# SDS-II

## HP-41C Software Development System For MS-DOS Computers

HP Portable Computer Division

January 30, 1986

SDS-II

## 2. Introduction

The new Software Development System (SDS-II) provides the necessary tools to collect and prepare your 41C programs for translation into an HP-41 ROM (Read-Only Memory) plug-in. Each plug-in consists of 1, 2, or 3 ROMs, each containing 4096 bytes of HP-41 program code and/or microcode.

The process of creating a plug-in ROM consists of three major steps:

- Writing HP-41 programs and saving them on mass storage.
- Reading your HP-41 programs into the SDS-II development system (the "READ41P" process).
- Building a ROM image containing your HP-41 programs and any necessary microcode support functions (the "BUILD" process).

Once the ROM image is built, it can be burned into EPROM boxes for testing, and, when fully tested, can be submitted to the HP custom ROM program for processing into plug-in ROMs. Full details on the procedures and expenses associated with producing custom ROMs are explained in an accompanying brochure.

# 3. Contents of SDS-II

## 3.1 What SDS-II Includes

SDS-II is a software package distributed on one of two possible media:

[1] Two 3.5" single-sided microfloppy MS-DOS disks, or

[2] Two 5.25" double-sided floppy MS-DOS disks in low-density (IBM PC) format.

SDS-II is compatible with all MS-DOS and PC-DOS computers running DOS version 2.0 or higher.

## 3.2 What SDS-II Does Not Include

SDS-II requires additional hardware, some of which is dependent on the choice of host system. Specifically:

[1] An HP-41C/CV/CX and HP-IL module (82160A).

[2] Mass storage for the HP-41 (82161A cassette drive or 9114 disk drive). The 9114 is strongly recommended over the 82161A.

[3] Depending on your configuration, you may require an accessory for communicating with the HP-41 mass storage device (detailed below, in the section on configuration).

[4] ROM emulation hardware.

- If you are using EPROMs for ROM emulation, you need an EPROM programmer, EPROMs, an EPROM eraser, and an EPROM emulator box. As a programmer, SDS-II supports and recommends the Data I/O 21A, a powerful, reasonably priced product that was recently introduced. For emulation, SDS-II supports and recommends the ERAMCO-???? because of its complete emulation of HP ROMs.

- If you are using RAM for ROM emulation, you need a RAM box and the facilities to load it. Recommended is the ERAMCO ES16S, which provides complete emulation of HP ROMs. Use of this box requires the ERAMCO MLDL software, usually distributed in the ERAMCO ESMLDL 1. (Note: EPROM emulation is recommended over RAM emulation, although support is provided for the latter, as explained in the chapter on advanced applications.)

[5] An interface for communicating with the EPROM programmer (typically an asynchronous communications port).

## 4. Comparison With Old SDS

SDS-II is intended as a replacement for and upgrade from the HP-85-based SDS. The system differs substantially from the original SDS in the following ways:

[1] The software runs under the MS-DOS or PC-DOS operating system instead of on the HP-85. The speed improvement is approximately 20x.

[2] The system no longer relies on specialized custom hardware for communications with the HP-41. Programs are read directly from HP-41 mass storage media, and ROM emulation is provided through commercially available EPROM- and RAM-boxes.

[3] SDS-II does not provide special editors. The DEFINE file is a text file that you create using any text editor (EDLIN, WORDSTAR, EMACS, etc.). This replaces the special editors used in the old SDS for creating the list of TODISK files, the XEQ list, the specified order list, and the key assignment list.

[4] SDS-II supports the bank-switching 12K ROM for the HP-41. This is explained in more detail in the chapter on bank-switching.

[5] SDS-II includes a comprehensive set of tools for ROM development, including an assembler, linker, and various related tools and utilities. These are explained in the section on advanced applications.

# 5. Configuring Your SDS-II System

SDS-II requires an MS-DOS or PC-DOS computer with 128K bytes of available memory (after DOS, device drivers, and other resident applications are loaded). This chapter tells you how to configure your computer for two important communications tasks:

- Receiving programs from your HP-41, in which SDS-II reads the HP-41 disk or cassette tape, and
- Controlling the EPROM programmer, in order to program EPROMs for testing your software in an EPROM box. Instructions here are *specific to the Data I/O 21A*. If your system includes a different programmer, the chapter on utilities explains how you can use SDS-II utilities for interfacing with your programmer.

## 5.1 Configuring the HP-150

SDS-II is not installed as an application in PAM (Personal Applications Manager). That is, it is only accessible through the MS-DOS commands. In order to use SDS-II, you must enter the MS-DOS command environment.

### 5.1.1 Receiving From HP-41

If you are using a 9114 with your HP-41, and your HP-150 has a double-sided micro-floppy (3.5") disk drive, you can directly read the HP-41 disk without any extra communications hardware.

Otherwise, your HP-150 can communicate directly with the 9114 or the 82161A through the Extended I/O Accessory (45643A). Installing the accessory consists of two steps:

- Physically installing the accessory card, and
- Installing the HP-IL driver software. To install the software, you must modify the CONFIG.SYS file (in the root directory of the boot disk) to include the directive:

  DEVICE = HPIL150.SYS

  If there is no CONFIG.SYS in your root directory, create one containing the directive. The driver software (HPIL150.SYS) is included on the disk that accompanies the Extended I/O Accessory, and must be copied to the root directory of the boot disk (alternatively, the "DEVICE =" directive can be modified to specify another disk and/or directory).

  Some important details to keep in mind:

- If the CONFIG.SYS file contains a "SHELL =" directive, the "DEVICE =" directive must occur *before* the "SHELL =" directive.

- Some editors (notably EMACS) do not automatically append a trailing <CR><LF> to the last line of a file. The "DEVICE =" directive will not work if it is the last line of a file without the trailing <CR><LF>.

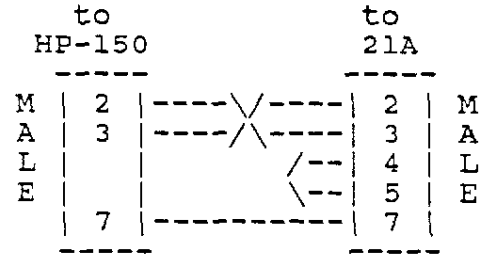
- The HP-IL driver redefines the "PRN:" device to be the first printer on the HP-IL loop. Any output directed to "PRN:" will be sent to that printer. If there is no printer on the loop, the "PRN:" device is not accessible. This is true both in the MS-DOS environment and in the PAM environment.

- The HP-IL loop can support up to eight mass storage devices. Because the HP-150 reserves disk drive ID's "A:" through "L:", mass storage devices on the loop are named "M:" through "T:". For example, if the loop contains a single 9114 disk drive, it is addressed as drive "M:".

Once CONFIG.SYS has been modified, the HP-IL driver will be installed whenever the HP-150 boots up (either from power-up or SHIFT-CTL-RESET). You will now have access to HP-IL devices connected to the accessory card.

### 5.1.2 Controlling the EPROM Programmer

Both the HP-150 and the Data I/O 21A have female RS-232 connectors configured for DTE. Communications between them requires a male-male RS-232 connector reversing the signals from pins 2 and 3. In addition, the 21A requires that pins 4 and 5 be tied together. The specific connections are:




## 5.2 Configuring the PORTABLE Series

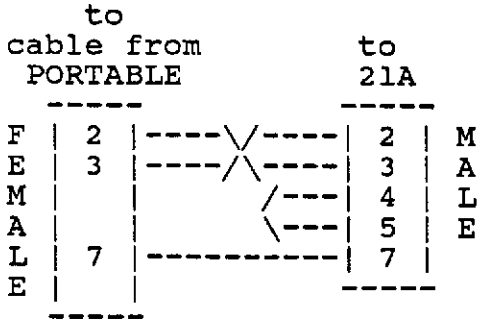
SDS-II is not installed as an application in PAM (Personal Applications Manager). That is, it is only accessible through the MS-DOS commands. In order to use SDS-II, you must enter the MS-DOS command environment.

### 5.2.1 Receiving From HP-41

The built-in HP-IL on the PORTABLE series is capable of direct communications with the 9114 and the 82161A. No additional hardware is needed. You must enter the System Config template (invoked as a softkey from PAM) and set the "External disk drives" entry to reflect the presence of one or more mass storage devices on HP-IL.

### 5.2.2 Controlling the EPROM Programmer

The optional RS-232 cable for the PORTABLE series is terminated with a male DTE connector. Communications with the Data I/O 21A requires a female-male RS-232 connector reversing the signals from pins 2 and 3. In addition, the 21A requires that pins 4 and 5 be tied together on its side. The specific connections are:




## 5.3 Configuring IBM, Vectra, and Compatibles

### 5.3.1 Receiving From HP-41

Your computer can communicate directly with the HP-9114 disk drive or the 82161A cassette drive through the HP-IL Interface card (82973A). Installing the accessory consists of two steps:

- Physically installing the accessory card, and
- Installing the HP-IL driver software. To install the software, you must modify the CONFIG.SYS file (in the root directory of the boot disk) to include the directive:

  DEVICE = HPIL.SYS

  If there is no CONFIG.SYS in your root directory, create one containing the directive. The driver software (HPIL.SYS) is included on the disk that accompanies the interface card, and must be copied to the root directory of the boot disk (alternatively, the "DEVICE =" directive can be modified to specify another disk and/or directory).

Some important details to keep in mind:

- If the CONFIG.SYS file contains a "SHELL =" directive, the "DEVICE =" directive must occur *before* the "SHELL =" directive.

- Some editors (notably EMACS) do not automatically append a trailing <CR><LF> to the last line of a file. The "DEVICE =" directive will not work if it is the last line of a file without the trailing <CR><LF>.

- The HP-IL loop can support up to eight mass storage devices. The exact drive designator will depend on system configuration. For example, a typical IBM PC with two floppy drives will address its HP-IL disks as "C:" (first drive on the loop) through "J:" (eighth drive on the loop). A typical IBM PC/AT with a hard disk drive at "C:" will address disks on the loop starting with "D:".

Once CONFIG.SYS has been modified, the HP-IL driver will be installed whenever the computer boots up. You will now have access to HP-IL devices connected to the HP-IL interface card.

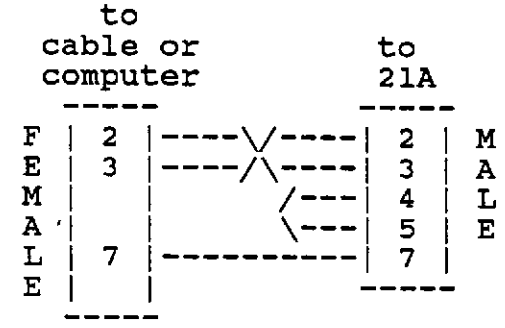
### 5.3.2 Controlling the EPROM Programmer

The IBM, Vectra and compatibles offer two different types of RS-232 interfaces:

- A built-in 25-pin interface, and

- A 9-pin "D-shell" interface, which requires a cable.

Both the cable and the built-in interface terminate with a male RS-232 connector configured for DTE. Communications with the Data I/O 21A requires a female-male RS-232 connector reversing the signals from pins 2 and 3. In addition, the 21A requires that pins 4 and 5 be tied together on its side. The specific connections are:



## 6. Step 1: Writing HP-41 Software

The software to be contained in the ROM will consist of various HP-41 programs written by you and microcode support programs obtained from the microcode library (explained in a separate chapter). As you write each piece of software, save it on the mass storage device using the WRTP command.

At this stage, the program is not in ROM form. Before you have a ROM image, SDS-II will pack the program, compile GOTO's for fast execution, and convert all global labels into ROM entries. That is, each global label will be associated with an XROM number, *and* XEQs referencing those global labels will be compiled into XROMs.

For now, however, your programs will contain alpha XEQs for referencing each other (i.e., when one of your programs calls another) and for keywords obtained from the microcode library.

## 7. Step 2: Reading HP-41 Software into SDS-II

Each program created on your HP-41 must be read into SDS-II using a program called READ41P. This program is included on the SDS-II distribution disk #1, and is invoked from the MS-DOS environment as:

READ41P <disk>:<programname> <filename>

READ41P will read the program from the mass storage device containing the HP-41 programs, analyze it, report any errors, print an informational listing, and create a file on the MS-DOS machine. If the programname contains blanks, you must replace those with a period (".") in the invocation. If the programname contains HP-41 special characters (such as not-equal, angle, or sigma), use the substitute characters explained in the appendix on special characters.

Since READ41P will usually generate more output than will fit on one screen (and faster than most people can read), it may be desirable to redirect its output to a file or a printer. The second example below demonstrates this.

EXAMPLE: The invocation

READ41P M:XYZ XYZ

will read the program "XYZ" from the mass storage medium "M:", process it, and produce a READ41P file called "XYZ.41T" in the current directory on the current disk.

EXAMPLE: The invocation:

READ41P B:A.B C:AB >PRN

will read program "A B" from the disk in drive "B:", process it, and produce a READ41P file called "AB.41T" in the current directory on disk "C:". Output from the READ41P program is directed to the computer's PRN device.

The READ41P processing detects several error or potential error conditions, and reports on them:

NOTICES A "NOTICE" is not an error, merely a warning that the program contains an XROM reference. This may be intentional (for example, use of an HP-IL function in an Advanced I/O ROM) or unintentional. The presence of this XROM reference in the final ROM will *require* that the referenced ROM be plugged in for your program to function properly.

ERRORS An error will be generated under the following conditions:

- The program being read contains multiple occurrences of a global label.
- The program contains an unresolved reference to a local label.
- A global label, alpha XEQ or alpha GTO contains an illegal character.
- The program contains more than 64 labels.

