SDS-II

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

Owner's Manual

June, 1986

NOTICE

Hewlett-Packard Company makes no expressed or implied warranty with regard to the documentation and program material offered or to the fitness of such material for any particular purpose. The documentation and program material is made available solely on an "as is" basis, and the entire risk as to its quality and performance is with the user. Should the documentation and program material prove defective, the user (and not Hewlett-Packard Company or any other party) shall bear the entire cost of all necessary corrections and all incidental damages in connection with or arising out of the furnishing, use, or performance of the documentation and program material.

Printing History

Edition 1.....May, 1986
Edition 2.....June, 1986

Contents

LIST OF TABLES

1. Introduction

The new Software Development System (SDS-II) provides the necessary tools to collect and prepare your HP-41 programs for translation into an HP-41 ROM (Read-Only Memory) plug-in. Each plug-in can contain 4K, 8K or 12K-words of HP-41 usercode and/or microcode.

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

1. Writing HP-41 programs on either:
   - The HP-41 calculator, saving the results on a disc or cassette, or
   - Your host MS-DOS machine, utilizing the User Code Compiler.
2. Preparing your HP-41 programs for the SDS-II development system (the **READ41P** process).
3. 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 separate literature.

2. Contents of SDS-II

2.1 What SDS-II Includes

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

- Two 3½" single-sided microfloppy MS-DOS discs, or
- Two 5¼" double-sided floppy MS-DOS discs in low-density (360K) format.

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

2.2 What SDS-II Does Not Include

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

1. Tools for developing HP-41 code. Either:
   - An HP-41C/CV/CX calculator and HP-IL interface module (HP-82160A).
   - Mass storage for the HP-41 (HP-82161A cassette drive or HP-9114 disc drive). The HP-9114 is strongly recommended over the HP-82161A.
   - Depending on your configuration, you may require an accessory for communicating with the HP-41 mass-storage device (see chapter 4 for more details).

   or:
   - A User Code Compiler[1] running on your host MS-DOS system.
2. 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. If you do not already have an EPROM programmer, we recommend the Data I/O 21A, a recently introduced, powerful, inexpensive product; this manual includes some instructions specific to using the 21A. For emulation, SDS-II supports and recommends the ERAMCO 16K bank-switching EPROM box because of its capability to emulate HP bank-switching ROMs.
   - If you are using RAM for ROM emulation, you need a RAM box and the facilities to load it. Recommended is the ERAMCO 16K RAM Storage Unit, which provides the same emulation capabilities as the 16K EPROM box. Use of this product requires the ERAMCO MLDL software, usually distributed in the ERAMCO ESMLDL 1. (Note: Because of the lower cost and higher reliability of EPROM boxes, their non-volatility, and the ability to distribute software updates without requiring an MLDL box, we recommend EPROM boxes over RAM boxes for emulation. However, support is provided for RAM-based emulation, as explained in section 13.2.)
3. An interface for communicating with the EPROM programmer (typically an asynchronous communications port).

---

1. From HandHeld Products, Inc., 6201 Fair Valley Drive, Charlotte, NC 28211.

3. 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. This results in an approximately 20x speed improvement.
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, MEMOMAKER, 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 chapter 11.
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 chapter 13.

4. 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:

1. Receiving programs from your HP-41, in which SDS-II reads the HP-41 disc or cassette tape. This step is not required if you are developing usercode with the User Code Compiler.
2. Controlling the EPROM programmer, in order to program EPROMs for testing your software in an EPROM box. The details of this task are dependent on your configuration and your particular brand of EPROM programmer. Some instructions are provided in this chapter specifically for the Data I/O 21A.

4.1 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.

4.1.1 Receiving from HP-41

If you are using a HP-9114 disc drive with your HP-41, and your HP-150 has a double-sided micro-floppy (3½") disc drive, you can directly read the HP-41 disc without any extra communications hardware.

