear to terminate polling.
              Non-zero to continue polling.
              The default is to continue; no action
              is needed in this case.
```

4.2.3 Generating a Polled Intercept

An example HANDI call:

```
JSB  =HANDI
VAL  EVENT#
BYT  ERROR#
```

HANDI polls each ROM's intercept routine to determine if it is interested in the event referred to by EVENT#. If none of the ROMS clears the HANDLD flag then the error ERROR# is generated.

HANDIO is similar to HANDI except that it does not generate an error. HANDIO is called:

```
JSE  =HANDIO
VAL  EVENT#
```

CHAPTER 5

Types of LEX Files

LEX files can exist in ROM or RAM.

5.1 ROM-Based Files

A plug-in ROM is formatted as a small file system similar to the system memory. Plug-in ROMs are 8K bytes long.

Address	Item
6000	RMHEAD--ROM Existence Header
6002	File Directory
	LEX File
	Other Files
	Checksum

ROM FORMAT

5.1.1 Existence Header

The ROM Existence header is used to indicate that a ROM is plugged in. The operating system assumes that the hardware will never produce the RMHEAD number when a ROM is not plugged in.

5.1.2 File Directory

The directory in a ROM consists of entries in the order the files appear in ROM followed by an end-of-directory marker. Directory entries are 18 bytes each (equated to DRENSZ) and are arranged as follows:

```
+------------------------------------------------------------------+
| Name  Offset                 Description                          |
+------------------------------------------------------------------+
|                                                                  |
| DR.LOC  0/1  Absolute address of the file in the ROM             |
|                                                                  |
| DR.SIZ  2/3  Size of the file in bytes including the  PCB but    |
|              not the divider byte.  If the  entire ROM is one    |
|              LEX file then this size should be no larger than    |
|              8169d since the rest is ROM overhead such as the    |
|              existence header, directory entry, and checksum.    |
|                                                                  |
| DR.TYP   4   Type of file, access permission.  See KR/GLO and    |
|              the  Kangaroo  Memory  Management document for a    |
|              list of possible file types.  The RAM bit should    |
|              be set appropriately.                               |
|                                                                  |
|              Access bits:                                        |
|                                                                  |
|              +-----+-----+-----+-----+-----+-----+-----+-----+   |
|              | RAM | RUN | EDT | LST | PUR | COP | LIN | TOK |   |
|              +-----+-----+-----+-----+-----+-----+-----+-----+   |
|                 7     6     5     4     3     2     1     0      |
|                                                                  |
| DR.TNM   5   Type of file, name of file type for CAT.            |
|              See KR/GLO for a list of mainframe defined          |
|              file type names.  For additional file types         |
|              see Custom Products in Marketing.                   |
|                                                                  |
|              File Type name bytes:                               |
|                                                                  |
|              A  APPointment             R  ROM image             |
|              B  BASic                   T  TEXt file             |
|              I  LIf type 1 file         V  VOLitile              |
|              L  LEX file                ?  Strange type          |
|                                                                  |
| DR.DAT  6/9  Date of creation in internal Kangaroo               |
|              form. (Number of elapsed seconds since 1900)        |
|                                                                  |
| DR.NAM 10/17 Name of the file right filled with blanks.          |
|              Legitimate file names are determined by FGETNM.     |
|              System files should have lowercase names.           |
|              Volatile files are not allowed in ROMs .            |
|                                                                  |
+------------------------------------------------------------------+
```

FILE DIRECTORY FORMAT

The last directory entry must correspond to the last file in the ROM. This is so that the size of the used portion of the ROM can be found. An end of directory is indicated by two bytes of 0 following the last directory entry.

5.1.3 The LEX File

ROM-based LEX files reside in the switchable plug-in address space 6000H-7FFFH. Absolute code may be used and absolute addresses are used in the header table, parse table, and runtime table. (See the next chapter for writing ROM-based assembly language.)

One and only one LEX file must be present in a plug-in ROM. The LEX file must be the first file in the directory, and under the standard must also immediately follow the directory with the following exception: if the LEX file is absolute, it may leave space between the directory and itself for future directory expansion. The identifier number in the LEX file is used as a unique identifier for the ROM. No other ROM or LEX file in the system can have this number.

A ROM-based LEX file will run in RAM only if the code is position independent. (See the section on Hybrids and the chapter on Coding Practices.) To prevent a system crash, the COPY bit in the file access byte in the directory entry for a ROM LEX file should be set to 0.

5.1.4 Other Files

Any other non-LEX files may also appear in a plug-in ROM, such as BASIC programs. BASIC programs may be executed from the ROM space, but files of other types in ROM may only be copied to RAM and are otherwise inaccessable to the system. For further information on BASIC programs in ROM, consult the Kangaroo Memory Management document. All files, including the LEX file, but not the last file, must have a byte following it to separate it from the following file. The value of this byte is undefined.

5.1.5 Checksum

The checksum is used by the diagnostic ROM and other programs to ensure that the plug-in ROM is working correctly. Under the standard the checksum appears immediately following the last byte in the last file in the ROM. The checksum is computed such that an 8 bit end-around-carry sum of all of the bytes in the ROM will yield OFFH.

KLOUT is a program which computes checksums of ROMs. It takes an absolute output file from KLINK and produces a core image with checksums. KLOUT searches for BSS instructions to replace with checksums. Since the BSS instruction generates peculiar code, it should only be used for this purpose. KR%SUM, a file containing only a BSS 1, can be linked such that it is the last file in a plug-in ROM. KLOUT will find the BSS code from KR%SUM and replace it with a checksum, thus guaranteeing that the ROM conforms to the standard.

5.2 RAM-Based LEX Files

RAM-based LEX files exist in the system's main memory. Since the operating system can move all files at will, RAM-based LEX files must be position independent. (See the next chapter.) All addresses in the header table, runtime table, and parse table are made relative to the start of the LEX file.

5.3 Hybrids

A LEX file which moves around in memory can be difficult to debug. One way around this problem is to do most of the debugging in ROM and then modify the LEX file for RAM; position independent code can be run from ROM. To go from ROM to RAM the LEX file must be modified so that its tables are relative to the start of the LEX file.

One convenient way to do this is to illustrated in the example LEX file. BASE is the base address of the LEX file, while RMBASE is used as the base of the address tables before RELMAR. For RAM based LEX files, RMBASE is made the same value of BASE by removing the 'equ 0' part of the line; this makes the table addresses realative. Also RAM based files should have the ABS and LOC statements removed from the first two lines of the file. ROM based lex files (like the example) have absolute tables so RMBASE is set to 0. Appendix C discusses a method for movement of LEX files into RAM.

CHAPTER 6

Coding Practices

6.1 ROM-based LEX Files

Table addresses preceding RELMAR in ROM-based LEX files are absolute.

A ROMJSB may be required to call a subroutine in another LEX file or in the operating system. Beware of passing a pointer to a parameter to a subroutine which requires a ROMJSB. If the parameter is in a switchable ROM and the ROM is switched out, invalid data will be used instead of the parameter.

A ROM-based LEX file must also not access data in the ROM which is not part of the ROM's file system since this area is undefined.

6.2 RAM-based LEX Files

A RAM-based LEX file can move whenever a file below it in memory changes size, is purged, or is created. Operating system files which change size include the workfile, calcprog, iofile, and devfile.

Because of this movement, the operating system requires table addresses preceding RELMAR in RAM-based LEX files to be relative to the start of the LEX file. This can be done by assembling from a base address of 0 or by subtracting a base address from all addresses in the table. In addition, LEX file movement requires position independent code, which adds several additional constraints.

Internal table references, subroutine jumps, and GTOs within a LEX file must be indexed. Indexing is done using ROMPTR, a global location which contains the start of the currently executing LEX file. Several examples will illustrate the use of ROMPTR with indexing.

6.2.1 Example 1

```
       LDMD R20,=ROMPTR          Get the start of this LEX file
       SBM  R20,=BASE            Now R20 is an index to use with
*                                references to other labels in
*                                the LEX file
       JSB  X20,DEST
*                                DEST is an internal subroutine
       LDMD R30,X20,TABLE        TABLE is an internal data
*                                table within the LEX file
```

6.2.2 Example 2

```
       LDMD R22,=ROMPTR
       JSB  X22,(DEST-BASE)
*                                DEST is an internal subroutine
        .
        .
        .
DEST   POMD R2,-R6               pop the return address;
*                                this makes the above
*                                subroutine jump into a GTO
```

6.2.3 Example 3

```
       LDMD R24,=ROMPTR
       SBM  R24,=(TABLE-BASE)
*                                R24 now points to the
*                                start of the TABLE
```

Altering the size of a file will cause movement of other files which reside after the altered file. Routines which alter file size should be called using ROMJSB, which will automatically update its return address and ROMPTR to compensate for the new location of a LEX file.

6.2.4 PC-Relative Addressing

Using PC-relative addressing in Kangaroo is difficult because the PC is undefined at the start of an instruction. This is because the CPU does not always update the PC when an instruction is executed. All Capricorn CPU instructions maintain an amount to update the PC in the N-register, a special 4-bit counter. Whenever a byte of an instruction is fetched, the N-register is incremented. Most instructions will add N to the PC, clear N and output the PC to the bus before the next instruction is fetched. However, the following special class of instructions do not update the PC: ARP, DRP, DCE, ICE, and CLE. All other instructions update the PC, but the programmer cannot assume that the the PC is not updated after a special class instruction. This is because an interrupt can occur which will update the PC.

Thus, the programmer must ensure that the PC is updated to use PC-relative addressing. The NUP instruction is useful for this.

CHAPTER 7

Identifier and Other Numbers

Do not simply assign a number of your choice to a LEX file you are writing. Each LEX file number should refer to a unique LEX file in order to avoid conflicts and a possible software crash in Kangaroo. LEX file identifier numbers are assigned by Custom Products in Marketing. Identifiers available for experimental or un-released LEX files are 100 to 199D. Under the LEX file standard, error numbers will range from 150 to 255 for all LEX files.

- 25 -

APPENDIX A

Example ROM-based LEX File

```
KARMA 06/09/82=============================================================
 ITEM    LOC    OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA    6/14/1982  9:51 AM  PG  1
===========================================================================
>>>>>>>>       Example LEX file                                          <<<<<
   2 0000                   abs              ; Used only for ROM
   3 0000                   loc  6000H       ; Used only for ROM
   4 6000             ****************************************************
   4 6000             *
   5 6000             *         A S S E M B L Y    I N F O R M A T I O N
   6 6000             *
   7 6000             * Note the careful use of upper and lower case here.  I
   8 6000             * use uppercase to denote external  and  global symbols
   9 6000             * and declared entry points.
   9 6000             *
   9 6000             ****************************************************
  10 6000                   ext  ERRORR      ; An Error reporter.
  11 6000                   ext  NUMVA+      ; Parse expression w/pre&post sc
  12 6000                   ext  ONEB        ; Pop binary# from r12 at runtim
  13 6000                   ext  OUTCHR      ; Output a chr to all disp devic
  14 6000                   ext  SFSCAN      ; Safe scan routine.
  15 6000                   ext  SYSJSB      ; Safe way to call system.
  16 6000                   ext  UNSEE       ; OK to clear LCD routine.
  17 6000
  18 6000             tylex  equ  00001101B  ; ROM,Purge,Copy,Token.
  19 6000
  20 6000             ****************************************************
  20 6000             *
  21 6000             *    File Directory  (Used only  for ROM LEX files)
  21 6000             *
  21 6000             ****************************************************
  22 6000
  23 6000 E3 1C             def  RMHEAD               ; Yes, there is a ROM.
  24 6002 16 60             def  lxstrt               ; Beginning of Lexfile.
  25 6004 EC 00             def  (lxend-lxstrt)       ; Size of file.
  26 6006 0D                val  tylex
  27 6007 4C                val  TYNLEX               ; LEX file in RAM.
  28 6008 90 B3 F3 9A       byt  90H,0B3H,0F3H,9AH    ; Date in sec since 1900
  29 600C             *                               ; 05/18/82  16:28:00
  30 600C 4C 45 58 45       asc  'LEXEXAM '           ; File name.
  30 6010 58 41 4D 20
  31 6014 00 00             def  (0)                  ; end of directory.
  32 6016
```

