Hewlett-Packard -- Portable Computer Division

Corvallis, Oregon

HP-71 HP-IL Module

Internal Design Specification

VOLUME I

Detailed Design and Entry Point Description

```
%%     %%   %%%%%%%             %%%%%%%%   %%
%%     %%   %%%%%%%%            %%%%%%%%   %%
%%     %%   %%     %%              %%      %%
%%     %%   %%     %%              %%      %%
%%     %%   %%     %%              %%      %%
%%%%%%%%%   %%%%%%%%   %%%%%%      %%      %%
%%%%%%%%%   %%%%%%%    %%%%%%      %%      %%
%%     %%   %%                     %%      %%
%%     %%   %%                     %%      %%
%%     %%   %%                     %%      %%
%%     %%   %%                  %%%%%%%%   %%%%%%%%%
%%     %%   %%                  %%%%%%%%   %%%%%%%%%
```

```
    %%     %%    %%%%     %%              %%%%%%
     %%   %%    %%  %%    %%                %%
      %% %%     %    %    %%                %%
       %%%      %%  %%    %%      %%%       %%
        %        %%%%     %%%%%%  %%%     %%%%%%
```

January 1984

HP Part No. 82401-90023

****** NOTICE ******

Table of Contents

HOW TO USE THIS DOCUMENT	CHAPTER 1

This document describes the software design of the HP-IL module for the HP-71. The HP-IL module is an optional plug-in for the HP-71 which adds I/O capability to the HP-71. The hardware of the HP-IL module includes an HP-71 ROM and an I/O processor. The HP-71 CPU communicates with the I/O processor through a mailbox. The I/O processor controls the HP-IL loop to perform the HP-IL operations.

When we use the term "HP-IL module", we mean the HP-71 ROM and the I/O processor. When we use the term "HP-IL ROM", we mean only the HP-71 ROM.

The role of the I/O processor is to take commands from the HP-71 and send the necessary messages on the HP-IL loop. The role of the HP-IL ROM is to provide additional keywords to the HP-71 for I/O operations. It also extends some of the keywords in the HP-71 to allow access to devices on the HP-IL loop.

The purpose of this document is to provide information to those users who want to access the routines in the HP-IL ROM. If an assembly language application is being written, all of the utility routines in the HP-IL ROM are accessible to the user. The utilities may be useful for adding keywords to the HP-71 or speeding up an application.

The second chapter provides some design notes of the HP-IL ROM. This chapter describes in detail the implementation of certain features in the HP-IL Module. The information includes:

- System RAM usage
- I/O buffer usage
- How interrupts are implemented
- How multiple loops are implemented
- How frame timeouts are implemented

The third chapter gives the loop specifier syntax for HP-IL keywords. At the present time, unless special hardware is provided, there is no way to plug another HP-IL mailbox into the HP-71. However, the firmware of the HP-IL module is capable of handling up to three mailboxes, each mailbox is treated as a separate loop. Whenever a device is specified, an optional loop specifier is also allowed, which specifies which loop the device is in. However, the syntax of the optional loop specifier was not published in the manual of the HP-IL module. The intention is to publish the loop specifier syntax with the future product which allows the user to plug in an additional HP-IL loop (such as a Port Extender).

The fourth chapter has some examples of what frames the HP-71 actually sends out to perform some basic HP-IL operations, such as powering up the loop, auto addressing the loop or searching for a device. This information may be useful to those people who want to implement an HP-IL interface in their device. They may want to know what frames to expect from the HP-71 for some simple operations.

The fifth chapter describes the lowest level utilities. It describes how to send messages to and receive messages from the I/O processor. All HP-IL commands and data transfers go through a mailbox to the I/O processor. If a poll handler or utility routine can not be found which implements a required special function, messages can be sent directly to the HP-IL mailbox. At this level, the user has control of the loop at a frame by frame level.

The sixth chapter describes all the polls answered by this module. The HP-IL ROM is a soft configured ROM, which means the routines in this module do not have a fixed address. The simplest way to access a routine in this module is by issuing a poll which the HP-IL module will answer. A poll can be issued without knowing the address at which the HP-IL module is configured. Please refer to the HP-71 IDS for more information about how to use polls.

The final chapter describes all of the utility routines in this module. There are many utility routines which can not be accessed through polling. These routines may be accessed by a direct call. As we mentioned earlier, the HP-IL ROM does not have a fixed address. To directly call a utility routine, first find the starting address of the HP-IL ROM in the configuration tables. Then add to the starting address, the offset of the utility routine into the HP-IL ROM and the jump to this address. To simply calling routines in the HP-IL ROM directly, we have provided a routine, JUMPER, in this chapter which will search the configuration tables for the address of the HP-IL ROM. Therefore, calling a utility routine in the HP-IL ROM is simplified to executing a GOSBVL to JUMPER and providing the offset of the utility routine from the start of the HP-IL ROM. The offset of each routine can be found in this chapter, along with the description of each routine.

```
+-----------------------------------------------+------------------+
|                                               |                  |
|   INTERNAL DESIGN NOTES                       |   CHAPTER  2     |
|                                               |                  |
+-----------------------------------------------+------------------+
```

The purpose of this chapter is to describe the implementation details for some of the features of the HP-IL ROM. These include system RAM and I/O buffer usage, standard display and print device assignments, interrupts, multiple loop capability and loop integrity maintenance.

2.1 System RAM usage

The following locations in system RAM are used by the HP-IL ROM:

```
ONINTR : Address of the ON INTR statement
IS-DSP : Display device assignment
IS-PRT : Print device assignment
MBOX^  : HP-IL mailbox address
LOOPST : HP-IL loop status
DSPSET : Display device set up
TERCHR : Terminating character for ENTER
```

2.1.1 ON INTR address

```
Symbol   : ONINTR
Location : #2F68D
Length   : 5 nibbles
Contents : Holds the address of last executed ON INTR statement
```

This address is set by the ON INTR statement. The address points to the ON INTR statement, not the interrupt service routine.

This address is cleared when RUN is executed. The ONINTR address is associated only with the current program. If a CALL statement is executed, the current value of the ONINTR location is saved and then the ONINTR location is cleared, before execution of the subprogram begins. The ONINTR address of the calling program is restored when an ENDSUB or END is executed.

2.1.2 DISPLAY IS assignment

```
Symbol   : IS-DSP
Location : #2F78D
Length   : 7 nibbles
Contents : Current standard display device assignment
```

Assignment encoding:

```
Nibs from
base adr:  Usage:
---------  ------
    2-0:   If device address known, address, loop # here
           If LOOP, nibs 1-0=0, nib 2 is loop #
           If NULL, F00
           If not known/not assigned/IO buffer, FFF
           If assigned, not HP-IL, Fxx, xx<>FF

      3:   If unassigned/not HP-IL, F
           If IO buffer with one entry, 4
           If address specified, 0
           If type specified, loop # + 1 (nib 3: 1,2,3)
           If this assignment has been "OFF"ed, bit 3 is 1

    6-4:   If type, nib 6: sequence #, nibs 5-4: Acc id
           If address, 6-4: address, loop #
           If IO buffer, 6-4: IO buffer #
           If unassigned (NOT "OFF"ed), FFF
           If not HP-IL and nib 3=F, not defined
```

If an IO buffer is used for the display assignment, it is holds the assignment as shown below:

```
 +---------------------------------------------------------+
 | Device ID/vol Lbl | search type | loop # | sequence # |
 +---------------------------------------------------------+
nibs:     16                 1           1         1
(high memory)                                 (low memory)
```

At cold start, if the HP-IL module is not present, the value of IS-DSP is set to FFFFFFF.

The initial value of IS-DISP at cold start with the HP-IL ROM present depends on whether a display device is found in loop 1. If no display device (with accessory ID 3X) is found the value will be set to 03F1FFF. If a display device is found, the value will be set to the address of the first display device in the loop. Whenever the HP-IL ROM detects it was just added to the HP-71, the initial value of the display device assignment will be set to this default value.

Anytime a loop broken condition is detected while trying to send characters to the display device, bit 3 of nibble 3 will be set to 1 and nibbles 2-0 are set to FFF. This setting will cause the HP-IL ROM to stop sending data to the display device. When a RESTORE IO is executed or the HP-71 is turned OFF and ON again, the HP-IL ROM will search for a display device. If the assigned display device is found, nibbles 2-0 will be set to the address of the device and bit 3 of nibble 3 will be cleared.

If the ATTN key is pressed while displaying, it will cause bit 0 of DSPSET (location #2F7B1) to be cleared and characters will no longer be sent to the display device. This bit will be set again when the HP-71 goes through the main loop trying to display the prompt character. At this time, the HP-IL ROM will try to restore the display device.

2.1.3 PRINTER IS assignment

```
Symbol   : IS-PRT
Location : #2F794
Length   : 7 nibbles
Contents : Current print device assignment
```

Assignment encoding:

```
 Nib from
 base adr:  Usage:
 ---------  ------
     2-0:   If device address known, address, loop # here
            If LOOP, nibs 1-0=0, nib 2 is loop #
            If NULL, F00
            If not known/not assigned/IO buffer, FFF
            If assigned, not HP-IL, Fxx, xx<>FF

       3:   If unassigned/not HP-IL, F
            If IO buffer with one entry, 4
            If address specified, 0
            If type specified, loop # + 1 (nib 3: 1,2,3)
            If this assignment has been "OFF"ed, bit 3 is 1

     6-4:   If type, nib 6: sequence #, nibs 5-4: Acc id
            If address, 6-4: address, loop #
            If IO buffer, 6-4: IO buffer #
            If unassigned (NOT "OFF"ed), FFF
            If not HP-IL and nib 3=F, not defined
```

If an IO buffer is used for the printer assignment, the assignment is saved in the following way:

```
+---------------------------------------------------------+
| Device ID/vol Lbl | search type | loop # | sequence # |
+---------------------------------------------------------+
nibs:      16               1           1          1
(high memory)                                (low memory)
```

If the HP-IL module is not present at cold start, the IS-PRT location is initialized to FFFFFFF.

The initial value of IS-PRT at cold start with the HP-IL ROM present, depends on whether a printer device is found in loop 1. If no printer device (with accessory ID 2X) is found the value will be set to 02F1FFF. If a printer device is found, the value will be set to the address of the first printer device in loop 1. Whenever the HP-IL ROM detects it has just been added to the HP-71, the printer assignment will be set to this default value.

Everytime a PRINT statement is executed, the loop is searched for the printer device, as specified by the current assignment.

2.1.4 Last mailbox address

Symbol : MBOX^
Location : #2F7A9
Length : 3 nibs
Contents : Mailbox address of last accessed mailbox

In executing many of the HP-IL keywords, the first step is to find the address of the mailbox in the configuration tables. This can be easily accomplished by calling the routine FNDMBX. FNDMBX saves the address of the mailbox in the system RAM location MBOX^. This eliminates the need to save the mailbox address in a CPU register during execution of a statement. The routine GETMBX will load the mailbox address into D0.

The mailbox address is 5 nibbles long. However the most significant nibble will always be a 2. This is because the mailbox is a Memory- mapped I/O type device and so it will always be configured in the address range 20000-2C000. The least significant nibble will always be a 0, since the memory size of a device is alway a multiple of 16 nibbles. Since the value of the top and bottom nibbles are always known, RAM is allocated for the middle three nibbles only. The routine GETMBX will supply the top and bottom nibbles of the mailbox address.

2.1.5 HP-IL loop status

Symbol : LOOPST
Location : #2F7AC
Length : 1 nibble
Contents : Holds the status of the HP-IL loop

Bit #	Meaning
3	Set by OFF IO command Cleared by RESTORE IO command
2	When set indicates the last mailbox accessed is in device mode. The routine START will set or clear this bit everytime it is called.
1-0	Cleared by the routine START everytime it is called. These two bits are not used at the present time.

This nibble is initialized to zero at cold start or when the HP-IL ROM is first added to the HP-71.

2.1.6 Display device status

Symbol : DSPSET
Location : #2F7B1
Length : 1 nibble
Contents : Indicates the type of device to which the display is assigned and the current status.

Bit #	Meaning
3	Set when the display device has been set up to receive data. The routine START clears this bit everytime it is called.
2	Set if the display device is a HP82163A Video Interface.
1	Set if the display device is a printer.
0	Set means the display device is OK. Clear if the ATTN key has been hit or the loop dies while displaying. Mainloop sets this bit again.

Bits 2 and 1 are used to indicate whether the display device is a HP82163A or a printer. If both bits are set, this indicates the display is neither a HP82163A nor a printer type device. In other words, the accessory ID is not 30 hex or 2X hex. This nibble is initialized to 7 at cold start and when the HP-IL ROM is first added to the HP-71.

2.1.7 ENTER terminating character

Symbol : TERCHR
Location : #2F97D
Length : 2 nibbles
Contents : Defines the terminating character for ENTER statement

This character is initialized to the line-feed character (0A) by the HP-IL ROM at cold start or when the module is added to the HP-71.

The HP-IL ROM does not provide any keywords for the user to change the terminating character. PEEK$ and POKE can be used to change it if required by the application.

2.2 I/O buffer usage

Several I/O buffers are used by the HP-IL ROM for various purposes:

bPILSV - HP-IL save buffer, a indication of the ROMs presence.
bPILAI - Contains ASSIGN IO names.
bSTMXQ - HP-IL statement execution buffer

2.2.1 HP-IL save buffer

Symbol : bPILSV
I/O buffer number : #80F

This buffer is created by the HP-IL ROM at cold start or when the ROM is first added to the HP-71. No information is stored in the HP-IL save buffer.

Everytime the HP-71 wakes up from deep sleep it will issue the deep sleep wake up poll. During this poll the HP-IL ROM will check to see if this buffer exists. If the buffer is not found, the HP-IL ROM will assume the I/O processor has not been initialized. The HP-IL ROM will create this I/O buffer and execute an initialization sequence which includes the following:

1. Initialize all the mailboxes found.
   *Set IDY timeout to 50 milliseconds.
   *Set up the accessory ID and the device ID.
2. Initialize DISPLAY IS and PRINTER IS assignments.
   *Write 03F1FFF to IS-DSP which indicates the display device

is unassigned but defaults to the 1st device in loop 1 with an accessory ID of 3X.
*Write 02F1FFF to IS-PRT which indicates the print device is unassigned but defaults to the 1st device in loop 1 with an accessory ID of 2X.
3. Set ENTER terminating character to line feed character (0A).

2.2.2 ASSIGN IO buffer

Symbol: bPILAI
I/O buffer number: #810

This I/O buffer is created by the ASSIGN IO statement. Its length is always 122 nibbles (30 entries * 4 + 2 nibs of 00). The ASSIGN IO statement can have up to 30 assign words. Each assign word takes 2 bytes in the ASSIGN IO buffer. The two bytes are the two characters of the assign word. If an assign word has only one character, the second character is zero filled.

2.2.3 HP-IL Statement Execution Buffer

Symbol: bSTMXQ
I/O buffer number: #811

This I/O buffer is allocated by the HP-IL ROM whenever data is received remotely as a device. When the HP-71 is a device and in remote mode, any data received will be interpreted as BASIC commands. The data received is put into this buffer by the HP-IL ROM for subsequent execution by the HP-71.

2.3 Decoding a device specifier

The method used to find a device on the loop, is dependent upon how the device is specified. The algorithm for decoding a device specifier is given below:

```
IF <dev spec> starts with a "."
   THEN <dev spec> is a volume label
IF <dev spec> starts with a "%" sign
   THEN <dev spec> is an accessory ID
IF <dev spec> starts with a "*" sign
   THEN <dev spec> is "*"
IF <dev spec> starts with a numeric character
   THEN <dev spec> is a HP-IL address
```

```
IF <dev spec> is one of the ASSIGN words
   THEN get the HP-IL address from the ASSIGN IO buffer.
IF <dev spec> is one of the reserved words
   THEN get the accessory ID from the reserved word table.
IF <dev spec> is "NULL" or "LOOP"
   THEN <dev spec> has no address
   ELSE <dev spec> is a Device ID.
```

2.3.1 How file and device specifiers are tokenized

File spec. tokenization:

```
1)    <string expression>
2) or  <tLITRL> [ <file name> ] <tCOLON> <device specifier>
3) or  <tLITRL> [ <file name> ] <tSEMIC> <volume label>
```

Device spec. tokenization:

```
1)    <string expression>
2) or <tCOLON> <HP-IL address>
3) or <tCOLON> <tLITRL> <device word> [ <tSEMIC> <loop number> ]
4) or <tCOLON> <t%> <num expr> [ <tSEMIC> <loop number> ]
5) or <tCOLON> <tLITRL> <assign word>
6) or <tCOLON> <tLITRL> <device ID> [ <tSEMIC> <loop number> ]
7) or <tCOLON> <t*>
```

2.3.2 Reserved device word table

The table entry structure is:

```
1 nibble: length of name minus 1, in nibbles (n-1)
n nibbles: name (Bytes in order!)
2 nibbles: accessory ID
```

The table consists of entries terminated by length nibble of 0.
The table is listed below:

```
NIBHEX 7            Length of "TAPE"
NIBASC \TAPE\       TAPE:TYPE=10
NIBHEX 01
NIBHEX D            Length of "MASSMEM"
NIBASC \MASSMEM\    MASSMEM:TYPE=1F (MASS MEM. CLASS)
NIBHEX F1
NIBHEX D            Length of "PRINTER"
NIBASC \PRINTER\    PRINTER:TYPE=2F (PRINTER CLASS)
```

```
        NIBHEX F2
        NIBHEX D             Length of "DISPLAY"
        NIBASC \DISPLAY\     DISPLAY:TYPE=3F (DISPLAY CLASS)
        NIBHEX F3
        NIBHEX 7             Length of "GPIO"
        NIBASC \GPIO\        GPIO:TYPE=40
        NIBHEX 04
        NIBHEX 9             Length of "MODEM"
        NIBASC \MODEM\       MODEM:TYPE=41
        NIBHEX 14
        NIBHEX 9             Length of "RS232"
        NIBASC \RS232\       RS232:TYPE=42
        NIBHEX 24
        NIBHEX 7             Length of "HPIB"
        NIBASC \HPIB\        HPIB:TYPE=43
        NIBHEX 34
        NIBHEX D             Length of "INTRFCE"
        NIBASC \INTRFCE\     INTRFCE:TYPE=4F
        NIBHEX F4
        NIBHEX D             Length of "INSTRMT"
        NIBASC \INSTRMT\     INSTRMT:TYPE=5F (INSTRMT CLASS)
        NIBHEX F5
        NIBHEX D             Length of "GRAPHIC"
        NIBASC \GRAPHIC\     GRAPHIC:TYPE=6F (GRAPHIC I/O)
        NIBHEX F6
  END OF TABLE INDICATOR...NULL
        NIBHEX 0
```

2.4 HP-IL ROM and Mailbox interface

The HP-IL module has a ROM containing an HP-71 LEX file and an I/O processor. The function of the ROM is to extend the BASIC language to include I/O capability on HP-IL. The HP-IL ROM talks to the I/O processor through a mailbox. It is just like a person would use the mailbox for their mail. If the HP-IL ROM wants to send something out, it will put the information in the mailbox and set a flag. If the I/O processor has a message for the HP-IL ROM, it will put it in the mailbox and set a different flag.

The implementation of the feature set of the module if shared by the HP-IL ROM and the I/O processor. The HP-IL ROM is responsible for moving data between the HP-71's memory (such as variables and files) and the mailbox. The I/O processor is responsible for moving data between the mailbox and the loop.

The HP-IL ROM and the I/O processor work together on things other than transferring data too. These include setting up frame timeouts, interpreting remote data and generating interrupts to the HP-71. The following section describes how these features are implemented.

2.4.1 How frame timeouts are implemented

The STANDBY statement takes two parameters:

1. Timeout period: defines how long the HP-71 waits for each HP-IL message to travel around the loop, back to the HP-71.
2. Verify interval: defines how often the HP-71 tests the loop's continuity by sending an HP-IL Identify message. The the Identify message travels around the loop quickly.

The I/O processor stores and uses the frame timeout values. These two parameters are not directly sent to the I/O processor. The HP-IL ROM converts the STANDBY parameters into three timeout parameters used by the I/O processor:

1. Frame timeout: Specifies how long to wait for the frame to timeout before sending out an IDY frame.
2. IDY timeout: Specifies the maximum time to wait for the IDY to return before setting the loop broken error.
3. Number of IDYs: Specifies the maximum number of frame time-outs to allow, before setting the frame timed out error.

The HP-IL ROM always sets the IDY timeout to 50 milliseconds. The STANDBY statement causes the Number of IDYs to be set to the CEIL(the timeout period/the verify interval) and sets the frame timeout to the verify interval. If only the timeout period is specified, it will be used to set the frame timeout and the number of IDY's is set to 1.

The HP-IL ROM initializes the frame timeout to 2 seconds and the number of IDY's to 30. This means if the loop is broken it will be detected within 2 seconds. If the loop is complete, a message sent has to return in 60 seconds. When a STANDBY OFF is executed, these default values are used. When STANDBY ON is executed, the frame timeout will be set to infinity. This means the loop will never been tested with an IDY message and the HP-IL module will wait forever for a message to return.

When the HP-IL ROM begins execution of a statement, it first clears the I/O processor error code by reading the error and ignoring it. From that point on, when the HP-IL ROM wants to send a message to the mailbox, it first checks the status of the mailbox by looking at the error bit. If the mailbox reports an error has happened, the HP-IL ROM will error out. If the mailbox is ready to receive a message, the HP-IL ROM will write the message to the mailbox. If necessary the I/O processor will send message(s) to the loop and wait for them to return. If a message takes too long to return, the I/O processor will start to send out IDY's to test the loop. When the message finally returns, the I/O processor will error check it. If the message does not return in time or a transmit error has been detected, the I/O processor will set the error bit in the mailbox. The HP-IL ROM will know an error has happened on the last transmission when it trys to send the next message.

2.4.2 Interpreting data when in remote mode

The HP-71 can operate as a device in the loop. There are several ways to cause the HP-71 to give up control of the loop:

1. Execute a CONTROL OFF.
2. Execute a PASS CONTROL.
3. Send the HP-71 an IFC message. Whenever an IFC is received which the HP-71 did not source, the HP-71 will give up control. Controller status is cleared. If the HP-71 is already a device, the HP-71 just executes the the IFC command (untalks and unlistens itself).

A controller can send BASIC commands to the HP-71 when it is a device. The controller has to put the HP-71 in remote mode to cause the HP-71 to interpret the ASCII data it receives as a BASIC statement. The implementation is described below:

1. When the I/O processor has data available, it will generate a service request line on the HP-71 processor bus. This is the same service request as would be generated by the timer.
2. The HP-71 will check first to see if the service request is generated by the timer. If it is not, then the HP-71 will issue the service request poll to give LEX files a chance to process the service request.
3. When the HP-IL ROM receives this poll, it will check if the service request is generated by the I/O processor. If it is, the handshake byte of the mailbox is read to see if the SRQ bit is set. If there are multiple mailboxes found in the configuration table, the HP-IL ROM will look for the first mailbox which has the service request bit set.
4. After finding a mailbox with the service request bit set, the HP-IL ROM will read its status and instruct the I/O processor to clear the service request bit in the mailbox.
5. The status of a mailbox will indicate the reason it is requesting service. There are two reasons the I/O processor may want to request service:
   a. An interrupt has occurred
   b. Data is available from the loop (only as a device).

If the reason for requesting service is interrupt occurred, the HP-IL ROM will set the system flag called Exception flag (S12) and return. Subsequently, the Exception flag will cause the HP-71 to issue the exception poll, and the interrupt will be serviced in the exception poll handler.

If the reason for requesting service is data available, the HP-IL ROM will implement the following checks:

a. If the HP-71 is not in remote mode, then the poll handler is completed. In this case the data will be held by the I/O processor and service will be requested until the data has been read by the HP-71 (usually by an ENTER statement).
b. If the HP-71 is in remote mode, the HP-IL ROM will then check if the HP-71 is idle. Idle means the HP-71 is not running a program, not in the CALC mode, and not executing an INPUT statement. If the HP-71 is not idle, the poll handler will return immediately as in the previous case.

   If conditions match on any of the above 2 cases then the poll handler will return without doing anything, the HP-71 will keep issuing the service request poll until there is no longer a service request on the HP-71 processor bus.

If the HP-71 is in remote mode and it is idle, the HP-IL ROM will first clear the key buffer and then put a single key code in the key buffer. The key code put into the key buffer is "FF". It is a key code the HP-71 won't recognize. Whenever the HP-71 finds an unrecognized key code in its key buffer, it will issue the KYDF (key def) poll to see if any LEX file knows how to interpret it. The HP-IL ROM checks the key code when the KYDF poll is issued. If the "funny" key code is "FF", then the HP-IL ROM knows it has data available in remote mode. The HP-IL ROM finds the mailbox which has data available, then creates an I/O buffer, reads the data from the mailbox and puts it into the I/O buffer. When this is done, the HP-IL ROM will set the input buffer pointer to the I/O buffer and return. The HP-71 will treat the ASCII characters in the input buffer as if they were typed in from the keyboard by the user. The ASCII string will be parsed and executed.

2.5 How interrupts are implemented

A user program can use the interrupt capability of the HP-IL module through the following keywords provided by the HP-IL ROM:

1. ON INTR GOTO/GOSUB <line number>
   This statement identifies and enables a branch to the

interrupt service routine.

2. ENABLE INTR <interrupt mask>
   This statement defines what events the program wants to enable for interrupts.

3. OFF INTR
   This statement clears the address of the interrupt service routine defined by the ON INTR statement. The effect is to cause an interrupt to become pending if it ever happens. This is a way to temporarily disable interrupts. The interrupt can be reactivated by the ON INTR statement.

4. READINTR
   This function can be used to find out what the caused the interrupt.

Execution of the ON INTR statement simply writes the address of the ON INTR statement to a reserved location in system RAM. In this way, the HP-IL ROM can tell whether a GOTO or a GOSUB should be executed when a branch to the interrupt service routine occurs.

The ENABLE INTR will send the interrupt mask to the I/O processor, since the I/O processor keeps track of the interrupt events. Two bytes are used by the I/O processor to monitor the interrupt events:

a. Interrupt mask byte
   This is the byte set by the HP-IL ROM to indicate to the I/O processor which of the 8 events are enabled to generate an interrupt. This byte will be cleared by the HP-IL ROM automatically in the following cases:
   1. Immediately before the end-of-line branch is taken to the interrupt service routine.
   2. At the end of program execution or whenever an EDIT is executed.

b. Interrupt cause byte
   There are total of 8 events that can cause interrupts. The 8 bits of this byte are a record of each of the 8 events. An event is recorded, regardless of whether or not that particular event is enabled to generate an interrupt.

Every time an interrupt event occurs, the corresponding bit in the interrupt cause byte is set to 1. The I/O processor will compare the interrupt cause byte and the interrupt mask byte. If any of the bits matched, the I/O processor will request service on the HP-71 processor bus. Everytime the HP-71 wakes up from light sleep or at the end of each statement execution, it will check for a service request on the HP-71 bus. If there is a request, the HP-71 will check if it is the timer. If it is, the request will be handled by the HP-71. Otherwise, the HP-71 will issue the service request poll to give external LEX files a chance to service the request.

When the HP-IL ROM receives this poll, it will check if the service request is generated by an I/O processor. If so, the handshake byte of the mailbox is read to see if the service request bit is set. If there is more than one mailbox found in the configuration table, the HP-IL ROM will look for the first mailbox which has the SRQ bit set. After finding a mailbox with the service request bit set, the HP-IL ROM will read its status.

The status of a mailbox will indicate the reason for the service request. There are two reasons the mailbox will request service:

a. An interrupt has occurred
b. Data is available from the loop (only as a device).

If the reason for requesting service is interrupt occurred, the HP-IL ROM will set the system flag called Exception flag (S12) and return. Subsequently, the Exception flag will cause the HP-71 to issue the exception poll. The interrupt branch is implemented in the exception poll handler.

The exception poll handler will implement the following:

1. Read the mailbox status to see if it has a pending interrupt. If more than one mailbox exists, the first mailbox with a pending interrupt will be serviced.

2. Check if the ON INTR address is non-zero. If it is zero, the Exception flag (S12) is set and the poll handler returns.

3. Check if the HP-71 is running a program. If is not, the Exception flag is set and the poll handler returns.

4. Check if statement which has just been executed is at the end of a line. If not, the Exception flag is set and the poll handler returns.