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

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

   DEVICE = HPIL150.SYS

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

   Some important details to keep in mind:

   - ☞ If the CONFIG.SYS file contains a **SHELL** command, the **DEVICE** command must occur *before* the **SHELL** command.
   - ☞ Some editors (notably EMACS) do not automatically append a trailing <CR><LF> to the last line of a file. The **DEVICE** command 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 disc drive IDs "A:" through "L:", mass-storage devices on the loop are named "M:" through "T:". For example, if the loop contains a single HP-9114 disc 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.

4.1.2 Controlling the EPROM Programmer

Most EPROM programmers communicate with a host computer through an asynchronous (RS-232) interface. The Data I/O 21A is capable of such communications, without hardware handshaking, at data rates up to 4800 baud. Following are instructions specific to the 21A — they *might* be useful in configuring other models or brands.

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:

TABLE 1. Connections between HP-150 and Data I/O 21A Programmer

Male to HP-150	Male to 21A	Function
3	2	Programmer→Computer
2	3	Computer→Programmer
	4-5	Tie RTS to CTS
7	7	Signal Ground

4.2 Configuring the HP-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.

4.2.1 Receiving from HP-41

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

4.2.2 Controlling the EPROM Programmer

Most EPROM programmers communicate with a host computer through an asynchronous (RS-232) interface. The Data I/O 21A is capable of such communications, without hardware handshaking, at data rates up to 4800 baud. Following are instructions specific to the 21A — they *might* be useful in configuring other models or brands.

The optional RS-232 cable for the HP-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:

TABLE 2. Connections between HP-Portable and Data I/O 21A Programmer

Female to Portable cable	Male to 21A	Function
3	2	Programmer→Computer
2	3	Computer→Programmer
	4-5	Tie RTS to CTS
7	7	Signal Ground

4.3 Configuring IBM, Vectra, and IBM-Compatible Personal Computers

4.3.1 Receiving from HP-41

If you are using a HP-9114 disc drive with your HP-41, and your SDS-II host system includes a double-sided micro-floppy (3½") disc drive, you can directly read the HP-41 disc without any extra communications hardware.

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

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

   DEVICE = HPIL.SYS

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

   Some important details to keep in mind:

   - ☞ If the CONFIG.SYS file contains a **SHELL** command, the **DEVICE** command must occur *before* the **SHELL** command.
   - ☞ Some editors (notably EMACS) do not automatically append a trailing <CR><LF> to the last line of a file. The **DEVICE** command 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 discs as "C:" (first drive on the loop) through "J:" (eighth drive on the loop). A typical IBM PC/AT with a hard disc drive at "C:" will address discs 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.

4.3.2 Controlling the EPROM Programmer

Most EPROM programmers communicate with a host computer through an asynchronous (RS-232)

interface. The Data I/O 21A is capable of such communications, without hardware handshaking, at data rates up to 4800 baud. Following are instructions specific to the 21A — they *might* be useful in configuring other models or brands.

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

- A built-in 25-pin interface, and
- A 9-pin "D-shell" interface, which requires an adapter cable.

Both the cable and the built-in interface terminate with an RS-232 connector (male or female) configured for DTE. Communications with the Data I/O 21A requires an 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.

While a generic BURN program might not require the handshaking connections to the EPROM programmer, the COMx drivers *do* require these connections. The connections shown below to pins 5 and 6 on the computer side (shown in boldface) are needed to allow the EPROM utility (explained in section 10.2) to send its output directly to COM1 or COM2.

The specific connections are:

TABLE 3. Connections between computer and Data I/O 21A Programmer

Female or Male to cable or computer	Male to 21A	Function
3	2	Programmer→Computer
2	3	Computer→Programmer
5	4-5	Tie RTS to CTS
		Programmer RTS to Computer CTS
6	**20**	**Programmer DTR to Computer DSR**
7	7	Signal Ground

5. 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 chapter 8). The first step is, of course, to write and save your software:

- If you are developing the software on an HP-41, save it on your mass-storage medium (disc or cassette) using the HP-41 WRTP command. SDS-II will read this medium using the procedures outlined in chapter 6.
- If you are developing the software with the User Code Compiler on your DOS machine, SDS-II will process the **.BIN** files created by the compiler (this procedure is described in section 13.1).