```
KARMA 06/09/82=============================================================
 ITEM    LOC    OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA    6/14/1982  9:51 AM  PG  2
===========================================================================
>>>>>>>>       Address Tables                                            <<<<<
  34 6016             ****************************************************
  34 6016             *
  35 6016             *             LEXEXAM Address Tables
  35 6016             *
  35 6016             ****************************************************
  36 6016
  37 6016             lexid  equ  0100d              ; The id of this LEX fil
  38 6016             errnum equ  0150d              ; Starting error number.
  39 6016
  40 6016             ****************************************************
  40 6016             *
  41 6016             *           L E X    F I L E    H E A D E R
  41 6016             *
  41 6016             ****************************************************
  42 6016             base                           * Assembly-time base add
  43 6016             rmbase equ  0                  * remove 'equ 0' for RAM
  44 6016             lxstrt
  45 6016 64 00             def  lexid               ; Number of this LEX fil
  46 6018 20 60             def  (runtab-rmbase-2)   ; Address of runtime tab
  47 601A 2C 60             def  (keywrd-rmbase)     ; Address of keyword tab
  48 601C 24 60             def  (partab-rmbase-2)   ; Address of parse table
  49 601E 37 60             def  (errmsg-rmbase)     ; Address of error table
  50 6020 60 60             def  (intcpt-rmbase)     ; Addr. of intercept cod
  51 6022
  52 6022             * Note  that  the addresses  of the  Runtime  and  Pars
  53 6022             * tables  are offset by two.  This is because the syste
  54 6022             * will  ignore  the  first table entry  in those tables
  55 6022
  56 6022             ****************************************************
  56 6022             *
  57 6022             *             R U N T I M E    T A B L E
  57 6022             *
  57 6022             ****************************************************
  58 6022             runtab
  59 6022 BC 60             def  (speak.-rmbase)     ; 1st keyword code addre
  60 6024 D4 60             def  (error.-rmbase)
  61 6026
  62 6026             ****************************************************
  62 6026             *
  63 6026             *               P A R S E    T A B L E
  63 6026             *
  63 6026             ****************************************************
  64 6026             partab
  65 6026 8B 60             def  (speak#-rmbase)     ; Addr. of parse-time co
  66 6028 98 60             def  (error#-rmbase)
  67 602A FF FF             def  RELMAR              ; Relocation Marker.
  68 602C
  69 602C             *     All items above RELMAR must be addresses only.
  70 602C
```

```
KARMA 06/09/82=============================================================
 ITEM    LOC    OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA    6/14/1982  9:51 AM  PG  3
===========================================================================
>>>>>>>>       Data Tables                                               <<<<<
  72 602C             ****************************************************
  72 602C             *
  73 602C             *             K E Y W O R D    T A B L E
  73 602C             *
  73 602C             ****************************************************
  74 602C
  75 602C             keywrd
  76 602C 53 50 45 41       asp  'SPEAK'              ; Invoking keyword #1.
  76 6030 CB
  77 6031 45 52 52 4F       asp  'ERROR'              ; Invoking keyword #2.
  77 6035 D2
  78 6036 FF                val  ff                   ; End of keyword table.
  79 6037
  80 6037             ****************************************************
  80 6037             *
  81 6037             *             E R R O R    M E S S A G E S
  81 6037             *
  81 6037             ****************************************************
  82 6037
  83 6037             errmsg
  84 6037 96                val  errnum               ; 1st error message numb
  85 6038 73 6F 72 72       asp  'sorry charlie!'     ; Error message errnum +
  85 603C 79 20 63 68
  85 6040 61 72 6C 69
  85 6044 65 A1
  86 6046 6D 61 6C 66       asp  'malfunction malfunction!' ;
  86 604A 75 6E 63 74
  86 604E 69 6F 6E 20
  86 6052 6D 61 6C 66
  86 6056 75 6E 63 74
  86 605A 69 6F 6E A1
  87 605E             * Note that above is the longest displayable error msg.
  88 605E FF                val  ff                   ; End of error messages.
  89 605F
```

```
KARMA 06/09/82=====================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG  4
===================================================================
>>>>>>>>      Code Attributes                                  <<<<<
  91 605F                 ***********************************************
  91 605F                 *
  92 605F                 *          C O D E     A T T R I B U T E S
  93 605F                 *
  94 605F                 * The byte  immediately preceeding the intercept code i
  95 605F                 * the code attributes byte.  This byte supplies informa
  96 605F                 * tion about the code used in this  LEX file as follows
  97 605F                 *
  98 605F                 *    0 = RAMable; bit 2 must also be set.
  99 605F                 *    1 = ROMable
 100 605F                 *    2 = Position-Independent code
 101 605F                 *    3 = Mergable
 102 605F                 *    4 = LEX ID Independent code
 103 605F                 *    5 = (reserved for future use)
 104 605F                 *    6 = (reserved for future use)
 105 605F                 *    7 = (reserved for future use)
 105 605F                 *
 105 605F                 ***********************************************
 106 605F
 107 605F 1F                     byt  00011111B          ;All of the above.
 108 6060
```

```
KARMA 06/09/82=====================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG  5
===================================================================
>>>>>>>>      Intercept Routine                                <<<<<
 110 6060                 ***********************************************
 110 6060                 *
 111 6060                 *        I N T E R C E P T     R O U T I N E
 112 6060                 *
 113 6060                 *   The intercept routine may trash: r0-3, arp, drp
 114 6060                 *   and  E register, and status.  Since HANDI saves
 115 6060                 *   and restores them on the r6 stack,  they cannot
 116 6060                 *   be used to pass out parameters.
 117 6060                 *
 118 6060                 *   Inputs:
 119 6060                 *
 120 6060                 *     bin
 121 6060                 *     drp 0
 122 6060                 *     R0       = Event number
 123 6060                 *     (ROMFL)  = Duplicate copy of event number.
 124 6060                 *     (ROMPTR) = Base address of this LEX file.
 125 6060                 *     (TOKEN)  = A pointer  to the  last token called b
 126 6060                 *                the interpreter.  If it is EROMTK, the
 127 6060                 *                it points to four bytes as shown below
 128 6060                 *
 129 6060                 *               +--------+--------+--------+--------+
 130 6060                 *               | EROMTK |    ROM NUMBER   | Token  |
 131 6060                 *               +--------+--------+--------+--------+
 132 6060                 *         TOKEN +    0        1        2        3
 133 6060                 *
 134 6060                 *     All other input will depend upon the intercept.
 135 6060                 *
 136 6060                 *   Output: bin
 137 6060                 *          (HANDLD) = clear to terminate polling.
 138 6060                 *                     non-zero to continue polling.
 138 6060                 *
 138 6060                 ***********************************************
 139 6060
 140 6060                 intcpt
 141 6060                        drp  !0                  ; From scanner.
 142 6060 C8 01                  cmb  r0,=V.WARM          ; Is it a warmstart?
 143 6062 F6 26                  rne                      ; NO!
 144 6064 5E B1 A3 82            ldmd r36,=ROMPTR         ; Load my base address.
 145 6068
 146 6068 1E C6 C8 00            jsb  x36,(mesout-base)  ; Tell user about it.
 147 606C 4B 61 6E 67            asc  'Kangaroo at your service!'
 147 6070 61 72 6F 6F
 147 6074 20 61 74 20
 147 6078 79 6F 75 72
 147 607C 20 73 65 72
 147 6080 76 69 63 65
 147 6084 21
 148 6085 0D 8A                  def  crlf!
 149 6087 CE FF FF               jsb  =UNSEE              ; OK to undisplay.
 150 608A 9E                     rtn
 151 608B
```

```
KARMA 06/09/82=====================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG  6
===================================================================
>>>>>>>>      Parse Routines                                   <<<<<
 153 608B                 ***********************************************
 153 608B                 *
 154 608B                 *            P A R S E     R O U T I N E S
 155 608B                 *
 156 608B                 * Input:  (from scanner)
 157 608B                 *         bin
 158 608B                 *         r10     = Pointer to ASCII text input.
 159 608B                 *         r14     = Rom token number.
 160 608B                 *         r15     = 0 (for multibyte adds).
 161 608B                 *         r41/42  = Two Byte ROM id number.
 162 608B                 *         r43     = ROM token number.
 163 608B                 *         r47     = Primary Attributes.
 164 608B                 *
 165 608B                 * Output: The R12 increasing stack contains the tokens
 166 608B                 *         corresponding to the keywords in this file.
 167 608B                 *
 168 608B                 * NOTE: The above registers must be correct on exit in
 169 608B                 *       addition to maintaining r4-13, r16/17.
 169 608B                 *
 169 608B                 ***********************************************
 170 608B
 171 608B                 ***********************************************
 171 608B                 *
 172 608B                 * Parse one  keyword with no parameters.  Put the token
 173 608B                 * onto the r12 stack in increasing addresses as follows
 174 608B                 *
 175 608B                 *       ROM switch token (EROMTK)
 176 608B                 *       ID number of this LEX file
 177 608B                 *       Ordinal of corresponding keyword.
 178 608B                 *  r12 -->
 178 608B                 *
 178 608B                 ***********************************************
 179 608B                 speak#
 180 608B 42 B1 A3 82            ldmd r2,=ROMPTR          ; Fetch base address.
 181 608F 02 C6 95 00            jsb  x2,(prslex-base)    ; Save lex switch tokens
 182 6093 CE FF FF               jsb  =SFSCAN             ; Get next token safely.
 183 6096 F0 0D                  jmp  pushme
 184 6098
 185 6098                 ***********************************************
 185 6098                 *
 186 6098                 * Parse one keyword with one numeric parameter.
 186 6098                 *
 186 6098                 ***********************************************
 187 6098                 error#
 188 6098 42 B1 A3 82            ldmd r2,=ROMPTR          ; Fetch base address.
 189 609C 02 C6 95 00            jsb  x2,(prslex-base)    ; Save lex switch tokens
 190 60A0 CE FF FF               jsb  =SYSJSB             ; Prescan, get a numeric
 191 60A2 FF FF                  def  NUMVA+              ; expression, postscan.
 192 60A5                 pushme
 193 60A5 6C 06 E3               pomd r54,-r6             ; Get lex switch tokens
 194 60A8 0A E5                  pumd r54,+r12            ; and put then onto r12.
 195 60AA 9E                     rtn
 196 60AB
```