If the Exception flag is set, the HP-71 will continue to issue the exception poll when it finishes executing the next statement or when it next wakes up from light sleep.

When all the above conditions are met, the HP-IL ROM will clear the interrupt mask and cause program execution to branch to the interrupt service routine.

The purpose of automatically clearing the interrupt mask is to prevent re-entering the interrupt service routine when already in the routine. The user program should reactivate the interrupt at the end of the service routine. If the last statement of the interrupt service routine is a RETURN, be sure to put the ENABLE INTR in the same line as the RETURN. Otherwise, if there is an interrupt pending, doing an ENABLE INTR will cause an end-of-line branch to take place before the RETURN is executed.

However, the interrupt cause byte will still function as usual, meaning it will still keep recording any interrupt events that occur. The interrupt cause byte will be cleared only when read by the READINTR function. Therefore, it is very important for the interrupt service routine to read the interrupt cause byte. If the interrupt cause byte is never read, it will not be zeroed. Then when interrupts are enabled at the end of the interrupt service routine, it will cause the interrupt branch to happen instantly. Everytime the interrupt mask is set, the I/O processor will compare the new mask and the interrupt cause byte. If there is a match in any of the bits, the I/O processor will generate an interrupt right away. So if the interrupt cause byte is not cleared, false interrupts will be generated.

2.5.1 Disabling interrupts

There are two ways to disable interrupts:

1. OFF INTR
   This statement clears the address set up by the ON INTR statement. It has no effect on the interrupt mask or cause byte. An interrupt becomes pending if it happens after an OFF INTR. Everytime an ON INTR is executed, a check is made for any pending interrupts. If there are pending interrupts an end-of-line branch will take place.

2. ENABLE INTR 0 (clears the interrupt mask)
   Zeroing the interrupt mask will prevent interrupt branching. Zeroing the mask byte will guarantee that no bits will be set when the mask byte is anded with the cause byte. So, the I/O processor will never generate a service request due to an interrupt.

2.6 HP-71 Requesting Service in Device Mode

The HP-71 can share control of the loop with other controllers. As a device, the HP-71 has the capability to get the attention of the active controller by requesting service. The REQUEST statement is a BASIC keyword which can cause the HP-71 to request service on the loop.

The REQUEST statement takes an integer parameter in the range 0 to 255. The parameter is the value of status which is returned to the controller whenever the HP-71 is polled for its status. The parameter is sent to the I/O processor where it is saved in a byte reserved for the current status. Everytime the I/O processor is polled for status, this byte is sent out automatically.

The status byte is initialized to zero at power on and will remain zero unless the I/O processor receives a new status byte from the HP-71 processor. When the I/O processor receives a new status byte from the HP-71 processor, it will do two things:

1. Save the new status byte in RAM.
2. If the I/O processor is in device mode, the loop service request status will be updated as the new status byte indicates. This process is described below:

   If the bit 6 of the status byte is set, it means service request is set on the loop. The service request bit is set on any IDY, DAB or END frames which pass by the HP-71. Also if the loop is in EAR mode (Enable Asynchronous Requests), the I/O processor will start sourcing IDY frames. EAR mode enables a device's ability to source IDYs when it needs service. This way the controller does not have to constantly send frame to monitor service requests.

   If the bit 6 of the status byte is zero, the IO processor will stop requesting service on the loop. If the I/O processor was not requesting service, no change is made.

   If the I/O processor is controller, service will never be requested on the loop.

A RESET HPIL will cause the status byte to be set to 0. Bit 7 indicates to the controller how to interpret the status byte. If bit 7 is set, it means bits 5-0 should be interpreted as system status. If the bit 7 is zero, it means bits 5-0 should be interpreted as device dependent status. Refer to the HP-IL Interface Specification document for the details on status responses.

2.7 Implementing Multiple Loops

The HP-71 only needs one HP-IL ROM which can communicate with more than one mailbox plugged into the HP-71. There can be more than one HP-IL ROM plugged into the HP-71, but only the first one will be accessed.

The interface between HP-71 processor and the I/O processor is through a mailbox. The mailbox is soft configurable in the HP-71 address space. This means the mailbox must be configured for the HP-71 processor to communicate with it. The HP-71 processor system is reconfigured whenever it is possible a change in the system plug-ins could have occurred, such as when turning on or when a module pulled interrupt occurs. A configuration table will be generated as the result of the configuration. From the configuration table, it is possible to find out how many mailboxes are configured and at what address they reside.

It is quite easy for the HP-IL ROM to handle more than one mailbox. If a device specifier does not specify the loop number, the HP-IL ROM will search the configuration table to find the address of the first mailbox in the table. The order the mailboxes appear in the configuration table is consistent with the order of the ports in which the loop reside. The port on the back of the HP-71 is port 0, and will always be the first mailbox in the configuration table. If a loop number has been specified, the HP-IL ROM searchs for the Nth mailbox's address in the configuration table.

2.7.1 Status Information Allocation

Certain status and assignments are saved by the HP-IL module. The following lists specify who saves the information, either the HP-IL ROM or the I/O processor.

The HP-IL ROM saves the global information (true for all loops):

1. PRINTER IS and DISPLAY IS assignments.
2. ASSIGN IO assignments.
3. OFF IO (it affects all the mailboxes).
4. Terminating character for ENTER.
5. Flags -21, -22, -23, -24.

Each I/O processor saves the following information:

1. Controller or device status.
2. Interrupt mask and interrupt cause byte.
3. Last received DDT or DDL frame.
4. Its own status, such as talker active or listener active.
5. Manual mode status. This status will be checked by the HP-IL ROM everytime it trys to talk to a mailbox. If the mailbox is in manual mode, the HP-IL ROM may generate an error and will send no messages to the I/O processor.
5. Frame Timeout settings.
6. Device ID and Accessory ID.
7. Loop address table.

(Ending AES address, ending AEP address, ending AAD address)
8. Whether or not the mailbox has been initialized.
9. Loop powered up status.

2.8 How to find out the capacity of a mass memory device

The HP-IL ROM is able to control one type of mass memory device. Only a device with accessory ID of 10 hex will be recognized by the HP-IL ROM. The HP-71 assumes a device of this Accessory ID uses the HP82161A digital cassette drive protocol.

In order to control a mass memory device larger than the HP82161A the HP-IL ROM assumes certain extensions will be implemented by these future products. The following extensions are described below.

The HP82161A protocol has been extended by the addition of the following DDCs:

DDT6: Send physical attributes -
When SDA is received after a DDT6, the device should send 12 bytes that represent the 6 words of the LIF extension field. The LIF extension field consists of the following information:
Words 12 and 13 contain the number of tracks per surface, with the most significant word being word 12. Words 14 and 15 contain the number of surfaces per medium, and words 16 and 17 containing the number of sectors per track.
The first byte sent should be the most significant byte of word 12 and the last byte the least significant byte of word 17. All the above numbers are in binary.

DDT7: Send maximum address -
When SDA is received after a DDT7, the device should send 2 bytes that represent the record number (in binary) of the last (highest) physical record. The most significant byte is to be sent first.

DDL11 (11 is decimal): Verify records -
After a DDL11 is received, the next two bytes are to be interpreted as the number of records to verify (verify means read and verify checksum is correct and record can be found). The first byte received is most significant 8 bits of the number of records. Verification starts at the next record (set previously by SEEK) and continues until all records are verified or an error is detected or end of medium is reached. Device status will reflect the results of the verify (all OK or checksum error or record not found)

One clarification on the DDL4 (SEEK) command:
The two bytes of data following a SEEK command are to be interpreted as a logical record number (where the device appears to the controller to be organized as an array of records numbered from 0 to MaxRec).

The HP-71 uses the extended HP82161A protocol only during medium initialization.

The HP-71 uses DDT6 to determine the information to be written into the LIF extension field. If the device responds to the subsequent SDA with an ETO, the HP-71 assumes the device is a (non-extended protocol) HP82161A which has LIF extension field: 00 00 00 02 00 00 00 01 00 00 01 00 (hex). These indicate that the digital cassette has 2 tracks per surface, 1 surface/medium, and 100 (hex) sectors/track.

The HP-71 uses DDT7 to determine the size of the device (for checking requested directory size for valid range). If the number of directory entries requested by the user is greater than the number of records that would be left in the data portion of the medium then an error is generated.

The HP-71 does not use the DDL11 (Verify) command.

EXTENDED COMMAND SYNTAX	CHAPTER 3

There is a hidden feature in the HP-IL module that has not been mentioned in the user manual of the module. That is the ability to handle multiple loops. At the time the manual was written, there was no hardware available to allow the user to plug in additional loops. So the user's manual did not include loop numbers in the syntax of the keywords. The purpose of this chapter is to furnish that information.

3.1 Loop Number Specifier

The HP-IL ROM firmware is capable of handling up to three HP-IL mailboxes. It can treat each HP-IL mailbox as a different loop which plugs in to the HP-71. So when you specify a device, you can also specify which loop the device is in.

The optional loop number can be 1, 2, or 3. If the loop number is not specified, the default is loop 1. The HP-IL module plugged into the back of the HP-71 is always loop 1. The 2nd and 3rd loops are determined by the order they are configured by the HP-71.

Generally, the way to specify the optional loop number is to append a colon and a loop number to the device specifier. For example,

```
PRINTER IS PRINTER:2
```

assigns the standard print device to the first printer in the second loop.

```
OUTPUT %16(2):3 ; "abcde"
```

outputs the string to the second device in the third loop that has the accessory ID of 16.

```
ENTER LOOP:2 ; A$
```

reads in data from the second loop.

There are two other cases. In some statements, such as CONTROL ON, no parameter is required. You only have to specify the loop number as a numeric expression in this type of statements.

CONTROL ON 2

sets the controller role in the second loop.

The other case is in a statement such as ENABLE INTR, where a parameter is required but not the parameter is not a device specifier. In this case, specify the loop number as the first parameter and separate it from the other parameter by a semicolon.

ENABLE INTR 2 ; 64

sets the interrupt mask to 64 in the second loop.

General rules of specifying the optional loop number:
----

1. If the statement requires a device specifier, append a colon followed by a numeric expression to the device specifier.
2. If the statement takes no parameter, simply specify the loop number by a numeric expression.
3. If the statement takes a parameter but not a device specifier, insert the loop number as the first parameter and follow it by a semicolon.

3.2 Syntax Identifier Definitions

This section describes the identifier words used under the SYNTAX section of the following keyword descriptions. The identifier words are listed here in alphabetical order along with their definitions:

<accessory ID> ::= 0 | 1 | ..... | 255

**

<assign word> ::= " [:] <alpha> [<alpha> | <digit>] "

<device> ::= { <HP-IL address> | <device word> | <device type> |
<assign word> | <device ID> }

<device ID> ::= <ID string> [(<sequence number>)]

<device specifier> ::= { : <device> [: <loop number>] |
. <volume label> [: <loop number>] }

<device type> ::= % <accessory ID> [(<sequence number>)]

<device word> ::= { MASSMEM | PRINTER | DISPLAY | INTRFCE |
INSTRMT | GRAPHIC | TAPE | TV }
[(<sequence number>)]

<file name> ::= <alpha> [.... [<alpha> | <digit>] ....]
(10 characters maximum)

<file number> ::= <digit> [<digit> [<digit>]]

<file type> ::= SDATA | DATA | TEXT

<file specifier> ::= <file name> [<device specifier>]

<HP-IL address> ::= <primary address > [. <secondary address>]

<loop number> ::= 1 | 2 | 3

<primary address> ::= 0 | 1 | .... | 30

<secondary address> ::= 1 | 2 | .... | 31

<sequence number> :: = 1 | 2 | .... | 16

<volume label> ::= <alpha> [.... [<alpha> | <digit>] ...]
(6 characters maximum)

** The quotes around the assign word are not required, but are recommended to prevent any ambiguities in specifying a device

3.3 ASSIGN

SYNTAX

ASSIGN # <channel number> TO <file specifier>

EXAMPLE STATEMENTS

ASSIGN # 1 TO NOTES:MASSMEM:L

3.4 CAT

SYNTAX

CAT [<file name>] <device specifier>

EXAMPLE STATEMENTS

```
CAT :TAPE(3):L
CAT .VOLUM1:L
```

3.5 CAT$

SYNTAX

CAT$(<file number> , <device specifier>)

EXAMPLE STATEMENTS

```
F$=CAT$(1,":TAPE:3")
CAT$(F,".DATA:1")
```

3.6 CLEAR

SYNTAX

CLEAR <device specifier>
CLEAR [LOOP [: <loop number>]]

EXAMPLE STATEMENTS

```
IF X(2) THEN CLEAR LOOP:L
CLEAR ":DISPLAY:2"
```

3.7 CONTROL OFF

SYNTAX

```
CONTROL OFF [ <loop number> ]
```

EXAMPLE STATEMENTS

```
IF NOT C THEN CONTROL OFF L
```

3.8 CONTROL ON

SYNTAX

```
CONTROL ON [ <loop number> ]
```

EXAMPLE STATEMENTS

```
IF C(I) THEN CONTROL ON L
```

3.9 COPY

SYNTAX

```
COPY [ { <file specifier> | <device specifier> |
         LOOP [ : <loop number> ] } ]
     TO { <file specifier> | <device specifier> |
         LOOP [ : <loop number> ] } ]
```

```
COPY { <file specifier> | <device specifier> |
         LOOP [ : <loop number> ] } ]
     [TO { <file specifier> | <device specifier> |
         LOOP [ : <loop number> ] } ]
```

EXAMPLE STATEMENTS

```
COPY START:TAPE(2):3
COPY TO BACKUPFILE.DATA1:L
COPY OLDFILE:CA:L TO NEWFILE
COPY TO :MASSMEM:L
```

3.10 CREATE

SYNTAX

```
CREATE <file type> <file specifier> , <filesize> [,<record length>]
```

EXAMPLE STATEMENTS

```
CREATE TEXT FILE6:1:L,500
CREATE DATA A$,10,50
```

3.11 DEVADDR

SYNTAX

```
DEVADDR ( <device specifier> )
```

EXAMPLE STATEMENTS

```
A=DEVADDR("PR(2):L") @ PRINTER IS A
```

```
B=DEVADDR("%16:3") @ COPY FILE1 TO :B
C=DEVADDR(D$)
```

3.12 DEVAID

SYNTAX

DEVAID (<device specifier>)

EXAMPLE STATEMENTS

```
T=DEVAID("HP82164A:3")
```

3.13 DISPLAY IS

SYNTAX

DISPLAY IS <device specifier>
DISPLAY IS LOOP [: <loop number>]

EXAMPLE STATEMENTS

```
DISPLAY IS 1.02:L
DISPLAY IS %48(2):L
```

3.14 ENABLE INTR

SYNTAX

ENABLE INTR [<loop number> ;] <interrupt mask byte>

EXAMPLE STATEMENTS

```
ENABLE INTR L ; L+I*2^N
IF E THEN ENABLE INTR 1;8 @ ENABLE INTR 2;8
```

3.15 ENTER

SYNTAX

```
ENTER <device specifier> [ USING {<string exp> | <line numb>} ]
      [ ; <variable> [ , <variable> ...] ]
ENTER LOOP [:<loop numb>] [ USING {<string exp> | <line numb>}]
      [ ; <variable> [ , <variable> ...] ]
```

EXAMPLE STATEMENTS

```
ENTER "HP82:2"; N,A$
ENTER %64:L USING "80A" ; X$,Y$
ENTER 3:L USING " ,B" ; A
ENTER LOOP:L ; B1$
```

3.16 INITIALIZE

SYNTAX

```
INITIALIZE [ <volume label> ] <device spec> [ , <directory size>]
```

EXAMPLE STATEMENTS

```
INITIALIZE A$,35
INITIALIZE DATA:TAPE:L,55
```

3.17 LOCAL

SYNTAX

```
LOCAL <device specifier>
LOCAL [ LOOP [ : <loop number> ] ]
```

EXAMPLE STATEMENTS

```
IF NOT R THEN LOCAL "T2:2"
LOCAL HP71(2):L
```

3.18 LOCAL LOCKOUT

SYNTAX

```
LOCAL LOCKOUT [ <loop number> ]
```

EXAMPLE STATEMENTS

```
IF NOT O THEN LOCAL LOCKOUT L
```

3.19 OUTPUT

SYNTAX

```
OUTPUT { <device specifier> | LOOP [ : <loop number> ] }
       [ USING { <string> | <line number> ]
       [;expression [ { , | ; } <expression> .... ] ] [;]
```

EXAMPLE STATEMENTS

```
OUTPUT DISPLAY:2;A$
```

3.20 PACK

SYNTAX

PACK <device specifier>

EXAMPLE STATEMENTS

```
IF V THEN PACK TAPE(2):3
PACK %16:L
```

3.21 PACKDIR

SYNTAX

PACKDIR <device specifier>

EXAMPLE STATEMENTS

```
PACKDIR :TAPE(3):L
PACKDIR A$
```

3.22 PASS CONTROL

SYNTAX

PASS CONTROL { <device specifier> | LOOP [: <loop number>] }

EXAMPLE STATEMENTS

```
PASS CONTROL %1:L
PASS CONTROL 3:2
```

3.23 PRINTER IS

SYNTAX

```
PRINTER IS <device specifier>
PRINTER IS LOOP [ : <loop number> ]
```

EXAMPLE STATEMENTS

```
PRINTER IS "PR(2):2"
PRINTER IS %32:L
```

3.24 PRIVATE

SYNTAX

```
PRIVATE <file specifier>
```

EXAMPLE STATEMENTS

```
PRIVATE TEST:TAPE:L
PRIVATE "FILE1.TAPE1:3"
```

3.25 PURGE

SYNTAX

PURGE <file specifier>

EXAMPLE STATEMENTS

```
PURGE BACKUP:TAPE(2):L
IF F$=A$ THEN PURGE A$
```

3.26 READDDC

SYNTAX

READDDC [(<loop number>)]

EXAMPLE STATEMENTS

```
X = READDDC(L)
IF BIT(READINTR,0) THEN A=READDDC(L)
```

3.27 READINTR

SYNTAX

READINTR [(<loop number>)]

EXAMPLE STATEMENTS

```
I=READINTR(L)
```

3.28 REMOTE

SYNTAX

```
REMOTE <device specifier>
REMOTE [ LOOP [ : <loop number> ] ]
```

EXAMPLE STATEMENTS

```
IF R THEN REMOTE "%66(2):3"
REMOTE LOOP:L
```

3.29 RENAME

SYNTAX

```
RENAME <old file specifier> TO <new file specifier>
```

EXAMPLE STATEMENTS

```
RENAME "FILE1027:TAPE:L" TO "FILE1028"
RENAME "POINTS" TO "DATA:1.02:3"
```

3.30 REQUEST

SYNTAX

```
REQUEST [ <loop number> ; ]  <numeric expr>
```

EXAMPLE STATEMENTS

```
REQUEST L ; 224
```

3.31 RESET HPIL

SYNTAX

RESET HPIL [<loop number>]

EXAMPLE STATEMENTS

```
IF LEN(A$)>L THEN RESET HPIL L
RESET HPIL 3
```

3.32 RESTORE IO

SYNTAX

RESTORE IO [<loop number>]

EXAMPLE STATEMENTS

```
IF A$=R$ THEN RESTORE IO 2
```

3.33 SECURE

SYNTAX

SECURE <file specifier>

EXAMPLE STATEMENTS

```
SECURE VER1:TAPE(2):L
SECURE "TEST:3:2"
```

3.34 SEND

SYNTAX

```
SEND [ <loop number> ; ] [ [ CMD expression [ , expression ] ...]
                           [ DATA expression [ , expression ] ...]
                           ...
                           ...
                           ...
```

EXAMPLE STATEMENTS

```
SEND 2; CMD A$ LISTEN 4 SAD 14,18 DATA X$
```

3.35 SPOLL

SYNTAX

```
SPOLL ( <device specifier> )
```

EXAMPLE STATEMENTS

```
A= SPOLL("3:1")
IF SPOLL("MASSMEM(1):2") = 220 THEN 100
```

3.36 STANDBY

SYNTAX

```
STANDBY [ <loop number> ; ] OFF
STANDBY [ <loop number> ; ] ON
STANDBY [ <loop number> ; ] <numeric expr> [ , <numeric expr> ]
```

EXAMPLE STATEMENTS

```
STANDBY 2;ON
STANDBY A;OFF
STANDBY L ; F,I
```

3.37 STATUS

SYNTAX

```
STATUS [ (<loop number>) ]
```

EXAMPLE STATEMENTS

```
X= STATUS(2)
IF BIT(STATUS(L),5) THEN GOTO 100
```

3.38 TRIGGER

SYNTAX

```
TRIGGER <device specifier>
TRIGGER [ LOOP [ : <loop number> ] ]
```

EXAMPLE STATEMENTS

```
IF T THEN TRIGGER 1:2
TRIGGER LOOP:2
```

3.39 UNSECURE

SYNTAX

```
UNSECURE <file specifier>
```

EXAMPLE STATEMENTS

```
UNSECURE DATA:%16:L
UNSECURE FILE1:HP82161A:L
```

```
+-----------------------------------------+-----------------+
|                                         |                 |
|   EXAMPLES OF HP-IL OPERATIONS          |   CHAPTER  4    |
|                                         |                 |
+-----------------------------------------+-----------------+
```

The purpose of this chapter is to describe at a frame level the messages the HP-71 sends out. The details of some basic operations are given, such as powering up the loop, addressing the loop and searching for a device.

This chapter also describes how to copy a file from or to another computer through HP-IB or RS232.

4.1 How the HP-71 powers up the loop

```
NOP, NOP, ....... /  IFC, IFC, ........     RFC
AAU, RFC, [AES, AEP,] AADn
[TADn, RFC, SDI ... / TADn, RFC, SAI ]
[TADn, RFC, SAI ]
```

The HP-71 will use either a NOP or IFC command frame to power up the loop. The NOP/IFC will be sent out at a rate of 50 milliseconds per frame until one returns. Up to 50 NOP or IFC frames will be sent out on the loop. If none return, the loop will be considered broken.

The power on sequence is always followed by the auto addressing sequence (unless flag -24 is set).

If there is a display device assigned, the search for the display device will follow auto addressing. The sequence used to search for the display device depends on how the display device was assigned. The display device may be searched for by either device ID or accessory ID. If the display device is the default value, it will be searched for by accessory ID.

After the display device is found, the HP-71 will read its accessory ID again, to determine what type of display it is.

The loop power up will be performed at the following times:

1. Everytime the HP-71 wakes up from deep sleep (turn on), and there is a display device assigned. The NOP Message is used to power up the loop in this case.
2. When CONTROL ON or RESTORE IO is executed. The IFC Message

is used to power up the loop in this case.
3. Everytime the HP-71 needs to use the loop and the loop has been broken or has been powered off. The NOP Message is used to power up the loop in this case.

If there is no display device assigned, the HP-71 will not try to power up the loop when it wakes up from deep sleep. It will try to power up the loop only when it needs to use the loop. The HP-IL module tracks of when the loop has been powered down or the loop has been broken, and will automatically power up the loop before any other frames are sent.

With flag -21 set, the HP-71 will not power down the loop when it is turned off. Therefore, it will not try to power up the loop when it is turned ON again, since the loop has never been powered down.

4.2 How the loop is addressed

AAU, RFC, [AES, AEP sequence,] AADn

The extended addressing sequence (AES, AEP) will only be sent out when flag -22 is set.

The HP-71 will auto address the loop at the following times:

1. After powering up the loop (refer to previous section about the power up conditions).
2. The loop has been unconfigured by with an AAU message (sent out by the SEND command).

If flag -24 is set, the HP-71 will not send out the auto addressing sequence, except when a RESTORE IO, CONTROL ON or ASSIGN IO statement is executed.

4.3 How the HP-71 searchs for a Device ID

TADn, RFC, SDI,.......[NRD]

This sequence is repeated until the Device ID the HP-71 is searching for is found or all of the devices have been polled. The HP-71 will read up to 8 characters of the Device ID. An NRD frame will be sent after 8 characters have been received.

4.4 How the HP-71 searchs for an Accessory ID

TADn, RFC, SAI

This sequence is repeated until the Accessory ID the HP-71 is searching for is found or all of the devices have been polled.

4.5 How the HP-71 reads a device's status (serial poll)

TADn, RFC, SST

This sequence may be preceded by the sequence of searching for a device, either by the device ID or accessory ID.

4.6 How to move files between computers

The COPY statement in the HP-IL ROM can be used to transfer files between:

1. HP-71 <=> Digital Cassette Drive
2. HP-71 <=> HP-71
3. Digital Cassette Drive <=> Digital Cassette Drive
4. HP-71 <=> Other computers

The HP-71 has the capability to transfer files to and from non-mass storage type devices (i.e. accessory ID is not 10 hex). This can be very useful for transferring file between the HP-71 and other computers. The computer may communicate with the HP-71 through a RS232 or HP-IB interface to HP-IL.

When the HP-71 sends a file to the Digital Cassette Drive, it knows how to find an empty space in the tape and position the tape to the right sector, all the details. But when HP-71 sends a file to a device other than a cassette drive, it does not know the commands which need to be sent to control the device. Rather than only allow file transfers to and from the cassette, the HP-71 will send out a file header followed by the contents of the file. The file header is sent first so the receiving device know the file name, file type and file size and can allocate space for the file. When a computer wants to send a file to HP-71, the HP-71 expects to receive the file in this same format.

The HP-71 has chosen to use the directory entry of the HP's Logical Interchange Format (LIF) as the file header format. The same entry is stored in the cassette drive directory.

The 32 bytes of the file header are as following:

Byte #	Meaning
0-9	File name (1-10 ASCII chars, trailing blanks)
10-11	File type
12-15	Starting address
16-19	Length of file
20-25	Time of creation (12 BCD digits)
26-27	Volume number
28-31	Implementation

File name - Characters are limited to digits (0-9) and upper case letters (A-Z). The first character must be a letter.

File type - HP71's file types in hex are:
00 01 - TEXT file
E0 D0 - SDATA file (same as HP-41 data file)
E0 F0 - DATA file
E2 04 - BINARY file
E2 08 - LEX file
E2 0C - KEY file
E2 14 - BASIC file (tokenized BASIC file)

Starting address - Always send 00 00 00 00.

Length of the file - These 4 bytes are a 32 bit unsigned integer. This number shows the file length in number of sectors. A sector is 256 bytes. The sectors usually are not the exact data length of the file. The data length is defined differently by file type (see Implementation below). But the file will be sent in blocks of 256 bytes.

Time of creation - 12 BCD digits of the form YYMMDDHHMMSS.

Volume number - Always send 80 01.

Implementation -

File type	Meaning

00 01	Always send 00 00 00 00.
E0 D0	Byte 28-29 - 16 bit unsigned integer shows the data length in # of registers. (byte 28 is the lower 8 bits) Byte 30 - If non-zero, the file is secured. Byte 31 - Unused.
E0 F0	Byte 28-29 - 16 bit unsigned integer shows the data length in # of logical records. (byte 28 is the lower 8 bits) Byte 30-31 - 16 bit unsigned integer shows the logical record length in bytes. (byte 30 is the lower 8 bits)
E2 04, E2 08, E2 0C, E2 14	Byte 28-30 - 20 bit unsigned integer shows the data length in # of nibbles. (byte 28 is lower 8 bits, 29 next, byte 30 is the high 4 bits) Byte 31 - Unused.

```
+------------------------------------------------+------------------+
|                                                |                  |
|   I/O PROCESSOR FIRMWARE SPECIFICATION         |   CHAPTER  5     |
|                                                |                  |
+------------------------------------------------+------------------+
```

This chapter contains the firmware specification for the I/O processor.

5.1 Basic Description

The I/O processor is a CMOS chip designed to be an interface between the HP-71 CPU and the HP-IL loop. Packaged with a 16K byte HP-71 ROM, it provides the interface to HP-IL for the HP-71 computer.

The I/O processor provides the low level interface to the loop. It takes care of the "simpler" tasks of sending and receiving frames, maintaining Talker, Listener and Controller status and error checking frames.