At this stage, the program is not in ROM form. Before you have a ROM image, SDS-II will pack the program, compile GTOs and XEQs 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. These steps are handled by the **READ41P** and **BUILD** procedures, described in chapters 6 and 7.

6. Step 2 — Reading HP-41 Software into SDS-II

Each program created on your HP-41 must be read into SDS-II using the **READ41P** program. **READ41P** has two modes of operation: it can read HP-41 programs from an HP-41 disc (described presently) or it can read **.BIN** files created by the HP-41 User Code Compiler (described in section 13.1). Whichever mode is used, this chapter contains important information about the operation and output of **READ41P**.

READ41P is included on the SDS-II distribution disc #1, and is invoked from the MS-DOS environment as:

READ41P <device>:<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 command line. If the <programname> contains HP-41 special characters (≠, ∠, or Σ), use the substitute characters explained in appendix C.

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

EXAMPLE 1: The command

READ41P M:XYZ XYZ

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

EXAMPLE 2: The command

READ41P B:A.B C:AB >PRN

will read program "A B"[2] from the disc in drive "B:", process it, and produce a **READ41P** file named "AB.41T" in the current directory on disc "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.

2. Substitution of the '.' character for blank is explained in appendix C.

- The program contains an unresolved reference to a local label.
- A global label or an XEQ or GTO referencing a global label contains an illegal character.
- The program contains more than 64 global 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

☞ *MS-DOS does not support the same filename characters as does the HP-41.* It is often impossible to use the HP-41 filename as the MS-DOS filename, either because of upper/lower case differences, name conflicts, or special characters (such as '=', '?', or ' '). You should therefore choose a name for the **READ41P** output file that reasonably resembles the original HP-41 program name, but conforms to MS-DOS file-naming conventions. The MS-DOS filename chosen has no effect on the contents (including labels) of the program.

Failure to specify a legal MS-DOS filename as the second command-line parameter will cause **READ41P** *to fail to open its output file.*

☞ The HP-41 program name in the **READ41P** command line 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.

☞ 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 XEQs into XROM references.
- 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**).

7. 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 up to six 4K ROM images, for programming into one or two bank-switching ROMs.[3]

The following command causes a ROM image to be built:

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

Using commands in the DEFINE file, **BUILD** collects the **READ41P** files (and microcode files, explained below) together into ROM image files. **BUILD** also creates a configuration file describing the programming configuration for the 1LG9 ROMs being programmed.

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 compiles label references, converts XEQs to XROMs, and so on.

7.1 Two Types of .41T Files

So far this document has dealt with **READ41P** files, which are created by the **READ41P** utility. 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 chapter 8; this chapter will restrict its discussion and examples to usercode.

7.2 The ROM Image Files

BUILD will create from one to six ROM image files. The files will be named with the first seven characters of <ROM-file-name> appended by the ROM sequence number (that is, 0, 1, 2, 3, 4, or 5), with an extension of "41R". These files are ready to be programmed into EPROMs for testing[4] and, eventually, programmed into custom ROMs.

7.3 The Configuration File

BUILD will create a configuration file containing text describing the configuration of the ROMs being defined. This file is used by HP, when the ROM is submitted, for purposes of programming the 1LG9 ROM(s). **BUILD** creates this file automatically; you must create it yourself if you are building a ROM using the advanced tools. The exact contents of the configuration file are described in appendix G.

The configuration file is named <ROM-file-name> (from the command line) with filename extension "41F".

7.4 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, MEMOMAKER, EMACS, etc.). For each 4K ROM image, the DEFINE file contains several parts, which *must* occur in the order shown:

---

3. For more information about bank-switching and the 1LG9 ROM, see chapter 11.
4. EPROM ROM emulation is explained in chapters 9 and 10.