READ41P will not generate an output file if any errors are found.

In addition to errors and notices, READ41P prints out an informational listing giving all global and local labels. The local label list includes information on how many times a local label is used and how many references appear to that label.

When all of your HP-41 programs have been collected in READ41P files, you can proceed to step 3 (BUILD) to assemble them into a ROM image.

*NOTE*

Some important details:

- The HP-41 program name in the READ41P invocation is case-sensitive. These two commands are *not* equivalent:

```
READ41P M:ABC ABC
READ41P M:abc ABC
```

  The name of the output file, however, is case-insensitive, since MS-DOS only supports uppercase filenames.

- The choice of MS-DOS filename for your READ41P file is entirely up to you. Good programming practice suggests that it have the same name as the HP-41 file, but this is not always possible (either because of upper/lower case differences, name conflicts, or special characters). When you do not use the same name, be sure to choose a name that suggests what the file contains. The choice of READ41P file name has absolutely no effect on the contents of the final ROM image.

- Some error and warning conditions cannot be detected by READ41P, but are noted in BUILD:
  - Local GTO's that are too distant to be compiled cannot be discovered until BUILD has packed the alpha XEQs into XROM references. This situation is explained more fully in the chapter on BUILD.
  - Multiple use of global labels in different programs cannot be detected until the BUILD phase.
  - XROM references (as pointed out in a NOTICE) to the ROM being built are illegal (for example, an occurrence of XROM 21,xx when you are BUILDing a ROM with an ID of 21). This cannot be detected until BUILD, when the ROM ID is assigned.
- The mass storage medium used by the HP-41 is in LIF format, not MS-DOS. Any attempt to access it as an MS-DOS medium (such as performing a DIR) will fail. Likewise, it is not possible to put a READ41P file on the LIF medium (e.g., READ41P D:PRG D:ABC).

# 8. Step 3: BUILDing the ROM Image

Once all of the READ41P files have been gathered, the ROM image can be generated. SDS-II will allow you to build a ROM image for a 4K, 8K, or 12K ROM plug-in (using 12K requires the use of bank-switching, explained in another chapter).

The following command causes a ROM image to be built:

BUILD <define-file-name> <ROM-file-name>

Following commands in the DEFINE file, BUILD collects the READ41P files (and MICROCODE files, explained below) together into a ROM image.

BUILD does its work in two passes. In the first pass, it reads all of the specified READ41P and MICROCODE files, copies them to a temporary working file, and collects all of the global labels. In the second pass, it does the "dirty work" of compiling label references, converting alpha-XEQ's to XROM's, and so on.

## 8.1 Two Types of .41T Files

So far this document has dealt with READ41P files, which are read from your HP-41 mass storage and stored on the host computer system with an extension of ".41T".

There is another type of ".41T" file that can be specified in the DEFINE file: MICROCODE. A MICROCODE file allows you to add assembly-language programs to your ROM. Use of MICROCODE files will be fully explained in another chapter; this chapter will restrict its discussion and examples to usercode.

## 8.2 The ROM Image File

Build will create 1, 2 or 3 ROM image files, depending on whether you are creating a 4K, 8K or 12K ROM. The files will be named with the first seven characters of <ROM-file-name> appended by the ROM sequence number (that is, 0, 1, or 2). The filename extension will be "41R". This file is ready to be programmed into EPROMs for testing (explained in another chapter).

## 8.3 The DEFINE File

The DEFINE file contains all of the instructions needed to assemble the ROM image. The define file is created using any text editor (such as EDLIN, WORDSTAR, EMACS, etc.). For each 4K ROM (there are 1, 2 or 3), the DEFINE file contains several parts:

- A ROM# directive with optional ROM header and optional privacy specifier. This is followed by a list of READ41P and MICROCODE

files.

- An optional ORDER directive, specifying how the global labels are to be ordered within the ROM catalog. This is sometimes followed by a list of labels and headers.

- An optional XEQ directive, used to specify any labels for which alpha XEQs will *not* be converted into XROMs. This is followed by a list of labels.

- An optional KEYS directive, used to specify key assignments to be set up by the ROM. This is followed by a list of key assignments.

- Optional comments anywhere within the DEFINE file.

All directives and comments are preceded by the "&" character. Following is a detailed explanation of each directive and its section of the BUILD file.

### 8.3.1 ROM# Directive

For each 4K ROM, this line specifies what ROM# is to be associated with the ROM. It is followed by an optional HEADER specification, which allows you to specify a CATalog header for the ROM, and an optional PRIVACY specification. For example:

```
&ROM# = 31, HEADER=CUSTOM.ROM
```

will assign ROM# 31 to this 4K ROM, and the catalog header "CUSTOM ROM". Use of the "." as a placeholder for a space is explained in the appendix on special characters. The first function in this ROM will be XROM 31,1; the second will be XROM 31,2 and so on. The directive:

```
&ROM# = 31, HEADER=MY.ROM, PRIVATE
```

will assign ROM #31 to this ROM with the catalog header "MY ROM". In addition, all programs in the ROM will be PRIVATE, preventing the user from viewing the contents. If your plug-in consists of multiple ROMs, PRIVATE must be specified in each &ROM# directive.

The following rules apply to this directive:

- The "&ROM#" must begin in the first character position on the line.

- Legal ROM ID's are 1-31. An ID of zero can be used only if the ROM contains no labels or headers -- that might occur if the ROM only contains MICROCODE files that do not define any keywords.

- The HEADER is optional. If no header is specified, the ROM will not have a catalog header (making it much harder for the user to find).

- If a HEADER is specified, maximum length is 11 characters. Special characters can be specified using the substitutes discussed in the appendix on special characters.

This directive is followed by a list of READ41P and MICROCODE files to be included in the ROM. BUILD will automatically append the "41T" file extension to the filenames. Specification of these files follows the usual MS-DOS rules of finding files: you can specify just the file name (if the file is in the current directory on the current disk) or a path (and disk drive designator, if appropriate).

### 8.3.2 ORDER Directive

This directive allows you to specify the order in which your global labels will appear in the ROM catalog. If you do not include this directive, the labels will appear in the order in which they are encountered while reading the READ41P files. If this directive is included, it takes the following forms:

```
&ORDER = E
```

to specify that the labels are to appear in the order encountered (the default).

```
&ORDER = A
```

to specify that the labels are to appear in alphabetical order. Special characters (sigma, angle, and not-equal) are sorted according to their internal HP-41 representation: sigma as ASCII 126, angle as ASCII 13, not-equal as ASCII 29.

```
&ORDER = S
```

to specify that the labels are to appear in a specified order. If (and only if) this last form is specified, the directive is followed by the list of labels, one per line, in the order in which they are to appear. In addition, this ORDER option allows something not allowed with the other options: specifying additional CATalog headers. That is, you can specify a header of up to 11 characters by prefixing it with a tilde ("~"). See the examples below for an illustration.

### 8.3.3 **XEQ Directive**

Normally, all alpha XEQ's that refer to labels within your ROM are compiled into XROMs. This saves space in the ROM and execution speed when the program is run. However, an XROM behaves differently from an XEQ. An XEQ command will first search user memory and then search all ROMs to find the named program; an XROM will always execute the program out of the ROM.

Sometimes it is desirable to prevent an alpha XEQ from compiling into an XROM. For example, you may want to allow the user to place a program in memory that overrides a function in the ROM. The XEQ directive allows you to specify that certain alpha XEQs not be compiled into XROM references.

The form of the XEQ directive is:

&XEQ

followed by a list of labels. Any alpha XEQ that refers to any of the specified labels will not be compiled into an XROM.

Alternatively, specifying:

&XEQ ALL

will prevent all alpha-XEQs from being compiled.

### 8.3.4 **KEYS Directive**

It is possible to specify that the calculator automatically assign certain keys on power-up. The directive:

&KEYS

can be followed by a list of keys to be automatically assigned by the ROM. Each item in the list is of three possible forms:

[1] Assigning an HP-41 function to a key:

<function-name> <keycode>

[2] Assigning an XROM function to a key (for example, a card reader function):

XROM <ROM-ID> <function-number> <keycode>

[3] Assigning a function from the ROMs being built to a key:

<function-name> <keycode>

Some important things to keep in mind about key assignments:

- The automatic key assignment occurs whenever the machine is turned on or memory is lost.
- The <keycode> is the same keycode displayed by the ASN function. A map of keycodes in shown in the appendices.
- Functions will automatically be assigned only to keys that do not have current assignments. If a key is currently assigned, this will not override that assignment.
- Specifying automatic key assignments requires the inclusion in the ROM of MICROCODE files that take up additional space. This is explained in the chapter on the microcode library.
- The XROM option can only be used to assign ROM numbers *not* in the ROMs being built. For example, if you are building a ROM with an ID of 21, you cannot assign an XROM 21,xx to a key. To assign functions in the XROM being built, use the function name.
- If you are building more than one ROM, all key assignments should be performed in the first ROM. It wastes space to include key assignments in more than one ROM, and assignments in the second ROM might be overriden by assignments in the first ROM.

### 8.3.5 Comments

Comments may be included anywhere within the DEFINE file. Their format is:

```
&& <comment>
```

## 8.4 Example of DEFINE File

The following example illustrates the various sections of the DEFINE file. Consider an 8K ROM to be built of three programs. All three programs were written on the HP-41 and read into SDS-II using the READ41P utility. The first program contains the following labels (these labels are made up; any resemblance to real HP-41 programs living or dead is purely coincidental):

```
"MAIN"
"S1"
"S2"
"S3"
"PRINT"
"RESET"
```

The second program contains the following labels:

```
"PROG2"
"FIXUP"
```

The third program contains the following labels:

```
"EDITOR"
"ADDLINE"
"EDTLINE"
"PACKFIL"
"PURGFIL"
"TIMEOUT"
"CLRFILE"
"RMVLINE"
```

The first program was read (by READ41P) into a file named MAIN.41T, the second program into PROG2.41T, the third into EDITOR.41T. The first two programs are to go into the first ROM, the third program into the second ROM. We wish to assign some keys and, for the second ROM, specify a CATalog order for the functions. In addition, we want XEQ "TIMEOUT" commands *not* to be compiled into XROMs (allowing the user to override "TIMEOUT" with his own program). The DEFINE file (with some comments added for clarity):

```
&ROM#=21,HEADER=--UTILITIES,PRIVATE
&& READ41P files in first ROM:
MAIN
PROG2
&KEYS
&& assign XROM "EDITOR" to sigma+ key
EDITOR 11
&& assign XROM "PRINT" to LN key
PRINT 15
&& assign mainframe FACT function to SIN key
FACT 23
&& assign mainframe E^X-1 function to f-SIN key
E^X-1 -23
&& assign XROM 1,5 to ENTER^ key
XROM 1 5 41
&ROM#=31,HEADER=--MY.EDITOR,PRIVATE
&& READ41P files in second ROM
EDITOR
&ORDER=S
EDITOR
~--FILE.CMDS
PACKFIL
PURGFIL
CLRFILE
~--LINE.CMDS
ADDLINE
```

```
EDTLINE
RMVLINE
~--TIMEOUT
TIMEOUT
&XEQ
TIMEOUT
```

The resulting catalog will be:

```
--UTILITIES
"MAIN"
"S1"
"S2"
"S3"
"PRINT"
"RESET"
"PROG2"
"FIXUP"
--MY EDITOR
"EDITOR"
--FILE CMDS
"PACKFIL"
"PURGFIL"
"CLRFILE"
--LINE CMDS
"ADDLINE"
"EDTLINE"
"RMVLINE"
--TIMEOUT
"TIMEOUT"
```

The catalog headers (all of which are prefixed with "--" in this example) serve to conceptually separate the sections of the ROM. While all catalog entries appear during a CAT 2 operation on the HP-41C and CV, only the catalog headers appear on the CX (as explained in the HP-41CX owners manual).

## 8.5 BUILD Errors

BUILD detects three levels of exceptional conditions:

ERRORS — Serious problems that must be corrected before the ROM can be built.

WARNINGS — Conditions that do not prevent ROM building, but which *may* be errors. All warnings should be investigated to insure that you have not introduced an inadvertent error.

NOTICES — Less serious than a warning, but a condition to be noted. All notices should be investigated to insure that you have not introduced an inadvertent error.

Following is a summary of exceptions that can occur during BUILD. When appropriate, the error message will indicate which line of the DEFINE file caused the offending error.

### 8.5.1 Errors in DEFINE File

**ERROR: cannot open READ41P file <filename>**
indicates that a file was specified that could not be found.

**WARNING: Header truncated to 11 chars**
A header longer than 11 characters was specified.

**NOTICE: Header has non-std chars**
A header (specified either in the "HEADER=" directive or in an "&ORDER=S" list) contains characters that are not legal in program names.

**ERROR: Duplicate ROM ID**
A "&ROM#" directive has specified the same ROM ID for more than one ROM.

**ERROR: Expected <something>**
indicates that something unexpected was encountered in the DEFINE file. The message will indicate on what line the error occurred. One of the <something>'s is end-of-file, which is expected if a "&ROM#" directive is encountered after three "&ROM#" directives have already been processed -- BUILD can define at most three ROMs.

### 8.5.2 Errors in READ41P Files

**ERROR: READ41P file is not recognizable**
indicates that the a READ41P file is not recognizable either as READ41P or MICROCODE.

**ERROR: Unexpected EOF in READ41P file**
**ERROR: READ41P file is corrupt**
**ERROR: Address not found for label "<label>"**
**ERROR: Unexpected global label on program line #<line#>**
all indicate that the READ41P file is corrupt or contains information that is internally inconsistent.