5.2 I/O Processor Configuration

The I/O processor is configured as follows:

CPU with cycle time of 1u sec
RAM - 272 bytes
ROM - 4096 bytes
HP-IL interface - highest priority interrupt
HP-71 BUS interface
HP-71 BUS Mailbox - low priority interrupt
TIMER - middle priority interrupt

5.2.1 HP-IL Capabilities

The I/O processor is a slave to the HP-71 CPU. Communication between the two CPUs is through a mailbox of 8 bytes. The mailbox is soft configured in the HP-71 address space. See the HP-71 IDS for more information on the configuration address of the mailbox. Following is a summary of all the functions the I/O processor implements:

As a CONTROLLER:

- Send out a frame
- Address devices to be talkers and listeners (including the I/O processor)
- Auto Address the loop (w/wo extended addressing)
- Poll a device for:
  - Status
  - Accessory ID
  - Device ID
- Pass control to another loop device
- Find the Nth device of accessory ID (or class) M
- Setup frame timeouts and IDY timeouts
- Start data transfers
- Setup terminating conditions for ending a data transfer (Transfers always terminate on an EOT):
  - Terminate after a certain number of frames
  - Terminate on an END frame
  - Terminate on a 1 character match
  - Terminate on loop service request
- Enable an IDY poll to monitor service request

As a DEVICE (Noncontroller):

- Send data to and receive data from the loop
- Setup Accessory ID response
- Setup Status Poll response
- Setup Device ID response
- Request Service on the loop
- Receive control from active controller

Additional Commands:

- Request service on certain interrupt conditions
- Read Status
- Read Error Message
- Perform diagnostics tests on itself
- Set Manual Mode for low level loop control
- Set the I/O processor into Scope Mode
- Set and clear system controller status

5.2.2 Mailbox Description

The mailbox between the HP-71 CPU and the I/O processor consists of 8 bytes of I/O area. Some of the nibbles may be written to by the I/O processor, some of them may be written to by the HP-71. All of the nibbles are readable by both processors. The mailbox is assigned an address in the HP-71 address space by the configuration routines. Routines exist in the HP-IL ROM to find the address of a particular mailbox. The mailbox configuration is shown below:

MAILBOX CONFIGURATION (HP-71 SIDE)

BASE ADDRESS OFFSET	CONTROLLED BY	NIBBLE DESCRIPTION
0	HP-71	Low nibble of msg from HP-71
1	HP-71	Low-mid nibble of msg from HP-71
2	HP-71	Mid-low nibble of msg from HP-71
3	HP-71	Mid-high nibble of msg from HP-71
4	HP-71	High-mid nibble of msg from HP-71
5	HP-71	High nibble of msg from HP-71
6	HP-71	Low handshake nibble from HP-71
7	HP-71 & I/O CPU	High handshake nibble for message from HP-71 to I/O processor
8	I/O CPU & HP-71	Low handshake nibble for message from HP-71 to I/O processor
9	I/O processor	High handshake nibble from I/O CPU
A	I/O processor	Low nibble of msg from I/O CPU
B	I/O processor	Low-mid nibble of msg from I/O CPU
C	I/O processor	Mid-low nibble of msg from I/O CPU
D	I/O processor	Mid-high nibble of msg from I/O CPU
E	I/O processor	High-mid nibble of msg from I/O CPU
F	I/O processor	High nibble of message from I/O CPU

Messages are passed through the mailbox in the following way: After the three message bytes are placed in the mailbox, the sender sets his message available bit. When the receiver reads a specific byte of the message, the sender's message available bit is zeroed automatically by hardware. Before modifying any of the mailbox bytes the sender must simply check his message available bit. If it is clear, then the previous message has been accepted and it is clear to write out the next message.

Two NRD (Not Ready for Data) bits are provided in the mailbox. One is maintained by the HP-71, the other by the I/O processor. This bit indicates to the sender that the receiver's buffer is full and no data messages should be sent. NRD only halts data messages and has no effect on other messages. This is the only bit in the HP-71 handshake nibble to which the I/O CPU can write. Also, this is the only bit in the handshake nibble from the I/O processor to which the HP-71 can write.

5.2.2.1 HP-71 Low Handshake Nibble

BIT NO.	DESCRIPTION	
3	Three Data Bytes	This bit is set by the HP-71 when 3 data bytes are in the mailbox. This bit is valid only when the HP-71 Message Available bit is set.
2	Single Data Byte	This bit is set when there is one data byte message in the mailbox. This bit is valid only when the HP-71 Message Available bit is set. The single data byte is in the low byte of the mailbox.
1	(Not used)	
0	(Not used)	

5.2.2.2 HP-71 High Handshake Nibble

BIT NO.	DESCRIPTION	
3	I/O CPU Reset Bit	HP-71 may reset the I/O processor by writing a '1' to this bit. After the I/O CPU is reset (whether by the HP-71 or power on), this bit remains set until the HP-71 clears it.
2	Mailbox Configured	This bit is controlled totally by hardware. It is set when the HP-71 mailbox is configured, and cleared when it is unconfigured.
1	I/O CPU NRD (Not Ready for Data)	This bit indicates the I/O CPU is not ready to receive data. There is not enough room in buffer to accept more data. It is cleared when the I/O CPU is ready to receive more data. This bit is controlled by the I/O CPU.

0	HP-71 Message Available	This bit is set by the HP-71 when it sends a message. It is cleared when the I/O CPU reads the low byte of the message. When this bit is set, the bits in the low handshake nibble indicate whether or not this message is data. The HP-71 should verify this bit is clear before writing the next message to the mailbox.

5.2.2.3 I/O CPU Low Handshake Nibble

BIT NO.	DESCRIPTION	
3 **	I/O CPU SRQ on HP-71 Bus	This bit is set by the I/O CPU to indicate it requires service. It will be set: (1) if a SRQ is present when the IDY poll is enabled or when the loop is in EAR mode, (2) when an enabled interrupt condition was met, (3) when there is data available in device mode (repeatedly set until data is read), (4) after a power on reset. This bit is cleared when the HP-71 reads status with the clear SRQ option.
2	Sleep Flag (I/O CPU or HP-71 CPU)	Controlled totally by hardware, its meaning is different on each side of the mailbox. Looking from the I/O CPU side, this bit is clear when the HP-71 is awake and set otherwise. From the HP-71 side, this bit is clear when the I/O CPU is awake and set otherwise. This bit provides information only, and has no effect on the execution of any commands.
1	HP-71 NRD	This bit is set by the HP-71 when it is not ready to receive data. It is also used to exit Scope Mode.
0 ***	I/O CPU Message Available	I/O CPU sets this bit to send a message to the HP-71. It is cleared when the HP-71 reads the high nibble of the message. If the Three Data Bytes bit in the high handshake

nibble is set, the message is 3 data bytes. Otherwise the message can be decoded by its opcode.

** I/O CPU may request service on the HP-71 bus by setting this bit.

*** Setting this bit generates a service request on the HP-71 bus.

5.2.2.4 I/O CPU High Handshake Nibble

BIT NO.	DESCRIPTION	
3	Three Data Bytes	Set whenever there are three data bytes in the mailbox from the I/O CPU. This bit is valid only when the I/O CPU Message Available bit is set.
2	Manual Mode	This bit is set to indicate the I/O CPU in Manual Mode or Scope Mode. It is clear otherwise.
1	SRQ received from loop.	This bit is valid only when the I/O CPU is an active controller. It is set when a service request is detected on the loop and cleared when no SRQ is pending on the loop. As a device, this bit is always clear.
0	Error Occurred	When set, this bit indicates an error has occurred. The bit is updated on every message to the HP-71, but is set immediately if a fatal error occurs. It is cleared when the HP-71 reads the error code.

5.3 Power On Sequence

At initial power on or whenever the I/O CPU is reset a self test is executed which includes a RAM test and a ROM test. If either of these tests fail, the error bit is set in the mailbox and the error code is set to the self test failed error code. (The mailbox test is only performed when the self test command is executed.)

The I/O CPU does not try to power up the loop at initial power on. The loop is not powered up until the HP-71 sends a command which uses the loop.

Defaults set at power on are:

HP-IL Status is active controller, not talker or listener
Loop Address is 31 at power on
Loop Address is 21 after an AAU
Status Response is 1 byte of value 0
Device ID Response has a length of 0
Accessory ID Response has a length of 0
Frame Timeout value is 2 seconds
IDY timeout is 255 milliseconds
Number of IDY timeouts is 30
IDY Poll timeout is 255 milliseconds
All special polls and modes are disabled

5.3.1 Powering Up the Loop

The I/O CPU automatically keeps track of the state of the loop (whether it is powered up or not). There is not a status bit to indicate to the master processor whether or not the loop is powered up. The intent is to let the I/O CPU keep track of the state of the loop.

Anytime a command is received which requires loop action, the I/O CPU first verifies the loop has been powered up. If not, it powers up the loop with a NOP frame. It can be verified the loop is powered up by sending the POWER UP THE LOOP command. If the I/O processor's internal status says the loop is already powered up, no loop action is taken and a status message is sent to the HP-71. Otherwise the I/O CPU powers up the loop with a NOP frame and then sends status to the HP-71. If the loop can not be powered up, the current command is aborted (no status message is sent to the HP-71) and the error bit will be set in the mailbox.

If it is desirable to power up the loop with another command frame (such as an IFC) the TAKE CONTROL command can be used. It allows the master processor to specify the data bits of the command frame used to power up the loop.

The routine used to power up the loop will send out up to 50 command frames. The time between sourcing the command frames is set by the IDY timeout value. If none of the command frames are received on the loop, then the loop is considered broken, and the error bit will be set in the mailbox. If a command frame is received, then the RFC frame is sent out and the power up sequence is completed.

5.4 Service Request on the HP-71 Bus

The I/O CPU has the capability to request service on the HP-71 bus. A bit in the mailbox is used exclusively for this purpose. The I/O CPU will request service on the HP-71 bus for various reasons and they are described in the following sections. Once the service request bit has been set in the mailbox, it will not be cleared by the I/O CPU until the HP-71 acknowledges it has seen the service request. This is done by reading the I/O CPU status with the clear service request option selected. For most cases the reason for the service request can be determined by reading I/O CPU status or reading the handshake nibbles from the I/O processor.

5.4.1 Power On Service Request

Whenever the I/O CPU executes a power on reset sequence, the SRQ bit will be set in the mailbox. This is to let the HP-71 know it has been reset and accessory ID and device ID values need to be setup.

5.4.2 Data Available Service Request

When the I/O CPU is in device mode and has data available in the input buffer, it will request service on the HP-71 bus. The service request bit will be set everytime through the main idle loop, so it will appear to the HP-71 to be set continuously, until the data in the input buffer has been read.

To determine if the service request is due to data available, read the I/O CPU status and check to see if the Data Available status bit set.

5.4.3 Interrupt Service Request

When an enabled interrupt condition has been met, the I/O CPU will request service on the HP-71 bus. However service will only be requested once due to an interrupt. Thus if an interrupt condition is met, but the Interrupt Occurred status bit is already set, the I/O CPU will not request service on the HP-71 bus.

To determine if the service request is due to an interrupt occurring, read the I/O CPU's status and check to see if the

Interrupt Occurred status bit is set.

5.4.4 Loop Service Request

When the I/O CPU is controller and a loop service request is detected, the I/O CPU may request service on the HP-71 bus for two specific cases. The first case is if the loop was in EAR mode when the service request was received. The second case is if the IDY Service Request poll was enabled, and when sending out an IDY, a service request was detected. For all other cases, when a loop service request is received, the I/O CPU will not request service from the HP-71.

A service request due to the loop service request, can be determined by looking at the handshake nibble from the I/O CPU to see if the Loop SRQ bit is set.

5.5 Terminating Data Transfers

When data transfers are terminated various messages may be sent to the HP-71 to indicate the transfer has completed. The message is dependent upon current HP-IL status and why the transfer was halted. The following table lists the message sent for the various cases:

Cause of Termination	Required HP-IL Status	Message sent by I/O CPU to HP-71
EOT frame received	Listener OR Controller Standby	EOT Received Message
Frame Count Exceeded	Active Controller AND Listener	None
	Active Controller AND Not Listener	Conversation Halted Message
Terminating Character was Matched	Listener	Terminator Character Matched
Terminate on END frame condition met	Listener	Terminator Character Matched
Terminate on SRQ frame	Active Controller	Conversation Halted Message
Send NRD Frame	Listener or Controller Standby	Conversation Halted Message

5.6 Frame Timeouts

When it is a controller, the I/O CPU will keep track of how long it takes a frame which is sent out on the loop to return and will generate an error if it takes too long. The I/O CPU can be setup to send out IDY frames to verify the loop is complete while waiting for a frame to return.

There are 3 parameters which affect the amount of time the I/O CPU waits for a frame to return and the number of IDY frames which will be sent out. These parameters are described below:

FRAME TIMEOUT VALUE - is the amount of time to wait for a frame to return before sending out an IDY frame and is also the time to wait between IDY frames.

NUMBER OF IDY TIMEOUTS - is the maximum number of IDYs plus one, to be sent out to verify the loop is complete when a frame takes longer than the frame timeout value to return.

IDY TIMEOUT VALUE - is the amount of time to wait for the IDY to return when it is sent out to verify the loop is still complete.

When a frame is sent out on the loop, the I/O CPU starts a timer loaded with the FRAME TIMEOUT VALUE. If the timer expires and the frame has not been received then an IDY frame is sent out to quickly check if the loop is complete. The length of time to wait for the IDY to return is the IDY TIMEOUT value. If the IDY does not return within this time period, the loop broken error is set and the command is aborted.

If the IDY is received, the I/O CPU again waits the frame timeout value for the frame to come in. The I/O CPU will repeat the timeout, send IDY sequence until the NUMBER OF IDY TIMEOUTS has been met. (Note: There will be NUMBER OF IDY TIMEOUTS frame timeout periods, but the number of IDY frames sent out will be one less than the value in the NUMBER OF IDY TIMEOUTS byte.) After the final FRAME TIMEOUT period has expired, the error frame timed out will be set and the current operation will be aborted.

When the I/O CPU is active listener or in controller standby mode, a frame timeout is not monitored. It is assumed the talker on the loop, will terminate the data transfer properly.

5.7 Error Handling

There are basically three types of errors that the I/O processor may detect:

- Data transfer errors, when sourcing data
- Fatal errors, eg. CMD frame not received as sent
- Nonfatal errors, eg. Device didn't respond to status poll

Each of these errors are handled in a slightly different way. However they all result in the error code being set to the appropriate number as soon as the error is detected.

If a data transfer error is detected when the I/O CPU is talker, an ETE will be sent out as soon as possible. The error bit and the NRD bit will be set in the mailbox, to let the HP-71 know the transfer was halted. The NRD bit will remain set until status or error message is read.

If a fatal error occurs, the current processing on the command is aborted, the error flag is set in the mailbox and the HP-71 will return to the main idle loop.

On a nonfatal error, the error code will be setup immediately. The error bit in the mailbox will be set on the next message to the HP-71.

5.8 Manual and Scope Modes

Beside "auto" mode, the I/O CPU may be set into a MANUAL Mode. In MANUAL Mode the HP-71 has complete control of the loop. All frames received are sent directly to HP-71, and only frames from the HP-71 are sourced on the loop. The I/O CPU does not maintain any loop status. The I/O CPU will execute all commands from the HP-71 which do not involve knowing loop status. (All commands with first nibble opcode of 2 through opcode of C are not executable in Manual Mode.) Manual Mode is terminated when the Go Into Auto Mode command is received.

Manual mode has a retransmission option which puts the I/O CPU into a tight Scope Loop. In this mode the I/O CPU will send all frames received to the HP-71 and also retransmit frames on the loop. No other commands from the HP-71 are processed, no loop status is maintained. The I/O CPU is an "invisible" device on the loop. The auto retransmit feature of HP-IL section will be used as long as possible. If the HP-71 can read the messages quickly enough, then frames will be automatically retransmitted. However if a message blocks the mailbox, no frames will be automatically retransmitted, to avoid losing frames. To exit Scope mode, set the HP-71 NRD bit in the mailbox.

The manual mode bit is set in the mailbox whenever the I/O CPU is in Manual or Scope mode. Scope mode may be entered when in Manual mode. However, exiting Scope Mode will exit Manual Mode also.

5.9 Mailbox Messages From HP-71

The mailbox commands from the HP-71 are divided into classes and are described in this section. The opcodes are listed with the low nibble being the leftmost nibble and the high nibble the rightmost nibble.

5.9.1 No Parameter Class

5.9.1.1 Nop

OPCODE: XXXX XXXX XXXX XXXX 0000 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command is merely a handshake message, it does not modify status or send any frames.

5.9.1.2 Read Address Table

OPCODE: XXXX XXXX XXXX XXXX 0001 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Three data bytes
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: The table read by this command is an address table which contains the range of addresses of devices on the loop. The table is created after the auto address command is executed. The I/O CPU sends back 3 data bytes. The low byte is the ending AAD address, the middle byte is the ending AEP address and the high byte is the ending AES address. If there are no devices with a particular type of address then the ending address returned will be zero. All addresses in the address table are zeroed at power on.

5.9.1.3 Request I/O Processor Status

OPCODE: XXXX XXXX XXXC XXXX 0010 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Status message, I/O CPU SRQ bit in the mailbox is cleared if C bit is set
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command allows the HP-71 to read twelve bits of the I/O CPU status as well as the current error number. (The bits are described under mailbox messages from the I/O CPU in the status class.) If the C bit is set in the message from the HP-71, the mailbox status bit which requests service from the HP-71 is cleared. This command returns exactly the same information as SEND ERROR message command.

5.9.1.4 End Of Message

OPCODE: XXXX XXXX XXXX XXXX 0011 0000

HP-IL FRAMES SENT: ETO
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: Active Talker

DESCRIPTION: If the I/O CPU is currently active talker an ETO frame is sent out on the loop.

5.9.1.5 Clear SRQ

OPCODE: XXXX XXXX XXXX XXXX 0100 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: Device mode

DESCRIPTION: This message will cause the I/O CPU to halt requesting service on DOE and IDY frames as a device.

5.9.1.6 Set SRQ

OPCODE: XXXX XXXX XXXX XXXX 0101 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: Device mode

DESCRIPTION: As a device, the I/O CPU will now request service on DOE frames and IDY frames. (This command is ignored as a controller.)

5.9.1.7 Send Error Message

OPCODE: XXXX XXXX XXXX XXXX 0110 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Status message sent, error bit cleared
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Returns a status message with the current error number. (See I/O CPU messages, status class.) The error number is cleared after the message is sent and the error bit in the mailbox is cleared before the message is sent.

5.9.1.8 Enter Auto End Mode

OPCODE: XXXX XXXX XXXX XXXX 0111 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets auto end mode, so that on the next output data transfer the last data byte sent is an END frame. To source an END frame, the HP-71 may set this mode, send the data to go out, and then send a non-data command, such as NOP. The buffer will be emptied before the NOP command is executed, with the last data byte sent as an END frame.

5.9.1.9 Go Into Manual Mode

OPCODE: XXXX XXXX XXXR XXXX 1000 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets the I/O CPU into Manual Mode. The only frames sourced on the loop are those sent directly from the HP-71 and any frame received on the loop is sent directly to the HP-71 for processing. If the R bit is set in this command the retransmission option is selected and the I/O CPU will enter a tight Scope loop. Entering Manual Mode or Scope mode will cause all talker and listener status to be cleared. For more information about Manual and Scope modes, please refer to the section "Manual and Scope Modes".

5.9.1.10 Go Into Auto Mode

OPCODE: XXXX XXXX XXXX XXXX 1001 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Exit manual mode, restores controller or device status.

5.9.1.11 Update System Controller Bit

OPCODE: XXXX XXXX SXXX XXXX 1010 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets System Controller bit equal to the S bit in the command.

5.9.1.12 Reset CURRENT Address

OPCODE: XXXX XXXX XXXX XXXX 1011 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Resets CURRENT Address to start of address table. CURRENT Address is zeroed at power on. It is set to the address of the first loop device when the command "Auto Address the Loop" is executed. It is modified in the command "Find Nth Device of Type M" and "Increment CURRENT Address". It may be used in the "Address P,S as Listener" and "Address P,S as Talker" commands.

5.9.1.13 Read CURRENT Address

OPCODE: XXXX XXXX XXXX XXXX 1100 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Address message sent.
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sends the CURRENT Address to the HP-71.

5.9.1.14 Increment CURRENT Address

OPCODE: XXXX XXXX XXXX XXXX 1101 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Address message sent (or an error)
POSSIBLE ERRORS: Illegal CURRENT Device Address
REQUIRED STATUS: none

DESCRIPTION: Increment CURRENT Address to the address of the next device on the loop. If the end of the address table has been reached, then an error is sent to the HP-71 and the CURRENT Address is reset to the address of the first device on the loop. If the end of table was not reached, then the CURRENT Address is incremented and sent to the HP-71. If the address table is not valid, then an Illegal CURRENT Address Error is sent to the HP-71.

5.9.1.15 Read My HP-IL Loop Address

OPCODE: XXXX XXXX XXXX XXXX 1110 0000

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Address message
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: I/O CPU's current HP-IL loop address is sent to the HP-71.

5.9.1.16 Take/Give Loop Control

OPCODE: DDDD DDDD XXLC XXXX 1111 0000

HP-IL FRAMES SENT: CMD (D),RFC (if L option selected)
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command allows the HP-71 to set or clear controller status. If the C bit is clear the all controller status is cleared, terminate on END frame and terminate on character match modes are enabled and the command is completed. If the C bit is set then the I/O CPU will set active controller status and then check the L bit. If the L bit is set, the I/O CPU will try to power up the loop with a command frame. The data bits of the command frame sent out on the loop are specified in the lower byte (D bits) of the command from the HP-71. The command frame will be sent up to 50 times before declaring the loop dead. The time between sourcing the command frames is the IDY timeout value. The loop will not be auto addressed. Setting controller status clears all terminator modes (terminate on SRQ, terminate on character match and terminate on END frame).

5.9.2 Frame Class

5.9.2.1 Send Frame

OPCODE: XXXX XXXX DDDD DDDD RCCC 0001

HP-IL FRAMES SENT: Frame sent from HP-71
MAILBOX RESPONSE: Frame received if R bit is set
POSSIBLE ERRORS: Illegal State
REQUIRED STATUS: Dependent upon frame

DESCRIPTION: Using this command the HP-71 may source frames on the loop. The D bits are the data bits, C bits are the control bits. The R bit indicates that the HP-71 wants to see the frame received in response to sourcing this frame. The R bit is valid only when the I/O CPU is controller.

If the I/O CPU is currently in Manual Mode, then any frame is legal. The R bit is ignored, since manual mode mode implies that all frames received go to the HP-71.

If the I/O CPU is not controller, then only a limited number of frames may be sent. They are:

DOEs if Active Talker.
EOTs if Active Talker.
NRDs if Listener.
IDYs if Asynchronous Requests are enabled.

If the I/O CPU is controller then almost any frame is valid. The following frames require a certain status:

DOEs require Active Talker status.
EOTs require Active Talker status.
NRDs require Listener or Controller Standby Status.

As controller, if the R bit is set, then the frame is sent out and the first frame received is sent back to the HP-71. In this case, the current timeout setting will be used. If a timeout is detected, the error bit in the mailbox is set. If the R bit is clear and the frame is a command frame then an RFC is automatically sent after the command.

5.9.3 Single Nibble Parameter Class

5.9.3.1 Address/Unaddress me as TL

OPCODE: XXXX XXXX XTLX XXXN 0000 0010

HP-IL FRAMES SENT: UNT,RFC (if address me as talker)

MAILBOX RESPONSE: none
POSSIBLE ERRORS: Illegal State
REQUIRED STATUS: Controller

DESCRIPTION: This command allows the HP-71 to set or clear T (talker) or L (listener) status of the I/O CPU. If the N bit is set then it is an unaddress command, otherwise it is an address me as T or L command. If the command is an address me as talker, then an UNT and RFC frame are sent out on the loop.

5.9.3.2 Power Down Loop

OPCODE: XXXX XXXX XXXX XXXX 0000 0011

HP-IL FRAMES SENT: [NOP,RFC (power up loop)] LPD,RFC
MAILBOX RESPONSE: clears Loop Powered Up bit
POSSIBLE ERRORS: Illegal State
REQUIRED STATUS: Controller

DESCRIPTION: If the loop is already powered down, this command is ignored. If the loop is in EAR mode the LPD (Loop Power Down) frame and RFC are sent. Otherwise this command first powers up the loop by sending out NOP command frame, followed by an RFC. Then a LPD and RFC are sent.

5.9.4 Address Class

5.9.4.1 Address P,S as Talker

OPCODE: XXXX XXXX PPPP SSSP XXSS 0100

HP-IL FRAMES SENT: TAD P,RFC [SAD S-1, RFC]
MAILBOX RESPONSE: none
POSSIBLE ERRORS: Illegal CURRENT Device Address or Status
REQUIRED STATUS: Controller

DESCRIPTION: Addresses a device on the loop as talker. The P bits specify the primary address, the S bits specify the secondary address+1. If the address passed is not zero in the primary or secondary parts then TAD P and RFC are sent (and if secondary address is not zero, then a SAD S-1 and RFC are sent out.) If the address passed is primary address zero and secondary address zero, then the CURRENT Address device is addressed as a talker. If the address table is not setup then an Illegal CURRENT Address Error will result. This command does not modify the CURRENT address.

5.9.4.2 Address P,S as Listener

OPCODE: XXXX XXXX PPPP SSSP XXSS 0101

HP-IL FRAMES SENT: LAD P,RFC [SAD S-1, RFC]
MAILBOX RESPONSE: none
POSSIBLE ERRORS: Illegal CURRENT Device Address or State
REQUIRED STATUS: Controller

DESCRIPTION: Addresses a device on the loop as listener. The P bits specify the primary address, S bits specify the secondary address+1. If the address sent is not zero in the primary and secondary parts then LAD P, RFC are sent (and if secondary address is not zero, then a SAD S-1 and RFC are sent out.) If the address passed is primary address zero and secondary address zero, then the CURRENT Address device is addressed as a listener. If the address table is not setup then an Illegal CURRENT address error will result. This command does not modify the CURRENT address.

5.9.4.3 Find Nth Device of Type M

OPCODE: XXXX XXXX MMMM MMMM NNNN 0110

HP-IL FRAMES SENT: UNL,RFC, {TAD,RFC, [SAD,RFC,] SAI} [UNT,RFC]
MAILBOX RESPONSE: Device Address or Error
POSSIBLE ERRORS: No Such Device, Illegal Status, Illegal CURRENT Address
REQUIRED STATUS: Controller

DESCRIPTION: This command finds the Nth device of a specific accessory ID on the loop. M specifies a class (top nibble) and a particular device within a class (bottom nibble). If the bottom nibble is F (hex) then the search is for matching class only.

All devices on loop are polled until a device of given class (or class and device) is found. If this is the Nth device of this type then the device address is sent to the HP-71 and the device is left addressed as a talker. If the device type or number is not found then a No Such Device Error message is returned to the HP-71 and an UNT, RFC sequence is sent out.

This command uses the CURRENT Address to keep track of which device is currently talker. If the device is found then the CURRENT Address will contain the address of that device, otherwise CURRENT Address will be reset to the address of the first device on the loop.

5.9.4.4 Auto Address the Loop

OPCODE: XXXX XXXX XXXX XXXX XXXS 0111

HP-IL FRAMES SENT: AAU,RFC, [AES,AEP sequence,] AAD
MAILBOX RESPONSE: Address of last device on the loop
POSSIBLE ERRORS: Invalid status
REQUIRED STATUS: Controller

DESCRIPTION: This command auto addresses the loop. If the S bit is clear then extended addressing and simple addressing are used. If S bit is set then only simple addressing is used. Addressing always begins with secondary address of 0, primary address of 1. The I/O CPU's loop address is set to primary address of 0 with no secondary address. The first frames sent out are an AAU, RFC to unaddress all devices.

For an auto extended addressing sequence, an AES 0 is first sent. If the frame returns unchanged then there are not extended addressed type devices and simple addressing sequence is sent out. Otherwise it is followed by an AEP 1. If the last AES frame received had an address of 31 then the sequence is repeated starting with AES 0, followed by an AEP (next primary address). This is repeated until an AES is received that has an address less than 31.

For an automatic addressing sequence an AAD (next primary address) is sent out.

If at any time during the addressing sequence a primary address of 31 is received, addressing is halted at that point and the last address is sent to the HP-71.

The address table is set up during execution of this command. The ending AES address, AEP address and AAD address are saved in the table. If there were no devices of a particular type, then the ending address is zero. After the loop has been addressed, the CURRENT Address is set to the address of the first device on the loop. The address of the last device on the loop is sent to the HP-71.