```
KARMA 06/09/82=====================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG  7
===================================================================
>>>>>>>>      Prslex - LEX Token Saver                         <<<<<
 198 60AB                 ***********************************************
 198 60AB                 *
 199 60AB                 * Pushes the LEX tokens onto R6 on entry.
 200 60AB                 *
 201 60AB                 *   In:  DRP 2
 202 60AB                 *        r14     = LEX token.
 203 60AB                 *        r41/42  = LEX id number.
 204 60AB                 *
 205 60AB                 *   Out:
 206 60AB                 *
 207 60AB                 *        ROM switch token (EROMTK)
 208 60AB                 *        ID number of this LEX file
 209 60AB                 *        Ordinal of corresponding keyword.
 210 60AB                 *  r6 -->
 211 60AB                 *
 212 60AB                 *   Trashes: R2/3, R55/57
 213 60AB                 *   Calls:   none
 214 60AB                 *   Errors:  none
 215 60AB                 *
 216 60AB                 * This routine is necessary to make this lex file indep
 217 60AB                 * endant of the LEX file number.  This allows  LEX file
 218 60AB                 * to change numbers  when copied from ROM  to  RAM.  Th
 219 60AB                 * user simply  calls this routine  on entry to the pars
 220 60AB                 * code,  and moves the tokens from the  R6 stack  to th
 221 60AB                 * R12 stack  before  returning to the  operating system
 221 60AB                 *
 221 60AB                 ***********************************************
 222 60AB
 223 60AB                 prslex
 224 60AB                        drp  !2                  ; From caller.
 225 60AB 06 E3                  pomd r2,-r6              ; Get return address.
 226 60AD 6D A8 B4               ldb  r55,=EROMTK         ; Load switch token.
 227 60B0 6E 21 A1               ldm  r56,r41             ; Fetch LEX id#.
 228 60B3 6D 06 E5               pumd r55,+r6             ; Save em both for later
 229 60B6 4C E4                  pubd r14,+r6             ; and current token also
 230 60B8 42 E5                  pumd r2,+r6              ; Save old return addres
 231 60BA 9E                     rtn
 232 60BB
```

```
KARMA 06/09/82===================================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG  8
=================================================================================
>>>>>>>>       Runtime Routines                                             <<<<<
 239 60BB                 ***********************************************************
 234 60BB                 *
 235 60BB                 *             R U N T I M E    R O U T I N E S
 236 60BB                 *
 237 60BB                 *   Input: BCD
 238 60BB                 *          drp 20
 239 60BB                 *          r10      = pointer 1 byte past current token
 240 60BB                 *          r12      = operational stack pointer
 241 60BB                 *          r16      = Machine state byte
 242 60BB                 *          r17      = contains stall flag
 243 60BB                 *          (ROMPTR) = base address of this LEX file
 244 60BB                 *          (TOKEN)  = same as for parse routines
 245 60BB                 *
 245 60BB                 * The runtime routines must maintain the integrity
 247 60BB                 * of registers r4-13, r16/17
 247 60BB                 *
 247 60BB                 ***********************************************************
 248 60BB
 249 60BB                 ***********************************************************
 249 60BB                 *
 250 60BB                 * 100  SPEAK   <cr>
 251 60BB                 *
 252 60BB                 * Output a silly message to all display devices
 253 60BB                 *
 254 60BB                 * In :   none
 255 60BB                 * Out:   message on display devices
 256 60BB                 * Calls: mesout
 256 60BB                 *
 256 60BB                 ***********************************************************
 257 60BB
 258 60BB                 ***********************************************************
 258 60BB                 *
 259 60BB                 * The attributes byte must immediately preceed the
 260 60BB                 * runtime code for that keyword.
 260 60BB                 *
 260 60BB                 ***********************************************************
 261 60BB A1                    byt  241                 ; Statement, THEN OK.
 262 60BC                 speak.
 263 60BC 98                    bin                      ; System entered in BCD.
 264 60BD                       drp  !20                 ; System supplied DRP.
 265 60BD B1 A3 82              ldmd r20,=ROMPTR         ; Fetch my base address.
 266 60C0 10 C6 C8 00           jsb  x20,(mesout-base)   ; Output the message.
 267 60C4 41 72 66 21           asc  'Arf' Arf!'
 267 60C8 20 41 72 66
 267 60CC 21
 268 60CD 0D 8A                 def  crlf!               ; Output EOL sequence.
 269 60CF CE FF FF              jsb  =UNSEE              ; Undisplay when done.
 270 60D2 9E                    rtn
 271 60D3
```

```
KARMA 06/09/82===================================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG  9
=================================================================================
>>>>>>>>       Error  - Reports the specified error                         <<<<<
 273 60D3                 ***********************************************************
 273 60D3                 *
 274 60D3                 * 100 ERROR <error number> <cr>
 275 60D3                 *
 276 60D3                 * Report the error specified after the keyword.
 277 60D3                 * Note that only the least significant byte is used.
 278 60D3                 *
 279 60D3                 * In:  R12 points to runtime parameter stack.
 280 60D3                 *      In this case an 8 byte number is expected there.
 281 60D3                 *
 282 60D3                 * Out: The error text on the display devices.
 283 60D3                 *
 284 60D3                 * Calls: ONEB, ERRORR.
 285 60D3                 *
 285 60D3                 *
 285 60D3                 ***********************************************************
 286 60D3 A1                    byt  241                 ; Statement, THEN OK.
 287 60D4                 error.
 288 60D4 CE FF FF              jsb  =ONEB               ; R76 := binary # on r12
 289 60D7 50 3E A0              ldb  r20,r76             ; Set up error number.
 290 60DA CE FF FF              jsb  =ERRORR             ; Output that error.
 291 60DD 9E                    rtn
 292 60DE
```

```
KARMA 06/09/82===================================================================
 ITEM    LOC   OBJECT CODE  SRC=BD&EXA OBJ=BD%EXA   6/14/1982  9:51 AM  PG 10
=================================================================================
>>>>>>>>       Mesout - Output Message Subroutine                           <<<<<
 294 60DE                 ***********************************************************
 294 60DE                 *
 295 60DE                 * Output a message to display devices.
 296 60DE                 *
 297 60DE                 *   Calling sequence:   JSB  =MESOUT
 298 60DE                 *                       ASP  'message'
 299 60DE                 *
 300 60DE                 *   Inputs:  BIN
 301 60DE                 *            Message bytes follow call with high
 302 60DE                 *            bit set on the final character.
 303 60DE                 *   Output:  Message on the all display devices.
 304 60DE                 *   Trashes: status
 305 60DE                 *   Calls:   OUTCHR ---> Does ROM switching'
 306 60DE                 *   Errors:  none
 306 60DE                 *
 306 60DE                 ***********************************************************
 307 60DE
 308 60DE                 crlf'  equ  8a0dh                   ;CRLF with terminator.
 309 60DE
 310 60DE                 mesout
 311 60DE 40 06 E5              pumd r0,+r6                 ;Save used registers.
 312 60E1 42 E5                 pumd r2,+r6
 313 60E3 58 E5                 pumd r30,+r6
 314 60E5 B5 F8 FF              ldmd r30,X6,(0-8D)          ;Fetch message address.
 315 60E8 5A E4                 pubd r32,+r6
 316 60EA                       loop
 317 60EA                        drp  !32
 318 60EA 18 E0                  pobd r32,+r30              ;Get nxt chr; bump coun
 319 60EC 9A                     sad                        ;Save flags.
 320 60ED 84                     llb  r32                   ;Clear MSB.
 321 60EE 86                     lrb  r32
 322 60EF CE FF FF               jsb  =OUTCHR               ;Output the character.
 323 60F2 9F                     pad
 324 60F3 F5 F5                 whps
 325 60F5 5A 06 E2              pobd r32,-r6                ;Restore used registers.
 326 60F8 58 B7 F8 FF           stmd r30,x6,(0-8d)          ;Save new return address
 327 60FC E3                    pomd r30,-r6
 328 60FD 42 E3                 pomd r2,-r6
 329 60FF 40 E3                 pomd r0,-r6
 330 6101 9E                    rtn
 331 6102                 lxend fin                         ;End of file.
```

```
KARMA 06/09/82===================================================================
      SYMBOL  VALUE    TYPE  COUNT   Symbol Table   6/14/1982  9:51 AM  PG 11
=================================================================================
      BASE     6016      LCL    4
      CRLF'    8A0D      EQU    2
      EROMTK   00B4    G EQU    1
      ERRMSG   6037      LCL    1
      ERRNUM   0096      EQU    1
      ERRORN   6098      LCL    1
      ERROR.   60D4      LCL    1
      ERRORR   FFFF      EXT    1
      FF       00FF    G EQU    2
      INTCPT   6060      LCL    1
      KEYWRD   602C      LCL    1
      LEXID    0064      EQU    1
      LXEND    6102      LCL    1
      LXSTRT   6016      LCL    2
      MESOUT   60DE      LCL    2
      NUMVA+   FFFF      EXT    1
      ONEB     FFFF      EXT    1
      OUTCHR   FFFF      EXT    1
      PARTAB   6026      LCL    1
      PRSLEX   60AB      LCL    2
      PUSHME   60A5      LCL    1
      RELMAR   FFFF    G DAD    1
      RMBASE   0000      EQU    9
      RMHEAD   1CE3    G DAD    1
      ROMPTR   82A3    G DAD    4
      RUNTAB   6022      LCL    1
      SFSCAN   FFFF      EXT    1
      SPEAKN   608B      LCL    1
      SPEAK.   60BC      LCL    1
      SYSJSB   FFFF      EXT    1
      TYLEX    000D      EQU    1
      TYNLEX   004C    G EQU    1
      UNSEE    FFFF      EXT    2
      V.WARM   0001    G EQU    1
      lo0007   60EA      LCL    1
      rtn004   60BA      LCL    1
```

```
KARMA 06/09/82================================================================
         HEADING           Table of Contents             PAGE          PG 12
==============================================================================
```



```
BD&LXA HAD  0 ERRORS  0 WARNINGS  95 LABELS SIZE   258  LAST ERROR AT    0
```

HP Confidential

- 39 -

APPENDIX B

RAM-based Intercept Handling

The following program is from RY/JSB. It suggests a possible way to handle ROM switching for intercept handling.