### 8.5.3 Key Definition Errors

**ERROR: Illegal key definition for XROM <xx>,<xx>**
A key definition was attempted for a ROM ID that is being built. For example, XROM 21,xx was assigned to a key while ROM ID 21 is one of the ROMs being built.

**ERROR: Cannot assign key to <function>; function not found**
A function specified in the key assignment list was not found either in the ROMs being built or in the HP-41 function list.

**NOTICE: ROM label "<label>" overrides HP-41 function for key assignment**
A key assignment was made to a ROM function that has the same name as an HP-41 mainframe function.

**KEY ASSIGN ERROR: Bad ROM number**
A ROM number was specified for an XROM key assignment that was not in the range from 1 to 31.

**KEY ASSIGN ERROR: Bad function number**
A function number was specified for an XROM key assignment that was not in the range from 0 to 63.

**KEY ASSIGN ERROR: Bad keycode**
An illegal keycode was specified for a key assignment. See the appendix on HP-41 keycodes for a map of legal keycodes.

**KEY ASSIGN ERROR: Illegal chars in label**
A function label in a key assignment line contains illegal characters.

**KEY ASSIGN ERROR: Multiple assignment to same key**
An attempt has been made to assign more than one function to the same key. This error will not occur if the multiple assignment occurs in two different ROMs (although, as explained above, all definitions should be performed in the first ROM).

### 8.5.4 Out-of-room Errors

The following errors can occur if there is not enough room in the ROM to hold all of the READ41P and MICROCODE files and the ROM overhead:

**ERROR: Not enough room for key assignment table**
**ERROR: ROM address space overflow**
**ERROR: Not enough space for MCODE**

### 8.5.5 ROM ID = 0

**ERROR: &ORDER=S not allowed with ROM ID = 0**
An "&ORDER=S" directive is not valid if the ROM ID specified in the "&ROM#" directive is zero.

**ERROR: Labels not allowed when ROM ID = 0**
READ41P and MICROCODE files containing any function labels are not allowed if the ROM ID is zero.

**ERROR: HEADER not allowed if ROM ID = 0**
A HEADER specification is not allowed in the "&ROM#" directive if the ROM ID is zero.

### 8.5.6 Specify Errors

The following errors can occur if you use the "&ORDER=S" directive:

**SPECIFY ERROR: Following labels not specified:**
Not all labels in the ROM were specified in the list.

**SPECIFY ERROR: Label "<label>" does not exist in this ROM**
A label was specified that does not exist in this ROM.

**SPECIFY ERROR: Label "<label>" already specified**
A label was specified more than once.

**SPECIFY ERROR: Illegal chars in label**
A label was specified that contained illegal characters.

**SPECIFY ERROR: Too many labels in ROM**
A header added in the specify list causes the number of labels + headers in the ROM to exceed 64.

### 8.5.7 XEQ Errors

The following error can occur if you use the "&XEQ" directive:

**XEQ ERROR: Label "<label>" does not exist in this ROM**
A label was specified that does not exist.

**XEQ ERROR: Illegal chars in label**
A label was specified that contained illegal characters.

### 8.5.8 Label ERRORS

The following errors relate to the function names used in the ROM:

**ERROR: Too many labels in ROM**
A ROM contains more than 64 labels + headers.

**ERROR: Duplicate label in this ROM**
A label occurs more than once in this ROM.

**ERROR: Duplicate label in previous ROM**
A label in ROM 2 or 3 also occurs in an earlier ROM.

### 8.5.9 Errors in HP-41 Program

**WARNING: Unresolved XEQ "<label>" on program line #<line#>**
An alpha-XEQ references a label that does not occur in the ROMs being built. The reference will not be compiled into an XROM. (When the statement is executed, it will search main memory and all ROMs to find the label.)

**WARNING: Unresolved GTO "<label>" on program line #<line#>**
An alpha-GTO references a label that does not occur in the ROMs being built. (When the statement is executed, it will search main memory and all ROMs to find the label.)

**WARNING: Label "<label>" conflicts with HP-41 mainframe keyword**
A label is used in the ROM that conflicts with an HP-41 mainframe keyword.

**NOTICE: GTO <label> on program line #<line#> jumps >127 bytes, not compiled**
A short-form GTO (GTO 00 through GTO 14) references a label that is more than 127 bytes away. The GTO will not be compiled, resulting in slower execution speed.

**NOTICE: XROM <xx>,<xx> on program line #<line#>**
The program contains an XROM statement. Execution of this statement will *require* that the corresponding ROM be plugged in.

**ERROR: Unresolved GTO/XEQ <label> at program line #<line#>**
A local GTO or XEQ references a label which does not exist. Since this situation is also trapped in READ41P, this error should never occur.

**ERROR: Illegal XROM <xx>,<xx> on program line #<line#>**
The program contains an XROM statement that references a ROM being built.

### 8.5.10 Microcode Errors

The following errors can occur if any MICROCODE files are included in the ROMs being built:

**ERROR: Unresolved reference(s) to <microcode-label>**
A MICROCODE file contains an unresolved reference to a label. This can occur if a MICROCODE file (such as ALENG) is included but the files it depends on (such as ALEN and BIND) are not. The chapter on the MICROCODE library specifies the dependencies that must be satisfied.

**ERROR: Reference to <microcode-label> out of range**
**ERROR: Internal reference out of range: address <hex-address>**
These errors will not occur with the MICROCODE library provided with SDS-II, but could occur with an independently-developed MICROCODE file.

**ERROR: MICROCODE label <microcode-label> multiply defined**
A global label at the microcode level occurs more than once. The message will list the offending modules. This error will only occur if a multiply-defined label is actually referenced.

**WARNING: ROM label <microcode-label> overrides HP-41 mainframe label**
A global label at the microcode label conflicts with a label in the HP-41 mainframe.

## 8.6 BUILD FATAL Errors

Certain conditions may cause BUILD to fail with a fatal error, immediately halting execution before completion of the current pass. These errors are generally related to the condition of the temporary (intermediate) files used by BUILD, and can usually be attributed to one of the following conditions:

- Default disk is write-protected, preventing BUILD from creating its temporary files. (BUILD creates its temporary files, UCODE.TMP and MCODE.TMP, in the current directory of the default disk, regardless of where the actual ROM image files are being created).
- Default disk is out of disk space or directory space to hold the temporary files.
- In some cases, corrupted ".41T" files can cause BUILD to fail with a "temp file is corrupt" message.

# 9. Microcode Library

This chapter contains important information if you are

- using files from the microcode library,
- creating a PRIVATE ROM,
- utilizing the automatic key definition capability, or
- writing your own microcode utilities.

In addition to collecting usercode programs into a ROM, SDS-II can collect microcode. Microcode files add two capabilities to the HP-41:

- **Definition of new keywords.** A microcode file can add a new function to the HP-41.
- **Special interrupt processing.** A microcode file can execute special processing at power-on, power-off, coldstart, and several other times. The automatic assignment of keys is an example of a microcode file that does special interrupt processing.

The purpose of this chapter is to describe the microcode library, which is included with SDS-II on disk #1. Privacy and key processing are special cases of microcode files. Information on creating your own microcode files is contained in the chapter on advanced applications.

Microcode files fall into three categories:

Type 2 Routines which can be executed from the HP-41 keyboard.

Type 1 Routines called only by other microcode routines.

Type 0 System microcode routines.

## 9.1 Type 2 MICROCODE Files

Most of the type 2 MICROCODE files implement popular functions that are available in the extended functions ROM and are already built into the HP-41CX. By including these functions in your ROM, however, you make them available for your application regardless of what machine it is plugged into.

**WARNING**

When using one of these functions, it is important that your program contain an alpha-XEQ, not an XROM. For example, if using the ALENG function in your application, your program must contain XEQ "ALENG", *not* ALENG. To insure that this happens, place the labels of the microcode functions you will use somewhere in the HP-41 program memory (not in the programs under development!). This will ensure that the HP-41 compiles them as alpha-XEQ's and not as references to the extended functions ROM.

Some of these type 2 microcode functions require type 1 or type 0 microcode functions. A table of these dependencies occurs at the end of the chapter.

### 9.1.1 AIP (ALPHA Integer Part)

*Not* from the Extended Functions ROM.

This function appends the integer part of the X-register to the ALPHA register. It ignores the fractional part and the sign, and is useful for constructing prompts.

### 9.1.2 ALENG (ALPHA Length)

This function exists in the Extended Functions ROM.

ALENG returns the number of characters in the ALPHA register to the X-register.

### 9.1.3 ANUM (ALPHA Number)

This function exists in the Extended Functions ROM.

ANUM scans the ALPHA register for an alpha-formatted number. If a number is found, its value is recalled to the X-register and user flag 22 is set. If no number is found, the X-register and flag 22 are unchanged.

The digits in the ALPHA register can represent values in any display format. Number separators and radix marks are interpreted according to calculator flags 28 and 29. For example, if the ALPHA register contains the string "PRICE: $1234.50", executing ANUM returns the following results, depending on the status of flags 28 and 29 (using '.' radix for consistency):

| Flag 28 | Flag 29 | Number Returned |
|---|---|---|
| set | set | 1234.5 |
| set | clear | 1234.5 |
| clear | set | 123450 |
| clear | clear | 1234 |

If the digits in the ALPHA register are preceded by a minus sign, a negative number will be placed in the X-register when ANUM is executed.

### 9.1.4 AROT (ALPHA Rotate)

This function exists in the Extended Functions ROM.

AROT rotates the contents of the ALPHA register by the number of characters in the X-register to the left (if the X-register is positive) or to the right (if the number is negative).

### 9.1.5 ATOX (ALPHA to X)

This function exists in the Extended Functions ROM.

ATOX shifts the leftmost character out of the ALPHA register and places its character code in the X-register. If the ALPHA register is empty, zero is placed in the X-register.

### 9.1.6 CLKEYS (Clear Keys)

This function exists in the Extended Functions ROM.

CLKEYS clears all USER mode key assignments.

### 9.1.7 ENROM1 (Enable ROM 1)

*Not* from the Extended Functions ROM.

This function is used to enable ROM 1 when a 12K (bank-switching) ROM is being used. It should *only* be placed in the first ROM of a three-rom plug-in. For more information, see the chapter on bank-switching.

### 9.1.8 ENROM2 (Enable ROM 2)

*Not* from the Extended Functions ROM.

This function is used to enable ROM 2 when a 12K (bank-switching) ROM is being used. It should *only* be placed in the first ROM of a three-rom plug-in. For more information, see the chapter on bank-

switching.

### 9.1.9 GETKEY (Get Key)

This function exists in the Extended Functions ROM.

When a program executes GETKEY, execution halts until a key is pressed or an interval of approximately ten seconds elapses. If a key is pressed, its keycode is placed in the X-register. If no key is pressed, a zero is placed in the X-register at the end of the timed interval.

GETKEY responds to the first key pressed, so there can be no shifted responses to GETKEY. If you press the gold key during a GETKEY pause, its keycode (31) is placed in the X-register.

GETKEY enables you to branch to a subroutine on the basis of an entry from the keyboard, even when the key pressed is not a digit key.

### 9.1.10 PASN (Programmable Assign)

This function exists in the Extended Functions ROM.

PASN enables you to assign functions or programs to a key location. However, PASN requires you to enter the keycode for the key which you wish to assign the function or program. PASN is executed after placing the keycode in the X-register. The function or program name is placed in the ALPHA register. To assign BEEP to the [COS] key, place 24 in the X-register and "BEEP" in the ALPHA register, then execute PASN.

### 9.1.11 PCLPS (Programmable Clear Programs)

This function exists in the Extended Functions ROM.

PCLPS clears one or more of the programs in main memory. All programs beginning with the one named in the ALPHA register (or the current program if the ALPHA register is clear) and continuing to the end of program memory are cleared. If a running program names itself (or clears the ALPHA register) and executes PCLPS, that program and all following it will be cleared and program execution will terminate.

### 9.1.12 POSA (Position in ALPHA)

This function exists in the Extended Functions ROM.

POSA scans the ALPHA register for the ALPHA character or string specified in the X-register. There are two ways to specify the character or string. You can enter the character code for a single

character, or you can enter an actual character or string of characters (up to 6 characters) using ASTO. If the specified character or string is found in the ALPHA register, the character position of the character (or the position of the leftmost character in the string) is returned in the X-register.

Character positions are counted from left to right, starting with position zero. If the specified string occurs more than once in the ALPHA register, only the position of the first occurrence is returned. If the target string is not found in the ALPHA register, negative one is returned.

### 9.1.13 PSIZE (Programmable Size)

This function exists in the Extended Functions ROM.

PSIZE works like the SIZE function provided with the calculator except that it can be executed from within a program. It makes it possible for a running program to reallocate the registers in main memory as required. To use: place the number of data storage registers desired into the X-register and execute PSIZE.

### 9.1.14 RCLSTFLG (Recall/Store Flags)

These functions exist in the Extended Functions ROM.

This file provides two functions: RCLFLAG and STOFLAG.

RCLFLAG recalls the status of flags 00 through 43 to the X-register as ALPHA data. The contents of the X-register can then be stored for later use. When RCLFLAG is executed, the display will not be intelligible.

If the flag status from a previously executed RCLFLAG is placed in the X-register, executing STOFLAG restores calculator flags 00 through 43.

If you want to restore only some of the flags, place the flag status in the Y-register and a number in the form (bb.ee) in the X-register. Executing STOFLAG will then restore flags starting with (bb) and ending with (ee).

### 9.1.15 REGMVSWP (Register Move/Swap)

These functions exist in the Extended Functions ROM.

This file provides two functions: REGMOVE and REGSWAP.