5.9.5 Conversation Class

In this class of commands, the HP-71 may start a data transfer with one of the five SOT (start of transmission) RDY frames, set the frame timeout value or set the frame count as a device. The first 4 commands which all start data transfers have a 20 bit field in which a frame count may be specified. This allows the HP-71 to set up a conversation of X number of frames. After X frames go by the I/O CPU will stop the transfer with a NRD sequence. If the count sent is FFFFF (hex), this is termed infinity and means don't count. This is useful if the transfer should be terminated by some other terminating conditions such as character match or EOT. If the frame count is set to 00000, then the transfer is halted after 1 data byte.

If the SOT frame returns to the I/O CPU unchanged, then a Device Not Ready Error message is sent to the HP-71. If an EOT is received, then an EOT received message is sent to the HP-71.

5.9.5.1 Start Data Transfer

OPCODE: CCCC CCCC CCCC CCCC CCCC 1000

HP-IL FRAMES SENT: SDA
MAILBOX RESPONSE: none
POSSIBLE ERRORS: Device Not Ready, Illegal Status
REQUIRED STATUS: Controller

DESCRIPTION: Sends out an SDA with frame count of C.

5.9.5.2 Start Status Poll

OPCODE: CCCC CCCC CCCC CCCC CCCC 1001

HP-IL FRAMES SENT: SST
MAILBOX RESPONSE: none
POSSIBLE ERRORS: Device Not Ready, Illegal Status
REQUIRED STATUS: Controller

DESCRIPTION: Sends out an SST with frame count of C.

5.9.5.3 Start Device ID

OPCODE: CCCC CCCC CCCC CCCC CCCC 1010

HP-IL FRAMES SENT: SDI
MAILBOX RESPONSE: none
POSSIBLE ERRORS: Device Not Ready, Illegal Status
REQUIRED STATUS: Controller

DESCRIPTION: Sends out an SDI with count of C.

5.9.5.4 Start Accessory ID

OPCODE: CCCC CCCC CCCC CCCC CCCC 1011

HP-IL FRAMES SENT: SAI
MAILBOX RESPONSE: none
POSSIBLE ERRORS: Device Not Ready, Illegal Status
REQUIRED STATUS: Controller

DESCRIPTION: Sends out an SAI with count of C.

5.9.5.5 Pass Control

OPCODE: XXXX XXXX XXXX XXXX XXXX 1100

HP-IL FRAMES SENT: TCT
MAILBOX RESPONSE: NOP or Device Not Ready Error
POSSIBLE ERRORS: Device Not Ready, Illegal Status
REQUIRED STATUS: Controller and not Talker

DESCRIPTION: Sends out a TCT frame to the active talker on the loop. If control is accepted by the device, then a NOP message is sent to the HP-71 to signal control was successfully passed. If the TCT frame was returned then a Device Not Ready Error message is sent to the HP-71. If control was successfully passed, then terminate on character match mode and terminate on END frame mode are automatically set.

5.9.5.6 Set Frame Timeout

OPCODE: TTTT TTTT TTTT TTTT TTTT 1101

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets up the frame timeout to T milliseconds. This is the amount of time to wait for a frame to return before sending out an IDY. The power on default value is 2 seconds. If the frame timeout value is set to all zeros, the timeout is infinite, the I/O CPU will wait forever for a frame to return and no IDYs will be sent out.

When controller, the I/O CPU will automatically verify the loop is complete if a frame takes a "long time" to return. For more information on this refer to the section on frame timeouts.

5.9.5.7 Set Frame Count

OPCODE: CCCC CCCC CCCC CCCC CCCC 1110

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command sets up the frame count. When in device mode the frame count is the number of bytes to send to the HP-71 from the input buffer. Data received as a listener in device mode, stays in the input buffer until the frame count is set to a non zero value. A frame count of all zeros means send none of the bytes from the input buffer. A frame count of all F's means send all the data from the buffer to the HP-71.

As a controller this frame count is used to specify the number of bytes which should go by in controller standby mode before the data transfer is halted. For example the frame count may be set to 5, then if a SDA frame is sent, the data transfer will be halted after 5 bytes. If the frame count is set to all F's then no frame count will be maintained. If the frame count is set to all 0's, the data transfer will be halted after 1 byte.

5.9.6 Multibyte Parameter Class

5.9.6.1 Set SOT Response

OPCODE: NNNN XSAI RRRR RRRR 0011 1111

HP-IL FRAMES SENT: none

MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets up the response to a SAI, SST and SDI poll as a device. The value to set is in R bits. The type of poll response being setup is specified in the SAI bits:

SAI	Set Response Byte of:
100	Status
010	Accessory ID
001	Device ID

N bits specify which byte of the response to set (0-15). Byte 0 is the length of each response. Byte 1 is the 1st byte sent out, byte 2 is second byte, etc. RAM has been set aside in the I/O processor for 1 byte of accessory ID, 2 bytes of status and 8 bytes of device ID.

If the first byte of the Status response is being set, then the I/O CPU's loop SRQ bit is updated. If bit 6 of this byte is set, then the I/O CPU will start requesting service on the loop. If bit 6 of this byte is clear, then the I/O CPU will stop requesting service on the loop.

At power on all lengths and values of the responses are zeroed. The only exception to this is the status length which is set to 1.

5.9.6.2 Set Terminator Mode

OPCODE: XXXX XXXX SEOT 0000 0100 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: I/O CPU may be setup to terminate input on an END frame and/or a character match. Using this command these modes may be set or cleared. (An END frame is a DATA frame with an extra bit set to indicate this is the last byte of a data block.)
The bits which set or clear the modes are:

Bit S: If set then this command is updating the terminate on END frame mode. When clear this command is updating terminate on character mode. This means that bit S is used to determine whether bit E or bit T is meaningful.

Bit E: Set if terminate on END frame mode is to be set. clear if terminate on END frame mode is to be cleared. Valid only when bit S is set.

Bit T: Set if terminate on character mode is to be set, clear if terminate on character mode is to be cleared. Valid only when S bit is clear.

Terminate on END frame and terminate on character match can be enabled simultaneously during a data transfer.

5.9.6.3 Set Terminator Character

OPCODE: XXXX XXXX CCCC CCCC 0101 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets character, C, which is the character to match when in terminate on a character mode. This character is not used unless terminate on character match mode is enabled. At power on this character is set to a line feed (0A hex).

5.9.6.4 Set Number of IDY Timeouts

OPCODE: XXXX XXXX NNNN NNNN 0110 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command sets the number of IDY frames-1 sent out to verify the loop is complete. When a frame times out on the loop an IDY is sent out to verify the loop is complete. If the IDY returns, the I/O CPU again waits for the frame timeout period. When it expires, another IDY is sent out. This command allows the HP-71 to set the number of timeout cycles. Setting this value to 2 means there will be two frame timeout periods and 1 IDY will be sent out on the loop. The power on default value is 29.

5.9.6.5 Set IDY Timeout

OPCODE: XXXX XXXX TTTT TTTT 0111 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets up the IDY timeout value in milliseconds. This is the amount of time to wait for an IDY frame to return when sourced as controller. It is also the time between sourcing command frames when powering up the loop. This timeout is initialized to 255 milliseconds at power on.

5.9.6.6 Clear Data Buffers

OPCODE: XXXX XXXX XXXX XXXX 1000 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Zeros data buffer counts and resets pointers to start of buffers.

5.9.6.7 Set IDY SRQ Poll Timeout

OPCODE: XXXX XXXX TTTT TTTT 1001 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Sets the time between sourcing IDYs when the IDY poll is enabled. Default value is 255 milliseconds. The IDY poll is active only while the

I/O processor is controller.

5.9.6.8 Setup Interrupt Mask

OPCODE: XXXX XXXX MMMM MMMM 1010 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Updates Interrupt Mask Byte. If the interrupt mask is being set to a non zero value, then the SRQ bit in the Interrupt Cause byte is cleared. This is to avoid duplicate interrupts due to one SRQ. Executing this command clears the Interrupt Occurred status bit.

5.9.6.9 Read Interrupt Cause

OPCODE: XXXX XXXX XXXX XXXX 1011 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Contents of RAM location message
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Returns the value of the interrupt cause byte. This byte is cleared after it is sent to the HP-71.

5.9.6.10 Read DDC Frame

OPCODE: XXXX XXXX XXXX XXXX 1100 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Contents of RAM location message
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: Allows the HP-71 to read the last DDC frame received. The DDC register is cleared after the contents are sent.

5.9.6.11 Update Terminate on SRQ Mode

OPCODE: XXXX XXXX 000M 0000 1101 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: If M bit is set, set terminate on loop SRQ mode, otherwise clears terminate on loop SRQ. This mode is active only when the I/O CPU is controller. If the I/O CPU is listener or in controller standby mode, it will stop the data transfer with an NRD sequence when a SRQ is detected. When the I/O CPU is a talker, it will stop sending data when SRQ is detected and will set the NRD bit in the mailbox and send the conversation halted message to the HP-71.

5.9.6.12 Power Up the Loop

OPCODE: XXXX XXXX XXXX XXXX 1110 1111

HP-IL FRAMES SENT: NOP (50 times, until one returns), RFC
MAILBOX RESPONSE: Status Message
POSSIBLE ERRORS: Loop Not Complete
REQUIRED STATUS: none

DESCRIPTION: If controller and the loop is not powered up, this command will power up the loop. The loop is powered up by sending NOP frames (up to 50), until one returns. The RFC frame is then sent. The time between sourcing command frames is the IDY timeout value. If the loop has been successfully powered up, the I/O CPU will send its current status to the HP-71. If the loop is broken, the error bit will be set in the mailbox.

5.9.6.13 Enable/Disable IDY Poll

OPCODE: XXXX XXXX XXXM XXXX 1111 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: none
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: If M bit is set, IDY poll is enabled, otherwise it is disabled. This poll is executed only when the I/O CPU is controller. An IDY will be sent every X msecs, if the I/O CPU is idle. The value of X may be set with the SET IDY SRQ POLL TIMEOUT command. This allows the loop to be monitored for SRQ without having to send frames or put the loop in EAR mode. If the IDY returns with the service request bit set, the I/O CPU will flag this in the mailbox by setting the Loop Service Request bit and by requesting service on the HP-71 bus. The IDY Poll will be automatically disabled at this point. If no service request is pending then polling will continue until it is disabled by the HP-71. If the loop is not yet powered up and the poll enabled, the poll is automatically disabled.

5.9.7 Diagnostic Class

5.9.7.1 Read RAM

OPCODE: AAAA AAAA RXXP XXXX 0000 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Contents of RAM location message
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command allows the HP-71 to read a byte of the I/O CPU RAM. The RAM page from which to read is specified by the RP bits in the command and the address is in the A bits. The value read is returned to the HP-71. This command is useful for development.

5.9.7.2 Write RAM

OPCODE: AAAA AAAA BBBB BBBB 0001 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Contents of RAM location message
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command allows the HP-71 to write to a RAM location on page 0 (low 256 bytes). The address is specified in the A bits and the value to write out is specified in the B bits. After the byte is written it is read by the I/O CPU and the contents are sent to the HP-71. This command is useful for development.

5.9.7.3 Self Test

OPCODE: XXXX XXXX XXXX XXXX 0010 1111

HP-IL FRAMES SENT: none
MAILBOX RESPONSE: Two test messages & self test results msg
POSSIBLE ERRORS: none
REQUIRED STATUS: none

DESCRIPTION: This command tells the I/O CPU to execute a self test. The following two test messages are sent to the HP-71:

high	mid	low	handshake	I/O CPU NRD
0101 0101	1010 1010	0101 0101	1010 1XX1	0
1010 1010	0101 0101	1010 1010	0101 1XX1	1

Then a RAM and ROM self test is performed and the results of the test are sent to the HP-71. See the Diagnostics class of messages from the I/O processor.

5.9.8 Data Class

Data from the HP-71 which is to be put in the output buffer passed as either a triple data byte or a single data byte. One of the two bits in the HP-71 handshake byte is set to indicate what type of transfer it is. If it is a triple data transfer then all three bytes in the mailbox contain a byte of data, with the low byte being the first. If it is a single byte of data, then the data byte is in the low byte of the mailbox.

5.10 Mailbox Messages from the I/O processor

The messages sent from the I/O CPU to the HP-71 are in response to a command the HP-71 has sent or a frame received on the loop. The opcode is in the Mid-low nibble of the message. The opcodes are shown in the messages following, with the low nibble of the mailbox on the left and the high nibble on the right.

5.10.1 Frame Class

OPCODE: DDDD DDDD 1CCC XXXX XXXX XXXX

STIMULUS: Manual or scope mode and frame received
Single data frame and listener
HP-71 command: send frame and send me frame

received in response

DESCRIPTION: Frame Class is a means for the I/O CPU to send a frame received on the loop to the HP-71. The C bits are the control bits and the D bits are the data bits of the frame.

5.10.2 Device Address Class

OPCODE: SSSP PPPP 01SS XXXX XXXX XXXX

STIMULUS: HP-71 command:
Auto Address the Loop
Find the Nth Device of Type M
Increment or Read Current Device Address
Read my HP-IL Loop Address

DESCRIPTION: Device class is a means for the I/O CPU to send a device address to the HP-71. The P bits contain the primary address, the secondary address + 1 is in the S bits. A secondary address of zero indicates there is no secondary address.

5.10.3 Status and Error Class

5.10.3.1 Current I/O Processor Status

OPCODE: STLC BPUI 0001 KRXV NNNN NNNN

STIMULUS: HP-71 command:
Send Status
Read error number

DESCRIPTION: This message is a means to let the HP-71 know the current HP-IL status and current error code (if any). The twelve bits of status are:

B Controller Standby Mode
P Set if IDY Poll is enabled or loop is in EAR mode
U Set if Address Table is not valid
I Interrupt Pending (set when an enabled interrupt has occurred, cleared everytime interrupt mask byte is set)

S System Controller
T Talker Active

L   Listener
C   Active Controller

K   Locked Out Mode
R   Remote Mode
X   Data in Output Buffer
V   Data Available in input buffer

The error codes sent in N bits are:

0   No error detected
1   No such device (HP-71 request to find a device)
2   Device not ready (HP-71 request to start a transfer)
3   Loop is not complete (IDY doesn't return)
4   Frame Lost (hardware detected)
5   Input to Output Overrun on HP-IL hardware
6   Frame sent out is not the same as frame received
7   Incorrect frame received, protocol violation
8   Frame Lost (software buffer overrun)
9   Illegal Status for command (e.g. not controller)
10 Partial Frame received
11 Frame Timed Out on the Loop
12 Illegal CURRENT Device Address or Loop is Unaddressed
13 Self Test Failure (set only at power on reset)

5.10.3.2 Nop

OPCODE: 0000 0000 0000 XXXX XXXX XXXX

REQUIRED STATUS: Pass control command successfully executed

DESCRIPTION: This is a handshake message only. It is sent to the HP-71 to indicate control has been passed successfully.

5.10.3.3 IFC Received

OPCODE: 0001 0000 0000 XXXX XXXX XXXX

REQUIRED STATUS: none

DESCRIPTION: This message is not currently used.

5.10.3.4 EOT Received

OPCODE: 001E 0000 0000 XXXX XXXX XXXX

REQUIRED STATUS: Controller and a data transfer was terminated with an EOT.

DESCRIPTION: This message is sent to the HP-71 only when the I/O processor is controller of the loop, a data transfer was started and the transfer was not halted by count or a terminating character match. The E bit is set if an ETE frame was received and clear if an ETO frame was received.

5.10.3.5 Data Transfer Halted

OPCODE: 0100 0000 0000 XXXX XXXX XXXX

REQUIRED STATUS: Controller and data transfer was stopped due to terminate on SRQ mode or an HP-71 command to Send a NRD frame or Frame count was met and the I/O CPU was not listener.

DESCRIPTION: Status message to indicate the data transfer was halted due to Send NRD frame command or terminate on SRQ and SRQ received or frame count met when controller and not listener.

5.10.4 Terminating Conditions Met

OPCODE: 0101 0000 0000 XXXX XXXX XXXX

REQUIRED STATUS: Terminate on END frame or terminate on character match mode must be set and matched when active listener.

DESCRIPTION: Message to indicate the terminating conditions were matched as listener for either END frame or character match.

5.10.5 Diagnostics Class

5.10.5.1 Self Test Results

OPCODE: 0RA0 0000 0010 XXXX XXXX XXXX

STIMULUS: HP-71 command to execute self test.

DESCRIPTION: This message reports the results of self test command. It is sent in response to a self test command from the HP-71. The ROM and RAM test results are indicated by the R and A bits respectively. If the bit is set then the test was successful.

5.10.5.2 RAM Value

OPCODE: MMMM MMMM 0011 XXXX XXXX XXXX

STIMULUS: HP-71 command received:
Read or Write Memory command received or Read DDC or Read Interrupt Cause byte received.

DESCRIPTION: This message returns the value of a RAM location to the HP-71 in M bits.

5.10.6 Data Class

Data from the I/O CPU will come in one of 2 flavors. A single data byte will be sent back with the opcode from the FRAME CLASS. Data may also be sent as a triple data message. This message is indicated by the Three Data Bytes bit set in the I/O CPU handshake byte of the mailbox. The first byte is in the low byte of the mailbox, the second byte is in the middle byte and the third byte is in the highest addressed byte of the mailbox. The message should be read from low byte to high byte. When the highest nibble of the message is read by the HP-71, the I/O CPU's message available will automatically be cleared.

5.11 I/O Processor as a Device

In device mode, the I/O processor retransmits frames on the loop and keeps track of the current HP-IL status. The I/O processor may be setup to request service on the HP-71 processor bus whenever certain states become true by setting the interrupt mask byte. When the HP-71 processor executes the SREQ? instruction, the second least significant bit will be set if the I/O processor is requesting service.

The bits in the interrupt mask are described below:

Bit Number	Description
7	IFC received which HP-71 didn't source
6	MLA received

5	TCT received
4	MTA and SDA received
3	Service Request on the Loop (controller)
2	DCL or SDC Received
1	GET Received
0	DDC Received

An interrupt cause byte is kept by the I/O CPU. Whenever one of above conditions is met, the corresponding bit is set in the interrupt cause byte. This byte may be read and is cleared automatically after it is read.

Whenever the interrupt mask byte is setup an AND of the mask byte and the cause byte is executed. If the result is not zero, the I/O processor will request service on the HP-71 bus by setting a bit in the mailbox. Otherwise service request will not be set until an interrupt condition is matched. A single service request is generated even though multiple interrupts may occur before the interrupt cause byte is read.

A bit in status, Interrupt Pending, indicates that an enabled interrupt has occurred. This bit is cleared whenever the interrupt mask byte is set.

This method of handling interrupts guarantees that no interrupts which may occur while in the interrupt processing routine will be lost, since the cause bits will continue to accumulate even after the interrupt routine has been entered.

It is desirable for all interrupt events to accumulate except service request. If the interrupt routine is entered due to a service request, when the interrupt register is read, the SRQ occurred status bit is cleared. When more frames are sent out on the loop to satisfy the SRQ, it may cause the SRQ bit in the interrupt register to be set. If the interrupt routine exits and enables the SRQ interrupt, another interrupt will be generated due to the original service request. To avoid this problem, everytime the interrupt mask is set to a non zero value, the bit in the interrupt cause byte which indicates that a SRQ was received is cleared.

5.11.1 HP-IL Frames and I/O Processor's Response

The following lists show all the currently defined HP-IL frames, the value of the data bits in HEX and the response of the I/O processor to each frame.

5.11.1.1 Univeral Command Group Frames

NOP (10) Nop Frame. No Response.

LLO (11) Local Lockout Frame. If in remote enabled state, the Local Lockout status bit is set.

DCL (14) Device Clear Frame. Clears input and output buffers. All data received from the loop and not read by the HP-71 will be lost. All data sent from the HP-71 to the I/O CPU which has not been sent out on the loop will be lost. If the DCL interrupt is enabled, the I/O processor will request service on the HP-71 bus. The DCL bit will be set in the interrupt cause register.

PPU (15) Parallel Poll Unconfigure. Disables the I/O CPU's response to a parallel poll.

EAR (18) Enable Asynchronous Request. I/O CPU enters asynchronous request mode. If at anytime this mode is enabled and the I/O CPU is requesting service from the loop, an IDY with service request will be sent out.

IFC (90) Interface Clear. Listener, Talker and Controller status are cleared.

REN (92) Remote Enable. Remote Enabled status is set.

NRE (93) Not Remote Enable. Remote Mode, Local Lockout and Remote Enable status bits are cleared.

AAU (9A) Auto Address Unconfigure. I/O CPU's loop address is set to 21 (decimal).

LPD (9B) Loop Power Down. No Response

5.11.1.2 Addressed Command Group Frames

NUL (00) Null Frame. No Response

GTL (01) Go To Local. Remote Mode status is cleared if active listener.

SDC (04) Selected Device Clear. If active listener, response is the same as for a DCL frame.

PPD (05) Parallel Poll Disable. If active listener, the I/O CPU's response to a parallel poll is disabled.

GET (06) Group Execute Trigger. If active listener and GET interrupt is enabled then the I/O CPU will request service on the HP-71 bus. The GET bit is set in the interrupt cause register.

ELN (0F) Enable Listener NRDs. If active listener, sets internal status, listener NRDs are enabled.

PPE (8X) Parallel Poll Enabled. If active listener, the I/O CPU's parallel poll response is setup and enabled according to the X bits in the PPE frame.

DDL (AX-BX) Device Dependent Listener. If active listener the frame will be saved in the last DDC frame register. If active listener and DDC interrupts are enabled, the I/O CPU will request service on the HP-71 bus. The DDC bit is set in the interrupt cause register.

DDT (CX-DX) Device Dependent Talker. If addressed talker the frame will be saved in the last DDC frame register. If addressed talker and DDC interrupts are enabled, the I/O CPU will request service on the HP-71 bus. The DDC bit is set in the interrupt cause register.

5.11.1.3 Listener/Talker/Secondary Command Group

LAD (2X-3X) Listener Address Frame. If this is my listen address then talker status is cleared and active listener status is set.

If the I/O CPU has a secondary address and this address matches its primary address, then an internal flag is set to indicate my primary listen address was just received.

If the listener active interrupt is enabled, the I/O CPU will request service on the HP-71 bus and the LA bit in the interrupt cause register is set.

UNL (3F) Unlisten Frame. Clears listener status.

TAD (4X-5X) Talk Address Frame. If this address is my talk address, then listener status is cleared and active talker status is set. If this address is not my talk address then clear all talker status.

If the I/O CPU has a secondary address and the address on the TAD frame matches its primary address, then an internal flag is set to indicate my primary talker address was just received.

UNT (5F) Untalk Frame. Clears all talker status.

SAD (6X-7X) Secondary Address Group. If the I/O CPU doen't have a secondary address this frame is ignored. If this address matches the I/O CPU's secondary address and it's primary listener or talker address was just received, then listener/talker status is set. If this address does not match the I/O CPU's secondary address and my primary talk address was just received, then talker status is cleared.

5.11.1.4 READY Frames

RFC (00) Ready For Command. Retransmit frame after previous command has been executed.

ETO (40) End of Transmission OK. I/O CPU sources this frame when active talker to terminate a data stream. It is only sent when instructed by the HP-71.

ETE (41) End of Transmission Error. I/O CPU sources this frame when talker immediately after it detects an data error. A data error occurs when a data frame received does not match the data frame sent out by the I/O CPU.

NRD (42) Not Ready For Data. If active talker and this frame is received, the NRD frame is retransmitted and when the data byte sourced is received an EOT (End of Transmission) is sent out.

SDA (60) Send Data Frame. If addressed talker, active talker status is set. Any data in the output buffer will be sent out. If talker active interrupt is enabled, service will be requested on the HP-71 bus and the TA bit is set in the interrupt cause register.

SST (61) Send Status. If addressed talker, current status is sent out. Up to 2 bytes of status may be sent. Default at power on is 1 byte of value 0.

SDI (62) Send Device ID. If addressed talker, current Device ID is sent out. At power on, the I/O CPU's Device ID is length 0. The HP-71 sets the Device ID to ASCII string "HP-71" followed by a carriage return and line feed whenever it detects an I/O CPU reset.

SAI (63) Send Accessory ID. If addressed talker, the current accessory ID is sent out. The I/O CPU does not have a accessory ID at power on. The HP-71 sets the accessory ID to 3 whenever it detects an I/O processor reset.

TCT (64) Take Control Frame. If addressed talker, then control of the loop is assumed. The I/O CPU will immediately power up the loop by sending out a NOP frame sequence followed by a RFC in response to to a TCT frame. If the controller interrupt is enabled, the I/O processor will request service on the HP-71 bus. The CA bit is set in the interrupt cause register.

* AAD (8X-9X) Auto Address Frame. If the I/O CPU is already auto addressed then this frame is ignored. If the address on the frame is 31 then the frame is ignored. If not auto addressed and the address is less than 31 then the I/O CPU takes the address on the frame for its own address, increments the frame address by 1 and passes it on to the next device.

* AEP (AX-BX) Auto Extended Primary Address. If the I/O CPU is already addressed or has not just received an auto extended secondary address then this frame is ignored. If this frame has an address of 31 then it is ignored. If the I/O CPU has just been assigned an auto extended secondary address and is waiting for a primary address then it takes this address for its primary address and passes the frame unmodified on to the next device.

* AES (CX-DX) Auto Extended Secondary Address. If the address on this frame is 31 of if the I/O CPU is already auto address configured, this frame is ignored. Otherwise, I/O CPU saves this address as its secondary address, increments the frame address and sends it on to the next device. Addressing will not be completed until the I/O CPU receives a primary address.

* To determine whether or not the I/O CPU has been assigned an address, the byte at address 35 hex (ADR-RMT-S) can be read and bit 4 (LOOP-UNADDRESS) can be tested. If it is 0 the I/O processor has a valid address, if it is 1 the I/O CPU is not auto addressed.

5.11.1.5 IDY Frames

IDY (XX) Identify Frame. If the I/O CPU is requesting service on the loop the SRQ bit is set before the IDY is retransmitted.

ISR (XX) Identify Frame with Service Request. No Response.

5.11.1.6 DOE Frames

DAB (XX) Data Frame.
DSR (XX) Data Frame with Service Request.
END (XX) End Frame.
ESR (XX) End frame with Service Request.

All of the frames in this class are processed alike. If the I/O CPU is not talker or listener, the frame is simply retransmitted. If the I/O CPU is active talker, the frame is error checked and the next data frame is send out. If the I/O CPU is listener the frame is put in the input buffer and retransmitted. If the I/O CPU is requesting service on the loop, the service request bit is set in the frame before it is retransmitted.

5.12 Additional Capabilities

By using the commands to Read and Write to the I/O CPU RAM and ROM, some additional capabilities can be realized. These are the described below:

I) Reallocation of RAM between the input and output buffers. There are 131 bytes of RAM available for buffer space. The default allocation is 66 bytes for the output buffer and 65 bytes for the input buffer. The buffers are adjacent in memory, so that by updating pointers, sizes and the dividing address between the 2 buffers, the sizes may be easily changed. At power on the input buffer is positioned in memory from address 7D hex to address BD hex and the output buffer extends from address BE hex to FF hex.

A recommended procedure would be:

(1) Verify that both the input and output buffers are empty. This can be accomplished by reading

status. Both status bits, Data Available and Data in Output Buffer should be zero.

(2) Update input buffer size and space bytes. This is the only tricky part to modifying the buffer sizes. It must be done in such a way that the input buffer count appears to be negative. If not the I/O processor will detect data in the input buffer and will begin sending it to the HP-71. Count is calculated by subtracting the buffer space from the buffer size. During the transition, it must be guaranteed that the buffer space is greater than the buffer size. Therefore follow the following logic:

```
IF current input buffer size > new input buffer
                               size

   THEN DO
          Write new input buffer size (@74 hex)
          Write new input buffer space (@78 hex)
        END
   ELSE DO
          Write new input buffer space (@78 hex)
          Write new input buffer size (@74 hex)
        END
```

(3) Set both input buffer pointers to start of input buffer. The input pointer is in RAM at address 76 hex and the output pointer is in RAM at address 77 hex. They should be set to 7D hex.