1. A **ROM#** command with optional ROM header, optional privacy specifier, and optional configuration specifier. This is followed by a list of **READ41P** and microcode files.
2. An optional **ORDER** command, specifying how the global labels are to be ordered within the ROM catalog. This is sometimes followed by a list of labels and headers.
3. An optional **XEQ** command, used to specify any labels for which XEQs will *not* be converted into XROMs. This is followed by a list of labels.
4. An optional KEYS command, used to specify key assignments to be set up by the ROM. This is followed by a list of key assignments.
5. Optional comments anywhere within the DEFINE file.

All commands and comments are preceded by the '&' character. The following sections describe each command and its section of the DEFINE file.

7.4.1 ROM# Command

The ROM# command begins defining the characteristics of a 4K ROM image. It occurs once in the command file for every 4K ROM image being defined. This command is followed by a list of **READ41P** and microcode files to be included in the ROM.

For each ROM# command, four attributes can be specified; three of them are optional, with default settings if not specified. As with all commands, this command begins with '&' *without* a trailing space. The clauses of this command *must* occur in the order shown:

1. The **ROM#** command *must* begin with a clause specifying the ROM number — a value between 0 and 31. A ROM number of zero is permissible *only* if the ROM image contains no functions or headers. The form of this clause is:

   ROM# = <romnumber>

   It is followed by a comma if any of the optional clauses (below) is to be specified.

2. The optional HEADER clause specifies a header that is to occupy function #0 of this ROM image. It is generally *very* desirable to define a header for at least the first ROM image of a plug-in; because this header is >7 characters long (blank-padded by **BUILD**, if necessary), it is found by the CAT 2 function on the HP-41CX. If the HEADER clause is not specified, the ROM image does not have a catalog header. Maximum length of the header is 11 characters. The format of the HEADER clause is:

   HEADER=<header>

   Wherever a blank is desired in the <header> field, it is represented by a '.' (period). A true blank terminates the <header> field. The HEADER clause is followed by a comma if any of the other optional clauses is to be specified.

3. The optional PRIVATE clause specifies that the usercode programs in this ROM image are private, and cannot be copied or viewed. Specifying PRIVATE causes **BUILD** to set some bits in the programs, and to include a microcode file (explained in the chapter 8). If the PRIVATE clause is not specified, the usercode files in the ROM image are not private — they can be copied, viewed, and single-stepped. The format of the PRIVATE clause is:

   PRIVATE

   This clause must be specified for *every* ROM image in which it is desired (i.e., in every **ROM#** command). This clause is followed by a comma if the optional CONFIG clause is to be specified.

4. The optional CONFIG clause must be specified in *none* of the **ROM#** commands or in *all* of them. If this clause is not specified, the ROM images will assume a default (and usually reasonable) configuration, as shown in appendix F. If the clause is specified, it takes the following format:

   CONFIG=<configuration>

   where <configuration> is a one- to three-character string of the following form (alternate choices for each character are stacked vertically):

   $$\begin{bmatrix}1\\2\end{bmatrix}\ \begin{matrix}L\\U\end{matrix}\ \begin{bmatrix}0\\1\\2\end{bmatrix}$$

   The first character specifies the chip number the ROM image is to occupy. If one, two or three ROM images are being specified, they will generally all occupy chip #1 (requiring bank-switching for the three-ROM case). Since a 1LG9 chip cannot hold more than 12K, defining more than three ROM images requires two chips (i.e., two plug-in modules). This character is optional; if not specified, it defaults to 1.

   The second character specifies whether the ROM image is to occupy the Lower or Upper half of the port address space. It is not optional.

   The third character specifies the bank number to be occupied by the ROM image (see chapter 11 for more information). It is optional; if not specified, it defaults to 0.

EXAMPLES

The following examples illustrate use of the **ROM#** command:

&ROM# = 21

ROM number is 21, no header specified, not private, default configuration as explained in appendix F.