REGMOVE sss.dddnnn copies a block of registers, specified by (nnn), beginning at register (sss), to a block of the same length, beginning at register (ddd). Any data that was already in the

destination block is lost. For example, to move ten registers of data from registers 2-11 to registers 20-29, place 2.020010 in the X-register and execute REGMOVE.

REGSWAP sss.dddnnn exchanges the contents of a block of (nnn) registers beginning at register (sss) with the contents of a block of the same length beginning at register (ddd). Executing REGSWAP with 2.020010 in the X-register would exchange registers 2-11 with registers 20-29.

### 9.1.16 SIZE (Determine Current SIZE)

This function exists in the Extended Functions ROM.

The MICROCODE file SIZE provides the SIZE? function. SIZE? places the number of registers currently allocated to data storage into the X-register.

SIZE? can be used within a program to inhibit execution of PSIZE when a memory reallocation is not required:

```
01 LBL ABC
02 SIZE?    The number of data storage registers presently
            allocated is placed in the X-register.
03 nn       The number of registers this program needs. The
            results of the previous step are now in the Y-register.
04 X>Y?     Is the number of storage registers required by the
            program (X-register) greater than the number presently
            allocated (Y-register)?
05 PSIZE    If so, this step is executed. If not, this step is
            skipped.
```

### 9.1.17 XTOA (X to ALPHA)

This function exists in the Extended Functions ROM.

XTOA, when executed with a character code in the X-register, appends the character represented by the character code to the right-hand end of the string in the ALPHA register. XTOA may be executed with any number from 0 to 255 in the X-register. The null byte, which corresponds to the decimal value 0, has a special meaning in the ALPHA register. Because of this, under some circumstances you cannot retrieve a null byte from the ALPHA register. This is discussed in more detail in volume 2 of the HP-41CX owners manual.

### 9.1.18 XF (X Exchange Flags)

This function exists in the Extended Functions ROM.

The MICROCODE file XF provides the function X<>F. X<>F uses the number in the X-register to set flags zero through seven. At the same time, it transfers the previous status of those flags to the X-register.

In the X-register, the flag status takes the form of an 8-bit number from 0 through 255. Each flag corresponds to one bit in that number. The number in the X-register is the sum of x(i) [i=0 to 7], where x(i)=0 if flag i is clear, and x(i)=2^i if flag i is set. The flags and their power-of-two equivalents are:

| Flag Number | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Equivalent | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

For example, suppose flags 0, 3, 5, and 7 are set, while flags 1, 2, 4, and 6 are clear. To determine what number is placed into the X-register when X<>F is executed, add up the numeric equivalents of the flags that are set:

| Flag | Numeric Equivalent |
|---|---|
| 0 | 1 |
| 3 | 8 |
| 5 | 32 |
| 7 | 128 |
| | --- |
| | 169 |

The number in the X-register would be 169.

If you enter zero in the X-register and execute X<>F, flags zero through seven are cleared, and their previous status is placed in the X-register.

You can use X<>F to create extended general purpose flags by storing numbers representing the status of flags zero through seven in a register. For example, to check the status of an extended flag, recall the flag status code in the X-register using RCL, execute X<>F, then execute [FS?] as usual.

X<>F enables you to use large numbers of flags in programs. Flags are grouped by eights and transferred into and out of the first

eight flag positions by means of X<>F. The number representing the status of a particular group of eight flags is placed in a storage register until it is needed. When it is needed, it is recalled to the X-register, exchanged with the flags presently in those eight positions, and the status of specific flags in that group can be examined or altered.

## 9.2 Type 1 MICROCODE Files

The type 1 microcode files are those which contain utilities used by two or more of the type 2 microcode files. For example, the file BIND contains a utility used by ALENG, AROT, ATOX, POSA, SIZE?, and X<>F. If one or more of those files is used, BIND must be included in the file list in the DEFINE file.

For each type 1 microcode file, this section will list the microcode labels that are defined within. This information can be used to determine which file is missing if BUILD fails with an unresolved microcode reference.

### 9.2.1 ALEN

ALEN defines the following microcode labels:

ALEN, CNTBYT, and FAHED.

### 9.2.2 ALNAM2

ALNAM2 defines the following microcode label:

ALNAM2.

### 9.2.3 BIND

BIND defines the following microcode label:

BIN_D.

### 9.2.4 XB

XB defines the following microcode labels:

X_256, X_999.

## 9.3 Type 0 MICROCODE Files

Type 0 microcode routines perform miscellaneous functions not covered by types 1 and 2.

### 9.3.1 AUTOST (Autostart)

AUTOST is an example of a file that does special interrupt processing. Unlike the type 2 files, AUTOST does not define any functions. Nor is it called by other routines (as are the type 1 files).

AUTOST, when included in your ROM, causes the HP-41, whenever it powers on, to search memory (user memory and ROMs) for a program named "RECOVER". When the program is found, it is executed. If it is not found, the calculator exhibits strange behavior -- *AUTOST should not be used in a ROM that does not contain a "RECOVER" program.*

AUTOST is useful for taking control of the machine as soon as it is turned on.

### 9.3.2 PRIVACY

PRIVACY is a short microcode file that must exist in every private ROM. This file is *not* included with the microcode library because it is built into BUILD -- automatically installed if the ROM is private. Its length is 13 bytes, plus one entry in the MCODE table (explained below).

### 9.3.3 KEYASN

KEYASN is a microcode file that must exist in every ROM which performs automatic key assignments. Like PRIVACY, KEYASN is build into BUILD. Its length is variable, requiring 150 bytes plus 2 bytes per key assignment. In addition, it requires 2 entries in the MCODE table (explained below).

### 9.3.4 MCODE

MCODE is a microcode file that must exist in every ROM in which a microcode file is doing special interrupt processing. All of the type 0 files mentioned above perform special interrupt processing. Its length is variable and dependent on several factors:

- For the second or third ROM of a 3-ROM plug-in, MCODE begins at ROM address 4014 and ends at 4083.
- For other ROMs (first ROM of a 3-ROM plug-in or any ROM of a 1- or 2-ROM plug-in), MCODE begins at ROM address 4020 and ends at 4083.

In addition, MCODE creates a table (immediately below the starting address) used for handling the special interrupt processing. The length of the table is 2n+1 bytes, where n is the number of table entries required by all of the MICROCODE files performing special

processing.

Additional information about creating MICROCODE files to perform special processing is contained in the chapter on advanced applications.

## 9.4 Microcode Library File Requirements

For each file in the microcode library, the table on the following page gives the type, number of bytes required, and list of dependencies. Files listed in the dependency column are type 1 and type 0 microcode files that must be included for the corresponding type 2 file to work. If they are not included, BUILD will fail with unresolved references. Files listed in parentheses are automatically included by BUILD, and should not be specified in the DEFINE file.

| Microcode File | Type | Bytes Required | Dependencies |
|---|---|---|---|
| AIP | 2 | 29 | |
| ALEN | 1 | 81 | |
| ALENG | 2 | 13 | ALEN<br>BIND |
| ALNAM2 | 1 | 98 | |
| ANUM | 2 | 114 | ALEN |
| AROT | 2 | 49 | ALEN<br>BIND |
| ATOX | 2 | 23 | ALEN<br>BIND |
| AUTOST | 0 | 98 | (MCODE) |
| BIND | 1 | 28 | |
| CLKEYS | 2 | 55 | |
| ENROM1 | 2 | 9 | |
| ENROM2 | 2 | 9 | |
| GETKEY | 2 | 60 | |
| KEYASN | 0 | see text | (MCODE) |
| MCODE | 0 | see text | |
| PASN | 2 | 83 | ALNAM2 |
| PCLPS | 2 | 127 | ALNAM2 |
| POSA | 2 | 84 | ALEN<br>BIND<br>XB |
| PRIVACY | 0 | 13 | (MCODE) |
| PSIZE | 2 | 95 | XB |

| Microcode File | Type | Bytes Required | Dependencies |
|---|---|---|---|
| RCLSTFLG | 2 | 108 | |
| REGMVSWP | 2 | 121 | |
| SIZE | 2 | 19 | BIND |
| XB | 1 | 29 | |
| XF | 2 | 42 | BIND<br>XB |
| XTOA | 2 | 18 | XB |

# 10. Emulating ROMs

SDS-II supports two techniques for emulating ROMs: EPROM boxes and RAM boxes.

## 10.1 EPROM Box ROM Emulation

A number of EPROM boxes are commercially available for emulating HP-41 ROMs. Through the BURN41 utility, SDS-II supports the ERAMCO-???? EPROM box. Unlike other commercially-available products, the ERAMCO-???? supports the bank-switching scheme used in the Hewlett-Packard bank-switching ROM (explained in the chapter on bank switching).

*** CONTINUE HERE with instructions for ERAMCO-???? ***

## 10.2 RAM Box Emulation

A number of RAM boxes are commercially available for emulating HP-41 ROMs. Typically, these boxes are programmed by the HP-41 using specialized software. For more information on using RAM boxes, see the sections on WRITMLDL and READMLDL in the chapter on advanced applications.

## 11. Burning EPROMs For The ERAMCO-????

Completion of this chapter awaits information about the product.
*** CONTINUE HERE ***

## 12. Bank-Switching

The HP-41 address space layout allows a plug-in ROM to use up to 8K of memory. Each plug-in port has an address space of 8K, divided into two 4K segments known as the "lower half" and the "upper half". In the past, a 4K or 8K plug-in was produced by using 1 or 2 (respectively) 4K ROM chips (HP part number 1LE9). Typically (although not always), a 4K application would use the lower half of the port's address space, and 8K applications would use the entire address space.

The HP-41 custom ROM program is now using a 12K ROM known as the 1LG9. The 1LG9 can be programmed to act either as a 4K, 8K, or 12K ROM. This not only reduces the chip count from 2 to 1 for an 8K plug-in, it allows even larger plug-ins: 12K.

Using a 12K plug-in in an 8K address space requires, understandably, special techniques to address the entire 12K. This is achieved through bank-switching. The following sections explain the requirements and limitations imposed by bank-switching.

### 12.1 A Word About Terminology

This document has been using the term "ROM" to describe a 4K block of HP-41 ROM address space, and "plug-in" to describe the collection of 4K blocks housed in a single package. Strictly speaking, the 1LG9 is a single ROM containing three programmable 4K blocks. For consistent terminology, however, "ROM" will designate a 4K block, and a 1LG9 plug-in conceptually treated as containing 1, 2, or 3 ROMs.

### 12.2 Basic Bank-Switching

Using SDS-II, you can specify that your plug-in consist of 1, 2, or 3 ROMs (by using 1, 2, or 3 "&ROM#" directives in the DEFINE file). The following table illustrates where these ROMs are addressed.

| # ROMs | ROM # | Where Addressed |
|---|---|---|
| 1 | 1 | lower half |
| 2 | 1 | lower half |
| | 2 | upper half |
| 3 | 1 | lower half |
| | 2 | upper half, bank 1 |
| | 3 | upper half, bank 2 |

The 1- and 2-ROM cases are straightforward. For a 3-ROM plug-in,

bank 1 is enabled when the plug-in is first inserted. When the ENROM2 command (from the MICROCODE library) is executed, bank 1 is disabled and bank 2 appears in its place. Similarly, the ENROM1 command enables bank 1 and disables bank 2.

In effect, there are two *different* ROMs occupying the upper half of the port, but only one is available at a time. Once a bank is enabled, it remains enabled until the opposite bank is enabled or until the ROM is removed from the machine. On being plugged back into a machine, BANK 1 is re-enabled.

Certain critical limitations apply to the ENROM1 and ENROM2 commands:

- These commands should be placed in the *first* ROM of the plug-in. Placing these commands in the bank-switching ROMs can cause unpredictable (and generally disastrous) results when they are executed.

- These commands will *only* affect the plug-in in which they are resident. If, for example, the HP-41 contains two bank-switching plug-ins in two different ports (say, ports 1 and 2), executing the ENROM2 keyword in port 1 will only affect the plug-in in port 1.

- Because of the possibility that more than one plug-in in the user's HP-41 will be bank-switching, and that ENROM1 and ENROM2 will subsequently be multiply defined, it is recommended that your application *not* rely on having the user execute the ENROM1 and ENROM2 commands. You should place all major labels (those to be XEQ'd by the user) in the first ROM, and only use ENROM1 and ENROM2 *within* your application to enable the ROMs containing your utilities.

## 12.3 Advanced Bank-Switching

The material in this section assumes a familiarity with HP-41 assembly-language programming, and with the architecture and operating system of the HP-41.

The 1LG9 has a number of configuration options, more fully explained in the section on 1LG9 configuration in the chapter on advanced applications. Each 4K core of the 1LG9 can be programmed as either bank 0 (always enabled), bank 1 (enabled on power-up), or bank 2 (alternately enabled/disabled with bank 1). While it is possible to create other configurations than that shown in the previous section (ROM 1=low/bank0, ROM 2=high/bank1, ROM 3=high/bank2), that configuration should be usable for all applications.

Using bank-switching places certain requirements on the code within the ROM:

- The bank-switching itself is accomplished through the use of the assembly-language instructions ENROM1 (instruction code 100H) and ENROM2 (instruction code 180H). These instructions *must* occur somewhere within the address space of the 1LG9 being bank-switched. That is, an ENROMx instruction will only affect the 1LG9 out of which it is read. (Note that the ENROMx instruction only takes effect when it is read as a CPU instruction, and not when it is read as data by the CXISA instruction.)

- The 1LG9 requires that the ENROMx instruction be preceded by an instruction whose high bit is zero. This is customarily handled by placing a "GOTO $+1" instruction before the ENROMx.