(4) Write to the address (@79 hex) which holds the dividing address between the input buffer and the output buffer. It should be set to the value 7D hex plus the input buffer size.

(5) Write to output buffer size byte (@75 hex). Update it to the new output buffer size.

(6) Set output buffer pointers (input pointer is at @7A hex, output pointer is at @7B hex). They need to be set to point anywhere in the new output buffer area, such as the last byte in the buffer at @FF hex.

II) Modify the point at which the I/O CPU NRD bit is cleared in the mailbox. Currently the I/O CPU NRD bit is cleared whenever there are 3 bytes available in the output buffer. The value of 3 is kept in a byte of RAM called NRD-INTR-VALUE. By writing to this byte, the point at which the NRD bit is cleared in the mailbox is changed. This byte is at hex address 3E.

This may be useful in an application which wants an interrupt on NRD bit clearing. If the value in the NRD-INTR-VALUE is set to 50, then NRD will be cleared only when the I/O CPU has 50 bytes available in the buffer. So a fast master processor would only be interrupted when the I/O CPU has a larger amount of space available.

III) Use different timer prescales. Under some conditions it may be desirable to modify the timeout period substantially. This can be accomplished easily by changing the prescale rate in the timer status register at @18 hex. The prescale value is initialized 2 places. The first is at cold start and the second is in talker. So as long as the I/O CPU is not active talker, the prescale can be modified simply by a write RAM instruction to lengthen or shorten timeouts significantly.

5.13 HP-IL Capability Subsets

The following are the list of HP-IL capablities that the I/O CPU implements as specified in the HP-IL Interface Specification:

C1,2,3,4,5,6,7	Basic controller capability, System Controller Capability, SRQ Detect Capability, Control Passing and Receiving Capability, Parallel Poll Capability, Asynchronous SRQ Capability
T1,2,3,4,6	Data Capability, Status Capability, Accessory ID Capability, Device ID Capability, Extended Talker Address Capability
L1,3,4	Basic Listener Capability, Extended Addressing Capability, Halt Data Transfer Capability
SR2	Full SRQ Capability
RL2	Basic Remote Local Capability with Lockout
AA1,2	Basic Auto Addressing Capability, Extended Addressing Capability
PD0	No Power Down Capability
PP1	Basic Parallel Poll Capability
DC2	Complete Device Clear Capability
DT1	Complete Device Trigger Capability
DD1	Complete Device Dependent Capability

5.14 Mailbox Messages Opcodes

The following two tables show the opcodes of the commands from the HP-71 and the opcodes of the messages from the I/O processor.

OPCODE TABLE FOR COMMANDS FROM HP-71

High Byte	Class or Command
0X	No Parameter Class
00	NOP
01	READ ADDRESS TABLE
02	REQUEST HP-IL STATUS
03	END OF MESSAGE
04	CLEAR SRQ
05	SET SRQ
06	SEND ERROR MESSAGE
07	ENTER AUTO END MODE
08	GO INTO MANUAL MODE
09	GO INTO AUTO MODE
0A	UPDATE SYSTEM CONTROLLER BIT
0B	RESET CURRENT ADDRESS
0C	READ CURRENT ADDRESS
0D	INCREMENT CURRENT ADDRESS
0E	READ MY HP-IL LOOP ADDRESS
0F	TAKE/GIVE LOOP CONTROL
1X	Frame Class
2X-3X	Single Nibble Parameter Class
2X	ADDRESS/UNADDRESS ME AS TL
3X	POWER DOWN THE LOOP
4X-7X	Address Class
4X	ADDRESS P,S AS TALKER
5X	ADDRESS P,S AS LISTENER
6X	FIND STATUS OF DEVICE
7X	AUTO ADDRESS THE LOOP
8X-EX	Conversation Class
8X	START CONVERSATION
9X	START STATUS POLL
AX	START DEVICE ID
BX	START ACCESSORY ID
CX	PASS CONTROL
DX	SET TIMEOUT VALUE
EX	FIND Nth DEVICE OF TYPE M

F0-F2	Diagnostics Message Class
F0	READ MEMORY
F1	WRITE MEMORY
F2	SELF TEST
F3-FF	Multibyte Parameter Class
F3	SET SOT RESPONSE
F4	SET TERMINATOR MODE
F5	SET TERMINATOR CHARACTER
F6	SET NUMBER OF IDY TIMEOUTS
F7	SET IDY TIMEOUT TO M MILLISECS
F8	CLEAR DATA BUFFERS
F9	SET IDY POLL TIMEOUT VALUE
FA	SETUP INTERRUPT MASK
FB	READ INTERRUPT CAUSE
FC	READ DDC FRAME
FD	UPDATE TERMINATE ON LOOP SRQ MODE
FE	POWER UP THE LOOP
FF	ENABLE/DISABLE IDY POLL
(opcode in handshake byte)	Data Class

OPCODE TABLE FOR MESSAGES FROM I/O PROCESSOR

Middle Byte (Low Nibble)	MESSAGE CLASS
1XXX	Frame Class
000X	Status and Error Class
01XX	Device Address Class
001X	Diagnostics Class
(opcode in handshake byte)	Triple Data Class

```
+-----------------------------------------------+------------------+
|                                               |                  |
|   HP-IL POLL INTERFACES                       |   CHAPTER  6     |
|                                               |                  |
+-----------------------------------------------+------------------+
```

6.1 Overview

The HP-IL ROM extends many of the file-related keywords in the HP-71 to allow access to HP-IL devices. This is implemented by answering polls which the HP-71 mainframes sends out. (Please refer to the mainframe IDS for details on the polling process.)

The HP-IL ROM answers 30 polls to perform various tasks on HP-IL. These poll handlers will implement a predefined function no matter who issues the poll. So an assembly language program can issue a poll and use the poll handlers in the HP-IL ROM for I/O on HP-IL.

Among these 30 poll handlers, we estimate only half of them are useful to an applications programmer as general I/O functions. A list of these poll handlers is in the following section. The rest are needed to complete the I/O functions of the HP-71. This chapter includes a list of all the polls the HP-IL ROM handles, along with a description of the poll handler.

6.1.1 Output and Input of data

pPRTCL - Print class poll handler
Sets up a device for receiving data and returns the address of a handler which will actually output the data.

pPRTIS - PRINT device poll handler
Sets up the PRINT device (defined by PRINTER IS) for receiving data and returns the address of a routine which will do the printing.

pENTER - Input data from the loop.
Given a device address, this poll handler will enter data from the device and save the data on the stack.

6.1.2 Files on a mass memory device

pCAT$ - Returns the catalog information of a file.

pCREAT - Creates a file in a mass memory device.

pCOPYx - Transfers a file to or from an HP-IL device.

pFINDF - Search for a file in a given mass memory device.

pFSPCx - Search for a file by a given file specifier.

pPURGE - Purge a file from a mass memory device.

pRDCBF - Read a record of the file into an I/O buffer.

pRDNBF - Write current record out and read in next record.

pRNAME - Rename a file in a mass memory device.

pFPROT - Secure or make private a file in mass memory device.

pWRCBF - Write the I/O buffer out to the record in a file.

6.1.3 Parse and Decompile

pDEVCp - Parse an HP-IL device specifier.

pFILDC - Decompile an HP-IL device specifier.

pFSPCp - Parse a file specifier.

6.1.4 Initialization and addressing the loop

pCLDST - Initialize standard output devices.

pDSWNK - Wakeup if HP-IL Mailbox requesting service.

pPWROF - Power down the loop.

6.2 pCAT - CAT execution poll handler

Poll Name: pCAT - CAT execution poll handler

Name of Handler: hCAT

Type: POLL (poll #06)

Purpose:
Execute the CAT statement for an HPIL mass memory device

6.3 pCAT$ - CAT$ function poll handler

Poll Name: pCAT$ - CAT$ function poll handler

Name of Handler: hCAT$

Type: POLL (poll #07)

Purpose:
Execute the CAT$ function for a HPIL mass memory device.

6.4 pCLDST - Cold start poll handler

Poll Name: pCLDST

Name of Handler: PILCST

Type: FPOLL (poll #FF)

Purpose:
1. Create the HPIL save buffer (bPILSV). The existence of this buffer indicates the HPIL module already initialized.
2. Initialize all the mailboxes found.
   *Set IDY time out to 50 msecs.
   *Set up accessory ID and device ID.

3. Initialize DISPLAY IS and PRINTER IS devices.
   *Write 03F1FFF to IS-DSP. This says the display device is unassigned but defaults to the 1st device in the loop with an accessory ID of 3X.
   *Write 02F1FFF to IS-PRT. This says the print device is unassigned but defaults to the 1st device in the loop with an accessory ID of 2X.
4. Set ENTER terminating character to Line-Feed character(0A).

6.5 pCONFG - Configuration poll handler

Poll Name: pCONFG - Configuration poll

Name of Handler: PILCNF

Type: FPOLL (poll #FB)

Purpose:
1. Search for the HPIL save buffer (bPILSV), do the following if the buffer not found:
   *Create the HPIL save buffer. The existence of this buffer indicates the HPIL module already initialized.
   *Initialize standard output device to DISPLAY IS DISPLAY, PRINTER IS PRINTER.
   *Set terminate character to line-feed for ENTER.
2. Search for the DISPLAY and PRINTER device.
3. Reclaim the ASSIGN IO buffer and device specifier buffer.
4. If there is a display device assigned, write the display routine address to system RAM.

6.6 pCOPYx - COPY execution poll handler

Poll Name: pCOPYx - COPY execution poll

Name of Handler: hCOPYx

Type: POLL (poll #08)

Purpose:
Handler for the execution of COPY statement.

6.7 pCREAT - Create a file in a mass memory device

Poll Name: pCREAT - Create a file in a mass memory device

Name of Handler: hCREAT

Type: POLL (poll #09)

Purpose:
Create a new file in a HPIL mass memory device.

6.8 pDEVCp - Parse an HPIL device specifier

Poll Name: pDEVCp - Device parse poll handler.

Name of Handler: DEVSPp

Type: POLL (poll #01)

Purpose:
Parse an HP-IL device specifier.

6.9 pDIDST - Store device specifier information

Poll Name: pDIDST - Store device specifier information

Name of Handler: hDIDST

Type: POLL (poll #0A)

Purpose:
Store device specifier information to a given RAM location.
Save this information when the device is found.

The specifier information is saved, so if the loop is reconfigured, a search for the device can be repeated.

6.10 pDSWNK - Deep Sleep Wakeup poll handler

Poll Name: pDSWNK - Deep Sleep Wakeup -- no key down

Name of Handler: PILWNK

Type: FPOLL (poll #FE)

Purpose:

The HP-IL module is capable of requesting service on the HP-71 bus. But HP-IL is not the only device which can request service on the HP-71 bus. The Timer or the keyboard may request service too.

Any time the HP-71 detects a service request and it is not because a key is down, it will issue this Poll to give other LEX files, like the HP-IL ROM, a chance to respond to it's service request.

The purpose of this poll handler is to cause the HP-71 to wake up from deep sleep. The only thing this handler will do is set the ATTN key hit flag to 1. This will cause the HP-71 to wakeup from deep sleep. After the HP-71 wakes up, it will discover that there is a service request pending. It will then issue the Service Request poll. The HP-IL ROM will actually process the service request during the Service Request poll.

6.11 pENTER - Enter data from HPIL

Poll Name: pENTER - Enter data from HPIL

Name of Handler: hENTER

Type: POLL (poll #12)

Purpose:

To read data from HP-IL and put it on the math stack.

6.12 pEXCPT - Exception poll handler

Poll Name: pEXCPT - Exception poll handler.

Name of Handler: hEXCPT

Type: FPOLL (poll #F8)

Purpose:
Perform ON INTR end-of-line branch. The interrupt mask is setup by the ENABLE INTR statement. When an interrupt event occurs, the HPIL module will request service from the HP-71. The HP-71 will in turn issue the service request poll. When the HPIL module responds to the service request poll, it only sets the "Exception" flag (S12) and then returns to the mainframe immediately. At the end of each statement execution, the mainframe will check the "Exception" flag. If it is set, the mainframe will issue the Exception poll. This poll handler will verify the interrupt condition again and take the end-of-line branch if possible. If the branch can't be taken, this handler will set the "Exception" flag again and return. Setting the Exception flag on return will cause the HP-71 to issue another Exception poll at the end of next statement execution.

The following conditions will cause the end-of-line branch to become pending (it can not be taken immediately):

1. No ON INTR been executed or OFF INTR been executed.
2. HP-71 is not running a program.
3. The last statement executed is not at the end of a line.

6.13 pFILDC - Decompile an HPIL device specifier

Poll Name: pFILDC - Decompile an HPIL device spec

Name of Handler: PILDC

Type: POLL (poll #02)

Purpose:

Decompile an HPIL device spec stored as literal

Input stream:

<t*>
or <t%> <num expr> [(<num expr>)]
or <num expr>
or <tLITRL> <literal data> [(<num expr>)]
or <tSEMIC> <volume label>

Output text:

*
or :%<num expr> [(<num expr>)]
or :<num expr>
or :<literal data> [(<num expr>)]
or .<volume label>

6.14 pFINDF - Find a file in an HPIL device

Poll Name: pFINDF - Find a file in an HPIL device

Name of Handler: hFINDF

Type: POLL (poll #17)

Purpose:

Find a specified file in a given mass memory device.

6.15 pFPROT - Secure a file or make a file private

Poll Name: pFPROT - File protect handler

Name of Handler: hFPROT

Type: POLL (poll #0B)

Purpose:

Execute the SECURE/PRIVATE statement for a file in an HPIL device.

6.16 pFSPCp - Parse a file specifier

Poll Name: pFSPCp - File spec parse

Name of Handler: FILSPp

Type: POLL (poll #04)

Purpose:
Parse a file specifier that contains HPIL device specifier.

```
File specifier syntax:
 Input stream:
     <string expression>
 or  [ <file name> ] : <device specifier>
 or  [ <file name> ] . <volume label>
 Token output:
     <string expression>
 or  <tLITRL> [ <file name> ] <tCOLON> <device specifier>
 or  <tLITRL> [ <file name> ] <tSEMIC> <volume label>
```

6.17 pFSPCx - Find a file from the file specifier

Poll Name: pFSPCx - File spec execute

Name of Handler: FILSPx

Type: POLL (poll #05)

Purpose:
Find the file from the file specifier.

6.18 pIMXQT - IMAGE execution poll handler

Poll Name: pIMXQT - IMAGE execution starts

Name of Handler: ENTUSG

Type: FPOLL (poll #1D)

Purpose:

Handle the poll to do formatted input for ENTER USING. This poll is issued by the execution of USING. This is the hook for a LEX file to use the IMAGE parse routine in the mainframe to do formatted input or output.
The execution of ENTER will jump back into the USING routine in the mainframe to parse the IMAGE, if the statement is ENTER USING. The USING routine will parse the IMAGE string first then issue this poll to see if any LEX file wants to continue from that point.
The HPIL ROM always answers this poll and checks if the statement executing it is ENTER. If it is, the HPIL ROM will take over from that point.
This handler does not return to the poller via a "RTN", it does a direct jump back to "USGrst" in the USING code.

6.19 pKYDF - Key definition poll handler

Poll Name: pKYDF - Key definition poll

Name of Handler: hKYDF

Type: FPOLL (poll #1B)

Purpose:

Catch the key definition poll to execute a BASIC command received from the Loop.

When a key is pressed on the keyboard, the HP-71 saves the keycode in the key buffer first, then processes the key code when it is idle. When it processes the key code, it issues this poll first to see if any LEX file wants to define the key code.
This is the hook used by HPIL to execute a BASIC command.

When the HPIL module receives data in remote mode, it will wipe out the key buffer and put a single key code into the key buffer. This key code won't be recognized by the HP-71. Moments later when the HPIL module responds to the key def poll, it will read the ASCII string into an I/O buffer and set the key buffer pointer to point to the I/O buffer. This will cause the BASIC command from the loop to be parsed and executed.

6.20 pMNLP - Mainloop poll handler

Poll Name: pMNLP - Mainloop

Name of Handler: PILMLP

Type: POLL (poll #FA)

Purpose:

Restore the display device if it was turned off by hitting the ATTN key once while displaying.

This poll is issued by the Mainloop every time it is ready to display the cursor character. The display device could be turned off several ways, e.g. by aborting out of an I/O operation by hitting the ATTN key.

The purpose of this handler is to restore the display device if it is offed by the ATTN key. The user doesn't have to do a RESTORE IO to restore the display device once it is turned off by the ATTN key.

6.21 pPRTCL - Print class poll handler

Poll Name: pPRTCL - Print class poll handler

Name of Handler: hPRTCL

Type: POLL (poll #0E)

Purpose:

This is the poll handler that can be used to output data to a device other than the standard output device. This poll handler will set the device up for receiving data and return an address of a routine which will actually do output the data (the routine name is "PRASCI").

6.22 pPRTIS - PRINT device poll handler

Poll Name: pPRTIS - PRINT device poll handler

Name of Handler: PRTIS

Type: POLL (poll #0F)

Purpose:
Sets up PRINT device for receiving data and returns the address of the routine which will actually do the printing. The PRINT device is defined by the PRINTER IS statement.

6.23 pPWROF - Power-off poll handler

Poll Name: pPWROF- Power-off poll handler

Name of Handler: PILPOF

Type: FPOLL (poll #FC)

Purpose:
1. Sets device codes (DISPLAY, PRINTER) to power off values to allow restart on next usage.
2. Sends power-down message to all HPIL modules if the HPIL module is not in manual or device mode and flag -21 is clear.

6.24 pPURGE - Purge a file in a mass memory device

Poll Name: pPURGE - Purge a file in a mass memory device

Name of Handler: hPURGE

Type: POLL (poll #10)

Purpose:
Purge a file in a mass memory device. If the file is opened to the File Information Buffer (FIB), the file start field in the FIB is zeroed. The poller should call the routine "PUGFIB" in mainframe to purge the FIB entry.

6.25 pRDCBF - Read a record from a mass memory device

Poll Name: pRDCBF - Read current record from mass memory

Name of Handler: hRDCBF

Type: FPOLL (poll #18)

Purpose:
Read a record (256 bytes) from a mass memory device into an I/O buffer.

This routine is designed to work with a file on a mass memory device. The file has to be opened to the File Information Buffer (FIB) first. This can be done by the ASSIGN # statement. The FIB will contain information about the file such as the current file pointer and the file size. For a file on a mass memory device, there is an I/O buffer associated with the file (also done by the ASSIGN # statement).

This poll handler can be used to read on a given record number from a file into the associated I/O buffer. The record this poll handler will read is the record pointed to by the current file pointer in the FIB. The FIB also contains the I/O buffer number associated with this file.

(Refer to HP-71 IDS for details about the FIB)

When this routine is exitted, the file access nibble in the FIB is zeroed, and the current file pointer is not changed.

6.26 pRDNBF - Write current, read next record

Poll Name: pRDNBF - Write current record and read next record.

Name of Handler: hRDNBF

Type: FPOLL (poll #19)

Purpose:

When writing or reading from a file I/O buffer and the end of the I/O buffer is reached, execute this poll. It will write the I/O buffer out to the file if necessary and read in the next record of the file into the I/O buffer.

This routine is designed to work with a file on a mass memory device. The file has to be opened to the File Information Buffer (FIB) first. This can be done by the ASSIGN # statement. The FIB will contain information about the file such as the current file pointer and the file size. For a file on a mass memory device, there is an I/O buffer associated with the file (also done by the ASSIGN # statement).

When opening a file, the first record (256 bytes) of the file is read into the associated I/O buffer. All accesses to the file are directly written to or read from the I/O buffer. When accesses reach the end of the I/O buffer, the next record will be read into the I/O buffer. If the data in the current I/O buffer has been altered, it will be written back to the file before the next record is read in.

To use this poll, the poller only needs to pass the FIB entry address of the file. This routine will check if it needs to write the I/O buffer back out to the file first, and then read in the next record.

6.27 pRNAME - Rename a file in a mass memory device

Poll Name: pRNAME - Rename

Name of Handler: hRENAM

Type: POLL (poll #11)

Purpose:
Rename a file in an HP-IL mass memory device.

6.28 pSREQ - Service request poll handler

Poll Name: pSREQ - Service Request poll handler

Name of Handler: PILSRQ

Type: FPOLL (poll #F9)

Purpose:
The HPIL module is capable of requesting service from the HP-71. But the Timer and Card Reader may also request service. When the HP-71 detects a service request and it is not by the Timer or Card Reader, it will issue this poll to give the plug-in module a chance to service the request. This is how the HP-71 gets control from the mainframe.

The HPIL module will request service in two cases:

1. An interrupt event occurs and it matches the interrupt mask set up by the ENABLE INTR statement.

   In this case, the service request poll handler will only set the "Exception" flag (S12) and return. The End-of-Line branch will be carried out by the Exception poll handler.

2. Receiving data from the loop while the HPIL module is a device in the loop.

   In this case, the service request poll handler will only generate a "funny" key code in the key buffer, that subsequently will cause the keyboard routine to issue the "KYDF" (key define) poll. Execution of the BASIC command will be carried out in the KYDF poll handler.

6.29 pVER$ - Version code poll handler

Poll Name: pVER$ - Version code poll handler

Name of Handler: hVER$

Type: FPOLL (poll #00)

Purpose:
To show the presence of the HPIL module and add the revision code to the VER$ function.

6.30 pWRCBF - Write a record to a mass memory device

Poll Name: pWRCBF - Write I/O buffer to current record

Name of Handler: hWRCBF

Type: FPOLL (poll #1A)

Purpose:
According to the FIB, write the file I/O buffer to where it came from in a mass memory device. Buffer contents, current position and record address in FIB are not changed by this operation.

This routine is designed to work with a file in a mass memory device. The file has to be opened to the File Information Buffer (FIB) first. This can be done by the ASSIGN # statement. The FIB will contain information about the file such as the current file pointer and the file size. For a file in a mass memory device, there is be an I/O buffer associated with the file (also done by the ASSIGN # statement).

To use this poll, the poller only needs to pass the FIB entry address of the file. This routine will find the I/O buffer and write it back to the proper place in the file. The difference between this poll handler and the "pRDNBF" is that this routine will not automatically read in the next record to the I/O buffer.

On exitting this routine the file access nib in the FIB is set to zero and the I/O buffer contents and the file pointer in the FIB are not changed.

6.31 pZERPG - Zero program information poll handler

Poll Name: pZERPG - Zero program poll

Name of Handler: hZERPG

Type: POLL (poll #F7)

Purpose:

Zero interrupt mask.

This poll is issued when zero program information due to an END, ENDALL, EDIT, Program Edit.

HP-IL ROM UTILITY ROUTINES	CHAPTER 7

7.1 Overview

This chapter describes the utility routines in the HP-IL ROM. The second section describes the JUMPER routine, which is used to access the utility routines. The following sections describe utility routines which are contained in the HP-IL ROM which may be useful to other applications.

Please note that ONLY those routines described in this chapter are guaranteed to reside at the entry addresses given. These are the only supported entry points in the HP-IL ROM. There are many more utility routines in the HP-IL ROM which are not described in this section. These utility routines may not reside at the same location in the HP-IL ROM from one version of code to the next. Therefore to insure any code developed will be compatible with all future releases of the HP-IL ROM, access only those entry points described in this chapter.

7.2 How to call a utility routine

Since the HP-IL ROM is a soft addressable ROM, its actual address is defined at configuration time. Therefore, a utility routine in the HP-IL ROM can not be called by a direct GOSBVL. To access a routine in the HP-IL ROM, first determine the starting address of the HP-IL ROM from the configuration tables. Then add the offset of the routine to be called to the starting address, to get the actual address of the routine in HP-IL ROM.

The following JUMPER routine is designed to make this whole process easier. This routine will search the configuration tables to determine the address of the HP-IL ROM. It adds the offset of the routine to the actual address of the HP-IL ROM and then jumps to this address.

The JUMPER routine can be included with any LEX files or ROMs which want to access utilities in the HP-IL ROM. The source code for the routine is given below.

7.2.1 JUMPER routine

```
**
** Name:        JUMPER - Jump to a routine in HPIL ROM
**
** Category:    ADDCAL
**
** Purpose:
**      By giving the offset of a routine entry in the HPIL
**      ROM, this routine will find the absolute start
**      address of the HPIL ROM and do an indirect jump to
**      specified routine.
**
** Entry:
**      RSTK points to the 5-nibble offset from the start of
**      the LEX file to the desired entry point.
**
** Exit:
**      LEX file found:
**        Jumps to desired routine with all CPU registers pre-
**        served, including carry and mode (DEC/HEX), with the
**        exception of SB ("Sticky Bit")
**        Execution will return to after the 5 nibbles offset.
**
**      LEX file not found:
**        Jumps directly to MFERR with error "XWORD Not Found"
**
** Calls:       I/OFND
**
** Uses.......
**  Inclusive: SNAPBF[44:0]
**
** Stk lvls:    2 (I/OFND)
**
** NOTE: 1) Stk lvls are used only within this routine and do
**          not apply to the destination routine (ie the use
**          is only a transient usage within this routine, and
**          nothing remains on RSTK when this routine jumps to
**          the target routine except whatever was on the RSTK
**          on entry to this routine)
**
**       2) The proper way to set up the RSTK as needed for
**          the entry conditions to this routine:
**
**
**             .
**             . {Assembly code preceding the call}
**
**             .
**             GOSUBL =JUMPER
**             CON(5) ({target addr})-({target LEX table addr})
```

```
**              .
**              . {Continue with assembly code here}
**              .
**
************************************************************
**
snapbf  EQU     #2F7F0          Snap buffer entry address
exword  EQU     #0023           Xword Not Found error number
i/ofnd  EQU     #118BA          I/OFND routine entry address
bserr   EQU     #0939A          BSERR routine entry address
lexpil  EQU     #FF             HPIL ROM LEX ID
*
*
=JUMPER
*
* Save D1, C[W], A[W], B[A], P, carry, and mode in SNAPBF
* (Total size of SNAPBF is 16+16+5+5+5, or 47 nibbles. This
* routine uses 45 of those nibbles)
*
        RSTK=C
        CD1EX
        D1=(5) =SNAPBF
        DAT1=C A                Write D1 @ SNAPBF
        D1=(2) (=SNAPBF)+5
        C=RSTK
        DAT1=C W                Write C[W] @ SNAPBF + 5
        D1=(4) (=SNAPBF)+21
        DAT1=A W                Write A[W] @ SNAPBF + 21
        D1=(2) (=SNAPBF)+37
        C=B     A
        CPEX    5               Save P @ SNAPBF + 42
        P=      6
        C=0     P
        GONC    JUMP05          C[6]="0" means carry clear
        C=C-1   P               C[6]#"0" means carry set
JUMP05  P=      7
        C=0     P
        C=C-1   P               C[7]="9" means decimal mode
        DAT1=C 8                Write B[A],P,Carry,mode@SNAPBF+37
        SETHEX                  Force HEX mode for I/OFND
*
* Now A[W],B[A],C[W],P and D1 are available for use
*
        P=      0
        LC(3)   =bLEX           Find the LEX buffer
        GOSBVL =I/OFND
        GONC    JUMP90          Not there!! (Error)
*
* Found the LEX buffer...D1 points to it
*
* Search the LEX buffer for the HPIL ROM LEX ID
```

```
*
        LC(2)   =LEXPIL         C[B] = HPIL ROM LEX ID
        B=C     A               B[B] = HPIL ROM LEX ID
        A=0     A
        A=A+1   A               A[B] = 1
*
JUMP10  C=DAT1 6
        ?C=0    B               End of LEX buffer?
        GOYES   JUMP90          Yes...exit
        ?B#C    B               Right ID?
        GOYES   JUMP20          No...try next one
*
* LEX ID number matches...check if the token # is in the range
*
        CSR     W
        CSR     A               C[3:0] is now the token range
        ?A<C    B               Too small?
        GOYES   JUMP20          Yes...keep looking
        CSR     A
        CSR     A
        C=C-A   B               If no carry, token # is in range
        GONC    JUMP30          In range...process offset
JUMP20  D1=D1+  11              Not in range...goto next LEX entry
        GONC    JUMP10          Go always
*_
*_
JUMP90  LC(4)   =eXWORD         "XWORD Not Found"
        GOVLNG  =BSERR          Do NOT return to caller if error
*_
*_
*
* Found the requested LEX table
*
JUMP30  D1=D1+  6               Point to address of main table
        C=DAT1  A               Read the address of table into C
        B=C     A               Put address of table into B[A]
*
* Now get offset from main table start from the RSTK pointer
*
        C=RSTK                  Get address of offset...
        D1=C                    ...into D1
        D1=D1+  5               Skip the offset field
        CD1EX
        RSTK=C                  Put return address back on RSTK
        C=DAT1  A               Read offset from main table
        C=C+B   A               Add address of main table
        RSTK=C                  Push desired address onto RSTK
*
* Now restore the registers and jump to the routine
*
        D1=(5)  (=SNAPBF)+21    Position to A[W] value save area
```

```
        A=DAT1 W              Restore A[W]
        D1=D1+ 16             Position to carry/mode/B[A] save
        C=DAT1 8
        B=C    A              Restore B[A]
        P=     7              Check mode
        C=C+1  P              If carry, hex mode
        GOC    JUMP50
        SETDEC                No carry = DEC mode
JUMP50  P=     6
        ?C#0   P
        GOYES  JUMP60         Set carry if C[7]#0
JUMP60  P=C    5              Restore P from C[5]
        D1=(4) (=SNAPBF)+5    Position back to C[W] save area
        C=DAT1 W              Restore C[W]
        D1=(2) =SNAPBF        Position to D1 save area
        RSTK=C                (Temporarily save C[A] on RSTK)
        C=DAT1 A
        D1=C                  Restore D1
        C=RSTK                (Restore C[A] from RSTK)
        RTN                   Jump to the routine
        END
```

7.3 Data Input and Output routines

PRASCI - Character outputting routine.