```
 1 0000
 2 0000               ****************************************************
 2 0000               *
 3 0000               * THIS CODE HAS NOT BEEN TESTED!!!  It should
 4 0000               * work, but NO guarantees are made.
 4 0000               *
 4 0000               ****************************************************
 5 0000
 6 0000                      EXT  COPY          move bytes
 7 0000                      EXT  DODO          i/o rom
 8 0000                      EXT  EVIL          save registers
 9 0000                      EXT  IOISVR        i/o interrupt service
10 0000                      EXT  ROOM?         check ram room
11 0000                      EXT  ROMJSB        jsb to rom
12 0000
13 0000               INTSIZ EQU  68D           interrupt shell size
14 0000               S10-77 DAD  -56D*256D+10
15 0000               INTERR DAD  (0)           error handling code
16 0000
17 0000
18 0000               * Code for safe ROMJSB:
19 0000
20 0000 9A            RMJSB+ SAD                 save status
21 0001 40 06 E5             PUMD R0,+R6         save R0/1
22 0004 42 E5                PUMD R2,+R6         save R2/3
23 0006 CE 10 00             JSB  =1F            do the guts
24 0009 42 06 E3             POMD R2,-R6         restore R2/3
25 000C 40 E3                POMD R0,-R6         restore R0/1
26 000E 9F                   PAD                 restore status
27 000F 9E                   RTN                 go back
28 0010
29 0010 CE FF FF      1H     JSB  =EVIL          save R10/77
30 0013 08 C8                  DEF  S10-77
31 0015 63 B1 9D 82          LDMD R43,=ROMFL     get some globals
32 0019 06 E5                PUMD R43,+R6        save them
33 001B B1 A2 82             LDMD R43,=(SLOT+1) get some more
34 001E E5                   PUMD R43,+R6        save them
35 001F 66 B1 4C 84          LDMD R46,=ROMNUM    and more globals
36 0023 E5                   PUMD R46,+R6        save them
37 0024 B1 4A 85             LDMD R46,=DIRECT    get the directory
38 0027 E5                   PUMD R46,+R6        save it
39 0028 93                   CLM  R46            clear the
40 0029 B3 4A 85             STMD R46,=DIRECT    directory
41 002C CE FF FF             JSB  =ROMJSB        do the safe romjsb
42 002F FF FF                  DEF  IOISVR         to interrupt service
43 0031 FF FF                  DEF  DODO           in rom
44 0033 66 06 E3             POMD R46,-R6        get directory
45 0036 B3 4A 85             STMD R46,=DIRECT    restore it
46 0039 E3                   POMD R46,-R6        get back some globals
47 003A B3 4C 84             STMD R46,=ROMNUM    restore them
48 003D 63 E3                POMD R43,-R6        get some more
49 003F B3 A2 82             STMD R43,=(SLOT+1) restore them
50 0042 E3                   POMD R43,-R6        and some more
```

```
KARMA 05/10/82==============================================================
 ITEM    LOC   OBJECT CODE  SRC=RY&JSB OBJ=RY%JSB   5/19/1982 11:14 AM  PG  1
============================================================================
 51 0043 B3 9D 82             STMD R43,=ROMFL     restore that too
 52 0046 9E                   RTN                 finish up
 53 0047
 54 0047          ****************************************************
 54 0047          *
 55 0047          * Notes for the interrupt service routine in rom:
 56 0047          *    Must save HANDLD if going to do HANDI calls.
 57 0047          *    If this routine interrupts itself, there will
 58 0047          *    be problems (and possible stack overflow).
 59 0047          *    Must save PROTEM if going to do SYSJSB or MELJSB.
 59 0047          *
 59 0047          ****************************************************
 60 0047
 61 0047
 62 0047
```

```
KARMA 05/10/82=================================================================
 ITEM    LOC   OBJECT CODE  SRC=RY&JSB OBJ=RY%JSB   5/19/1982 11:14 AM  PG  2
===============================================================================
    0047                 * Code to move ROMJSB shell to ram:
    0047
    0047 40 C8 01    INTSET CMB  R0,=V.WARM    warmstart?
    004A F6 30              IFEQ               yes,
    004C 5A A9 44 00          LDM  R32,=INTSIZ   get the size
    0050 CE FF FF             JSB  =ROOM?        room for us?
    0053 F8 AB                JEN  INTERR        no, handle it
    0055 5C B1 5B 82          LDMD R34,=LAVAIL   location to copy from
    0059 58 1C A1             LDM  R30,R34       calculate place to move to
    005C 1A C5                SBM  R30,R32         to move to
    005E CE FF FF             JSB  =COPY         move lavail to lwamem
    0061 70 10 A1             LDM  R60,R20       move size to R60 (ignore gar
    0064 32 A3                STM  R60,R62         and propagate (ignore R70/
    0066 D3 5B 82             ADMD R60,=LAVAIL   adjust the
    0069 B3 5B 82             STMD R60,=LAVAIL     pointers
    006C 58 B1 61 82          LDMD R30,=LWAMEM   get ram start
    0070 5A A9 44 00          LDM  R32,=INTSIZ   get the size
    0074 5C A9 00 00          LDM  R34,=RMJSB+   get the code
    0078 CE FF FF             JSB  =COPY         move it
    007B 9E                   RTN                all done
    007C                    ENDIF
    007C
    007C                    FIN
```

```
KARMA 05/10/82=================================================================
     SYMBOL  VALUE    TYPE  COUNT   Symbol Table    5/19/1982 11:14 AM  PG  3
===============================================================================
     1H0001  0010      LCL    1
     COPY    FFFF      EXT    2
     DIRECT  854A    G DAD    3
     DODO    FFFF      EXT    1
     EVIL    FFFF      EXT    1
     INTERR  0000      DAD    1
     INTSET  0047      LCL    0    <--- NOT REFERENCED??
     INTSIZ  0044      EQU    2
     IOISVR  FFFF      EXT    1
     LAVAIL  825B    G DAD    3
     LWAMEM  8261    G DAD    1
     RMJSB+  0000      LCL    1
     ROMFL   829D    G DAD    2
     ROMJSB  FFFF      EXT    1
     ROMNUM  844C    G DAD    2
     ROOM?   FFFF      EXT    1
     S10-77  C808      DAD    1
     SLOT    82A1    G DAD    2
     V.WARM  0001    G EQU    1
     ij0003  007C      LCL    1

RY&JSB HAD  0 ERRORS  0 WARNINGS  28 LABELS SIZE   124  LAST ERROR AT    0
```

HP Confidential

APPENDIX C

BU%LEX and Hybrid LEX files

BU%LEX is a LEX file which allows the creation of RAM-based LEX files using the HP1000 development systems (Systems 31 and 32 in the lab.) BU%LEX is an unsupported product. To use BU%LEX, follow the instructions which appear in the file and which are also reproduced here.

> This routine copies specially formatted ROM files into LEX files which can by accessed by Kangaroo. Non-lex files can also be copied, but they must be KLINK compatable. The workfile is purged and a CLEAR VARS is done after the file is copied which will make the lex file much more stable in RAM. (Hence easier to debug')
>
> To use this facility:
>
> 1) The source files must consist of relocatable modules. A file directory, including the ROM existance header, must be first. Make sure that the runtime tables are relative to base address zero.
>
> 2) Run KLINK. Link this module first followed by the modules containing your code.
>
> 3) Once you have loaded the resulting ROM code onto the development system, run the operating system and ON Kangaroo type: LEX <cr>
> The file will be created from the directory information contained at the beginning of your ROM.

HP Confidential

APPENDIX D

References

1. Comments contained in the source code
2. Entry Point documentation
3. HANDI call documentation
4. Interpreter document
5. Kangaroo Cross Reference
6. Kangaroo Memory Management document
7. Kangaroo Owner's Manual
8. Kangaroo System Hooks and Handles document
9. KR/GLO (Kangaroo global file)
10. Parser document

HP Confidential

Table of Contents

HP Confidential

The Kangaroo Rom Switching Guide

Jack Applin IV

3:47 July 19, 1982

Big Picture

```
INSIDE KANGAROO                 :   OUTSIDE KANGAROO
................................:.........................
                                :
   64K +-----------+            :
       | I/O space |            :
63.75K +-----------+            :
       |           |            :
       | System    |            :
       | Rom 4     |            :
       | (BASROM)  |            :
   56K +-----------+            :
       |           |            :
       | RAM       |            :
       |           |            :
   48K +-----------+            :
       |           |            :
       | RAM       |            :
       |           |            :
   40K +-----------+            :
```

```
    |           |                   :
    | RAM       |                   :
    |           |                   :
32K +-----------+  +-----------+    :  +-----------+    +-----------+
    |           |  |           |  | :  |           |    |           |
    | System    |  | System    |  | :  | Plug-in   |    | Plug-in   |
    | Rom 3     |  | Rom 5     |  | :  | Rom       |    | Rom       |
    | (ALTROM)  |  | (MELROM)  |  | :  |           |    |           |
24K +-----------+  +-----------+    :  +-----------+    +-----------+
    |           |                   :
    | System    |                   :
    | Rom 2     |                   :
16K +-----------+                   :
    |           |                   :
    | System    |                   :
    | Rom 1     |                   :
 8K +-----------+                   :
    |           |                   :
    | System    |                   :
    | Rom 0     |                   :
 0K +-----------+                   :
```


In the Kangaroo system, the addresses 24K-32K contain the switching roms. Only one of these roms should be enabled at one time. The software keeps track of which rom is enabled at any given time. The address 56K-63.75K contains what is l_o_g_i_c_a_l switching rom, the BASROM, but it in fact is non-switching. The fact that the BASROM contains a lexfile makes it a logical rom.

Enabling and Disabling Roms

Roms are enabled and disabled by writing to I/O addresses. A any data whatsoever may be written, it is the act of writing that the hardware notices. The FF30s, FF40s, and FF50s are the I/O addresses that control rom enable/disable.

Unfortunately, there are two different revisions of the roms, the "old" roms (part numberss 1LD4 & 1LF9) and the "new" roms (part number ????). The original CMOSV roms (part # 1LD4) were designed by Bruce Schoeber. These roms were modified for CMOSC (part # 1LF9) by Liz Myers. Then, Liz modified them to become "new" roms (part # ????).

Also, there are the 16K CMOSC 1LH8 roms for use in MEMIC.

--- OLD ---

The old roms were only manufactured to fit in the FF4X addresses. These roms were enabled by writing to an FF4<even> address and were disabled by writing to the corresponding FF4<odd> address. Hence, to disable a rom, you have to write to either the corresponding FF4<odd> address (assuming you know what rom's enabled) or write to all the FF4<odd> addresses. It is possible to have two roms enabled at once.

--- NEW ---

The new roms are manufactured to fit in the FF3X, FF4X, or FF5X addresses. They are enabled by writing to an FF3<even>, FF4<even>, or FF5<even> address. They are disabled by writing to a ny address FF30-FF5F except for FF55, which is a test address. This should make it difficult to enable two roms at once, since the act of enabling any rom disables all others.

--- MEMIC ---

The MEMIC roms are manufactured to fit in only the FF5X addresses. They are enabled by writing to an FF5<even> address. They are disabled by writing to FF4F only.

NOTE

Since it is unknown whether the I/O space auto-increments or not, (and in fact it varies between different revisions of the roms) you should only perform one-byte writes to the I/O addresses.

The routine UNROM writes to the FF4<odd> addresses to disable all the roms. This should work for any of the roms.

NOMAS

NOt MAnufacturer Supported
recipient agrees NOT to contact manufacturer

The Rom Switching Software

The rom switching software consists of several routines and a few pointers. The routines are used to switch between roms. The pointers can be used to see what rom/lexfile you're in.

For example, if routine ALPHA used ROMJSB to call routine BETA in rom 2222 (in rom, not ram), BETA could look at:

ROMNUM to find out the ID number (2222) of the lexfile BETA's in
SLOT to find out the enable address of the rom BETA's in

If routine ALPHA used ROMJSB to call routine GAMMA in lexfile 3333 in ram, GAMMA could look at:

ROMNUM to find out the ID number (3333) of the lexfile GAMMA's in
ROMPTR to find out the address of the lexfile GAMMA's in

NOTE

Since each rom must contain a lexfile as its first file
(and no other lexfiles)
I refer to the lexfile's ID number (first two bytes of the lexfile)
as the rom's ID number.
Technically speaking, only lexfiles have ID numbers, not roms.
This is pretty sloppy, but it works.

Rom Switching Routines

CYCLE: cycle through the lexfiles

CYCLE calls a given subroutine once with each lexfile enabled.

GETROM: enable a lexfile

GETROM enables the lexfile whose ID is in R0. If the lexfile is in a rom, the rom is enabled. If the lexfile is in ram, only the pointers (ROMPTR, ROMOFF, etc.) are changed and no hardware rom switching is done.

GETRM2: enable a lexfile

GETRM2 enables the lexfile whose ID is in R22. Otherwise the same as GETROM above.

MELJSB: call a routine in the MELROM

MELJSB saves the current lexfile/rom status, enables the MELROM, calls the routine given after the call to MELJSB, re-enables the previous rom, and returns to the caller. MELJSB is transparent to the called subroutine, that is, the exact status of the calle (ARP, DRP, E, everything!) is passed to the called subroutine, and the exact returning status of the subroutine is passed bac to the caller of MELJSB.

NXTROM: get the next rom

NXTROM, given a pointer to the rom table ROMTAB, enables the next rom (not lexfile) and returns an updated pointer. It is used by NXTKWF & CYCLE to loop through lexfiles for HANDI.

ROMJSB: call a routine in any lexfile

ROMJSB saves the current lexfile/rom status, enables the lexfile given in the call, calls the routine given in the call, re-enables the previous rom, and returns to the caller. ROMJSB is transparent to the called subroutine, that is, the exact statu of the caller (ARP, DRP, E, everything!) is passed to the called subroutine, and the exact returning status of the subroutine i passed back to the caller of ROMJSB.

ROMRTN: enable the system rom

ROMRTN enables BASROM, setting the pointers (ROMPTR, ROMNUM, etc.) to reflect this. Note that it does physically enable ALTROM (at 24K) but the pointers are set to BASROM (at 56K).

ROMRTN is the same as calling GETROM to enable the BASROM.

ROMSET: set up the table of roms

ROMSET (called at coldstart/warmstart) initializes ROMTAB, the table of roms. ROMTAB contains the slot address of all currently existing roms, followed by a 0000 for end-of-table. The entry for BASROM (at 56K) is 0001.

RUNROM: enable the lexfile for the current environment

RUNROM looks in the current environment for the current program's lexfile and calls GETROM to enable it.

SAVEME: save me from rom switching

SAVEME is used like this:

```
SUB  JSB  =SAVEME          save me from rom switching
     ...
     JSB  =GETROM          enable some other rom
     ...
     RTN                   return with previous lexfile enabled!
```

When SAVEME is called, it saves the current lexfile status. Then it calls the rest of it's caller (SUB, in our example) as a subroutine. The caller executes, possibly performing rom switching, and then returns. This return takes us back into SAVEME! At this point, SAVEME restores the previous lexfile (it saved the lexfile status on the stack previously) and returns. This takes us back to the caller of SUB with the previous lexfile enabled.

SYSJSB: call a routine in the BASROM

SYSJSB saves the current lexfile/rom status, enables the BASROM, calls the routine given after the call to SYSJSB, re-enables the previous rom, and returns to the caller. SYSJSB is transparent to the called subroutine, that is, the exact status of the calle (ARP, DRP, E, everything!) is passed to the called subroutine, and the exact returning status of the subroutine is passed bac to the caller of SYSJSB.

Please note that SYSJSB enables BASROM (at 56K) and not ALTROM (at 24K) and so ROMPTR and ROMNUM are set accordingly. It does physically enable ALTROM, but the pointers are set to BASROM.

UNROM: physically disable all roms

UNROM physically disables all roms by writing to the FF4<odd> addresses. It doesn't modify the pointers (ROMPRT, ROMNUM, etc.).

Rom Switching Software Globals

HANDLD

The flag for a routine to stop a HANDI call. If a intercept routine (called by HANDI) clears HANDLD, then HANDI stops cycl through the intercept routines and signals success to the caller of HANDI. If HANDLD is never set, HANDI goes through all intercept routines.

NOTE: intercept routines called by HANDI0 can also set HANDLD to stop its cycling, but this is not reccomended.

ROMNUM

The ID number of the current lexfile. The ID number is contained in the first two bytes of the lexfile.

ROMOFF

The "offset" for the current lexfile. It is added to the routine addresses in the lexfile's tables to make them absolute addresses. In lexfiles in rom, ROMOFF=0. In lexfiles in ram, ROMOFF=ROMPTR=the address of the lexfile.

ROMPTR

The pointer to the current lexfile. For the BASROM, ROMPTR=56K+2 (just after the rom header bytes). For all other lexfiles in rom, ROMPTR=24K+2. For lexfiles in ram, ROMPTR=the address of the lexfile (no header needed for lexfiles in ram).

ROMTAB

The table of slots that contain roms. The table contains a series of slot addresses followed by a zero to indicate end-of-table. The BASROM is indicated by 0001 in this table. The table is initialized at coldstart/warmstart.

SLOT

The enable address of the currently enabled rom (not lexfile). If a lexfile is currently pointed to by ROMPTR/ROMOFF, SLOT still contains the enable address of the previously enabled rom, which is still physically enabled.

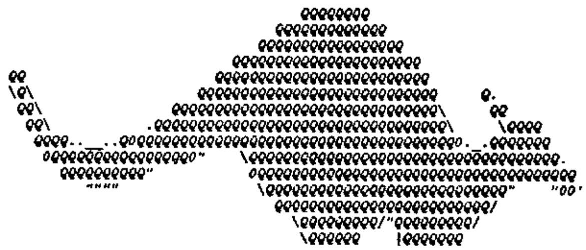
Kangaroo Rom Switching Guide

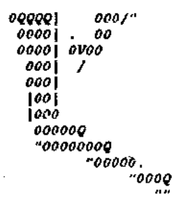
Other documents

Joey's Big Book of Roms (KR"ROM) by Dave Barrett & Seth D. Alford
This tells you how to write a lexfile

Time & date stuff

Raan Young
07/09/82





TIME$ and DATE$ use essentially the same code, the only difference being the delimiter used (which also serves as a flow flag at one point). The process gets some ram for the string and gets the current time/date. Depending on the delimiter, it converts the hours/years to ASCII and outputs them to the string, followed by the delimiter. The process is repeated for the minutes/month and second/days fields. Finally, the string address and length are put on the stack.

DATE gets the current time, converts the month to number of days by going through a loop which adds up the number of days in each month, adds the day to get number of days since start of year, and adjusts as needed for leap year. The bottom two digits of the year are multiplied by 1000 and added to the day count to produce the Julian date YYDDD. This integer is put on the stack.

TIME gets the current time, multiplies the hours by 60, adds the minutes, multiplies the results by 60, and adds the seconds. This gives the integer number of seconds since midnight. Then we get the fractions of seconds by adding the RTC to the time base and throwing away everything above the 14th bit. This number is converted to floating point and divided by 16384 (1 sec in 2^-14 second ticks). The result is added to the integer seconds, the new result is left shifted 3 decimal points, truncated and right shifted back (this truncates to milliseconds). The real number is left on the stack.

TRANSFORM

Gary K. Cutler

2:20 PM THU., 15 JULY, 1982

TRANSFORM	CHAPTER 1

1.1 TRFRM. -- The runtime routine

TRFRM. is the runtime entry point for TRANSFORM. At the time of entry there are six bytes on the R12 stack (unless the user is transforming the current editfile, in which case only the <type> will be on the stack). Four bytes which contain location and length of the filename and two bytes describing the attributes of the <type>.

SYNTAX: TRANSFORM ["<filespecifier>"] INTO <type>

PARSED: ["<filespecifier>"], <type>, INTO, TRANSFORM

```
               |--------------------|
               |  filename length   |
               |--------------------|
               |  filename length   |
               |--------------------|
               | filename location  |
               |--------------------|
               | filename location  |
               |--------------------|
               |       <type>       |
               |--------------------|
               |       <type>       |
               |--------------------|
r12 ---------> |                    |
               |--------------------|
```

1.2 INITIAL STEPS

I. the type is popped off the stack and examined. This triggers one of three routines. CONAS-, BAS_AS, or CONLIF.

2:20 PM THU., 15 JULY, 1982

II. the file directory entry is located and the file type is determined.

III. conversion is initiated and if no errors occur the file is converted and the directory is updated

1.3 CONTROL AND ACTION

The control is directed by the specified <type>. If <BASIC> is specified then control is passed to CONAS-; <TEXT> sends control to BAS_AS; and <LIF1> passes control to CONLIF.

I. <type>::=BASIC

a) <file type>::=BASIC; action: none

b) <file type>::=TEXT; routine: CONAS-

action: Each text line is parsed. If no errors occurred then the original text line is replaced with the tokenized line. If an error has occurred, then the original text line will be interpreted as a line of comment and initiated with '! ?'.

c) <file type>::=LIF1; routines: CLIF ---> CONAS-

action: The lif1 file is stripped of its filler. The PCB (Program Control Block) is inserted. Then the ASCII format is converted to internal text form. This is accomplished by converting the line# to BCD and swaping position with the line length. Finally the text file is converted to basic as stated above.

II. <type>::=TEXT

a) <file type>::=TEXT; action: none

b) <file type>::=BASIC; routine: BAS_AS

action: Each basic line is decompiled. The original line is then replaced with the decompiled version.

c) <file type>::=LIF1; routine: CLIF

action: The PCB is inserted, ASCII line numbers are converted to BCD and the relative positions between the length byte and BCD line number are reversed.

2:20 PM THU., 15 JULY, 1982

III. <type>::=LIF1

a) <file type>::=LIF1; action: none

b) <file type>::=TEXT; routine: CONLIF

action: The PCB is deleted and each text line is converted to lif1 format. The internal endline is replaced with a binary length of -1 (FF,FF internally). The length of logical file is calculated (not including the lif1 header) and the appropriate number of bytes are added to the file in order to conform to sector length.

c) <file type>::=BASIC; routines: BAS_AS, CONLIF

action: The basic file is decompiled into text. The text file is then converted to lif1 as described above.

<current type> to <type>: <routine>

basic	text :	BAS_AS
text	basic:	CONAS-
text	lif1 :	CONLIF
basic	lif1 :	BAS_AS ---> CONLIF
lif1	text :	CLIF
lif1	basic:	CLIF ---> CONAS-

2:20 PM THU., 15 JULY, 1982

1.4 ERRORS

ERROR CONDITIONS	ERRORS
a) improper <type> i) not BASIC,TEXT,LIF1	wrong file type
b) improper file type i) file not BASIC,TEXT,LIF1	wrong file type
c) not sufficient memory	not enough memory
d) improper LIF1 structure i) no line #s ii) invalid line lengths	 bad statement line too long
e) text line greater than 255	line too long
f) TRANSFORM <workfile> into LIF1	invalid filespec
g) ROM missing during BASIC to TEXT	ROM missing
h) <filespec> containing device spec	invalid filespec

1.5 GLOBALS

Name	Location	Description
DCOVFL	82C8	overflow flag for input buffer
ENDLIN	A999	internal endline
ENLINR	83D1	current endline (stopping criterion)
INPBUF	8180	used as decompile buffer
KLUDGE	8249	updated location during memory moves
MISSNG	82B7	ROM missing flag
ORIGIN	83B2	file origin TEXT or LIF1
PARERR	82B8	parsing error flag
RNFILE	8247	location of current running file
STSIZE	8255	loc for parsing DEF FN statements
TOS	8257	current top of stack (R12)

1.6 CROSS REFERENCES

Global File	ER&GLO
Source File	GC&TFM
Utilities File	GC&UTL

1.7 FLOW DIAGRAMS

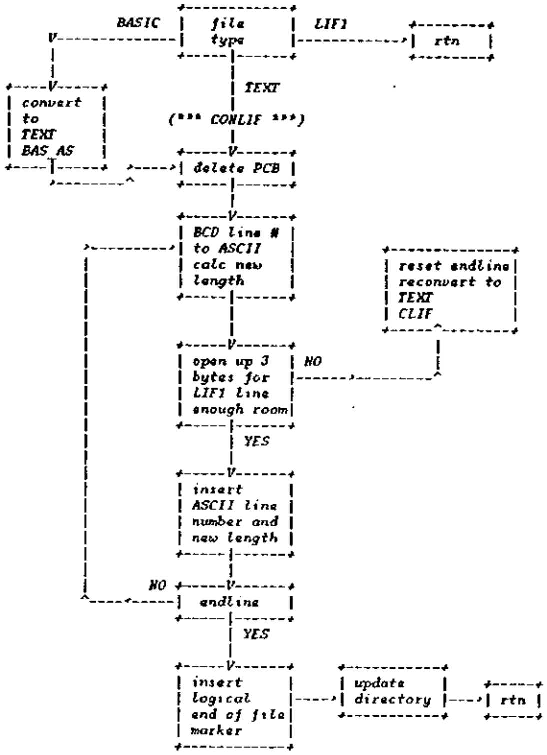
```
                         FLOW DIAGRAMS
                       -----------------