- In general, an ENROMx instruction should not occur within a bank-switching ROM. It can, however, be done with careful planning. Keeping in mind that executing an ENROMx instruction will immediately enable the selected ROM, instructions can be placed within both ROMs to insure that execution continues properly. For example, using the typical 3-ROM configuration described above, if the CPU executes an ENROM1 instruction from address F00H in BANK 2, it will read the following instruction from address F01H in BANK 1.

- Requirements of the self-test ROM and production testing impose the following restrictions (these are both handled automatically by BUILD, but must be accounted for if you create your own ROM images using the advanced tools):

    - The data at address FFDH within any core must contain a '1' in at least one of the upper two bits *if and only if* that core is a bank-selecting core.

    - If a core is bank-selecting, it must contain the following instructions at the following addresses:

```
FC7   ENROM1
FC8   RTN
FC9   ENROM2
FCA   RTN
```

The restrictions mentioned in the previous section about placement of major label are not absolute; they can also be circumvented through careful planning. This was done, for example, in the HP-41 ADVANTAGE ROM.

The ADVANTAGE places major labels in the first two ROMs. The third ROM has a ROM ID of zero and, subsequently, an empty FAT table. It *only* contains microcode, which is always called from BANK 0 after performing an ENROM2. The microcode in the ROM *never* relinquishes control with BANK 2 enabled. Rather, whenever code in BANK 2

relinquishes control *or* calls a mainframe function that might not return, it does so *through* code in BANK 0 that re-enables BANK 1 before relinquishing control (or executing the call). Obviously, such techniques require writing code to jump between pages in HP-41 ROM space.

These steps have the effect of completely hiding BANK 2 from the user, and making two ROMs (and therefore two FATs) available for functions.

# 13. SDS-II Basic Utilities

SDS-II disk #1 contains the following utilities in addition to READ41P, BUILD, and BURN41.

## 13.1 CHECKSUM

The checksum utility can be used to verify the checksum of a ROM image file. The invocation is:

CHECKSUM <filename> [<filename>...]

This utility accepts filename wild-carding in the command line. For example,

CHECKSUM *

will verify the checksums of all *.41R files in the current directory.

## 13.2 EPROM

The EPROM utility generates data to be downloaded to an EPROM programmer. It is used by the BURN41 program. Instructions on using the EPROM utility itself can be found in the utilities section in the chapter on advanced applications.

## 13.3 LIFPACK

The LIFPACK utility allows you to pack an HP-41 mass storage medium, reclaiming space lost when files are purged by the HP-41. Its use is not recommended for the 82161A cassette drive. To invoke:

LIFPACK <disk_designator>

For example, to pack the LIF disk in drive C:

LIFPACK C:

## 13.4 LISTFAT

The LISTFAT utility lists the catalog of a ROM image file. Invocation:

LISTFAT <ROMfilename> [<ROMfilename>]

If a second ROMfilename is specified, LISTFAT will correctly find functions in the second ROM whose FAT (function address table) entry is in the first ROM, and vice versa. Such ROMs will never be created by BUILD, but can be created using the advanced programming tools.

## 13.5 SDSCAT

The SDSCAT utility provides a catalog of HP-41 program files and MLDL-format files (created by WRITMLDL, explained in the chapter on advanced applications) on an HP-41 mass storage medium. Both catalogs are listed in alphabetical order, *not* the order the files are encountered on the disk. HP-41 program files are listed with the special characters substituted as described in the appendix on special characters.

Invocation:

SDSCAT <disk_designator>

For example:

SDSCAT C:

# 14. Advanced Applications

This chapter describes the advanced tools supplied with SDS-II for ROM development, as well as the tools needed to support the RAM-based ROM emulator.

The advanced programming tools are provided on an AS-IS basis. HP makes no warranty, expressed or implied, as to their performance. HP provides no support for assembly-language code development, and shall not be responsible for any loss or damage to the user, its customers or any third parties caused by inaccuracies in the materials or documentation.

## 14.1 Using the RAM-Based ROM Emulator

Several RAM-based devices, known as Q-ROM, have been marketed to allow ROM emulation. Recommended for use with SDS-II is the ERAMCO ES16S, which emulates the full functionality of the 1LG9 ROM, including the bank-switching capabilities.

Data is loaded in Q-ROM devices by writing to them from the HP-41. Software is also available for programming Q-ROM devices. For example, the GETROM keyword in the MLDL operating system (which is distributed on EPROM for use in the ERAMCO ESMLDL 1) allows transfer of a ROM image from an HP-41 mass storage medium into an Q-ROM device.

To emulate the bank-switching 1LG9 with the ERAMCO Ram Storage Unit, load the images from ROM 1 and ROM 2 (respectively) into bank 1, and the images from ROM 1 and ROM 3 (respectively) into bank 2. By virtue of its presence in both banks, ROM 1 is always present, providing emulation of 1LG9 bank 0.

Details on using the various Q-ROM devices are included in the documentation with each product, and will not be discussed here.

SDS-II includes two programs on disk #2 to support use of Q-ROM devices: READMLDL and WRITMLDL.

### 14.1.1 WRITMLDL

The WRITMLDL utility will copy a ROM image file created by BUILD (or the advanced utilities) onto a LIF medium in the "standard format". That is, it will create a file on the HP-41 medium that is directly readable by the GETROM keyword (mentioned above) and other such utilities.

The invocation is:

```
WRITMLDL <ROMfile> <disk_designator>:<filename>
```

For example,

```
WRITMLDL MYROM1 C:ROMIMAGE
```

will take ROM image file MYROM1.41R and create file ROMIMAGE on the HP-41 media in drive C: containing the ROM image in a format readable by GETROM.

**A word of caution:** Under certain circumstances, the GETROM keyword can be fooled into reading the wrong file. The problem, which is not easily repeatable, can best be characterized with an example:

If the HP-41 medium has two files, named ABCDEF and ABCDEFG, and ABCDEFG occurs earlier in the disk directory than ABCDEF, attempting to retrieve ABCDEF with GETROM will *sometimes* retrieve ABCDEFG. This problem can be avoided by appending a space to the filename specified in the ALPHA register.

### 14.1.2 READMLDL

READMLDL is the inverse of WRITMLDL. It will read a ROM image file from an HP-41 mass storage medium into a .41T ROM image file. The ROM image file can then be manipulated using such tools as LISTFAT, EXTRACT, etc.

Invocation:

```
READMLDL <disk_designator>:<filename> <ROMfile>
```

## 14.2 Other Advanced Utilities

This section assumes prior knowledge of HP-41 assembly language, and the HP-41 architecture and operating system. Recommended reading on this topic is the manual for the ZENROM (from Zengrange Ltd., in England).

### 14.2.1 ASSEMB41

ASSEMB41 is an HP-41 assembler. It assembles source files (suffixed with .41A) into relocatable object files (.41O) that can either be:

- Collected into ROM image files (.41R) using LINK41, or
- Turned into microcode library files (.41T) using MUCODE.

The assembler uses HP mnemonics, which differ in many ways from the mnemonics used in many third-party products. The following subsections list the mnemonics and their opcodes, which should facilitate translation from other assembly languages.

#### 14.2.1.1 Invocation

The assembler is invoked by:

ASSEMB41 [-ls8] [-o <outputfile>] <inputfile>

Command line options:

-l Produce a source code listing.

-s Print a symbol table.

-8 Print addresses and opcodes in octal. If not specified, the assembler will print addresses and opcodes in hex.

-o Use the following argument as the name of the output file. If this option is not specified, the input filename will be used. In either case, the output file will have extension ".41O".

The <inputfile> must have the extension ".41A" to be found by ASSEMB41.

If the -l or -s option is specified, the listing will be formatted for a printer, and should be redirected to one using the '>' command line feature of MS-DOS.

#### 14.2.1.2 Assembler Syntax Conventions

Following are the general syntax rules for ASSEMB41.

##### 14.2.1.2.1 Comments

Any line beginning with a "*" is interpreted as a comment.

A semicolon (';') can be used to begin an in-line comment.

##### 14.2.1.2.2 Fields

A line consists of three fields: label, opcode, and operand.

- A label must begin in column 1, must begin with an alphabetic character, and can contain any number of alphanumeric characters. Only the first 20 characters of a label are used by ASSEMB41.

- An opcode can begin anywhere but in column 1. If a label is used, there must be at least one space (or tab) between the label and the opcode.

- The operand, if required for the opcode, must be separated from the opcode by one or more spaces (or tabs).

If a field begins with a semicolon, the remainder of the line is treated as a comment and ignored by the assembler.

#### 14.2.1.2.3 Expressions

Most opcodes that can take numeric operands (with the exception of pseudo-ops SPACE, FILLTO, BSS, and ORG) can take arbitrary expressions combining labels, constants, and special symbols.

CONSTANTS — Constants can be in hex (terminated with 'H'), octal (terminated with 'O' or 'Q') or decimal. A hex constant beginning with a non-decimal digit must be prefixed with a zero to avoid confusion with labels (for example, 0FH).

LABELS — Local labels (those that can be resolved within this module) can be included in expressions.

SPECIAL — The special symbol '$' designates the current address. For example, GOTO $+1 means GOTO the next statement.

The following operators can be used in expressions: +, -, *, /, and % (modulus). An expression consisting of one label or '$' plus or minus a constant is considered a "relative" expression. All other expressions are considered "absolute". The purpose of this distinction becomes clear in the section below on global labels. If the '-s' option is specified, labels with "absolute" values are indicated in the symbol table with a "*".

The pseudo-ops mentioned above that cannot take arbitrary expressions *can* take constants in decimal, hex, or octal.

#### 14.2.1.2.4 Global Labels

ASSEMB41 supports global references for the following opcodes:

- All branches (short and long).
- CON.
- DEFP4K, DEFR4K, DEFR8K, U4KDEF, U8KDEF.
- GSB41C, GSBSAM, GOL41C, GOLSAM.
- LC3.

A global reference is one that is resolved by the linker (LINK41) rather than by the assembler. A global expression takes the form of a label preceded by '='. For example, to call the mainframe routine CLLCDE, use "GOSUB =CLLCDE". A global expression must only contain a single label; it cannot contain any arithmetic.

To declare a label as global, the GLB opcode (explained below) is used. When the linker resolves global references, it updates relative expressions to reflect the load address of the assembly module; absolute expressions are not updated.

In addition to supporting global references, the opcodes mentioned above support "relocation fixups". This means that a local reference to a relative expression (such as CON <label>) is updated by the linker to reflect the load address of the assembly module. An error message in LINK41 or BUILD about an internal reference out of range is caused by a relocation fixup being out of range.

### 14.2.1.3 Mnemonics

The mnemonics used here reflect the history of mnemonics used in past internal HP-41 software; they do not reflect a conscious choice made for this product (exception: the addition of the WMLDL mnemonic). The type 0 opcodes are presented both in alphabetical and numeric order, to facilitate understanding this set of mnemonics.

For the opcodes that take an operand, only the base value of the compiled word is shown. The actual choice of bytes is dependent on the value of the operand. Where appropriate, this table gives the legal range of operands.

#### 14.2.1.3.1 Type 0 Opcodes -- Alphabetical Order

See the commentary after this list for an explanation of opcodes designated with '*'.

| MNE | OPRND | OPC | MNE | OPRND | OPC |
|---|---|---|---|---|---|
| ?F0=1 | | 3ACH | IFCR? | | 16CH |
| ?F10=1 | | 0ECH | INCPT | | 3DCH |
| ?F11=1 | | 1ACH | LC | 0-15 | 010H |
| ?F12=1 | | 36CH | *LC3 | <expr> | 010H, 010H, 010H |
| ?F13=1 | | 2ECH | LDI | | 130H |
| ?F1=1 | | 32CH | LLD? | | 160H |
| ?F2=1 | | 22CH | M=C | | 158H |
| ?F3=1 | | 02CH | MCEX | | 1D8H |
| ?F4=1 | | 06CH | N=C | | 070H |
| ?F5=1 | | 0ACH | NCEX | | 0F0H |
| ?F6=1 | | 16CH | NOP | | 000H |
| ?F7=1 | | 2ACH | ORAV? | | 0ECH |
| ?F8=1 | | 12CH | P=Q? | | 120H |
| ?F9=1 | | 26CH | PFAD=C | | 3F0H |
| ?LLD | | 160H | POWOFF | | 060H, 000H |
| ?P=Q | | 120H | PT= | 0-13 | 01CH |
| ?PT= | 0-13 | 014H | PT=? | 0-13 | 014H |
| ?S0=1 | | 38CH | PT=A | | 3E8H |