PREND - Closing part of the PRASCI routine.

REDCHR - Character inputting routine.

7.3.1 PRASCI - Character outputting routine.

Name: PRASCI - Send ASCII characters to the loop

Entry Address: 107F Hex

Purpose:
Send the ASCII characters to the loop (already set up)

Entry:
MBOX^ points to the desired mailbox
A[A] contains the length of the string in bytes
D[A] is the start address of the string

Exit:

If loop error, jumps to ERRORX
P=0
D1 positioned following last character sent

Calls: GETMBX,WRITIT,TSAVD0,TRESD0,<ERRORX>

Uses.......
Inclusive: A[A],C,D1,P,FUNCD0,ST[8,3:0]

Stk lvls: 3 (pushed D0;WRITIT)(pushed D0;TRESD0)

7.3.2 PREND - Closing part of the PRASCI routine.

Name: PREND - Clean up the loop after PRINT/OUTPUT

Entry Address: 10B7 Hex

Purpose:
Clean up the loop after a PRINT/OUTPUT sequence

Entry:
Device(s) are addressed as listener(s)
MBOX^ points to the mailbox used

Exit:
D0 points to the mailbox used
Carry clear (P may be non-zero)

Calls: D1=SR0,SAVEIT,UTLEND

Uses.......
Inclusive: A,B,C,D,R2,R3,D0,D1,P,ST[3:0]

Stk lvls: 4 (UTLEND)(SAVEIT)

7.3.3 REDCHR - Character inputting routines.

Name: REDCHR - Read characters from the loop
Name: RED-LF - Read characters from the loop until <Lf>
Name: SKP-LF - Read & discard characters from the loop
Name: REDC00 - Read characters from the loop until <Lf>
Name: RDST01 - Read characters from the loop to stack

Entry Address: REDCHR - 22F7 Hex
RED-LF - 22E4 Hex

SKP-LF - 22DD Hex
REDC00 - 22E7 Hex
RDST01 - 2301 Hex

Purpose:

Read data from the loop onto the stack

Entry:

REDCHR,REDC00,RED-LF,SKP-LF only:
The 7 nibble device specifier is stored on the bottom (highest address) of the math stack.

RDST01 only:
R1[6:0] is the 7-nibble device specifier

(All entries)

D1 points to current top of math stack. Data read will be stored on top of stack (last character placed at lowest address)

Available memory on stack will be checked.

S5 (BytCnt):
- 1:Read a specified number of characters
  - A[A] is the number of characters to read
- 0:Terminate by END frame or terminating char match
  - A[B] is the terminating character

S6 (Trash):
- 1:Ignore the data which is read
- 0:Save the data which is read on the stack

S7 (ChrTrp):
- 1:Detect a special character in incoming data
  - B[B] is the character to be detected
  - If B[3:2]=00, ignore the character; otherwise replace the character with B[3:2]
- 0:No special character processing

If system flag -23 is set:
Terminate by ETO, terminating character is ignored

If S5 (BytCnt)=0, S6 (Trash)=0, and S-R0-3[0]>2 (the destination is a string), then R3[A] is the maximum number of chars to read before interrupting the conversation with an NRD. R3[S] must not be "F".

If S5 (BytCnt)=1 or S6 (Trash)=1, then flag -23 has no effect other than to terminate on an ETO instead of the terminator character.

```
        If { S-R0-3[0]<=2 (not string dest) and S5 (BytCnt)=0 }
          or { in device mode (not controller) },
          then flag -23 has no effect (it is ignored).
```

Exit:

```
     HEX mode.
     XM=0.
     Carry clear:
       D1 points to the last character read
       Number of chars read=(FORSTK)-D1
       S4 (Memerr)=0
     Carry set:
       S4 (Memerr)=1: Insufficient memory (Need to load eMEM)
       S4 (Memerr)=0: P,C[0] is the error code
```

Calls: FSTK-7,SFLAG?,STGART,CHKSTK,GETDev,CLMODE,CS=TYP,
            PUTC,SETTRM,PUTEFC,YTML,PUTE,GETX,FRAME-,CLMDUT

Uses:
 Inclusive: A,B[15:14,B],C,D[15:13,5:0],R1,R2,D0,D1,P,ST[7:0]

Stk lvls: 4 (START)

NOTE: B[B] is modified only if an error has occurred

7.4 Display routines

BDISPJ - Character-oriented display routine

7.4.1 BDISPJ - Character-oriented display routine

Name: BDISPJ - HPIL Character-oriented display routine

Entry Address: 3637 Hex

Purpose:
     Routine to display characters on HPIL devices

Entry:
     A[B] is a data byte
     HEX mode

Exit:
    A[B] is the data byte from entry
    Display status bits restored
    HEX mode, carry clear

Calls: CHKASN,SETLP,FNDMBX,START,GTYPE,MTYL,FINDA,
GETMBX,WRITIT,SENDIT,SENDI+,PUTD,PUTX,END,
MOVCUR,MOVCU+,D0=CUR,D0@CUR,Clear?,SendBf,
BLANKC,LCleft,DSPCL?

Uses.......
 Exclusive: A[15:2],B[W],C[W],D[A], D0,D1,P,(ST)
 Inclusive: A[15:2],B[W],C[W],D[15:13],D[5:0],D0,D1,P,(ST)

Stk lvls: 4 (START)

NOTE:
    Does not alter A[B], returns (DSPSTA+3) in STatus bits

7.5 Mass memory routines

BLDCAT - Build catalog entry given directory entry.

CHKMAS - Check if a device is a mass memory device.

DSPCAT - Display a CAT test string.

ENDTAP - Clean up the loop after mass memory action.

FINDFL - Find file on mass memory device.

FORMAT - Format medium in the specified drive.

GDIRST - Locate the start of directory and get its length on a mass memory device.

GETDIR - Get the Nth entry in a tape directory.

INITFL - Initialize a file in a mass memory device.

LSTENT,NXTENT - Move to the last/next directory entry.

MOVEFL - Move a file between two devices.

NXTENT - Move to next directory entry.

NEWFIL - Create a file on mass memory device.

READR# - Read a specified record from a mass memory device.

SEEKA - Seek to a record.

SEEKRD - Seek for a record, then read it.

TSTAT - Check the tape drive's status.

WRITE# - Write to a specified record.

7.5.1 BLDCAT - Build CAT text from directory entry.

Name: BLDCAT - Build CAT text, given directory entry

Entry Address: 6395 Hex

Purpose:
Build the CAT[$] string on the [MATH] stack, using the directory entry in SCRTCH[63:0]

Entry:
SCRTCH contains the directory entry for the file

Exit:
Carry clear, CAT text on stack, AVMEME at CAT text

Calls: D1@AVE,TSAVD0,BLANKC,SWAP01,GT2BYT,FTYPF#,HTODX,
WRTASC,GETBYT,GT2BY0,A-MULT,TRESD0

Uses.......
Exclusive: A[W],B[W],C[W],D[S],R0,D1,P
Inclusive: A[W],B[W],C[W],D[S],R0,D1,P,FUNCD0

Stk lvls: 3 (FTYPF#)

7.5.2 CHKMAS - Check for mass memory type device.

Name: CHKMAS - Check if D[X] is mass storage device

Entry Address: 42F1 Hex

Purpose:
Check if a device (at D[X]) is mass storage

Entry:
    D[X] is device address
    D0 points to the mailbox

Exit:
    Carry clear:
      Device is mass storage (Acc ID=#10), P=0
    Carry set:
      Not mass storage OR loop error
      (P, C[0] are error code - if P= =ePIL, C[0]=eDTYPE,
      than C[1] is device class, A[B] is full Acc ID)

Calls: GTYPE

Uses.......
 Exclusive: C[W],P
 Inclusive: A[A],C[W],P,ST[3:0]

Stk lvls: 3 (GTYPE)

7.5.3 DSPCAT - Display a CAT text string.

Name: DSPCAT - Display a CAT text string from @ D1

Entry Address: 6606 Hex

Purpose:
    Send 40 bytes (starting at D1) to the display

Entry:
    D1 @ start of data

Exit:
    P=0

Calls: D0=FR0,SWAP01,CKINF-,SEND20,CURSFL,CRLFND

Uses.......
 Inclusive: A-D,R0,D0,D1,all FUNCxx except FUNCR0,STMTR0,P

Stk lvls: 5 (CURSFL)

7.5.4 ENDTAP - Loop clean up after mass mem action.

Name: ENDTAP - Clean up the loop after mass mem action

```
Entry Address:  456E Hex

Purpose:
     Check status of a drive, rewind it, and unaddress all
     talkers and listeners

Entry:
     D[X] is device address
     D0 points to the mailbox

Exit:
     Carry clear:
       P=0, all OK
     Carry set:
       Error...P, C[0] are error code

Calls:       TSTAT,MTYL,DDL,<UTLEND>

Uses.......
 Exclusive: C[W],P,ST[3:0]
 Inclusive: C[W],P,ST[3:0]

Stk lvls:    3 (TSTAT)
```

7.5.5 FINDFL - Find file on mass storage device.

```
Name:        FINDFL - Set up loop, get a directory entry
Name:        FINDF+ - Set up loop, get directory entry (MS)
Name:        FINDFx - Find a file on a mass storage device

Entry Address:  FINDFL - 4734 Hex
                FINDF+ - 473B Hex
                FINDFx - 47C7 Hex

Purpose:
     Find file on external device (for FINDF+ and FINDFx,
     the device must be a mass storage device)

Entry:
     FINDFL,FINDF+:
       First 8 characters in A[W], last 2 in R0[3:0]
       D[A] is device address (set up by FILSPx poll handler)
     FINDFx:
       D[X] is mass storage device address
       D0 points to the mailbox
       First 8 chars of name in R0, last 2 in R1[3:0]
```

```
Exit:
     Carry clear:
       File directory entry in =SCRTCH[32]
       A[A] is starting record (A[4]=0)
       C[A] is number of records (C[4]=0)
       D1 points to file type
       B[3:0] is directory pointer for file (B[3:1] is
         record number, B[0] is entry within record)
     Carry set:
       P=0: Names don't match (same conditions as carry clear)
       P#0: Error (P, C[0] are error code)

Calls:      START,CHKBIT,CHKMAe,YTML,D1=SCR,READSU,hCPY5s,
 FINDFx --> GETDR!,NXTEN+,CSRC5,CSLC5,GETDIR,GETZER

Uses.......
 Exclusive: A,B,C,        D1,P,              ST[5]
 Inclusive: A,B,C,D[15:5],D1,P,SCRTCH[63:0],ST[5:0]

Stk lvls:   5 (GETDR!)
```

7.5.6 FORMAT - Format medium in the specified drive.

```
Name:         FORMAT - Format medium in specified drive

Entry address: 4326 Hex

Purpose:
     Format medium in specified drive (initialize it)

Entry:
     R0 contains vol label ([11:0]), # of entries ([15:12])
     Drive address is in D[X]
      D[X](lower five bits) = device's primary address
      D[X](middle five bits) = device's secondary adrs(0 if none)
      D[X](top 2 bits) = Loop # ( 0 is loop #1)
     D0 points to the mailbox

Exit:
     Carry clear:
       P=0, drive is rewinding (successful formatting)
     Carry set:
       Error (P, C[0] are error code)

Calls:      DDL,DDT,READI3,WRITIT,PRMSGA,CLLOOP,CLEARN,
            MTYL,YTML,TSTAT,SEEKA,PUTALR,PUTDX,PUTD,PUTE,
            GETD,ChkEOT,DdlWrt,D1=SCR,F->SCR,PUTDIR,
            CSLC4,CSLC5,CSRC5,ASLC4,ASRC4,YMDHMS,<ENDTAP>
```

```
 Uses:
  Exclusive: A,B,C,D,R0,   R2,D1,P
  Inclusive: A,B,C,D,R0,R1,R2,D1,P,SCRTCH[63:0],ST[8:0]

 Stk lvls:   4 (CLEARR)
```

7.5.7 GDIRST - Locate the start, length of directory

```
Name:        GDIRST - Get directory start and information

Entry Address:  48D8 Hex

Purpose:
     Locate the start of directory (and length) on mass mem
     and return both to the caller

Entry:
     D[X] contains the drive address
     D0 points to the mailbox

Exit:
     Carry clear:
       B[W] contains:
         Directory start pointer in [3:0], [15:12]
         Start of data area in [7:4]
         Zero in [11:8]
       D[W] contains:
         Drive address in [A] (No change)
         Number of directory records in [8:5]
         Address of LAST data record + 1 [12:9]
         Zero in [15:13]
     Carry set:
       Error (P, C[0] are error code)

Calls:       SEEKA,DdtRd,READSC,D1=SCR,GETALR,ASLC9,ASRC4,
             GETZER,(GDIRSM),ASRC9,CSRC8,ASRC3,ASLC3,CSLC4

Uses.......
 Exclusive: A,B,C,D[15:5],D1,P
 Inclusive: A,B,C,D[15:5],D1,P,SCRTCH[63:0],ST[3:0]

Stk lvls:   3 (SEEKA)(GDIRSB)
```

7.5.8 GETDIR - Get the Nth entry in a tape directory.

```
Name:       GETDI! - Get first directory entry from drive
Name:       GETDIR - Get the next directory entry from drive
Name:       GETDR" - Get the next directory entry @ B[3:0]
Name:       GETDR# - Get the next directory entry @ A[3:0]
Name:       GETDR+ - Get the next directory entry @ A[S]

Entry Address:  GETDR! - 486C Hex
                GETDR" - 4873 Hex
                GETDR+ - 488E Hex
                GETDIR - 48B5 Hex
                GETDR# - 4875 Hex

Purpose:
     GETDR!: Get the first entry in an LIF directory
     GETDR": Get the B[3:0]th entry in an LIF directory
     GETDR#: Get the A[3:0]th entry in an LIF directory
     GETDR+: Get the A[S] entry in the current record
     GETDIR: Get the next entry in an LIF directory

Entry:
     D[X] is the drive address
     D0 points to the mailbox
     GETDIR: Drive is addressed as talker, me as listener
     GETDR": B[3:0] is the directory entry #
     GETDR#: A[3:0] is the directory entry #
     GETDR+: A[S] is the directory offset nibble in record

Exit:
     Carry clear:
       Directory entry in =SCRTCH[32]
       A[W] is first 8 chars of filename
       D1 points past first 8 chars of filename
     Carry set:
       Error (P, C[0] are error code)

Calls:      GDIRST,SEEKA,DDT,MTYL,PUTD,YTML,TSTATA,READSC,
            D1=SCR

Uses.......
 Exclusive: A,  C,       P
 Inclusive: A,B,C,D[15:5],P,SCRTCH[63:0],ST[4:0]

Stk lvls:   GETDR!: 4 (GDIRST)
Stk lvls:   GETDR": 3 (SEEKA)(TSTATA)
Stk lvls:   GETDR#: 3 (SEEKA)(TSTATA)
Stk lvls:   GETDR+: 3 (TSTATA)
```

Stk lvls: GETDIR: 3 (TSTATA)

7.5.9 INITFL - Initialize a file

Name: INITFL - Initialize a file on external device

Entry Address: 6979 Hex

Purpose:
Initialize an external file after creation

Entry:
R1[S] = Create code of the file
Tape is positioned at the start of the file data area
R2[A] is # of sectors in the file

Exit:
Carry clear:
The file will be filled with zeros or all FF's
Create code = 2 - filled with zeros
Otherwise - filled with all FF's
Carry set:
Error...P, C[0] are error code

Calls: SENDIT

Uses:
Exclusive: A[W],C[W],D1, FUNCR1[15:0],P
Inclusive: A[W],C[W],D1,ST[3:0],FUNCR1[15:0],P

Stk lvls: 2 (SENDIT)

7.5.10 LSTENT,NXTENT - Move to directory entry.

Name: NXTENT - Move to next directory entry
Name: LSTENT - Move to previous directory entry

Entry Address: NXTENT - 4AB3 Hex
LSTENT - 4AC9 Hex

Purpose:
Increment/decrement to next/last directory entry

Entry:
C[3:0] is the current entry

Exit:
    C[3:0] is next/last entry
    P=0
    Carry set if crossed record boundary, else clear

Calls: None

Uses.......
 Inclusive: C[3:0],P

Stk lvls: 0

7.5.11 MOVEFL - Move a file between two devices

Name: MOVEFL - Move a file between two HPIL devices

Entry Address: 4606 Hex

Purpose:
    Move a block of "records" from one HPIL device to another

Entry:
    R1[A] = device addr of destination device (from FILSPx)
    R2[A] = device addr of source device (from FILSPx)
    R3[A] = record address of destination if mass mem
    B[A] = record address of source if mass mem
    R3[9:5] = number of records to copy

Exit:
    P#0!
    Carry clear: OK
    Carry set: error (P, C[0] are error code)

Calls: CSLC5,D1=AVE,CSRC10,CSLC10,START,GETDev,SEEKA,
            CHKBIT,DdtRd,READSU,D1@AVS,CSRC5,MTYL,DDL,ASRC10,
            WRITIT,hCPY5s,ASRC5,YTML

Uses.......
 Exclusive: A[W],C[W],D[A],R3[14:10],R4,D0,D1,P,ST[4:0]
 Inclusive: A[W],C[W],D[W],R3[14:10],R4,D0,D1,P,ST[8],ST[4:0]

Stk lvls: 3 (SEEKA)(hCPY5s)

Detail:
    COUNT# is R3[14:10] - # of records this transfer

```
COUNTD is R4[9:5]    - # of records already finished
COUNTR is R4[14:10]  - # of records remaining
COUNT  is R3[9:5]    - # of records to move (total)
```

7.5.12 NEWFIL - Create a file on mass memory device.

Name: NEWFIL - create a file on mass memory device

Entry Address: NEWFIL - 4AFA Hex
NEWFI+ - 4ADE Hex

Purpose:
Create a new file on a medium, given a pointer to the file data and all info needed to create the directory entry. If NEWFIL is called by CREATE, the file will be initialized according to its create code.

Entry:
```
     ST[=sOVERW]=1 if overwrite existing file, 0 if error on
        existing file
     D[X] is device address (D[B]=0 if LOOP)
     R0 is first 8 chars of name
     R4[15:12] is last 2 chars of name
     R1[5:0] is new file size in bytes
     R1[9:6] is new file type
     R1[14:10] is new file data start (RAM address)
        (If zero, don't copy any file...check CCode)
     R1[15] = 0 if called by COPY with device spec,
        "F" if called by COPY with LOOP or non-mass storage
          device (D[B]#0 means non-mass storage device)
        create code if called by CREATE
     R2[7:0] is data for implementation bytes ([B] is first
        byte of implementation field...byte 28)
     (R2[B] is FIRST byte of implementation info)
  NEWFIL:
     D0 points to the mailbox
```

Exit:
```
     Carry clear:
       P=0, R3 is file information (B[W] internally):
         [3:0]: Current directory pointer (of no value)
         [7:4]: Pointer to start of data area for file
         [11:8]: Pointer to old directory location (if found)
         [15:12]: Pointer to new directory location of file
       R1 is unchanged from entry conditions
         (If R1[S]="F" and R1[B]#"00" then R1[5:2] has been
          incremented, R1[B]=0)
       The file has been created on the mass storage medium
```

```
      Carry set:
        Error (P,C[0] are error code)

Calls:       START,CHKBIT,GDIRST,SEEKA,DdtRd,READSC,GT2BYT,
             NXTENT,PT2BYT,YMDHMS,MTYL,<ENDTAP>,I/OFND,PURFIB,
             FTYPF#,CHKSEC,CHKSIZ,PUGFIB,NEWF80,NEWF84,NEWF90,
             NEWF.0,GETMBX,D1=SCR,F->SCR
             CSRC3;4;5;8;9;12,ASRC4,CSLC3;4;5;8;12

NEWF80 -->v ASRC4;8,CSRC2;3;12,CSLC3,YMDHMS,PT2BYT,DdlPwr,
             SEEKA,MTYL,DDL,PUTD,PUTC,D1=SCR
NEWF84 -->v PT2BYT,CSLC2;6,MTYL,GT2BYT,CSRC13
PUTDR# -->v SEEKA,MTYL
NEWF90 -->v DdlPwr,DDL,PUTD
PUTDIR ---> DDL,D1=SCR,<NEWF.3>

NEWF.0 -->v CSRC4;10,SEEKA,MTYL,DDL,<INITFL>
NEWF.3 ---> WRITIT,GETST,PUTC,<TSTAT>

Uses.......
 Exclusive: A,B,C,D,R0,R2,R3,R4,D0,D1,P
 Inclusive: A,B,C,D,R0,R2,R3,R4,D0,D1,P,SCRTCH[63:0],ST[8,4:0]

Stk lvls:    5 (PUGFIB)(Only if deleting FIB entry:file existed)
Stk lvls:    4 (GDIRST)(NEWF80;YMDHMS)

Detail:
     Consolidates into one pass through the directory the
       following actions for mass storage:
            1. Find the file on the medium (if present)
            2. Find a space on the medium sufficient to hold
                 the file, giving preference to the place
                 it was before (if found in 1.)
            3. Purge the old directory entry, if not using
                 same entry for new file
            4. Write the new directory entry
            5. Copy the file to the data area of the medium
```

7.5.13 READR# - Read specified record from mass mem

```
Name:        READR# - Read a record from mass mem into RAM

Entry Address:  4594 Hex

Purpose:
     Read a specific record number
```

```
Entry:
     D1 points to the destination buffer
     A[3:0] contains the record number
     D[X] contains the drive address
     D0 points to the mailbox

Exit:
     Carry clear: OK (P=0)
     Carry set: Error (P, C[0] are error code)

Calls:       TSTAT,SEEKA,DdtRd,DDT,READSU,<TSTATA>

Uses.......
 Exclusive:      C[W],   P
 Inclusive: A[W],C[W],D1,P,ST[3:0]

Stk lvls:   3 (TSTAT)

Note: This routine will always read the device status first
      and ignore any device error that is reported initially
```

7.5.14 SEEKA - Seek a record.

```
Name:        SEEKA - Seek a record (record # in A[3:0])
Name:        SEEKB - Seek record (drive=listener,me=talker)

Entry Address:  SEEKA - 42C7 Hex
                SEEKB - 42CE Hex

Purpose:
     Seek to the specified record

Entry:
     SEEKA: Desired record # is in A[3:0]
     SEEKB: Desired record # is in A[3:0], drive is talker,
            I am listener
     Drive address in D[X]
     D0 points to the mailbox

Exit:
     Carry clear:
       Drive is talker, I am listener, P=0
     Carry set:
       Error (P,C[0] are error code)

Calls:       MTYL,DDL,PUTD,<TSTAT>

Uses.......
```

 Exclusive: C[W],P
 Inclusive: C[W],P,ST[3:0]

Stk lvls: 2 (MTYL) <TSTAT>

7.5.15 SEEKRD - Seek for a record, then read it.

Name: SEEKRD - Seek to a record, then read it

Entry Address: 636D Hex

Purpose:
 Seek a record on the mass memory device and read it

Entry:
 C[3:1] is the record # desired
 D0 points to the mailbox
 D[X] is the device address

Exit:
 Carry clear:
 P=0, record has been read into buffer 0 of device
 Carry set: Error (P=error #)
 Error (P,C[0] are the error code)

Calls: TSTAT,SEEKA,DDT,TSTATA

Uses.......
 Exclusive: A[A],C[W],P
 Inclusive: A[A],C[W],P

Stk lvls: 3 (TSTAT)(SEEKA)(TSTATA)

7.5.16 TSTAT - Check the tape drive's status.

Name: TSTAT,TSTATA - Check the drive status

Entry Address: TSTAT - 4293 Hex
 TSTATA - 429A Hex

Purpose:
 Check status of mass storage device

Entry:

D[X] contains the address of the drive
D0 points to the mailbox

Exit:
Carry clear:
Drive is addressed as a talker
Status in C[B]
Carry set:
Error (P, C[0] are error code)

Calls: YTML,PUTE,GETD (YTML only for TSTAT)

Uses.......
Exclusive: C[W],P
Inclusive: C[W],P,ST[3:0]

Stk lvls: 2 (YTML;PUTC)(GETD;GET)

7.5.17 WRITE# - Write to a specified record.

Name: WRITE# - Write to a specific record

Entry Address: 45D4 Hex

Purpose:
Write to a specific record on a mass mem device

Entry:
D1 points to the input buffer
A[3:0] contains the record number to be written
D[X] contains the drive address
D0 points to the mailbox

Exit:
Carry clear if OK (P=0)
Carry set if error (P, C[0] are error code)

Calls: TSTAT,SEEKA,MTYL,DdlWrt,DDL,WRITIT

Uses.......
Exclusive: A[A], P
Inclusive: A[A],C[W],D1,P,ST[8],ST[3:0]

Stk lvls: 3 (TSTAT)

Note: This routine always reads the device status first and ignores any initial device error.

7.6 Device searching routines

CHKAIO - Check if a string is an ASSIGN WORD.

CHKASN - Check a HPIL standard output device assignment.

DEVPAR - Decodes parsed device specifier, returns address.

FXQPIL - Get file name from program memory.

GADDR - Given a device specifier, finds address of the device.

GADRRM - Get HPIL address from program memory.

GADRST - Get address from stack.

GETDID - Fetch the device ID from a statement.

GETDVW - Get device word off the math stack.

GETID - Get the device ID from a device.

GETLPs - Get loop specifier, check mailbox status.

GETPIL - Get and evaluate an HPIL file specifier.

GHEXBT - Get hex value of 1 byte.

GTYPE - Get the accessory ID of a device.

GTYPST - Get device type from stack.

PROCDW - Process device word.

PROCLT - Process a device specifier from a literal.

PROCST - Process a device specifier from a string expression.

ROMTYP - Check if the string is a RESERVED WORD.

SAVEIT - Save standard output device descriptor entry.

SETUP - Build standard output device descriptor string.