                           TRANSFORM

        BASIC   +-------------+        LIF1
# 1 <-----------|   <type>    |------------------> # 2
                +-----|-------+
                      | TEXT
                      |
                +-----V-------+
        LIF1    |    file     |    TEXT        +-------+
# 3 <-----------|    type     |----------------|  rtn  |
                +-----|-------+                +-------+
                      | BASIC
                      |
             (***  BAS_AS  ***)
                      T
                +-----V-------+
  >------------>|  decompile  |
  |             |   current   |
  |             |    line     |
  |             +-----|-------+
  |                   |                   +----------------+
  |             +-----V-------+           | reset endline  |
  |             |     ROM     | YES       | reconvert to   |
  |             |   missing   |------>----| basic          |
  |             +-----|-------+           | GOTO CONAS-    |
  |                   | NO                +-------^--------+
  |                   |                           |
  |             +-----V-------+                   |
  |             | insert new  | NO                |
  |             | line        |------>------------^
  |             | enough room |
  |             +-----|-------+
  |                   | YES
  |                   |
  |             +-----V-------+
  |             | delete old  |
  |             | line        |
  |             +-----|-------+
  |                   |           +-------------+
  |          NO +-----V-------+ YES| update      |   +-----+
  ^-------<-----|   endline   |--->| directory   |-->| rtn |
                +-------------+    +-------------+   +-----+
```



```
                                             +-----------------+
            (***  # 1  ***)                  |reset endline    |
                                             |reconvert to text|
            <type>:=BASIC                    |GOTO BAS_AS      |
                                             +-------^---------+
            +-------------+                          |
     LIF1   |    file     |  BASIC     +-----+       |
# 4 <-------|    type     |----------->| rtn |       |
            +-----|-------+            +-----+       |
                  | TEXT                             |
          (*** CONAS- ***)                           |
                  |                         >-------^---<
            +-----V-------+             NO  |           |
            |    parse    |          +------|-----+     |
 >--------->|   current   |          |insert  ! ? |     |
 |          |    line     |          |enough room |     |
 |          +-----|-------+          +---^-----|--+     |
 |                |                      |     | YES    |
 |          +-----V-------+ YES          |     |        |
 |          |   errors    |------->------^     |        |
 |          +-----|-------+                    |        |
 |                |                            |        |
 |          +-----V-------+                    |        |
 |          | insert      |                    |        |
 |          | tokenized   | NO                 >--V     |
 |          | line        |----------->---------V-----^
 |          | enough room |                    V--V
 |          +-----|-------+                    |
 |                | YES                        |
 |          +-----V-------+                    |
 |          | delete      |                    |
 |          | text line   |                    |
 |          +-----|-------+                    |
 |                |                            |
 |       NO +-----V-------+                    |
 ^----<-----|   endline   |---------<----------+
            +-----|-------+
                  | YES
            +-----V-------+
            | update      |
            | directory   |
            +-----|-------+
                  |
            +-----V-------+
            |     rtn     |
            +-------------+
```

(*** # 2 ***)

<type>:= LIF1



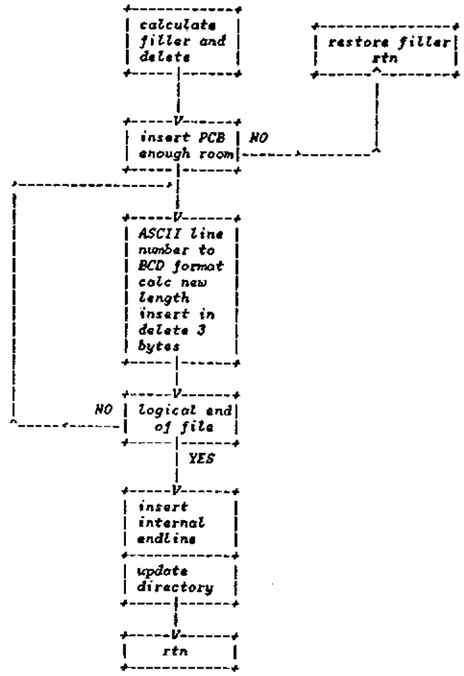
2:20 PM THU., 15 JULY, 1982

(*** # 3 ***)

<type>:= TEXT

file type:= LIF1

(*** CLIF ***)

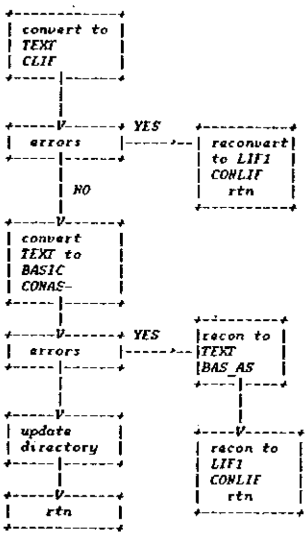


2:20 PM THU., 15 JULY, 1982

(*** # 4 ***)

<type>:= BASIC

file type:= LIF1



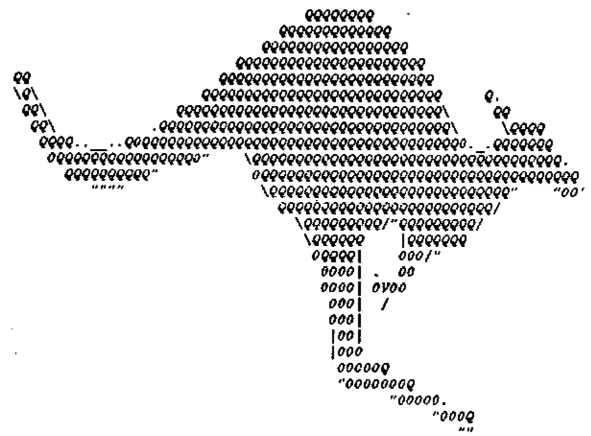
2:20 PM THU., 15 JULY, 1982

Table of Contents

2:20 PM THU., 15 JULY, 1982

ON TIMER documentation

Raan Young
07/09/82



Timers allow a basic program to cause execution of basic statements at specified real time intervals. Subject to restrictions on minimum duration and compleation of the current line, the program will GOSUB to the timer statement at the interval specified.

GENERAL FLOW:
The ON TIMER statement is initialized when the ON TIMER token is executed. N seconds later, the comparator interrupt will trigger the timer. This trigger sets up the next timer, sets R10 and PCR to point to the code in the ON TIMER statement, and sets the flag TMRFLG to indicate the timer wants to run (this is only used when a timer wakes up the machine). The first token of the timer statement clears the flag, and excution continues until the invisible pop or invisible return sends us back to where we were.

PARSING:
The ON TIMER statement parses into the ON TIMER token, parameters, invisible clear token, execution tokens, invisible RETURN token. If there is a GOTO or ON ... GOTO token then an invisible POP token is placed in front of it.

SETUP:
The runtime for the ON TIMER (ONTMR.) token sets up the timer entry in the timer file ('timers'). The format of an entry is:

byte	entry	comment
1-2	line # (bcd)	same as timer #
3	line length	
4-10	interval	in 2^-14 sec ticks
11-17	absolute time	of next interrupt (in ticks)
18	busy flag	0=not busy, #0=busy
19-20	relative PCR	of ON TIMER line
21-22	relative R10	of clear token in ON TIMER statement
23-30	runfile name	of program declaring ON TIMER

Busy flag is used to prevent a timer from interrupting itself (see trigger section). The runfile name is used to ensure that timers only try to execute when in the program that delcared them (see trigger section).

The interval is fetched from the R12 stack using ONE7+B to format it into ticks. If the number is < 1/10 second, then 1/10 second is used instead (this is to prevent timers from constantly interrupting). The timer number is also on the R12 stack, and is checked for 0<=N<=1000 (if not, then error). The line is added to the timer file by FREPLS, which creates the file if needed. After the entry is added, the timer file is searched for the next absolute time of interrupt, and this is sent to the comparator as the timer entry (see KR"CMP). The number of this timer is saved in TMRNMB for later identification. The execution then skips to the next line, bypassing the rest of the ON TIMER statement.

TRIGGER:
The trigger routine for ON TIMER (TMTRIG) is called by CMPCHK when the timer flag in CMPFLG is set. The timer file is located, and the timer which caused the interrupt is found using TMRNMB. The interval is added to the old absolute time to get the next time of interrupt. If the busy flag is not set, then the relative PCR and R10 values are fetched, and the timer program name is checked against the current program name. If the names match, then the busy flag is set. The next timer is then found and set to the comparator (TMRNMB is updated). If the names did not match or the timer was busy, this is all that is done. Otherwise, the current PCR and R10 are saved by SUBSTF, which also puts the timer number on as the flag. Next, the program rom is asserted (this is in case the timer woke us up), the new PCR and R10 are absolutized and set up, the jump is traced, and TMRFLG is set to indicate a timer wants processing. (TMRFLG is only tested when a timer woke us up.)

PROCESSING:
The normal flow of BASIC will pick up with the clear timer token in the ON TIMER statement after the trigger has set things to this token. The TMRCLR routine for this token clears the TMRFLG to indicate the timer has been processed. The rest of the line executes normally until the invisible POP or invisible RETURN is encountered. This token will take the information on the GOSUB stack (put there by the trigger) and return (or discard) to the line where the interrupt occured. The timer busy flag is cleared as part of the stack clean up (the timer number pushed on the stack is used to locate and clear the flag (done by TMRDNE)). (See KR"ERR for more on this GOSUB stack flag.) In the case of the timer waking us up, the call to CMPCHK is done in the going to sleep stuff and we are not in the CRUNCH loop. This is why the TMRFLG is used, it is tested by the sleep stuff after the call to CMPCHK. If it is set, then the machine returns to edit mode and continues the program instead of going to sleep. (The machine will wakeup and go right back to sleep when woken by the comparator unless diverted - see wakeup and sleep documents for more.)

SUSPENSION:
Timers are suspended when the program is stalled (this is handled by TMRSS). The timer entry in the comparator is 0'd (turned off), and the time of suspension is saved in TMRRTC (this is used at restart time). Finally, the timer flag is cleared to prevent any problems if the machine goes to sleep.

RESTART:
The timers are restarted when the program is continued (by TMRGO). This gets the current time, and subtracts the value saved in TMRRTC, to compute the duration of the suspension. This value is added to each timer's absolute time to adjust for the suspension. The next timer is then found and loaded.

OFF TIMER:
The individual timer is turned off by TMROF., which deletes that entry from the timer file. All timers are offed at dealloc-all time by TMRKIL, which purges the timer file.

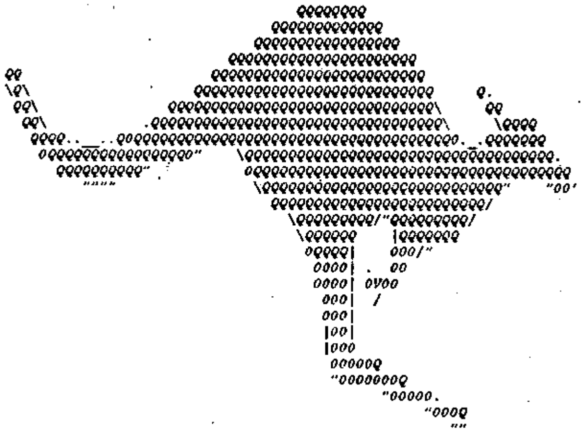
TIMER CONFLICTS:
There are three ways the timer processing can be skipped:

* timer is busy -- the timer time is updated, but processing is skipped
* program is not the one which declared the timer -- the timer is updated but processing is skipped
* timers suspended by stall -- timer processing is missed (because timers aren't running) but the suspension time is 'invisible' to the timers so they will continue as if never stopped.

TIMER ROLLOVER:
The rollover of the clock initiates adjustment of the timers by calling TMRRLV (see KR"CMP for more on this). The suspend time is adjusted (in case timers are suspended while rollover), and then each timer's absolute time is adjusted (by adding the complement of the rollover time).

Time mode command processing

Raan Young
07/09/82



Time mode consists of three parts: the part which handles the time display, the part which gets user command input and then calls the third part which handles the parsing and processing of time mode commands.

DISPLAY:
The time display is driven by the comparator. When time mode is entered, the clock entry is set up so that an interrupt for clock triggering will occur every second on the second. This trigger will update the display of the time through CSTRIG. This is the normal operation for time mode and will continue until time mode is left or a command is entered. (See the Pocket Secretary documentation for more on this).

INPUT:
Input is done in a 5 character LCD window, with the time being simultaneously displayed (and updated) on the LCD to the left of this window. The command input is done with GETTEM, and is checked for type of terminator key. If the terminator was RTN, the command processor TMECMD is called. If the terminator was CLR or other non-mode key, the command is erased and ignored. If it is a mode key, then time mode is left and we switch to that mode. (See the Pocket Secretary documentation for more on this).

COMMANDS:
Valid commands are EXACT, SET, ADJST, STATS, and RESET. The command is blank filled, uppercased, and then the command table is searched for a match, (if not found then an error is reported and the command is returned to the display for correction). If the command is not recognized in the mainframe, the HANDI call V.TMCX is issued to allow for extensions to the command list. If the command is recognized, then "ticking" (display update) is turned off (interrupts still happen but the display is inhibited). and the command process routine is called via table lookup. After the routine returns, ticking is reenabled, and the clock tick is forced by setting the clock flag in CMPFLG (this prevents any delay after the command is finished). The display inhibit is controlled by a bit in PSSTAT and is toggled by TCKTGL.

STATS:
This command displays/modifies the current clock status (month/day mode, 12/24 hr mode, exactness, and calendar mode). The display is first built up by assuming default states for the modes and moving the default display line to the input buffer. Then each mode is checked (flags are stored in PSSTAT) and the display line is adjusted appropriately. Then line is displayed as a protected field and the user is allowed to modify certain parts of the display. If the user terminates input with a CLR then the original status is redisplayed and we go around again. If terminated with anything else (other than RTN), STATS command is aborted and nothing is changed. If terminated with return, then each field is examined and the corresponding flag is changed to reflect the new field contents (or no change if the field was not altered). If there is an error, the offending field is indicated, and input is restorted. Otherwise, the flags are saved in PSSTAT.

SET:
This command allows the user to reset the time. The time set template is displayed, with its format determined by the current mode settings. The user is then allowed to fill in the desired changes. As with STATS, the terminator determines the disposition of the input. If terminated with return, the current absolute time is saved, and the input is processed for date and time information. The current time is used to fill in any unspecifed values. If there are any errors, the error is reported, and input is restarted. If all is ok, the new time is used to compute the error and adjustment values, and the new time base is set to new time - RTC. (See error/adjust for more).

ADJST:
This command is used to adjust the clock by a relative amount, rather than setting it to an absolute time. The template is built and displayed for the user to fill in. Termination character is processed as usual, if RTN then the adjust is summed up by adding the portions together (hrs, mins, sec, tenths). The sign is checked to see if the adjust is forward or backward, the sum is complemented if backward. The adjust type flag is fetched. If there are any errors, the error is flaged, and input is restorted. The adjust value is added to the time base, and then separated into errors and adjustments. (See error/adjust for more).

EXACT:
This command establishes an end point for the sample period, and (if not the first exact), computes the error over the last sample period. Error is the difference between the time now (which the user has said is correct by doing the EXACT) and the uncorrected time we thought it was. This error is equivalent to the accumulated errors from SETs and ADJSTs. The time of the exact is computed by taking the current time and subtracting the accumulated adjustments. This gives the time with all time zones removed. This time is saved as the start of the next sample period. The exact flag is set, and the interval is computed by subtracting the last exact time from this time. (If result is negative then a warning is reported, and the adjust period is not changed.) If the accumulated errors are 0, then the adjust period is set to 0 (which turns off the adjustment interrupt). If the error is #0 then the interval is adjusted by subtracting the errors. (This makes negative errors [clock fast] increase the interval, and positive errors [clock slow] decrease the interval). If the error is negative, the decrement flag (slow down clock) is set in PSSTAT and the error is complemented. The interval is multiplied by 2^12 (1/4 second in ticks) and divided by the error. This result is the time (in ticks) between clock adjustments of 1/4 second (+ or -), and is sent to the comparator.

In formula form:

```
Exact_time=time-accumulated_adjusts

Sample_period=new_exact_time-old_exact_time

Adjust_period =
  ((sample_period-accum_errors)*2^12)/accum_errors

or:

Adjust_period =
  (sample_period-accum_errors)/(accum_errors*1/4*2^-14)
```

RESET:
This command simply clears out the exact information to get back to a pre-EXACT state. The comparator interrupt is turned off, the exact flag is cleared, and the accumulators are cleared.

Comparator setup:
The value to be sent to the comparator for the clock adjust is checked to insure it is not less than 1/2 second (excluding 0). If it is, then 1/2 second is sent instead. This is to insure that interrupts don't come in so fast that nothing else can operate.

Error/adjust:
The seperation of time difference into errors and adjustments is based on the assumption that any change evenly divisible by 30 minutes is a time zone change and not an error. The difference therefore has as many 30 min hunks removed as is possible (these are added to the adjust ac-

cumulator), and the amount left over is taken as error. If the error is greater than 15 minutes then one more 30

minute hunk is removed and the error is taken to be negative (errors are added to the error accumulator).

Examples:

Difference	Adjust amount	Error amount	Comment
30 minutes	30 minutes	0	zone ahead
5 minutes	0	5 minutes	5 min slow
35 minutes	30 minutes	5 minutes	zone ahead slow
25 minutes	30 minutes	-5 minutes	zone ahead fast
-30 minutes	-30 minutes	0	zone back
-5 minutes	0	-5 minutes	5 min fast
-35 minutes	-30 minutes	-5 minutes	zone back fast
-25 minutes	-30 minutes	5 minutes	zone back slow

Other routines:

There are two other routines in this file which are not directly related to time mode. CRTMDT returns the current time/date in R44/47 in internal format (number of 2^-14 ticks since the turn of the century). DTECNV takes an internal format date and converts it to two strings: date (MO/DY/YR or DY\MO\YR depending on month/day mode) and time (HR:MN, always 24hr). Strings are returned in R40/47 and R72/77 respectively, input is in R44/47.

User function operations

Raan Young
07/09/82

The general flow of user functions begins with a call (X=FNA(B)). The call token checks each supplied parameter against the parameters defined in the function definition. If any do not match in type, or if there are too many or too few, an error is reported and the token stops. The token also checks if this function has already been called (recursive call). If all is ok, the parameter values are moved from the R12 stack to the function parameter value location pointed to by the function parameter block (in the program). The call token then saves the return address, PCR, TOS, R16, and E.RMEM and does an imitation GOSUB to the definition. Execution continues with the definition until an END DEF token is encountered. In the case of a multi-line function, a LET token must be encountered to assign a value to the function variable, or it will return undefined. One-line definitions have an invisible END DEF token and the value is already on the stack (from evaluation of the expression), so no value is fetched from the function variable. The END DEF token makes sure the value is on the stack, restores all the saved information (TOS, E.RMEM, R16, PCR, return address) and returns to where the function was called. Note that the one-line string function is not subject to the 32 char length limit, because it does not acually use the function variable for storage.

User functions are quite complex in their internal format. For this reason, each phase will be handled separately. The following table shows the routines for each phase.

PHASE: \ KEYWORD:	DEF FN	LET	END DEF	call
PARSE	DEF	FNILET	FNEND	FN$CAL #1 FNCALL #2
PALLOC	DEFFN	FNASN	DEFEND	FUNCAL
RUN	SKPDEF	FNLET.	FNRET. #3 FNRTN.	FNCAL$ #1 FNCAL. #2
DEALLOC	DALFN	FRRET	FRET	DALFNC
DECOM	UFNDEF	FNASGN	FNRTN #3 FNEND	UFNCAL

In addition, ENV alloc is handled by INVFN, and variable

init is handled by INIVLP for the function variable. Parameters are not initialized at allocation because they allways have a value loaded into them before they are accessed (see RUN for CALL token).

#1: string function. #2: numeric function. #3: invisible token

PARSE:

DEF FN:

Parsing of the function definition insists on program mode and on being the first thing on the line. FNNAME pushes the name and scans for parameters. (If SCAN finds a real number it automatically puts it out, so FNNAME takes it back off the stack.) FNNAME then puts out 2 bytes filler (for rel jump value -- see allocation phase of END DEF). If the function was simple numeric (from initial SCAN) then it was numeric with no parameters and a '0' is output for the count/type. If the function was simple array (from initial SCAN) then it was numeric with parameters, and a '2' is output for the count/type. If the function was string (from initial SCAN) then the string type is set and FNNAME looks for a '(' (beginning of parameter list). If found then a '3' is output, otherwise a '1' is output (for the count/type). If the function is not any of the above types, then FNNAME returns with E=0 and an error is reported, otherwise E=1.

FNNAME returns with 'NAME(2), ZERO(2), COUNT/TYPE' output to the stack. If there were any parameters (count/type>1) then the location of the count/type byte on the stack is saved (for later modification), and a loop to parse the parameters is entered. This loop checks each parameter for type (must be string or simple numeric, otherwise an error is reported) and pushes the internalized form of the name. String parameters are also checked for user defined length, and the length is pushed. Next 2 bytes of filler are pushed for later addition of the value pointer (see allocation phase for DEF FN). The parameter counter is incremented by 2 (bottom bit is function type) for each parameter found. This loop continues until no comma is found after the parameter just parsed. After the loop ends, the count/type location is retrieved, and the count/type is updated to reflect the count found.

After insuring a trailing ')', the parsing puts out 2 bytes of filler for the relativized PCR (see allocation phase of DEF FN), and checks for an '=' (is this a one-line definition?). If not found, then we are done parsing the DEF FN, and insist that nothing else follow. If found, then we insist on an expression consistent with the type of function being parsed. After the expression we output the invisible function and and 2 bytes of filler for the function value pointer (see the allocation phase of END DEF). Finally, we insist on being the last thing on the line.

DEF FN parsed form, not allocated: (see RH"ICE)