| | | |
|---|---|---|
| ?S10=1 | | 0CCH |
| ?S11=1 | | 18CH |
| ?S12=1 | | 34CH |
| ?S13=1 | | 2CCH |
| ?S1=1 | | 30CH |
| ?S2=1 | | 20CH |
| ?S3=1 | | 00CH |
| ?S4=1 | | 04CH |
| ?S5=1 | | 08CH |
| ?S6=1 | | 14CH |
| ?S7=1 | | 28CH |
| ?S8=1 | | 10CH |
| ?S9=1 | | 24CH |
| ALARM? | | 36CH |
| C=C!A | | 370H |
| C=C&A | | 3B0H |
| C=C.A | | 3B0H |
| C=CORA | | 370H |
| C=DATA | | 038H |
| C=G | | 098H |
| C=KEYS | | 220H |
| C=M | | 198H |
| C=N | | 0B0H |
| C=REGN | 1-15 | 038H |
| C=ST | | 398H |
| C=STK | | 1B0H |
| CGEX | | 0D8H |
| CHKKB | | 3CCH |
| CLRABC | | 1A0H |
| CLRST | | 3C4H |
| CMEX | | 1D8H |
| CNEX | | 0F0H |
| *CON | <expr> | 000H |
| CRDFLG | | 3E8H |
| CRDINF | | 268H |
| CRDOHF | | 1E8H |
| CRDWPF | | 168H |
| CSTEX | | 3D8H |
| CXISA | | 330H |
| DADD=C | | 270H |
| DATA=C | | 2F0H |
| DECPT | | 3D4H |
| DISOFF | | 2E0H |
| DISTOG | | 320H |
| DSALM | | 2A8H |
| DSWKUP | | 228H |
| ENALM | | 2E8H |
| ENREAD | | 0A8H |
| ENROM1 | | 100H |
| ENROM2 | | 180H |
| ENWKUP | | 268H |
| PT=B | | 3A8H |
| RABCL | | 3F8H |
| RABCR | | 3B8H |
| RCR | 0-13 | 03CH |
| RCTIME | | 078H |
| RDALM | | 0B8H |
| RDINT | | 178H |
| RDSCR | | 138H |
| RDSTS | | 0F8H |
| RDTIME | | 038H |
| READEN | | 178H |
| REGN=C | 0-15 | 028H |
| RSTKB | | 3C8H |
| RTN | | 3E0H |
| RTNC | | 360H |
| RTNNC | | 3A0H |
| S0= | 0-1 | 384H |
| S10= | 0-1 | 0C4H |
| S11= | 0-1 | 184H |
| S12= | 0-1 | 344H |
| S13= | 0-1 | 2C4H |
| S1= | 0-1 | 304H |
| S2= | 0-1 | 204H |
| S3= | 0-1 | 004H |
| S4= | 0-1 | 044H |
| S5= | 0-1 | 084H |
| S6= | 0-1 | 144H |
| S7= | 0-1 | 284H |
| S8= | 0-1 | 104H |
| S9= | 0-1 | 244H |
| SB=F | | 298H |
| SELP | | 0A0H |
| SELPF | 0-15 | 024H |
| SELQ | | 0E0H |
| SETDEC | | 2A0H |
| SETHEX | | 260H |
| SLLABC | | 1A8H |
| SLLDAB | | 168H |
| SLSABC | | 3E8H |
| SLSDA | | 2A8H |
| SLSDAB | | 368H |
| SLSDB | | 2E8H |
| SPOPND | | 020H |
| SRLABC | | 128H |
| SRLDA | | 028H |
| SRLDAB | | 0E8H |
| SRLDB | | 068H |
| SRLDC | | 0A8H |
| SRQR? | | 2ACH |
| SRSABC | | 3A8H |
| SRSDA | | 1E8H |

| | | | | | |
|---|---|---|---|---|---|
| ENWRIT | | 028H | SRSDAB | | 328H |
| F=SB | | 258H | SRSDB | | 228H |
| FEXSB | | 2D8H | SRSDC | | 268H |
| FLG=1? | 0-13 | 02CH | ST=0 | 0-13 | 004H |
| FLLABC | | 138H | ST=1 | 0-13 | 008H |
| FLLDA | | 038H | ST=1? | 0-13 | 00CH |
| FLLDAB | | 0F8H | ST=C | | 358H |
| FLLDB | | 078H | STARTC | | 368H |
| FLLDC | | 0B8H | STK=C | | 170H |
| FLSDA | | 2B8H | STOPC | | 328H |
| FLSDAB | | 378H | STPINT | | 1E8H |
| FLSDB | | 2F8H | STREAD | | 0E8H |
| FLSDC | | 1B8H | STWRIT | | 068H |
| FRAV? | | 12CH | TCLCRD | | 368H |
| FRNS? | | 26CH | TRPCRD | | 328H |
| FRSABC | | 3B8H | TSTBUF | | 2E8H |
| FRSDA | | 1F8H | WDTIME | | 068H |
| FRSDAB | | 338H | WMLDL | | 040H |
| FRSDB | | 238H | WRALM | | 0A8H |
| FRSDC | | 278H | WRSCR | | 128H |
| G=C | | 058H | WRSTS | | 0E8H |
| GOKEYS | | 230H | WRTEN | | 2F0H |
| GOTOC | | 1E0H | WRTIME | | 028H |
| HPIL=C | 0-7 | 200H | WSINT | | 168H |
| HPL=CH | 0-7 | 024H | | | |

The CON mnemonic compiles into the low 10 bits of the expression in the operand field. Unlike most of these opcodes, it does not perform a range check on the operand to require that it be in the range 0-3FFH.

The LC3 compiles into three successive LC's, loading nibbles 2, 1, and 0 of the expression. For example, LC3 123H compiles into LC 1/LC 2/LC 3. Like CON, it does not perform a range check on the operand to require that it be in the range 0-0FFFH.

### 14.2.1.3.2 Type 0 Opcodes -- Numeric Order

| MNE | OPRND | OPC | MNE | OPRND | OPC |
|---|---|---|---|---|---|
| CON | <expr> | 000H | C=STK | | 1B0H |
| NOP | | 000H | FLSDC | | 1B8H |
| S3= | 0-1 | 004H | CMEX | | 1D8H |
| ST=0 | 0-13 | 004H | MCEX | | 1D8H |
| ST=1 | 0-13 | 008H | GOTOC | | 1E0H |
| ?S3=1 | | 00CH | CRDOHF | | 1E8H |
| ST=1? | 0-13 | 00CH | SRSDA | | 1E8H |
| LC | 0-15 | 010H | STPINT | | 1E8H |
| LC3 | <expr> | 010H, 010H, 010H | FRSDA | | 1F8H |
| ?PT= | 0-13 | 014H | HPIL=C | 0-7 | 200H |
| PT=? | 0-13 | 014H | S2= | 0-1 | 204H |

| | | |
|---|---|---|
| PT= | 0-13 | 01CH |
| SPOPND | | 020H |
| HPL=CH | 0-7 | 024H |
| SELPF | 0-15 | 024H |
| ENWRIT | | 028H |
| REGN=C | 0-15 | 028H |
| SRLDA | | 028H |
| WRTIME | | 028H |
| ?F3=1 | | 02CH |
| FLG=1? | 0-13 | 02CH |
| C=DATA | | 038H |
| C=REGN | 1-15 | 038H |
| FLLDA | | 038H |
| RDTIME | | 038H |
| RCR | 0-13 | 03CH |
| WMLDL | | 040H |
| S4= | 0-1 | 044H |
| ?S4=1 | | 04CH |
| G=C | | 058H |
| POWOFF | | 060H, 000H |
| SRLDB | | 068H |
| STWRIT | | 068H |
| WDTIME | | 068H |
| ?F4=1 | | 06CH |
| N=C | | 070H |
| FLLDB | | 078H |
| RCTIME | | 078H |
| S5= | 0-1 | 084H |
| ?S5=1 | | 08CH |
| C=G | | 098H |
| SELP | | 0A0H |
| ENREAD | | 0A8H |
| SRLDC | | 0A8H |
| WRALM | | 0A8H |
| ?F5=1 | | 0ACH |
| C=N | | 0B0H |
| FLLDC | | 0B8H |
| RDALM | | 0B8H |
| S10= | 0-1 | 0C4H |
| ?S10=1 | | 0CCH |
| CGEX | | 0D8H |
| SELQ | | 0E0H |
| SRLDAB | | 0E8H |
| STREAD | | 0E8H |
| WRSTS | | 0E8H |
| ?F10=1 | | 0ECH |
| ORAV? | | 0ECH |
| CNEX | | 0F0H |
| NCEX | | 0F0H |
| FLLDAB | | 0F8H |
| RDSTS | | 0F8H |

| | | |
|---|---|---|
| ?S2=1 | | 20CH |
| C=KEYS | | 220H |
| DSWKUP | | 228H |
| SRSDB | | 228H |
| ?F2=1 | | 22CH |
| GOKEYS | | 230H |
| FRSDB | | 238H |
| S9= | 0-1 | 244H |
| ?S9=1 | | 24CH |
| F=SB | | 258H |
| SETHEX | | 260H |
| CRDINF | | 268H |
| ENWKUP | | 268H |
| SRSDC | | 268H |
| ?F9=1 | | 26CH |
| FRNS? | | 26CH |
| DADD=C | | 270H |
| FRSDC | | 278H |
| S7= | 0-1 | 284H |
| ?S7=1 | | 28CH |
| SB=F | | 298H |
| SETDEC | | 2A0H |
| DSALM | | 2A8H |
| SLSDA | | 2A8H |
| ?F7=1 | | 2ACH |
| SRQR? | | 2ACH |
| FLSDA | | 2B8H |
| S13= | 0-1 | 2C4H |
| ?S13=1 | | 2CCH |
| FEXSB | | 2D8H |
| DISOFF | | 2E0H |
| ENALM | | 2E8H |
| SLSDB | | 2E8H |
| TSTBUF | | 2E8H |
| ?F13=1 | | 2ECH |
| DATA=C | | 2F0H |
| WRTEN | | 2F0H |
| FLSDB | | 2F8H |
| S1= | 0-1 | 304H |
| ?S1=1 | | 30CH |
| DISTOG | | 320H |
| SRSDAB | | 328H |
| STOPC | | 328H |
| TRPCRD | | 328H |
| ?F1=1 | | 32CH |
| CXISA | | 330H |
| FRSDAB | | 338H |
| S12= | 0-1 | 344H |
| ?S12=1 | | 34CH |
| ST=C | | 358H |
| RTNC | | 360H |

| | | |
|---|---|---|
| ENROM1 | | 100H |
| S8= | 0-1 | 104H |
| ?S8=1 | | 10CH |
| ?P=Q | | 120H |
| P=Q? | | 120H |
| SRLABC | | 128H |
| WRSCR | | 128H |
| ?F8=1 | | 12CH |
| FRAV? | | 12CH |
| LDI | | 130H |
| FLLABC | | 138H |
| RDSCR | | 138H |
| S6= | 0-1 | 144H |
| ?S6=1 | | 14CH |
| M=C | | 158H |
| ?LLD | | 160H |
| LLD? | | 160H |
| CRDWPF | | 168H |
| SLLDAB | | 168H |
| WSINT | | 168H |
| ?F6=1 | | 16CH |
| IFCR? | | 16CH |
| STK=C | | 170H |
| RDINT | | 178H |
| READEN | | 178H |
| ENROM2 | | 180H |
| S11= | 0-1 | 184H |
| ?S11=1 | | 18CH |
| C=M | | 198H |
| CLRABC | | 1A0H |
| SLLABC | | 1A8H |
| ?F11=1 | | 1ACH |
| SLSDAB | | 368H |
| STARTC | | 368H |
| TCLCRD | | 368H |
| ?F12=1 | | 36CH |
| ALARM? | | 36CH |
| C=C!A | | 370H |
| C=CORA | | 370H |
| FLSDAB | | 378H |
| S0= | 0-1 | 384H |
| ?S0=1 | | 38CH |
| C=ST | | 398H |
| RTNNC | | 3A0H |
| PT=B | | 3A8H |
| SRSABC | | 3A8H |
| ?F0=1 | | 3ACH |
| C=C&A | | 3B0H |
| C=C.A | | 3B0H |
| FRSABC | | 3B8H |
| RABCR | | 3B8H |
| CLRST | | 3C4H |
| RSTKB | | 3C8H |
| CHKKB | | 3CCH |
| DECPT | | 3D4H |
| CSTEX | | 3D8H |
| INCPT | | 3DCH |
| RTN | | 3E0H |
| CRDFLG | | 3E8H |
| PT=A | | 3E8H |
| SLSABC | | 3E8H |
| PFAD=C | | 3F0H |
| RABCL | | 3F8H |

### 14.2.1.3.3 Arithmetics

All of these mnemonics require a time-enable field, consisting of one of the following: PT, X, WPT, W, PQ, XS, M, S.

| MNE | OPRND | OPC |
|---|---|---|
| ?A#0 | TE | 342H |
| ?A#C | TE | 362H |
| ?A<B | TE | 322H |
| ?A<C | TE | 302H |
| ?B#0 | TE | 2C2H |
| ?C#0 | TE | 2E2H |
| A#0? | TE | 342H |
| A#C? | TE | 362H |
| A<B? | TE | 322H |
| A<C? | TE | 302H |
| A=0 | TE | 002H |
| B#0? | TE | 2C2H |
| B=0 | TE | 022H |
| B=A | TE | 082H |
| B=C | TE | 0E2H, 0C2H |
| BAEX | TE | 062H |
| BCEX | TE | 0E2H |
| BSR | TE | 3A2H |
| C#0? | TE | 2E2H |
| C=-C | TE | 282H |
| C=-C-1 | TE | 2A2H |
| C=0 | TE | 042H |

```
A=A+1   TE      162H            C=A     TE      0A2H, 102H
A=A+B   TE      122H            C=A+C   TE      202H
A=A+C   TE      142H            C=A-C   TE      242H
A=A-1   TE      1A2H            C=B     TE      0C2H
A=A-B   TE      182H            C=C+1   TE      222H
A=A-C   TE      1C2H            C=C+A   TE      202H
A=B     TE      062H, 082H      C=C+C   TE      1E2H
A=C     TE      102H            C=C-1   TE      262H
ABEX    TE      062H            CAEX    TE      0A2H
ACEX    TE      0A2H            CBEX    TE      0E2H
ASL     TE      3E2H            CSR     TE      3C2H
ASR     TE      382H
```

14.2.1.3.4 **Pseudo-Ops**

The following pseudo-ops are used in ASSEMB41:

```
BSS     <number>                LEGAL
EJECT                           LIST
END                             ORG     <number>
EQU     <expr>                  SKIP    <number>
FILLTO  <number>                SPACE   <number>
GLB     <label>                 TITLE   "<text>"
LCDCHAR "<text>"                UNLIST
```

An explanation of their functions:

BSS — Fill specified number of words with zeroes and skip them. Operand field specifies number of words.