7.6.1 CHKAIO - Check if a string is an ASSIGN WORD.

Name: CHKAIO - Check if device is an ASSIGN WORD

Entry Address: 411B Hex

Purpose:
Check if a string is an ASSIGN WORD (if so, return its value)

Entry:
B contains a string (B[B] is the first character, any unused characters are #00)

Exit:
P=0
Carry set if buffer not found or not an ASSIGN WORD
Carry clear if found...address in C[X]

Calls: CSLC5,ASRC5,I/OFND

Uses.......
Exclusive: A[W],C[W],P
Inclusive: A[W],C[W],P

Stk lvls: 1 (I/OFND)(CSLC5)(ASRC5)

7.6.2 CHKASN - Check an HPIL device assignment.

Name: CHKASN - Check if an HPIL assignment is active

Entry Address: 3CEC Hex

Purpose:
Check if the assignment is none, HPIL, or "other"
(If "OFF"ed, returns as if no assignment)

Entry:
C[6:0] is the assignment table value

Exit:
Carry set if not assigned/not HPIL/"OFF"ed/LOOP/NULL
Carry clear if assigned...B[W],C[X] set up for START
If C[S]<>0, this is a FIND (Address unknown)

```
Calls:      I/OFND

Uses.......
 Exclusive: B[W],C[W],P
 Inclusive: B[W],C[W],P

Stk lvls:   2 (pushed D1;I/OFND)
```

7.6.3 DEVPAR - Parse a device specifier.

```
Name:       DEVPAR - Parse a device specifier on the stack
Name:       DEVPR$ - Parse a string device spec on stack

Entry Address:  DEVPAR - 1C85 Hex
                DEVPR$ - 1CCB Hex

Purpose:
     Decode a device parameter (for functions which accept
     one parameter, either string or numeric, for device
     specifier)

Entry:
     P=0
     DEVPAR:
        D1 points to the parameter on stack
     DEVPR$:
        D1 points to string header (String is reversed)
        ST(sSTK)=1

Exit:
     FUNCD0 contains the calling routine's D0 value
     Carry clear: OK...D[X] is address (0 if not found)
            D1 set up for 1 numeric parameter return
     Carry set: Error...P, C[0] set up for ERRORX

Calls:      TSAVD0,POP1N,GADRRM,REVPOP,<DEVPR$>
     DEVPR$:TSAVD1,GETDIX,TRESD1

Uses.......
 Inclusive: A,B,C,D,R0-R3,D1,P,FUNCD0,FUNCD1,MLFFLG,ST[7,4:0]

Stk lvls:   3 (GETDIX - two levels saved in R0)
```

7.6.4 FXQPIL - Get the file name from program memory.

```
Name:         FXQPIL - Get a file name from memory (file spec)

Entry Address:  73E4 Hex

Purpose:
     Fetch a filename from program memory

Entry:
     Exit conditions from GETSTR
     (ST[sSTK]=0: literal in memory, =1:string on stack)
     (P=0)

Exit:
     D0/D1 set to first non-character item
     Carry clear (filename found):
       R0[W] is the first 8 chars, A[3:0] the last 2
       (Both are blank-filled)
     Carry set (no filename found):
       A,R0 are zeroed

Calls:        FXQPnm,FXQPn+

Uses.......
 Exclusive: A[W],     C[W],R0,      P
 Inclusive: A[W],B[W],C[W],R0,D0,D1,P

Stk lvls:   3 (FXQPnm)

Algorithm:
     Check if literal and no file name; if so, return zero
     Get the first 8 chars; put in R0; if reached end, set
       A[3:0]=\  \, return
     Get last 2 chars; put in A[3:0]; return
```

7.6.5 GADDR - Find the address of a device on loop.

```
Name:         GADDR - Get the address of a device from loop

Entry Address:  0994 Hex

Purpose:
     Get device address, given search information for the
```

```
      device

  Entry:
      D0 points to the HPIL mailbox
      D[B] is the search type (#1F,3F,5F,7F,9F)
         #1F: (Device type) -B[B] is accessory ID
         #3F: (Device ID)   -B[W] is device ID
         #5F: (Volume label)-B[W] is the label
         #7F: (Null)        -B[W] is "don't care"
         #9F: (LOOP)        -B[W] is "don't care"
      D[2] is the sequence number
      D[3] is the loop number
      D[S]=0 (for search type at exit)

  Exit:
      Carry clear:
        HPIL handshake in ST[3:0]
        Device address,(mailbox #)*1024 in D[X]
        D[S] is search type (1=device type, 2=device ID,
          3=volume label,4=NULL,5=LOOP)
        D[3] is sequence number (was in D[2] at entry)
      Carry set: P, C[S] are error code

  Calls:      PUTGF+,UNLPUT,PUTC+,GETERR,GETID,PUTGF-,UNT,
              TSTAT,SEEKA,DDT,TSTATA,READRG,ASRC4,MTYL,DDL

  Uses.......
   Exclusive: A[A],C[W],D[15:14],D[5:0],P
   Inclusive: A[W],C[W],D[15:13],D[5:0],P,ST[3:0]
              (If volume label, blankfills B[W],uses B[15:12])

  Stk lvls:   3 (GETID)(TSTAT)(SEEKA)
```

7.6.6 GADRRM - Get HPIL address from program memory.

```
  Name:       GADRRM - Get HPIL address from program memory
  Name:       GADRR+ - Get HPIL address from stack value

  Entry Address:  GADRRM - 4040 Hex
                  GADRR+ - 404F Hex

  Purpose:
      Get an HPIL address from program memory

  Entry:
      ST(sSTK)=0: D0 points to the expression in program mem
      ST(sSTK)=1: A[W] contains a floating number
```

```
Exit:
     Carry clear: C[X] is the HPIL address, P=0
     Carry set: Error (P is error #)

Calls:      EXPEX+,RESTST,AVM+16,GHEXB+

Uses.......
 Exclusive: A,B,C,D,                    P
 Inclusive: A,B,C,D,R0,R1,R2,R3,R4,D0,D1,P,FUNCxx

Stk lvls:   5 (EXPEX+)
```

7.6.7 GADRST - Get address from string on math stack.

```
Name:       GADRST - Get address from stack

Entry Address:  70F9 Hex

Purpose:
     Similar to GTYPST, except that the first 2 digits
     after the decimal point, if any, are used as the
     secondary address

Entry:
     D1 @ first character
     D[A] @ end of spec

Exit:
     Carry clear:
       C[X] is address
       D1 @ first unused character
       Skips trailing digits
       P=0
     Carry set:
       P, C[0] are error code

Calls:      NXTCHR,BAKCHR,RANGEN,DTOH,CSRC2

Uses.......
 Exclusive: A,B,C,   P
 Inclusive: A,B,C,D1,P

Stk lvls:   1 (NXTCHR)(BAKCHR)(RANGEN)(DTOH)(CSRC2)

Algorithm:
     Read a number from the stack until non-digit OR full;
```

```
    Check if "."...if not, return
    Get another number from the stack (2 digits)
    Combine the two numbers as one address, return
```

7.6.8 GETDID - Fetch the device ID

```
Name:       GETDID - Get device ID (specifier)
Name:       GETDIX - Get device ID (String expr on stack)

Entry Address:  GETDID - 6E19 Hex
                GETDIX - 6E37 Hex

Purpose:
     GETDID fetches a device ID, given D0 pointing to the
     ID in program memory

Entry:
     D0 points to the ID in program memory

Exit:
     Carry clear: Address/type in D[X], device type/ID in B
       If D[X]=0, then device id = "" OR *
       P=0
       FUNCD0 contains the D0 value after evaluating ID
     Carry set: error, P=error number

Calls:      GETSTR,PROCLT,NXTCHR,BAKCHR,Procst,TSAVD0,START

Uses.......
 Inclusive: A-D,R0-R4,D0,D1,P,STMTD1[3:0],STMTR1,FUNCxx,ST[11:0]

Stk lvls:   GETDID: 6 (GETSTR)
Stk lvls:   GETDIX: 4 (PROCST)
```

7.6.9 GETDVW - Get device word off the math stack

```
Name:       GETDVW - Get device word

Entry Address:  71C8 Hex

Purpose:
     Get a device word, given a pointer to the word

Entry:
```

```
      ST(=sSTK)=0:
         D0 points to first letter of device word in memory
      ST(=sSTK)=1:
         D1 points to first letter of device word on stack
         D[A] points to the end of the specifier

  Exit:
      Carry clear:
        Device word in B[W], zero-filled, first letter in B[B]
        P=0, carry clear if no error
        D0/D1 @ next character
      Carry set:
        Error (P, C[0] are error code)

  Calls:      NXTCHR,BAKCHR,UCRANG,RANGEN

  Uses.......
   Exclusive:     B[W],          P
   Inclusive: A[A],B[W],C[A],D0,D1,P (sSTK=0: D0; sSTK=1: D1)

  Stk lvls:   2 (UCRANG)
```

7.6.10 GETID - Get the device ID for a device.

```
  Name:       GETID - Read 8 bytes data into A after YTMLL
  Name:       READRG - Read 8 bytes data into the A register
  Name:       GETID+ - Read 8 bytes data into A after YTML

  Entry Address:  GETID  - 68A3 Hex
                  READRG - 689A Hex
                  GETID+ - 688F Hex

  Purpose:
      Read up to 8 bytes of data from a device and put it
      into A[W] (GETID and GETID+ strip Cr and trailing
      characters)

  Entry:
      D[X] is address of the device
      D0 @ mailbox

      READRG: Conversation is already set up

  Exit:
      Carry clear:
        Up to 8 bytes in A[W], number of bytes in D[S]
        P=0
```

Carry set:
  Error (other than device not ready)
  P,C[0]= Error #

Calls: YTML(GETID+),YTMLL(GETID),PUTE,GETX,FRAME-

Uses.......
 Exclusive: A[W],C[W],D[S],D[13],P
 Inclusive: A[W],C[W],D[S],D[13],P

Stk lvls: 2 (YTMLL)(YTML) (READRG uses only 1 level)

7.6.11 GETLPs - Get loop number, check status.

Name: GETLPs - Get (optional) loop #, check status

Entry Address: 1DAA Hex

Purpose:
Check if a loop number was passed to a function; if so, get that mailbox, else get first mailbox.
Check the status of the mailbox (reset?, etc)

Entry:
P=0
D1 points to the top of the stack
C[S] is the parameter count (0 or 1)
If C[S]=1, there is a numeric value on top of the stack

Exit:
Carry clear:
  P=0
  D0 points to the mailbox
  Mailbox status in C[X]
  D1 at (new) top of stack (loop number is popped off)
  FUNCD0 contains the caller's D0
Carry set:
  Error (P, C[0] are the error code)

Calls: TSAVD0,POP1N,GHEXB+,<FNDCHK>

Uses.......
 Inclusive: A,B,C,D,R0,D0,D1,P,FUNCD0,ST[3:0]

Stk lvls: 2 (TSAVD0)(GHEXB+)(<FNDCHK>)

7.6.12 GETPIL - Extract file name & device ID, acc ID

```
Name:        GETPIL - Evaluate an HPIL file specifier
Name:        GETPI+ - Get an HPIL file specifier from stack

Entry Address:  GETPIL - 6EA0 Hex
                GETPI+ - 6EA9 Hex

Purpose:
     This routine extracts the file name and the device
     and returns with the device type/device ID in B[W],
     address/type in D[X]

Entry:
     D0 points to the file specifier in program memory

Exit:
     ST(sDevOK) set if device spec was ok, else clear
     Carry clear:
       Filename in R0, R4[15:12]
       Device type in B[X]/B[W], address in D[X]
       If address = X00, then this is a * or a ""
       AVMEME collapsed back to starting point
     Carry set:
       Error (P,C[0] are error code)

Calls:       GETSTR,FXQPIL,NXTCHR,PROCLT,PROCST,ASRC4,D1=AVS,
             D1@AVE,CSRC12,GETDI5,ASLC12

Uses.......
 Inclusive:  A-D,R0-R4,D0,D1,P,STMTD1[3:0],STMTR1,ST(sDevOK),
             FUNCxx

Stk lvls:    6 (GETSTR)
```

7.6.13 GHEXBT, GTYPRM - Get hex value from 1 byte.

```
Name:        GTYPRM - Get one-byte hex value from literal
Name:        GTYPR+ - Clear status bits 11:0, GTYPRM
Name:        GHEXBT - Pop number off stack, get hex byte value
Name:        GHEXB+ - Use A[W] as value, convert to hex byte

Entry Address:  GTYPRM - 4003 Hex
                GTYPR+ - 4001 Hex
```

GHEXBT - 4012 Hex
GHEXB+ - 4016 Hex

Purpose:
Given D0 pointing to a numeric expression in program memory, return the HEX value of the expression

Entry:
ST(sSTK)=0: D0 points to the expression
ST(sSTK)=1: A[W] contains a floating number

Exit:
If carry clear, B[B] is the HEX type, B[4:2]=0,P=0,
C[B]=(DevTyp), C[XS]=0
If carry set, error (P=type)

Calls: EXPEX+,RESTST,AVM+16,FLTDH

Uses.......
Exclusive: A,B,C, P
Inclusive: A,B,C,D,R0,R1,R2,R3,R4,D0,D1,P,FUNCxx

Stk lvls: 5 (EXPEX+)

7.6.14 GTYPE - Get the accessory ID of a device.

Name: GTYPE - Get the device type (Acc id) from loop

Entry Address: 0C94 Hex

Purpose:
Get the accessory id of a device (address in D[X])

Entry:
D0 points to the HPIL mailbox
D[X] contains the address of the device to be checked

Exit:
Carry clear:
P=0
Device type in A[B] (if 2 byte response, A[3:2] is first byte received, A[B] is second)
If device does not respond to Acc ID, A[A]=0
Carry set: error (P, C[0] are error code)

Calls: YTML,PUTE,PUTGF

Uses.......

```
 Exclusive: A[A],C[W],P
 Inclusive: A[A],C[W],P,ST[3:0]

Stk lvls:   2 (YTML)(PUTGF)
```

7.6.15 GTYPST - Get device type (acc ID) from stack.

```
Name:       GTYPST - Get type from stack

Entry Address:  7088 Hex

Purpose:
     Given a pointer to the start of the type, return the
     numeric value of the type

Entry:
     D1 @ first digit of type
     D[A] @ end of specifier

Exit:
     Carry clear:
       Type in B[X], D1 @ first unused item
       C[X]=(=DevTyp)
       P=0
     Carry set:
       error (P, C[0] are error code)

Calls:      NXTCHR,BAKCHR,DTOH,RANGEN

Uses.......
 Exclusive: A[W],B[W],C[W],   P
 Inclusive: A[W],B[W],C[W],D1,P

Stk lvls:   1 (NXTCHR)(BAKCHR)(DTOH)(RANGEN)
```

7.6.16 PROCDW - Process device word.

```
Name:       PROCDW - Process device word

Entry Address:  7215 Hex

Purpose:
     Given a device word in B[W], figure out what it is
     (ASSIGN WORD, RESERVED WORD, NULL, LOOP, DEVICE ID)
```

Entry:
    B[W] contains the device word

Exit:
    P=0
    Carry set if sequence number is permissable after this
    Carry clear if sequence number is not permissable

Calls: CHKAIO,ROMTYP,(PRDWsb)

Uses.......
 Exclusive: C[W],P
 Inclusive: A[A],B[B],C[W],P

Stk lvls: 2 (CHKAIO)(ROMTYP)

Detail:
    Try in following order: ASSIGN WORD, RESERVED WORD,
      NULL,LOOP,(other=DEVICE ID)

7.6.17 PROCLT - Process literal.

Name: PROCLT - Process literal device spec

Entry Address: 7263 Hex

Purpose:
    Given a pointer to a device spec in memory, process it!

Entry:
    D0 @ device spec

Exit:
    Carry clear:
      P=0
      Device type/device id in B[X]/B[W]
      IF device type="*", *, or "" THEN C[X]=0
      ELSEIF address THEN C[X] is address+loop*1024
      ELSEIF LOOP then C[X] is "9F"+loop*4096
      ELSEIF NULL then C[B] is "7F"
      ELSEIF volume label THEN C[X] is "5F"+loop*4096
      ELSEIF device type THEN C[X] is "3F"+loop*4096
      ELSEIF device ID THEN C[X] is "1F"+loop*4096
    Carry set:
      Error (P, C[0] are error code)

Calls: NXTCHR,BAKCHR,GETDVW,PROCDW,SAVEAC,EXPEX+,
       GHEXBT,GADRR+,RESTST,SAVE2C,RESTD1,REST2C

```
Uses.......
 Exclusive: A,B,C,     R1,R2,      D0,   P
 Inclusive: A,B,C,D,R0,R1,R2,R3,R4,D0,D1,P,STMTD1[3:0],STMTR1,
            FUNCxx, all RAM available to FCNS

Stk lvls:   4 (EXPEX+ {saves a level on GOSUB stack first})
```

7.6.18 PROCST - Process a string device specifier

```
Name:       PROCST - Process string device specifier

Entry Address:  6F50 Hex

Purpose:
     Process a device specifier from a string expression

Entry:
     ST(sSTK)=1
     R0[W], R4[15:12] are filename
     D1 points to next item of string
     D[A] is the end of the string
     HEXMODE

Exit:
     Carry set if error (P,C[0] are error number)
     Carry clear:
            P=0
            Device type/device id in B[X]/B[W]
            IF device type="*", *, or "" THEN C[X]=0
            ELSEIF address, THEN C[X] is address+loop*1024
            ELSEIF LOOP, THEN C[X] is "9F"+loop*4096
            ELSEIF NULL, THEN C[B] is "7F"
            ELSEIF volume label THEN C[X] is "5F"+loop*4096
            ELSEIF device type THEN C[X] is "3F"+loop*4096
            ELSEIF device id THEN C[X] is "1F"+loop*4096

Calls:      NXTCHR,BAKCHR,UCRANG,GETDVW,PROCDW,GTYPST,GADRST

Uses.......
 Exclusive: A[W],B[W],C[W],R1,R2,   P
 Inclusive: A[W],B[W],C[W],R1,R2,D1,P

Stk lvls:   3 (GETDVW)
```

7.6.19 ROMTYP - Check if a string is a reserved word.

Name: ROMTYP - Check if device is a RESERVED WORD

Entry Address: 4167 Hex

Purpose:
Check if the string in B[W] is a RESERVED WORD; if so, return the value that corresponds to that word

Entry:
B contains the string (B[B] is the first character)

Exit:
P=0
Carry clear: B[B] is the device type; B[XS]=0
Carry set: not found

Calls: None

Uses.......
Inclusive: B[A],C[W],P (B[A] only if found)

Stk lvls: 1 (Internal call)(internal push)

7.6.20 SAVEIT - Save device descriptor entry.

Name: SAVEIT - Save device info at (D1) (7 nibbles)

Entry Address: 3E4B Hex

Purpose:
Save device descripter entry @ D1

Entry:
D1 @ destination entry
B,C are exit conditions of SETUP

Exit:
Carry clear, P=0 (Error exits directly)

Calls: CSRC3;4;5,CSLC4;9,I/OALL,I/OFSC,I/ODAL

Uses.......
Exclusive: A,B,C,D,R2,R3,D0,D1,P

```
 Inclusive: A,B,C,D,R2,R3,D0,D1,P

Stk lvls:   3 (I/OALL)(I/ODAL)

Algorithm:
            Check if entry will fit in 7 nibbles:
            If will not fit, goto SAVEI1
     SAVEI0:Read old entry; write new entry
            If old entry used buffer, deallocate the buffer
            RTNCC
           ----
     SAVEI1:Create a buffer for the entry
            Write the entry
            Build the info for the 7 nibble field
            Goto SAVEI0
```

7.6.21 SETUP - Build a recall string in C[6:0].

```
Name:       SETUP - Given info from START, set up C[6:0]

Entry Address:  3DC8 Hex

Purpose:
     Build a recall string in C[6:0] (carry set if buffer
     required to store this)

Entry:
     D is the info returned from START
       D[X] is address, (loop #) * 1024
       D[S] is type (0=address, 1=device type, 2=device ID,
       3=volume label, 4=NULL, 5=LOOP)
       D[3] is sequence # for types 1 and 2
     B is as returned from START

Exit:
     C[6:0] is the information to put into an IS-xxx entry
     P=0
     C[S]=0 if entry will fit in IS-xxx, else C[S]#0

Calls:      CSLC5,CSRC4,CSLC3

Uses.......
 Inclusive: C[W],P

Stk lvls:   1 (CSLC5)(CSRC4)(CSLC3)
```

7.7 Loop addressing routines

CHKSET - Check if a Mailbox has been reset and initialize it.

LISTEN - Address a device as listener.

MTYL - Address me as talker, one listener.

RESTOR - Clears offed status of standard output devices.

RESTRT - Set to research addresses of standard output devices.

START - Set up entry conditions for the loop.

UTLEND - Unaddress talker & listener, clean up.

YTML - Address a talker, me as listener.

7.7.1 CHKSET - Check if this Mailbox has been reset.

```
Name:          CHKSET - Check if this mailbox has been reset
Name:          CHKST+ - Set up this mailbox after reset

Entry Address:  CHKSET - 31DE Hex
                CHKST+ - 31F5 Hex

Purpose:
     Check if this mailbox has been reset...if so, set up
     device ID and accessory ID

Entry:
     D0 @ mailbox

Exit:
     D0 pointing to mailbox
     Carry clear:
       All OK (If mailbox had been reset, it has been set up)
     Carry set:
       Error...P, C[0] are error code

Calls:         PUTC,PUTE

Uses.......
 Exclusive: A[W],C[W],P
```

Inclusive: A[W],C[W],P

Stk lvls: 1 (PUTC)(PUTE)

Detail:
Check if RESET bit is set...if not, return, carry clear
Set IDY timeout = 50 mS
Set Accessory ID = (mSETAI)
Set Device ID = (vDEVID)&Cr&Lf

7.7.2 LISTEN - Address a device as listener.

Name: LISTEN - Address D[X] as listener
Name: ULYL - Unaddress listeners, address D[X] as Listener

Entry Address: LISTEN - 0CF1 Hex
ULYL - 0CEA Hex

Purpose:
Unaddress all listeners, address D[X] as listener

Entry:
Desired listener address in D[X]
D0 points to mailbox

Exit:
Carry clear: OK, P=0
Carry set: error (P=error #)

Calls: PUTC

Uses.......
Inclusive: C[W],P,ST[3:0]

Stk lvls: 1 (PUTC)

7.7.3 MTYL - Address me as talker, one listener.

Name: MTYL - Unaddress listeners, me talk, D[X] listen
Name: MTYLL- Address me as talker, D[X] as listener

Entry Address: MTYL - 0D18 Hex
MTYLL - 0D1F Hex

Purpose:

Address me as talker, D[X] as listener

Entry:
D[X] is the address of the device to be listener
D0 points to mailbox

Exit:
Carry clear: OK, P=0
Carry set: error (P=error code)

Calls: UNLPUT,LISTEN,<PUTC>

Uses.......
Inclusive: C[W],P,ST[3:0]

Stk lvls: 1 (UNLPUT)(LISTEN)

7.7.4 RESTOR - Reactive all devices.

Name: RESTOR - Clear "OFFED" bits in IS table entries

Entry Address: 3EF1 Hex

Purpose:
Reactivate all devices (clear their OFFED bits)

Entry:
Nothing

Exit:
Carry clear

Calls: Nothing

Uses.......
Inclusive: C[XS],D0

Stk lvls: 1 (Internal GOSUB)

NOTE: Does not alter P!

7.7.5 RESTRT - Restart all HPIL devices.

Name: RESTRT - Restart all HPIL devices (readdress)

```
Entry Address:  308D Hex

Purpose:
     Restart all device addresses in the HPIL system
     (set to search for address at next access)

Entry:
     P=0, HEXMODE

Exit:
     P=0
     Carry clear

Calls:      RESTRs,CSRC5,CSLC5,FIBOFF

Uses.......
 Exclusive:      C[W],D0,P
 Inclusive: A[W],C[W],D0,P

Stk lvls:   2 (FIBOFF)
```

7.7.6 START - Set up entry conditions for the loop.

```
Name:       START - Set up entry conditions for the loop
Name:       START+ - Set up loop information (loop # in C[S])
Name:       START- - Set up loop (loop # in C[S], sReadd=1)

Entry Address:  START  - 087D Hex
                START+ - 0883 Hex
                START- - 0886 Hex

Purpose:
     Set up the loop, given the device specifier

Entry:
     D[3:0] contains the device address (if known).
         If the address is not known, D[B]=#1F/3F/5F/7F/9F
            #1F: (DevTyp) B[X] is the accessory ID
            #3F: (DevID)  B[W] is the device ID
            #5F: (VolLbl) B[W] is the volume label
            #7F: (Null)   B[W] is "don't care"
            #9F: (Loop)   B[W] is "don't care"
            D[2] is the sequence number for #1F and #3F
     If D[X] is an address, bits 8 and 9 are the mailbox #
     If D[X] is not an address, D[3] is the mailbox #

Exit:
     Carry clear:
```

```
        Device address in D[X] (+mailbox*1024)
        D[S] is 0 if address given, 1 if device type,
          2 if device ID, 3 if volume label, 4 if NULL,
          5 if LOOP
        Sets D0 to the HPIL mailbox
        ST(sReadd) set if loop was readdressed, else clear
      Carry set:
        Error (P, C[0] are error code)

Calls:      SETLP,FNDCH-,GETDev,PUTGF-,PUTE,GETERR,GETST,
            SFLAG?,RESTRT,GETMBX,SWAP01,I/OFND

Uses.......
 Exclusive:      C[W],D[15   ],       D0,P,ST[4  ]
 Inclusive: A[W],C[W],D[15:13],D[5:0],D0,P,ST[4:0]

Stk lvls:   3 (RESTRT)(FNDCH-)<GADDR>

Algorithm:
     START: Derive loop # from D[X] (into C[S])     (SETLP)
     START+:Set flag (sReadd) to not force readdressing
     START-:Find mailbox, check for reset, OFFED    (FNDCH-)
            Check if controller...if so, goto STARTn
            Check if NULL, LOOP, or zero (if not, error)
            goto START3
            ----
   (Controller)
     STARTn:
            If force readdressing (sReadd=1)
              then send IFC to power up the loop
              else send power up the loop message (NOP frame)
     STARTS:Check if error powering up the loop    (GETERR)
     START!:Get Diamond status bits
            If sReadd=1 then goto START2
            If loop is unconfigured (sUNCNF)
              then
                If (supress readdress)=1 then goto START2
                Set all internal addresses=unknown (RESTRT)
                Set D0 to mailbox address          (GETMBX)
            goto START3
            ----
   (Readdressing the loop)
     START2:
            Set all internal addresses=unknown     (RESTRT)
            If (extended address flag=0) or
               (an ASSIGNIO is active)
              then readdress the loop, primary only
              else readdress the loop, extended addresses
            Send readdress message, get result     (PUTGF-)
            If address not returned by Diamond then error
   (Check the device specifier)
```

```
START3:If not (find device)
         then return (all OK)
         else goto GADDR (Get device address)
```

7.7.7 UTLEND - Unaddress talker & listener, clean up.

```
Name:        UTLEND - Unaddress talkers&listeners, clean up
Name:        ENDFN - Clean up the loop, preserve C[W] in R0

Entry Address:  UTLEND - 0861 Hex
                ENDFN  - 0855 Hex

Purpose:
     Clean up after accessing a loop

Entry:
     MBOX^ points to the mailbox used by this routine

Exit:
     Carry clear:
       D0 at last mailbox used before call
       ENDST: Jumps to NXTSTM
       ENDFN: Restores value of C[W] (saved at entry)
       UTLEND: First unaddress talkers/listeners, then END
     Carry set:
       Error (P, C[0] are error code)

Calls:  END:GETMBX
      ENDST:END
     UTLEND:UNT,UNLPUT
      ENDFN:UTLEND

Uses.......
 Inclusive: C[W],D0,P,ST[3:0]

Stk lvls:   END: 0 <GETMBX>
Stk lvls:   ENDST: 1 (END)
Stk lvls:   UTLEND: 1 (UNT)(UNLPUT)<END>
Stk lvls:   ENDFN: 2 (UTLEND)
```

7.7.8 YTML - Address a talker, me as listener.

```
Name:        YTML - "You" (D[X]) talk, "me" listen

Entry Address:  0D30 Hex
```

Purpose:
Address D[X] as talker, me as listener

Entry:
D0 points to mailbox
D[X] contains the address of the device to be talker

Exit:
Carry clear: P=0
Carry set: Error # in P

Calls: UNLPUT,PUTC,<PUTC=D>

Uses.......
Inclusive: C[W],P,ST[3:0]

Stk lvls: 1 (UNLPUT)(PUTC)

7.8 Communicating with I/O CPU routines

CHKSTS - Check Mailbox status and clear error mailbox bit.

DDL,DDT- Send a device dependent command to loop.

FNDMBX - Find an HPIL Mailbox in configuration table.

FRAMEE - HPIL frame encode from ASCII to 11 bit value.

FRAME+ - Evaluate an HPIL MB message, return message type.

GET - Get a message from Mailbox.

GETD - Get data.

GETDev - Check if the HPIL module is in device mode.

GETERR,GETST - Get error/status from I/O CPU.

GETHSS - Get 2 handshake nibbles from a Mailbox.

GETST - Get Mailbox status

GETMBX - Get the HPIL Mailbox address from RAM, put it in D0.