```
(1) DEF FN token
(2) function name
(2) filler (relative jump to end)
(1) count/type
optional parameters:    repeated count times
  (2) param name form
      optional string parameter:
      (2) string max length
  (2) filler (param value pointer)
(2) filler (relative PCR)
(1) EOL or ! token     OR   (n) expression
                       /    (1) invisible END DEF token
    one-line def fn ----    (2) filler (fn value pointer)
                       \    (1) EOL or ! token
```

LET:
Parsing for the LET and invisible LET tokens is the same (SCAN having decided which we have before it gets to us). After checking for program mode, we check that the function type is string or numeric (error if not), and handle the appropriate type of right hand expression.

LET parsed form, not allocated:

```
(1) LET or invisible LET token
(1) variable token, string or numeric
(2) function name
(n) expression
(1) store token, string or numeric
(1) EOL, !, or @ token
```

END DEF:
Parsing for this insists on program mode, pushes the token and 2 bytes of filler for the function value pointer (see allocation of END DEF), and then demands that it be the last thing on the line.

END DEF parsed form, not allocated:

```
(1) END DEF token
(2) filler (function value pointer)
(1) EOL or ! token
```

CALL:

String and numeric function calls differ only in what token is initially loaded. Parsing insists on program mode. After saving parse information, the routine FNNAM+ is called (FNNAM+ is just like FNNAME, see DEF FN parsing, except it does an initial SCAN). FNNAM+ puts the function name, filler, and count/type on the stack, and R30 has the type.

Parsing checks to see if the type of the function called is compatable with the expression it is being used in. If not (or if the name was invalid), the values saved at the start are restored, and the call is flaged as an error for the parser to deal with.

If all is ok so far, the information on the stack from FNNAM+ is removed, and we check to see if any parameters were specifed. If so, the parameter information is parsed and the parameter type is saved on the R6 stack for later (at this point, the real number information removed by FNNAM+ is restored also). Parsing is handled by assuming a numeric expression and trying a string if numeric fails (if string fails also, then clean up the stack and error out). This loop continues as long a comma follows the parameter. After the last parameter a ')' is checked for. Now the call token, function name, and parameter count are pushed out (along with a byte of garbage which is backed over). Using the parameter count, we compute the location on R12 of the last parameter and position R12 there. Now we proceed to pop the parameter types off the R6 stack and move them to the R12 stack (last one first). Finally we restore R12 to the end of this mess, and we are done.

CALL function parsed form, not allocated:

```
(m) preceding part of expression
    optional parameters:  repeated for count parameters
    (n) parameter expression
(1) call token
(2) function name
(1) parameter count
    optional parameters:  repeated for count parameters
    (1) parameter type flag
(p) rest of expression
```

ALLOCATION: (see GC"ALO)
DEF FN:
Allocation first checks to see if we are already allocating a function (if so, we give up and error). If ok, save the pointer to the name (follows the DEF FN token), and get the name, filler, and count/type. The count/type is massaged into the function type flag. Next, the CRDF? routine is called to search for the function variable, and create one if needed. The name is also converted to internal form. If the function variable has not already been encountered (in a call), then CRDF? will allocate a variable pointer block using ALVENT. The parameters to ALVENT are set up assuming a numeric function and then we check the type (if string, the parameters are modified). The call to ALVENT sets up the variable pointer entry for the function variable. The last thing CRDF? does is rearrange the pointer block to look like:

```
+-----------+--------------+----------------+--------------+
| name from | (max str len) | rel value pntr | rel def pntr |
+-----------+--------------+----------------+--------------+
```

If there were no errors in CRDF?, then we check to see if the function variable was found. If so, check to see if the def pntr is non-zero (if it is, the function is already defined and we error on the duplicate). If the def pntr is zero, the relative pointer to the count/type byte in the definition is placed there (using the name pointer saved earlier).

The function name in the definition is replaced with the relative pointer to the pointer block. This pointer is also saved in CURFUN for use by the END DEF allocation and the LET allocation. The pointer to the first function parameter (in the function parameter area) is saved in DFPAR1. This tells the variable search routine to look first in the parameter area (and where that area is). DFPAR1 is also used by the END DEF allocation to fill in the rel jump value and by LET allocation to see if the LET is inside a function definition. The type is stripped out of the count/ type info, and a loop is entered which allocates space for the value of each parameter, and saves the relative value pointer in the parameter variable block. After all parameters are handled, the pointer to the end of the parameter variable block is saved in DFPAR2 (this tells the variable search routine where the end of the block is!). Finally, the filler for the relative PCR is filled in.

LET or invisible LET:
The variable token (following the LET token) is massaged into the function type flag. Then we check to see if DFPAR1 is 0 (if so, error on LET outside function def). If ok, the name is changed to internal form and the variable pointer is compared to the one saved in CURFUN to make sure it is the right function variable (if not, error is reported). The value of CURFUN (which is the same as the relative pointer) is substitued for the name.

END DEF:
First, DFPAR1 is checked to see if a function is being defined (if not, error is reported). The pointer in DFPAR1 is then adjusted to point to the filler for the relative jump. The jump value is computed by taking the current position (absolute - just after END DEF token) and adjusting to skip the following filler and make pointer relative. The relative jump address is then placed in the filler in the function definition. The value in CURFUN is saved in the filler for the function value pointer. Finally, DFPAR1 is cleared to indicate the end of the function definition.

CALL:
The allocation massages the call token to be the appropriate function type flag. CRDF? is called to set up a partial variable entry (if needed, see DEF FN allocation for more on CRDF?). If there were no errors, the name is replaced with the variable pointer, and the parameter type list is skipped over.

Environmental allocation:
INVFN zeros out the function state information. This is called by INIVLP in addition the initialization of the function value to undefined.

RUN:
DEF FN:
Skip over the variable pointer to get to the relative jump address, and jump to that address (EOL or ! following the END DEF token).

LET or invisible LET:
Skip over the variable type token, get and absolutize the variable pointer. Check the name form to see if it is string or numeric. If string, call FTSTLS to set up the R12 stack. If numeric, skip over the pointer, get the absolute value address and put it on R12 as relative address with the name form. The stack is now set up for normal variable storage, and there is nothing further which is different for the function variable.

END DEF:

The invisible function end is slightly different at runtime from the END DEF token. It tests to see if there is a value on the stack (result of evaluating the one-line definition). If there is not, then the invisible end is treated like an END DEF (fetch the value from the function variable). If there is, FNGET is called to restore the machine to the state the function was called in, and then return to where function was called (via FNDRTN).

The END DEF token first calls FNGET. FNGET finds the function state info by locating the function variable value and using the 9 bytes in front of the value (which contain the function state). The rel return address is retrieved (and the value 0'd - see CALL runtime for more on this), if this is already zero then we error out (function was jumped into). Next, the relative PCR is fetched. Then the relative value of TOS at call time is retrieved, absolutized and restored (see CALL runtime for more). Then the old value of E.RMEM is retrieved and updated by the amount accumulated by the function (see runtime CALL for more - this will cause all rsmem'd memory to be released when the expression containing the function call is finished). Finally, R16 is restored to the state of the function call. (I don't know why -- Capricorn saved this, so we do too).

FNDRTN expects the registers set up to imitate a GOSUB return and takes care of absolutizing the pointers, setting the new R10 and PCR, and tracing the return.

CALL:
The runtime stuff for call first gets the variable block pointer as absolute, and then asks the system if there is LEEWAY memory left. This is done by calling ROOM! asking for 0 space at a location known to be below the current R12. If ROOM! returns an error, then there is not enough room left to evaluate a one-line function, so we give up. Next, we locate the function variable value and backup to point to the function state info. If the function return address is already #0 then function calls itself and we error out. The parameter count in the definition and the parameter count in the call are compared (if not the same we have an error), and we go into a loop to process the parameters.

The end of the call parameter type list is calculated and saved for later. The loop backs through the parameter values on the R12 stack assuming the parameter is a string and then adjusting if the type flag from R10 indicates it is a number. This continues until the value of R10 equals the computed end of the list. The new value of R12 is saved, and R10 is put back to the beginning of the list. Starting with the first parameter, the type of the call parameter is compared to the type of the def parameter (if no match, then error out). If the parameter is a string, then the def parameter info (length, value pointer) are fetched and the call parameter info retrieved from R12. (Note that the stack is NOT being used as a stack, so GETAD+ must be compensated for in its absolutization of the addresses on R12). FETCHA and STOST- are used to move the value to the parameter (TRFLAG is changed to turn off trace mode - this prevents garbage display to the user). If the parameter is a number, then ONER- is used to make sure the number is real and the value is moved to the parameter. This continues until the parameter count is down to 0.

After the parameters have been processed, the return address (after the function call) is relativized and saved in the function state info. Then the PCR is processed similarly. The current value of TOS is saved as the difference between TOS and R12 (the END DEF will add this to the new TOS to get back to the old one) and the new TOS is set to R12. The current E.RMEM is saved, and E.RMEM is 0'd (the END DEF will add this saved value to the ending E.RMEM to get a total). R16 is saved (I don't know why). R10 is set to point to the token following the relative PCR in the definition, and the PCR is set to the absolutized PCR from the definition. Finally, the function call is traced.

DEALLOCATION:
DEF FN:
The deallocation of the DEF FN simply replaces the function variable pointer with the function variable name, and skips over the parameter block and relative PCR.

LET and invisible LET:
Does nothing (just an RTN).

END DEF:
Just skip over the variable value pointer.

CALL:
Restore the function variable name, and skip over the parameter type list.

DECOMPLILE:
DEF FN: Output a 'DEF FN' for the token, get the name, skip over the relative jump, get the count/type, and output the name with '$' if necessary. If there are any parameters, output a '(', and go into a loop which translates the parameter name form to a name and puts it with appropriate token and stack marker on the stack for later processing. If the parameter is a string then the user defined string length is also put on the stack (if not 32). This continues until all parameters have been handled, then the ')' is stacked and UNSTAK is called to output the mess on the stack. The trailing blank is removed, and we check for a one line definition. If the next token is not an EOL or !, we assume so, and output a "="; otherwise we put the blank back.

LET or invisible let:
The keyword is moved to the output (LET FN or FN respectively), and the variable token is left for FETVAR to decompile as a normal variable.

END DEF:
The keyword is moved for END DEF (this step is skipped for the invisible end) and the variable pointer is skipped over.

CALL:
Get the function variable name and save it. The parameter type list is skipped through, changing each corresponding marker on the R12 stack (for the parameter expressions - already decompiled and stacked) to a comma. Next, find the start of the parameters on the R12 stack, and insert the function variable name, 'FN', '$' (if necessary), and '('. Finally, add a ')' at the end of the stack. If there are no parameters, then a stack marker and the function name, 'FN', and '$' (if necessary) are added to stack. The call to UNSTAK at the end of the expression containing the function call takes care of outputting all this stuff.