EJECT — Formfeed the listing.

END — End of source; do not read rest of the file.

EQU — Equate a label with a value. For example, "ABC EQU 5" will equate the label ABC to the absolute value 5.

FILLTO — Fill the object file with zeroes up to the address specified in the operand field. This pseudo-op fills to the specified address *relative to the start of the file*. For example, if ORG 1000H was specified, then FILLTO 0F00H will actually fill from the current address to address 1F00H.

GLB — Declares the label specified in the operand field to be global, which allows it to be found by the linker during the LINK41 phase. Its use is illustrated in the examples.

LCDCHAR — The expression in quotes is encoded in the LCD character format. This pseudo-op only accepts characters that are legal in labels. A '!' is used to designate that the

character following it should be encoded with bit 7 set. The special characters sigma, not-equal, and angle use the alternate representation explained in the appendix on special characters. The use of this pseudo-op is illustrated in the examples.

LEGAL — Normally, the assembler complains if certain potentially erroneous combinations of opcodes exist. For example, a LDI followed by anything other than a CON causes an error. A test of any sort followed by anything other than a conditional branch/return causes an error. A GOTO, GOLONG, or GOSUB preceded by a command that *might* set carry causes an error. By placing the LEGAL pseudo-op before the offending code, these errors are suppressed.

LIST — If the -l option was specified, this pseudo-op turns off the UNLIST mode. Default behavior is to list until an UNLIST is encountered.

ORG — Specifies that the relocatable file is to start at an absolute address. This forces the linker to place the module at that address, and all labels within the module to be considered absolute expressions. Operand field specifies the address. Files assembled with the ORG directive cannot be used as microcode library files.

SKIP — Same as EJECT.

SPACE — Skip the specified number of spaces in the output listing. Number of spaces specified in the operand field.

TITLE — Specify a title to appear on each page of the listing. Title is also stored in the object file, and displayed by LINK41 and ASMBINFO when this file is referenced. The title string must appear between quotes.

UNLIST — Turn off listing (if -l option is specified) until a LIST directive is encountered.

#### 14.2.1.3.5 FAT Entries

The following pseudo-ops create two-word entries for the Function Address Table (FAT):

| MNE | OPRND | OPC | MNE | OPRND | OPC |
|---|---|---|---|---|---|
| DEFP4K | <expr> | 000H, 100H | U4KDEF | <expr> | 200H, 000H |
| DEFR4K | <expr> | 000H, 000H | U8KDEF | <expr> | 200H, 000H |
| DEFR8K | <expr> | 000H, 000H | | | |

Their functions are as follows:

DEFR4K Creates a FAT entry for a microcode function somewhere within the current 4K block. <expr> is the execution address of the function. Immediately preceding the target address is the function name, in LCD character representation, backwards, terminating with bit 7 set on the last character.

DEFR8K Like DEFR4K, but capable of pointing to a function in the adjacent 4K block as well. Can be used to create a FAT entry in the lower half of the port address space pointing to a function in the upper half, and vice versa.

U4KDEF Creates a FAT entry for a usercode function somewhere within the current 4K block. <expr> is the address of the GLOBAL token in a LBL statement.

U8KDEF Like U4KDEF, but capable of pointing to a function in the adjacent 4K block as well. Can be used to create a FAT entry in the lower half of the port address space pointing to a function in the upper half, and vice versa.

DEFP4K The purpose of this mnemonic is lost to the modern memory.

14.2.1.3.6 **Branches**

These opcodes provide branching of various sorts:

| MNE | OPRND | OPC | MNE | OPRND | OPC |
|---|---|---|---|---|---|
| GOC | <expr> | 007H | GOSUB | <expr> | 001H, 000H |
| GOL41C | <expr> | 001H, 000H, 000H | GOTO | <expr> | 003H |
| GOLC | <expr> | 001H, 003H | GSB41C | <expr> | 001H, 000H, 000H |
| GOLNC | <expr> | 001H, 002H | GSBSAM | <expr> | 001H, 000H, 000H |
| GOLONG | <expr> | 001H, 002H | GSUBC | <expr> | 001H, 001H |
| GOLSAM | <expr> | 001H, 000H, 000H | GSUBNC | <expr> | 001H, 000H |
| GONC | <expr> | 003H | | | |

Explanation:

GOC Local goto target address if carry is set.

GONC Local goto target address if carry is clear.

GOTO Same as GONC, but the assembler complains if it follows anything that might set the carry.

GOLC Long goto target address if carry is set.

GOLNC Long goto target address if carry is clear.

GOLONG Same as GOLNC, but the assembler complains if it follows anything that might set the carry.

GSUBC Long gosub target address if carry is set.

GSUBNC Long gosub target address if carry is clear.

GOSUB Same as GSUBNC, but the assembler complains if it follows anything that might set the carry.

GSB41C Compiles into GOSUB GOSUB0/GOSUB1/GOSUB2/GOSUB3/GOSUB (whichever is appropriate) followed by CON <addr>. Uses HP-41 mainframe routines to achieve local gosub within current 4K block.

GSBSAM Compiles into GOSUB GOSUB followed by CON <addr>. Assembler complains if target address is not within current 1K block.

GOL41C Like GSB41C, but for GOSUB GOL0/GOL1/GOL2/GOL3/GOL.

GOLSAM Like GSBSAM, but for GOSUB GOL.

#### 14.2.1.3.7 Peripheral Commands

These commands are for smart peripherals.

| MNE | OPRND | OPC | MNE | OPRND | OPC |
|---|---|---|---|---|---|
| ?PFSET | 0-15 | 003H | PRINTC | | 007H |
| C=HPIL | 0-7 | 024H, 03AH, 003H | RDPTRN | | 03AH |
| CH= | 0-255 | 001H | RDPTRR | | 03BH |
| PFSET? | 0-15 | 003H | RTNCPU | | 005H |

### 14.2.1.4 EXAMPLES

#### 14.2.1.4.1 An Assembly-Language Keyword

The following example, the code for the AIP keyword, illustrates some of the assembler's features:

```
            TITLE   "AIP function"
            GLB     xqAIP
            LCDCHAR "!PIA"   ; AIP
xqAIP       C=REGN  3        ; Read X
            GOSUB   =CHK_NO_S        ; Check for alpha data
            C#0?    XS       ; Exponent negative?
            GONC    AIP10    ; No.
            C=0     W
AIP10       ST=1    5
            GOSUB   =INTFRC ; C=integer part, A.X=exponent
```

```
          PT=     13
AIP20     B=A     X         ; Save exponent in B
          LC      3
          G=C               ; G=ASCII'ized digit
          PT=?    2         ; Down at exponent?
          GONC    AIP30     ; No.
          INCPT             ; Yes. Stay here.
AIP30     M=C               ; Hold mantissa
          SELQ
          GOSUB   =APNDNW   ; Append to alpha register
          SELP
          C=M               ; Retrieve mantissa
          ABEX    X
          A=A-1   X         ; Done?
          GONC    AIP20
          RTN
```

#### 14.2.1.4.2 A Function Address Table

The following code, which would occur at the beginning of the ROM image, illustrates a ROM with ID=21, a header, and a single function (AIP, above).

```
          CON     21        ; ROM ID=21
          CON     2         ; 2 entries in FAT
          DEFR4K  HDR       ; Point to my header
          DEFR4K  =xqAIP    ; Point to my function
          CON     0
          CON     0         ; End of table
*
          LCDCHAR "!MOR YM--"      ; "--MY ROM"
HDR       RTN
```

### 14.2.2 LINK41

The LINK41 utility is used to collect one or more assembler output files into a ROM image file. Even if an assembler output file has no external references, it must be run through LINK41 to put it into the proper form. LINK41 expects all of its assembler input files to have the filename extension ".41O", and it creates ROM image files with the filename extension ".41R".

LINK41 is command-driven, either from a command file or from the keyboard. The output is formatted for printing (page headers, formfeeds, and such), and should be re-directed to a printer using the '>' command line feature of MS-DOS.

Invocation:

LINK41 [<command_file>]

If a command_file is specified, that file is opened and LINK41 commands are read from it. If not, commands are accepted from the console (prompting is provided).

LINK41 creates from 1 to 4 ROM image files. All commands can be abbreviated to their first two characters. The commands are:

```
NEwrom [<pagenumber>]
OUtput <ROMfilename>
LOcate <address>
CHecksum [<address>]
SEarch <assemblyfilename>
RElOc <assemblyfilename>
LIst XRef
SUppress XRef
COmment
ENd
?
```

and have the following meanings:

NEWROM — Analogous to the "&ROM#" directive in BUILD. Used to begin a new 4K ROM image. The optional <pagenumber> can be from 0 to 15, and determines the starting address of the ROM code. This information is not encoded in the output file in any way, but can be important for LINK41's resolving of references. With a few exceptions (noted below), most of the other commands cannot occur before the first NEWROM command.

OUTPUT — Designates the output file (extension ".41R" will automatically be appended) to contain the current ROM image (that designated by the most recent NEWROM command). If not specified, a ROM image file for this 4K block is not created.

LOCATE — The address, specified in hex (without a trailing 'H'), determines where the next RElocated file will go. It is analogous to the FILLTO command in the assembler.

CHECKSUM — This command instructs LINK41 to compute a checksum word for the current 4K ROM image. If the optional address is specified, the checksum word will be placed at that address. If not, the checksum word will be placed into its customary position in the last word of the 4K block.

SEARCH — This command can occur before the first NEWROM command. Its action is independent of where it occurs in the command file. The command causes the specified filename to be searched for labels to be resolved. The file MFENTRY.41O, included on disk #2, contains the HP-41

mainframe entry points.

RELOC This command causes the named ".41O" file to be read into the current ROM image (that designated by the most recent NEWROM command). Normally, files are RElocated successively in address space, without any dead space between then. This can be overriden either by the LOCATE command (above) or use of the ORG directive in the assembler file.

LIST XREF If this command occurs before the first NEWROM command, it causes listing of a cross-reference table to occur for all ROM image files. Otherwise it causes listing of a cross-reference table to occur for the current ROM (that defined by the most recent NEWROM command).

SUPPRESS XREF If this command occurs before the first NEWROM command, it suppresses listing of a cross-reference table for all ROM image files (this is the default condition). Otherwise it suppresses listing of a cross-reference table for the current ROM.

COMMENT Causes this line of the command file to be treated as a comment. That is, ignored.

END Indicates the end of the command file; anything left in the file is not read.

? Displays the command list.

The format of the ".41R" file is straightforward: each word of HP-41 ROM is represented by two bytes. The first byte contains the upper two bits, the second contains the lower eight.

### 14.2.3 **ASMBINFO**

The ASMBINFO utility is used to dump information about assembly files and SDS-II microcode files. The invocation is:

ASMBINFO <filename>

Unlike most of the other SDS-II utilities, ASMBINFO requires that the *full filename and extension* be specified. This is because ASMBINFO works on both ".41O" files and ".41T" files containing microcode. It will not work on ".41T" files containing usercode.

The information dumped by ASMBINFO consists of global labels, external references, fixups, and other miscellanea of interest to LINK41 and BUILD.

### 14.2.4 DISASM41

The DISASM41 utility will disassemble an entire file, interpreting it as consisting of HP-41 assembly-language. The invocation is:

DISASM41 <filename> [<filename> [<filename>]]

Like ASMBINFO, DISASM41 requires the full filename; no extension is assumed. This utility is most useful for disassembling ROM image files, which only contain HP-41 code. The overhead contained in other types of files (such as ".41O" and ".41T") not only disassembles into meaningless garbage, but may put the disassembler one byte out of sync when it reaches the actual code.

The output consists of an address, followed by the opcode's representation in hex, octal, decimal, ASCII, LCD characters, and finally, HP-41 mnemonic. Some other points:

- When the target address of a long branch is a recognized mainframe entry point, DISASM41 provides the name of that entry point.
- When a word disassembles into a two-word command, the second word is shown, disassembled, in parentheses.
- DISASM41 does not properly interpret the smart peripheral commands.
- DISASM41 errors out if the file has an odd length.

### 14.2.5 EPROM

The EPROM utility is used by BURN41, hence its inclusion on disk #1. This utility produces EXORMACS format listings of ROM image files. The invocation is:

EPROM [-lhc] [-o <outputfile>] <filename> [<filename>...]

Options:

-l Output low 8 bits of each word.

-h Output high 2 bits of each word, packed four per byte.

-c Output CMT format: low 8 followed by high 2 (unpacked).

-o Send output to specified file. If this option is not used, output is to standard out. By sending output to an asynchronous port, you can download the ROM image to an EPROM programmer through its RS-232 interface.

Following the options is a list of files. This list supports wild-carding. The extension ".41T" is automatically appended to the filenames, so, for example, specifying '*' would specify all ".41R" files in the current directory.

The EXORMACS format output by the program is compatible with many of the EPROM programmers on the market. The data records are of the following format:

S1xxyyyyddddddddd....ddcc

where:

xx is the byte count (3 + number of data-bytes).

yyyy is the start address.

dd are the data bytes.

cc is a checksum of all bytes (including xx and yyyy). The bytes on the line should sum to FFH (without wrap-around carry).

All of the above characters represented by x, y, and d are ASCII representations of a hex nibble: 0-F.

The end record consists of:

S9030000FC

The exact usage of this utility depends on the configuration of your EPROM box.

*** CONTINUE HERE *** with instruction specific to the HHP-????.

### 14.2.6 **EXTRACT**

The EXTRACT utility extracts usercode from a ROM image file in a format compatible with ASSEMB41. This allows you to use READ41P and BUILD to create a ROM-format image of the usercode program (with GOTO's and XROM's compiled, links resolved, etc.), and then incorporate that program into a ROM being developed with ASSEMB41 and LINK41.