GETX - Fast data input routine.

GFTYPE - Get frame type from RAM.

GLOOP# - Get loop # from RAM (if one present).

PRMSGA - Print message contained in C-reg to loop.

PUTARL - Put message in A register to loop.

PUTC - Put a command (4 nibs) to the Mailbox.

PUTD - Put a single data byte to the loop.

PUTDX - Put multiple data bytes to Mailbox filling with zeros.

PUTE - Put an extended message (6 nibs) to Mailbox.

PUTEN - Send message to Mailbox, ignore error bit.

PUTX - Send 3 bytes of data to the loop.

READIT - Read data bytes from the loop.

SENDIT - Send 1 or 2 character sequence to the loop.

SETLP - Determine loop number for FNDMBX routine.

WRITIT - Output data to loop from RAM.

7.8.1 CHKSTS - Check Mailbox status, error, etc.

```
Name:          CHKSTS - Check Diamond status, errors, etc
Name:          FNDCHK - Find a mailbox, CHKSTS
Name:          FNDCH- - Check OFFED, Find a mailbox, CHKSTS

Entry address: CHKSTS = 0C24 Hex
               FNDCHK = 0C1B Hex
               FNDCH- = 0C10 Hex
```

Purpose:
Check that the status is OK for messages (ie NOT in manual mode), clear the error bit in Diamond, set/clear bit for device/controller

Entry:
FNDCH-:C[S] is mailbox desired
FNDCHK:C[S] is mailbox desired
CHKSTS:D0 points to mailbox

```
Exit:
     Carry clear:
       P=0, C[X] is Diamond status
       CHKSTS:D0 unchanged
       FNDCH-,FNDCHK:D0 points to mailbox
     Carry set: error (P, C[0] are the error #)

Calls:       GETHS2,CHKSET,GETERR,GETST,GETMBX

Uses:
 Exclusive:       C[X],P
 Inclusive: A[W],C[W],P,ST[3:0], bit(Device) of LOOPST

Stk lvls:    2 (GETST)(GETERR)(CHKSET)(pushed status;GETMBX)
```

7.8.2 DDL,DDT- Send a device dependent command.

```
Name:        DDT - Send a Device Dependent Command
Name:        DDL - Send a Device Dependent Command

Entry address:    DDT  = 6BC9 Hex
                  DDL  = 6BBA Hex

Purpose:
     Send a DDL/DDT as determined by P (these routines are
     only good for DDL/DDT 0-15)

Entry:
     P contains the DDL/DDT number desired
     Loop is set up
     D0 @ mailbox

Exit:
     Same as PUTE

Calls:       None

Uses:
 Inclusive: C[W],ST[3:0],P

Stk lvls:    0
```

7.8.3 FNDMBX - Find an HPIL Mailbox.

```
Name:        FNDMBX - Find an HPIL mailbox (C[S] is #)
```

```
Name:       FNDMB- - Find mailbox, clear disp bits, chk OFF
Name:       FNDMBD - Find an HPIL mailbox, clear disp bits
Name:       FNDMB+ - Find an HPIL mailbox (D[A] is spec)

Entry address: FNDMBX = 3C75 Hex
               FNDMB- = 3C40 Hex
               FNDMBD = 3C5F Hex
               FNDMB+ = 3C3C Hex

Purpose:
     Search the configuration tables to find a HPIL mailbox
     (C[S] is the number of the mailbox minus 1 - if C[S]
     is 2 then find the 3rd mailbox!)

Entry:
     FNDMBX,FNDMB-,FNDMBD:
       C[S] is the mailbox number -1
     FNDMB+:
       D[A] is the device spec

Exit:
     Carry clear: D0 points to the mailbox, (MBOX^) is set
             to the mailbox
     Carry set: Mailbox and/or configuration buffer not
             found (P is the error number)

Calls:      CNFFND  (FNDMB+ also calls SETLP)

Uses:
 Exclusive: C[W],D0,P
 Inclusive: C[W],D0,P

Stk lvls:   1 (CNFFND)(SETLP)
```

7.8.4 FRAMEE - HPIL frame encode.

```
Name:       FRAMEE - Encode an HPIL frame from its mnemonic

Entry address: 6BD8 Hex

Purpose:
     HPIL frame encode (given the ASCII for the frame and a
     value, produce the appropriate 11-bit frame)

Entry:
     C[S] is length of ASCII character string
       C[S] = String length in nibbles - 1
     C[13:0] is the ASCII character string
       The string is right justified.
```

If set P to C[S], C[P:0] is the character string.

A[B] is the value included with the frame (if none, 0)

Exit:
P=0
Carry clear: C[X] is the frame value
B[B] is the mask value for the frame
C[S] is WP length of name
Carry set: Error...not found

Calls: None

Uses:
Inclusive: B[W],C[W],P

Stk lvls: 1 (Internal push)

7.8.5 FRAME+,FRAME- - Returns type of HPIL message.

Name: FRAME+ - Evaluate an HPIL message, return type
Name: FRAME- - Evaluate a message, return type (not 3data)

Entry address: FRAME+ = 07C2 Hex
FRAME- = 07D0 Hex

Purpose:
Parses a frame

Entry:
C[6:0] contains the input frame from GET
ST[3:0] contains the HPIL handshake nibble

FRAME+: C[S] is the status nibble from DIAMOND

Exit:

Frame type in P:	MNEMONIC:
0: ACKNOWLEDGE	(pACK)
1: CURRENT PIL STATE	(pSTATE)
2: DIAGNOSTIC (TEST RESULTS)	(pDIAGR)
3: DIAGNOSTIC (LOCATION CONTENTS)	(pDIAGL)
4: ADDRESS	(pADDR)
5: IFC RECEIVED (NOT SYS CONTROLLER)	(pIFC)
6: ETO RECEIVED	(pEOT)
7: CONVERSATION HALTED (COUNT, NOT L)	(pHALTD)
8: TERMINATOR MATCH	(pTERM)
9: ETE REVEIVED	(pETE)
10: UNRECOGNIZED TYPE	(pUTYPE)

```
          11: DATA/END FRAME                 (pDATA )
          12: COMMAND RECEIVED               (pCMD  )
          13: READY FRAME                    (pRDY  )
          14: IDY FRAME                      (pIDY  )
          15: THREE BYTE DATA TRANSFER       (p3DATA)
      If illegal frame or error, sets carry; else clears it
```

Calls: None

Uses:
 Inclusive: C[S],P (C[S] only for FRAME+)

Stk lvls: 0

7.8.6 GET,GETNE - Get a message from Mailbox.

Name: GET - Get a message from Diamond
Name: GETNE - Get a message without checking error bit

```
Entry address:   GET   = 67E6 Hex
                 GETNE = 67D0 Hex
```

Purpose:
        Utility to read the mailbox message

Entry:
     DO points to the HPIL mailbox

```
Exit:
     Carry clear:
       Contents of mailbox in C[7:0]
       Handshake nibble in ST[3:0]
       Status nibble in C[S]
     Carry set:
       Error (P=error number)
```

Calls: None

Uses:
 Inclusive: C[W],ST[3:0] (P only if error)

Stk lvls: 0

7.8.7 GETD - Get data.

```
Name:        GETD - Get data message
Name:        GETEND - Get EOT message

Entry address: GETD   = 685D Hex
               GETEND = 687A Hex

Purpose:
     Read a data/EOT message from Diamond

Entry:
     Expecting data/EOT from the mailbox
     D0 points to the mailbox

Exit:
     Carry clear:
       Frame in C[X]
       Frame type in C[S]
     Carry set:
       GETD: Not a data frame/aborted/error bit set
       GETEND: Not an EOT frame/aborted/error bit set

Calls:       GET,FRAME+

Uses:
 Exclusive: C
 Inclusive: C,ST[3:0]  (P only if error)

Stk lvls:    1 (GET)(FRAME+)
```

7.8.8 GETDev - Check if the HPIL module is a device.

```
Name:        GETDev - Get device status bit from LOOPST

Entry address: 0BF0 Hex

Purpose:
     Indicate whether the last call to CHKSTS found Diamond
     in device or controller mode

Entry:
     None

Exit:
     LOOPST in ST[3:0]
```

Carry set if device, clear if controller

Calls: None

Uses:
Inclusive: ST[3:0]

Stk lvls: 1 (internal push)

7.8.9 GETERR,GETST - Get Mailbox error/status.

Name: GETST - Get status from Diamond
Name: GETERR - Get error message from Diamond
Name: GETST- - Read status message from mailbox without checking the error bit

Entry address: GETST = 681C Hex
GETERR = 6826 Hex
GETST- = 6833 Hex

Purpose:
Get status/error message from Diamond

Entry:
D0 points to the HPIL mailbox

Exit:
Carry clear: PIL status in C[X], error # in C[3]
P=0
Carry set: Error (# in P,C[0])

Calls: PUTC+N,GETNE,FRAME+

Uses:
Exclusive: C[W], P
Inclusive: C[W],ST[3:0],P

Stk lvls: 1 (PUTC+N)(GETNE)(FRAME+)

7.8.10 GETHSS - Get 2 handshake nibbles from Mailbox.

Name: GETHSS - Get 2 handshake nibbles from Diamond

Entry address: 31CF Hex

Purpose:
Read the two handshake nibbles from Diamond to HP-71 and put into ST[7:0]

Entry:
D0 points to HPIL mailbox

Exit:
The two handshake nibbles from Diamond are in ST[7:0]
Carry clear

Calls: None

Uses:
Inclusive: ST[7:0]

Stk lvls: 0

7.8.11 GETMBX - Set D0 to the HPIL Mailbox address

Name: GETMBX - Get address of mailbox (last FNDMBX)

Entry address: 3BF7 Hex

Purpose:
Get the HPIL mailbox address from RAM and put it in D0

Entry:
Nothing

Exit:
C[A], D0-->Mailbox
Carry clear

Calls: None

Uses:
Inclusive: C[A],D0

Stk lvls: 0
NOTE: Does not alter P!

7.8.12 GETX - Fast data input routine.

Name: GETX - Fast DATA input routine

Entry address: 6745 Hex

Purpose:
    Fast data input routine...read DATA bytes as quickly as possible

Entry:
    D0 points to the mailbox
    Conversation is set up and started

Exit:
    If carry clear:
        P=0: C[B] is a data byte
        P=2: C[5:0] is three byte quantity; C[B] is first!
    If carry set:
        P=0: C[6:0] is message, C[S] is status*2
        P#0: Aborted (P= =eABORT)

Calls: None

Uses:
 Inclusive: C[W],P,ST[3:0]

Stk lvls: 0

7.8.13 GFTYPE - Get frame type from RAM.

Name: GFTYPE - Get frame type from RAM

Entry address: 2E2B Hex

Purpose:
    This routine return the mnemonic of a message in a statement.
    This routine is used by the SEND statement.

Entry:
    D0 points to string of chars (<=7)

Exit:
    A contains the string (A[S] is WP value)
    Carry SET if error

Calls: CONVUC,RANGEA

Uses:
 Exclusive: A[W],C[W],P,D0
 Inclusive: A[W],C[W],P,D0

Stk lvls: 2 (CONVUC)

7.8.14 GLOOP# - Get loop # from RAM (if one present).

Name: GLOOP# - Get loop # from RAM (if one present)

Entry Address: 2DEF Hex

Purpose:
Get loop number from memory

Entry:
D0 points to next token

Exit:
P=0
D0 points to next item on line
C[S] is loop # [0-2]
Carry set if no loop # given

Calls: GTYPRM

Uses.......
Inclusive: A,B,C,D,R0,R1,R2,R3,R4,D0,D1,P,ST[11:0],FUNCxx

Stk lvls: 6 (GTYPRM)

7.8.15 PRMSGA - Print message from C-reg.

Name: PRMSGA - Output message from C (uses A)

Entry Address: 0D4E Hex

Purpose:
Output message from C (ASCII) (use A[W] to store it)

Entry:
C[W] has an ASCII string, C[B] is the first character
Message is terminated by a #00 character
D0 points to mailbox

Exit:
Carry clear: OK, P=0
Carry set: error (P,C[0] are error code)

Calls: PUTD

```
Uses.......
 Inclusive: A[W],C[W],ST[3:0]

Stk lvls:   1 (PUTD)

Algorithm:
     PRMSGA:Copy C[W] to A[W]
     PRMSG1:shift A[W] right twice (next char in A[B] now)
            output the character in C[B]            (PUTD)
            if next character (A[B]) <> #00 then goto PRMSG1
            return
```

7.8.16 PUTARL - Put data from A[W] to Mailbox.

```
Name:       PUTARL - Put data from A[W] (Right to left)
Name:       PUTALR - Put data from A[W] (Left to right)

Entry Address:  PUTARL - 0EBA Hex
                PUTALR - 0ED2 Hex

Purpose:
     Output data from A[W] to the HPIL loop

Entry:
     D0 points to mailbox
     I am talker on loop
     P is a count of bytes to be output from A[W]
     PUTARL outputs bytes starting with A[B]
     PUTALR outputs bytes starting with A[15:14]

Exit:
     Carry clear: P=0, all OK
     Carry set: error (P, C[0] are error code)

Calls:      PUTD

Uses.......
 Exclusive: A[W],C[A],P
 Inclusive: A[W],C[W],P,ST[3:0]

Stk lvls:   1 (PUTD)
```

7.8.17 PUTC - Put a command (4 nibs) to Mailbox.

Name: PUTC+ - Put a command (1 byte) to the mailbox
Name: PUTC - Put a command (2 bytes) to the mailbox

Entry Address: PUTC - 6BB1 Hex
PUTC+ - 6BAD Hex

Purpose:
Put a command (1 or 2 bytes) to the mailbox

Entry:
D0 points to the HPIL mailbox
PUTC+: C[B] contains the command to send (1 byte)
PUTC: C[3:0] contains the command to send (2 bytes)

Exit:
Same as PUTE

Calls: None

Uses.......
Inclusive: C[W],ST[3:0],P

Stk lvls: 0

7.8.18 PUTD - Put a single data byte to the loop.

Name: PUTD - Put a single data byte on the loop

Entry Address: 6B43 Hex

Purpose:
Send a single data byte on the loop (Check NRD first)

Entry:
C[B] contains the data byte
D0 points to the HPIL mailbox

Exit:
Handshake nibble in ST[3:0]
Carry set if error, clear if OK

Calls: None

```
Uses.......
 Inclusive: C[W],ST[3:0]

Stk lvls:   0
```

7.8.19 PUTDX - Put multiple data bytes to Mailbox.

```
Name:       PUTDX - Output multiple data bytes (P is count)

Entry Address:  0EEA Hex

Purpose:
     Output data to the loop: first the contents of C[B],
     then P-1 zero bytes

Entry:
     D0 points to mailbox
     I am talker
     P contains the total number of bytes to send

Exit:
     P=0
     Carry set if error (P is error #)

Calls:      PUTD

Uses.......
 Exclusive: C[A],P
 Inclusive: C[W],P,ST[3:0]

Stk lvls:   1 (PUTD)
```

7.8.20 PUTE - Put long message (6 nibs) to Mailbox.

```
Name:       PUTE - Put extended message (6 nibbles)
Name:       PUTEX - Put extended message (6 nibs + 2 hs)

Entry Address:  PUTE  - 6B55 Hex
                PUTEX - 6B5D Hex

Purpose:
     PUTE:Put extended mailbox message (given full 6 nibs)
     PUTEX:Put a full message, INCLUDING HANDSHAKE!!!!

Entry:
```

PUTE: C[5:0] is message
PUTEX: C[7:0] is message
D0 points to the mailbox

Exit:
Carry clear: OK (P=0 for PUTX)
Carry set: error (P=error #)

Calls: None

Uses.......
Inclusive: C,ST[3:0] (PUTE sets P=0)

Stk lvls: 0

7.8.21 PUTEN - Send message to Mailbox, ignore error.

Name: PUTEN - Put message in C[5:0], don't check error
Name: PUTCN - Put message in C[3:0], don't check error
Name: PUTC+N - Put message in C[B], don't check error

Entry Address: PUTEN - 6B86 Hex
PUTCN - 6B81 Hex
PUTC+N - 6B7D Hex

Purpose:
Put a message without checking for the Diamond error bit (otherwise same as PUTE)

Entry:
D0 points to the HPIL mailbox

PUTEN: Message in C[5:0]
PUTCN: Message in C[3:0]
PUTC+N: Message in C[B]

Exit:
Carry clear:
Handshake nibble in ST[3:0]
Carry set:
P=error #

Calls: None

Uses.......
Exclusive: C[W]
Inclusive: C[W],ST[3:0]

Stk lvls: 0

7.8.22 PUTX - Send 3 bytes of data from C[5:0]

Name: PUTX - Send 3 bytes of data from C[5:0] to loop

Entry Address: 6A97 Hex

Purpose:
Output three bytes from C[5:0] to PIL

Entry:
C[5:0] is the three data bytes (C[B] is first byte)
D0: HPIL mailbox

Exit:
Carry clear: done
Carry set: error (P is error #)

Calls: None

Uses.......
Inclusive: C[W],ST[3:0]

Stk lvls: 0

7.8.23 READIT - Read data bytes from the loop.

Name: READIT,READSU - Read into RAM from loop

Entry Address: READIT - 66DE Hex
READSU - 66D2 Hex

Purpose:
Read data, given a buffer to put it into, and a count of how many bytes to enter

Entry:
D0 points to mailbox
D1 points to the input buffer
A[A] is the number of bytes to read
A[5] is the converstion type for Diamond

READSU: C[5:0] is start message and count

```
     READIT: the conversation is started

Exit:
     Carry clear: D1 points past the last character
                   A[A] is zero
     Carry set:   Error...A[A] is the number of bytes left
                     in the buffer
                   If P= =ePIL, C[6:0], [S] is status msg
                     from Diamond ([S] has been doubled)
                   Else C[W] is undefined

Calls:       PUTE,GETX,FRAME-

Uses.......
 Exclusive: A[5:0],C[W],D1,P
 Inclusive: A[5:0],C[W],D1,P,ST[3:0]

Stk lvls:    1 (FRAME-)(GETX)(PUTE)

Algorithm:
     READSC:Save conversation descriptor in A[5:0]
     READS+:Start the conversation                  (PUTE)
     READIT:If no more data to read (A[A]=0) then RTNCC
            Get a message from Diamond              (GETX)
            If not data, check the message:         (FRAME-)
              If EOT or terminator match, GOTO READS+
                 else error
            (data)
            If P#0 then write out 3 data bytes
              else write out 1 byte
            Increment D1 past data just written
            GOTO READIT
```

7.8.24 SENDIT - Send data from B[W].

```
Name:        SENDIT - Send a 1 or 2 char sequence from B[W].
Name:        SENDI+ - Find mailbox, send a sequence of chars

Entry Address:  SENDIT - 6A24 Hex
                SENDI+ - 6A1E Hex

Purpose:
     Send a sequence of 1 or 2 characters (in B[7:0])
     Number of characters to send in A[A]

Entry:
```

A[A]=count of characters
B[7:0]=sequence (B[B]=first char, B[3:2]=second char,
B[5:4]=first char, B[7:6]=second char)
D0 points to mailbox
ST(=LoopOK) set if abort on 1 ATTN, else clear

Exit:
Carry set if Attn or error, else clear
If carry set and P=0, then ATTN key hit ONCE

Calls: PUTX,PUTD,CK=ATN (SENDI+ also calls GETMBX)

Uses.......
Exclusive: A[A],C[W]
Inclusive: A[A],C[W],ST[3:0]

Stk lvls: 1 (PUTX)(PUTD)(CK=ATN)(GETMBX)

NOTE: This routine can be speeded up SLIGHTLY...see WRITIT documentation)

7.8.25 SETLP - Setup loop number for FNDMBX routine.

Name: SETLP - Set up C[S] for FNDMBX from D[A] info

Entry Address: 3C12 Hex

Purpose:
Given D[A] set up for device search, return the loop # minus one in C[S]

Entry:
D[A] is device info (see START documentation)

Exit:
Carry clear
P=0
Mailbox # in C[S]

Calls: None

Uses.......
Inclusive: C[A],C[S],P

Stk lvls: 0

7.8.26 WRITIT - Output data to loop from RAM.

Name: WRITIT - Write data from RAM to the mailbox

Entry Address: 69AF Hex

Purpose:
Output data to the Diamond, given a buffer of data in RAM and a pointer (D1) to the buffer

Entry:
D0: Diamond mailbox
D1: Data buffer start
A[A]: Number of bytes of data to send from at D1
Loop is addressed, set up for this transfer
ST(=LoopOK) set if should abort on one ATTN, else clear

Exit:
Carry clear:
Transfer complete, D1 points past end of buffer, A[A]="000FF", P unchanged from entry
Carry set: Error - P is the error number, A[A] is the number of data bytes not sent (may be low by up to 3)
(If Attn key hit ONCE, then carry set, P=0)

Calls: PUTX,PUTD,CK=ATN

Uses.......
Exclusive: A[A],C[W],D1
Inclusive: A[A],C[W],D1,ST[3:0]

Stk lvls: 1 (PUTX)(PUTD)(CK=ATN)

NOTE: this routine can be SLIGHTLY speeded up by calling PUTX one statement later (after the CPEX 15)...at the cost of setting P=0 unconditionally

7.9 Parse and decompile routines

DVCSPp - Device specifier parse routine.

FRASPd - Decompile a frame specifier.

FRASPp - Frame spec parse for HPIL frame descriptors.

LOOP#d - Decompile optional loop number.

LOOP#p - Parse optional loop specifier.

NAMEp - Parse a name or device word.

PRNTSd - PRINTER IS decompile routine.

PRNTSp - PRINTER IS parse routine.

7.9.1 DVCSPp - Device spec parse

Name: DVCSPp - Parse a device specifier (: optional)

Entry Address: 79BA Hex

Purpose:
Device spec parse...string expr, *, and [:] OK

Entry:
D1 points to the ASCII character string
D0 points to the location where the tokens go
D[A] is the end of available memory
P=0

Exit:
D0 positioned past last token output by this routine
D1 positioned past last character accepted
Carry clear
P=0
Exits through ERRORP if error

Calls: EOLCK,RESPTR,OUTBYT,CKSTR,BLANK,DVSPp,DVLBp

Uses.......
Inclusive: A,B,C,D[15:5],R0-R3,D0,D1,P,ST[11,10,8,7,3:0],
FUNCD0,PRMCNT[0]

Stk lvls: 5 (CKSTR)(DVSPp)

7.9.2 FRASPd - Decompile a frame specifier.

Name: FRASPd - Decompile a frame spec

Entry Address: 7D5E Hex

Purpose:
Frame spec decompile routine

Entry:
D0 points to the output buffer
D1 points to the input buffer (tokens)
D[A] is the end of available memory
A[B] is the next token (at D1)
P=0

Exit:
A[B] is next token
Carry clear if frame spec found, set if not found
D0,D1 updated to current position

Calls: ?A=CLN,OUT1TK,RANGEA,Outblk

Uses.......
Exclusive: A,C, D1
Inclusive: A,C,D0,D1

Stk lvls: 2 (OUT1TK)(Outblk)

7.9.3 FRASPp - Frame spec parse for HPIL frames.

Name: FRASPp - Parse an HPIL frame specifier

Entry Address: 7769 Hex

Purpose:
Frame spec parse for HPIL frame descriptors

Entry:
A[B] is next character (at D1)
D1 points to the ASCII character string
D0 points to the location where the tokens go
D[A] is the end of available memory
P=0

```
Exit:
     A[B] is next item (at D1)
     If carry set, not valid input (D0,D1 restored)
     If carry clear, output <tCOLON><text string>.
         ST(StrOK) is set if string OK next, clear if not
         ST(EolOK) is set if EOL is OK next, else clear
         ST(ExprOK) is set if expression makes sense next
     D0 positioned past last token output by this routine
     D1 positioned past last character accepted
     P=0

Calls:       UCRANG,OUTBYT,FRAMEE,OUTNBS,<BLANK>

Uses.......
 Inclusive: A,B,C,R0,R1,P

Stk lvls:    2 (UCRANG)(OUTBYT)(FRAMEE)(OUTNBS)
```

7.9.4 LOOP#d - Decompile optional loop number.

```
Name:        LOOP#d - Decompile an optional loop #

Entry Address:  7D3F Hex

Purpose:
     Decompile a loop number, if any. If none present, exit
     with carry set (Leaves next token in A[B])

Entry:
     D1 points to the (optional) loop #
     D0 points to the output buffer
     D[A] is the end of available memory
     A[B] is the next token (at D1)

Exit:
     D0,D1 positioned after the loop #, if found
     A[B] is the next token
     Carry set if no loop #, clear if loop # found

Calls:       EXPDC+,OUT2TC

Uses.......
 Exclusive: A,  C,           D1
 Inclusive: A,B,C,R0,R1,R2,D0,D1,P,ST[0,3,8,10,11]

Stk lvls:    5 (EXPDC+)
```

7.9.5 Loop#p - Parse optional loop specifier.

Name: LOOP#p - Parse an optional HPIL loop specifier

Entry Address: 773C Hex

Purpose:
Parse an optional loop number...if one present, output the tokens for it

Exit:
A[B] is next char, D1 points at next character
If <loop #> found, compiled code generated

Entry:
D1 points to the ASCII character string
D0 points to the location where the tokens go
D[A] is the end of available memory
P=0

Exit:
A[B] is next character (at D1)
D0 positioned past last token output by this routine
D1 positioned past last character accepted
P=0
Carry clear

Calls: SVD0D1,OUTBYT,CKNUM,OUT1TK,RSD0D1,BLANK

Uses.......
Inclusive: A,B,C,D[15:5],R0-R3,D0,D1,P,ST[11,7,3:0],FUNCD0,
PRMCNT[0]

Stk lvls: 5 (CKNUM)

7.9.6 NAMEp - Parse a name or device word.

Name: NAMEpb - Skip leading blanks, parse device word
Name: NAMEp - Parse a device word (C[S] is # chars)

Entry Address: 7A2D Hex

Purpose:
Parse a device word: <letter > {<letter> | <digit >} *n

```
Entry:
     C[S] is max number of characters to accept
     D1 points to the ASCII character string
     D0 points to the location where the tokens go
     D[A] is the end of available memory

Exit:
     First character not used in A[B] (char @ D1)
     Carry set if length exceeded or first char is a digit
     A[S]=0 if no chars, #F if characters
     D0 positioned past last character output by this routine
     D1 positioned past last character accepted
     P=0

Calls:       BLANK,CATC++,OUT1TK

Uses.......
 Inclusive: A[S,B],C[S,B],P,D0,D1,ST[2:1]

Stk lvls:   3 (CATC++)
```

7.9.7 PRNTSd - PRINTER IS decompile routine.

```
Name:        PRNTSd - PRINTER IS decompile routine
Name:        PACKd  - PACK decompile (device spec,OUTELA)

Entry Address:  PRNTSd - 7BD3 Hex
                PACKd  - 7BDF Hex

Purpose:
     Decompile the PRINTER IS/PACK statements

Entry:
     D1 points to tokenized device spec
     D0 points to output buffer
     D[A] is end of available memory, P=0

Exit:
     Exits through OUTELA
     Carry clear, P=0

Calls:       OUT3TC,?A=CLN,PILDC,?A=CMA,OUTCMA,EXPRDC

Uses.......
 Exclusive: A, C
 Inclusive: A,B,C,R0,R1,R2,D0,D1,P,ST[0,3,8,10,11]

Stk lvls:   6 (PILDC)
```

Detail:
    Decompiles 1 or more device specs (separated by commas)

7.9.8 PRNTSp - PRINTER IS parse routine.

Name: PRNTSp - Parse the PRINTER IS statement

Entry Address: 74FD Hex

Purpose:
    Parse the PRINTER IS (and DISPLAY IS) statement

Entry:
    D1 points to the ASCII character string
    D0 points to the location where the tokens go
    D[A] is the end of available memory
    P=0

Exit:
    D0 positioned past the last token output by this routine
    D1 positioned past the last character accepted
    P=0
    Exits through ERRORP if error

Calls: NTOKEN,<DVCPy*>

Uses.......
 Inclusive: A,B,C,D[15:5],R0,R1,R2,D0,D1,P,ST[11,10,8,7,3:0],
 FUNCD0,PRMCNT[0]

Stk lvls: 5 (DVCPy*)

Version 79.10.13 of RUNIT's INDEX program

B

C

D

E

F

G

O

P