The invocation is:

EXTRACT [-<blocknumber>] <funcnumber> <ROMfile> [<ROMfile>]

The optional parameters will never be necessary for a ROM image produced by BUILD, but they add flexibility to the program. First, the mandatory parameters:

FUNCNUMBER — Is the function number to be extracted from the file. If the global label referenced by the function number is not at the beginning of the program, EXTRACT will nevertheless extract the entire program.

ROMfile — The ROM image file from which the program is to be extracted.

Using the mandatory parameters, EXTRACT will extract a program from the 4K ROM image specified by ROMfile. The optional parameters allow specifying two files, for a total 8K image. This is useful for extracting a program which exists in one 4K block but whose FAT entry is in the other 4K block.

The optional <blocknumber> indicates in which 4K block the FAT entry lies. If zero (default), the FAT of the first 4K block is used, if one, the FAT of the second block is used.

EXTRACT sends its output to standard out, which can be redirected to a file with the '>' command line feature of MS-DOS. The output consists of CON statements defining the actual words, with comments (to the right) identifying the usercode being compiled.

A global assembler label is placed at each GLOBAL in the program. EXTRACT is not completely intelligent about this: the label is simply the text of the usercode label, and *might* not be a legal assembler label.

### 14.2.7 MUCODE

The MUCODE utility allows you to create MICROCODE files for inclusion into a ROM image file being created by BUILD. The difference between an assembly file (.410) and a MICROCODE file (.41T) is, largely, the addition of information at the front of the MICROCODE file identifying keywords and interrupt handlers contained therein.

The invocation is:

MUCODE <assemblyfilename> [<microcodefilename>]

The <assemblyfilename> will automatically have the extension ".410" appended. The MICROCODE file will have the same name as the assembly file (if <microcodefilename> is not specified) or <microcodefilename>; in either case, the extension will be ".41T".

In order to create a useful MICROCODE file, it is necessary to identify where the labels occur, and where the entry points are for the interrupt handlers. This is done through the use of special global labels:

xq..... Any global label beginning with "xq" (lowercase only) is recognized by MUCODE as the beginning of a function. The examples subsection of the ASSEMB41 section shows a function, AIP, for which this is done. (MUCODE will examine the code to verify that the microcode label is valid, printing an error message if it is not.)

epPSLOOP If the global label epPSLOOP occurs in the assembly file, it is recognized as the entry point of the interrupt handler for the pause loop interrupt. Please see below for important considerations about writing interrupt handlers.

epMRLOOP Like epPSLOOP, but for the main running loop interrupt.

epDSWNK Like epPSLOOP, but for deep-sleep wakeup no-key interrupt.

epPWROFF Like epPSLOOP, but for the power-off interrupt.

epIOSRV Like epPSLOOP, but for the I/O service interrupt.

epDSWKUP Like epPSLOOP, but for the deep-sleep wakeup interrupt.

epCOLDST Like epPSLOOP, but for the coldstart interrupt.

When processing a file, MUCODE creates a list of function labels and addresses in the format needed by BUILD. It is not necessary (or even possible) to LINK41 the assembly file before using MUCODE; BUILD will resolve all global references between MICROCODE modules. In addition, BUILD contains a list of the mainframe entry points contained in MFENTRY.41A, and will resolve all references to those entry points.

The "encountered order" of labels within the file (as far as BUILD is concerned) is alphabetical order *of the microcode labels*. For example, if function "ABC" occurs at microcode label xqRST, and function "XYZ" occurs at microcode label xqBCD, then "XYZ" appears before "ABC" in the "encountered" order of functions in the file.

Writing interrupt handlers for BUILD is very different from writing conventional interrupt handlers. BUILD is designed to accept an arbitrary number of interrupt handlers; it works by building a table of all interrupt handlers found in the various MICROCODE files, and, through the MCODE driver, calling them all at appropriate times.

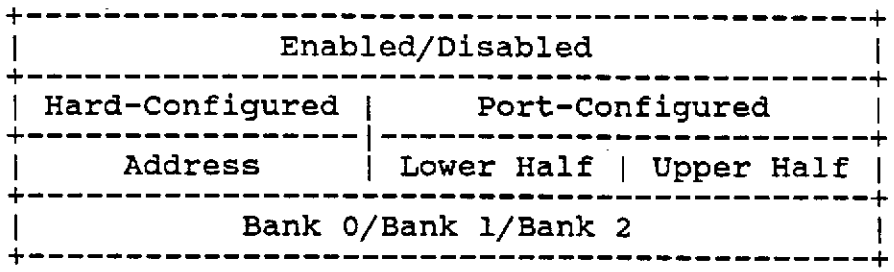
This raises two very important considerations for writing interrupt handlers:

- The handler *must* terminate with GOTOC (returning control to MCODE) instead of the conventional GOLONG =RMCK10.

- The handler *must* preserve the C-register, *and* meet the following return conditions: HEX mode, P selected, status set 0 up, chip 0 selected.

## 14.3 1LG9 Configuration

The 1LG9 12K ROM chip consists of three 4K cores, with a variety of configuration options. For most ROMs being built, the standard configurations shown in the basic bank-switching section of the bank-switching chapter are adequate. However, it is possible to request alternate configurations. For each 4K core, the following options are available:




An explanation of the options:

- If a core is disabled, it does not exist for the HP-41. This is how the 1LG9 is used for 4K and 8K ROMs.

- If the core is hard-configured, a configuration address must be selected (it must be on a 4K boundary).

- If the core is port-configured, it must be configured for the lower or upper half of the port's address space.

- A core (whether hard- or soft-configured) can be placed in:

  bank 0 Always present.

  bank 1 Present at power-up; enabled with ENROM1; disabled with ENROM2.

  bank 2 Not present at power-up; enabled with ENROM2; disabled with ENROM1.

  Note that, as mentioned earlier, the ENROMx instruction only affects the 1LG9 out of which it is read. It can, however, be read out of any core of the target 1LG9 to affect all of the cores.

## A. HP-41 Keycodes

This chart shows the keycodes for the primary (unshifted) keys. The keycode for a shifted key is obtained by prefixing the unshifted keycode with a minus. For example, the keycode for the shifted ENTER^ key is -41.

| *** HP-41C *** | | | | |
|---|---|---|---|---|
| on | user | | prgm | alpha |
| 11 | 12 | 13 | 14 | 15 |
| 21 | 22 | 23 | 24 | 25 |
| | 32 | 33 | 34 | 35 |
| 41 | | 42 | 43 | 44 |
| 51 | 52 | 53 | | 54 |
| 61 | 62 | 63 | | 64 |
| 71 | 72 | 73 | | 74 |
| 81 | 82 | 83 | | 84 |

## B. Special Characters

Following are most of the HP-41 display characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ=? abcde%<>^$-+*/0123456789
```

There are a few special characters, however, that are not defined as part of the ASCII character set, and cannot be displayed or entered on the MS-DOS computer hosting SDS-II. For purposes of data entry and display, the following substitutes are used for these characters:

```
APPEND CHARACTER :: 'x' (normally represented by ASCII 127)
MU               :: 'm' (normally represented by ASCII 12)
NOT EQUAL        :: 'n' (normally represented by ASCII 29)
SIGMA            :: 's' (normally represented by ASCII 126)
ANGLE            :: 'g' (normally represented by ASCII 13)
OVERBAR          :: 'o' (normally represented by ASCII 0)
```

Only three of these characters, NOT EQUAL, SIGMA, and ANGLE are legal characters in a global label; the others can, however, be specified in a header.

In addition, whenever the space character is to be used, either in a program invocation or in a DEFINE file, it is replaced with "." as a placeholder.

EXAMPLE: To read in a program named "A<NOT EQUAL>B" from the HP-41 disk, use:

```
READ41P M:AnB ANEQB
```

The program will be read into READ41P file ANEQB.41T. Note that this does not change the program itself -- the labels will be the same. It merely provides a handle for referencing the programs from the host MS-DOS machine.

EXAMPLE To read in a program named "A B" from the HP-41 disk, use:

```
READ41P C:A.B AB
```

The program will be read into READ41P file AB.41T.

## C. SDS-II ROM Image Submission Form

Include this form with the Custom ROM submission paperwork.

Size of ROM: ___ 4K ___ 8K ___ 12K

Name of BUILD file(s): _____.41R [ROM #1]

_____.41R [ROM #2] (8K & 12K only)

_____.41R [ROM #3] (12K only)

Configuration: ___ Standard ___ Custom:

**FOLLOWING SECTION FOR CUSTOM CONFIGURATION ONLY**

**ROM #1**

___ Bank 0 / ___ Bank 1 / ___ Bank 2

___ Hard-Configured: Configuration Address (in octal): _____

___ Port-Configured: ___ Lower Half / ___ Upper Half

**ROM #2**

___ Bank 0 / ___ Bank 1 / ___ Bank 2

___ Hard-Configured: Configuration Address (in octal): _____

___ Port-Configured: ___ Lower Half / ___ Upper Half

**ROM #3**

___ Bank 0 / ___ Bank 1 / ___ Bank 2

___ Hard-Configured: Configuration Address (in octal): _____

___ Port-Configured: ___ Lower Half / ___ Upper Half

## D. Handing STACK OVERFLOW Errors

Although unlikely, a STACK OVERFLOW error can occur with the SDS-II utilities. In this case, the problem can usually be corrected by adding "=<stacksize>" to the command line, where <stacksize> is the size of stack the program should use. The default stack size is 2048 bytes.

Several of the SDS-II utilities use unbalanced binary trees as data structures. The recursion used in traversing such a tree can be responsible for a stack overflow if the tree is filled in a worst-case or near worst-case order. This might occur in ASSEMB41 if the file being assembled contains hundreds of labels, and they occur in alphabetical or reverse alphabetical order. Other than ASSEMB41, the default stack space in SDS-II utilities is believed to be sufficient for worst-case behavior.

## E. Program Invocation Summary

Where a specific filename extension is specified, that extension is assumed by the utility. Where "ext" is indicated, extension must be specified in the command line.

**ASMBINFO <file.ext>**

**ASSEMB41 [-ls8] [-o <object.41O>] <source.41A>**

**BUILD <DEFINE_file_name> <output_file.41R>**
(<output_file> name appended by BUILD with 0, 1, or 2)

**BURN41 ????**

**CHECKSUM <file.41R> [<file.41R>...]**
(wild-carding supported in file name)

**DISASM41 <file.ext> [<file.ext> [<file.ext>]]**
(wild-carding supported in file name; max 12K words)

**EPROM <file.41R> [<file.41R>...]**
(wild-carding supported in file name)

**EXTRACT [-<blocknumber>] <funcnumber> <file.41R> [<file.41R>]**

**LIFPACK <disk_designator>**
(disk_designator must include ':')

**LINK41 [<command_file>]**

**LISTFAT <file.41R> [<file.41R>]**
(wild-carding supported in file name; max 8K words)

**MUCODE <file.41O> [<file.41T>]**

**READ41P <disk>:<41_prog_name> <file.41T>**

**READMLDL <disk>:<filename> <file.41R>**

**SDSCAT <disk_designator>**

**WRITMLDL <file.41R> <disk>:<filename>**

## F. ROM ID Allocation

In order to prevent unpredictable results, it is important that all of the ROMs plugged into the machine at a given time have a different ROM ID. Since every one of the available 31 ROM ID's has been used (some many times), it is difficult to recommend a good choice.

Following is a list of ROM ID's used in ROMs from HP:

| ROM ID | Assignment |
|---|---|
| 1 | Math |
| 2 | Statistics |
| 3 | Surveying |
| 4 | Finance |
| 5 | Standard |
| 6 | Circuit Analysis |
| 7 | Structures |
| 8 | Stress Analysis |
| 9 | Home Management |
| 10 | Games |
| 11 | Real Estate |
| 12 | Machine Design |
| 13 | Thermal and Transport Sciences |
| 14 | Navigation |
| 15 | Petroleum |
| 16 | Petroleum |
| 17 | Plotter |
| 18 | Plotter |
| 19 | Securities Structures |

| ROM ID | Assignment |
|---|---|
| | Clinical Lab<br>Aviation |
| 20 | |
| 21 | Reserved for custom modules |
| 22 | HP-IL Development<br>ADVANTAGE |
| 23 | Extended I/O |
| 24 | HP-IL Development<br>ADVANTAGE |
| 25 | Extended Functions |
| 26 | Time |
| 27 | Wand |
| 28 | Mass Storage |
| 29 | Printer |
| 30 | Card Reader |
| 31 | Reserved for custom modules |

## G. Contents of Disks

The two disks shipped with SDS-II contain the following files:

### G.1 Disk 1

BUILD.EXE
BURN41.EXE
CHECKSUM.EXE
EPROM.EXE
LIFPACK.EXE
LISTFAT.EXE
READ41P.EXE
SDSCAT.EXE
AIP.41T
ALEN.41T
ALENG.41T
ALNAM2.41T
ANUM.41T
AROT.41T
ATOX.41T
AUTOST.41T
BIND.41T
CLKEYS.41T
ENROM1.41T
ENROM2.41T
GETKEY.41T
PASN.41T
PCLPS.41T
POSA.41T
PSIZE.41T
RCLSTFLG.41T
REGMVSWP.41T
SIZE.41T
XB.41T
XF.41T
XTOA.41T

### G.2 Disk 2

ASMBINFO.EXE
ASSEMB41.EXE
DISASM41.EXE
EXTRACT.EXE
LINK41.EXE
MUCODE.EXE
READMLDL.EXE
WRITMLDL.EXE
MFENTRY.41A