&ROM# = 21, HEADER = --MY.ROM, PRIVATE, CONFIG = L

ROM number is 21; header is "--MY ROM"; usercode files are private; this ROM image occupies the first (and probably only) 1LG9 chip, the lower half of the address space, bank 0.

&ROM# = 31, HEADER = SERENDIPITY, CONFIG = 2U2

This is probably the last 4K of a 24K custom ROM. The ROM number is 31; header is "SERENDIPITY", usercode files are not private, it occupies bank 2 of the upper half of the *second* 1LG9 chip.

7.4.2 ORDER Command

This command allows you to specify the order in which your global labels will appear in the ROM catalog. If you do not include this command, the labels will appear in the order in which they are encountered while reading the **READ41P** files. If this command 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 (Σ, ∠, and ≠) are sorted

according to their internal HP-41 representation: Σ as ASCII 126, ∠ as ASCII 13, ≠ 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 command 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 example below for an illustration.

7.4.3 XEQ Command

Normally, all XEQ's that refer to global 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 main 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 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 command allows you to specify that certain XEQs not be compiled into XROM references.

The form of the **XEQ** command is:

&XEQ

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

Alternatively, specifying:

&XEQ ALL

will prevent all XEQs from being compiled.

7.4.4 KEYS Command

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

&KEYS

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

- Assigning an HP-41 function to a key:

  <function-name> <keycode>

- Assigning an XROM function to a key (for example, a card reader function):

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

- 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 that is displayed by the ASN function. A map of keycodes in shown in appendix B.

- ☞ 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 section 8.3.3.
- ☞ 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 overridden by assignments in the first ROM.

7.4.5 Comments

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

&& <comment>

7.5 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 file **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 Σ+ 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 29,20 (PRX from printer ROM) to ENTER^ key
XROM 29 20 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*).

7.6 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. You should investigate all warnings to insure that you have not introduced an inadvertent error.

NOTICES Less serious than a warning, but a condition to be noted. You should investigate all notices 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.

7.6.1 Errors in DEFINE File

ERROR: cannot open READ41P file <filename>
Indicates that the specified file 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** clause or in an **ORDER=S** list) contains characters that are not legal in program names.

ERROR: Duplicate ROM ID
A **ROM#** command has specified the same ROM ID for more than one ROM. This error is only a warning if the duplicate IDs occur in two complementary switched banks.

ERROR: Non-default configuration partly specified
The CONFIG clause was specified in some, but not all, of the **ROM#** commands.

ERROR: Illegal 1LG9 configuration: <message>
An illegal configuration was specified. For example:

- More than three ROM images in one 1LG9 chip.
- Overlapping ROM images on a given page (upper or lower half).
- Bank 1 without bank 2 (or vice versa) on a given page.

ERROR: Expected <message>
Indicates that something unexpected was encountered in the DEFINE file. The message will indicate on what line the error occurred. One possible <message> is **end-of-file**, which appears if a **ROM#** command is encountered after six **ROM#** commands have already been processed — **BUILD** can define at most six ROMs.

7.6.2 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.

7.6.3 Key Definition Errors

ERROR: Illegal key definition for XROM <xx>,<yy>
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 mainframe 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 appendix B 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).

7.6.4 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

7.6.5 ROM ID = 0

ERROR: &ORDER=S not allowed with ROM ID = 0

An **ORDER=S** command is not valid if the ROM ID specified in the **ROM#** command 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#** command if the ROM ID is zero.

7.6.6 Specify Errors

The following errors can occur if you use the **ORDER=S** command:

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
The specified label does not exist in this ROM.

SPECIFY ERROR: Label "<label>" already specified
This 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.

7.6.7 XEQ Errors

The following error can occur if you use the **XEQ** command:

XEQ ERROR: Label "<label>" does not exist in this ROM
The specified label does not exist.

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

7.6.8 Label ERRORS

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

ERROR: Too many labels in ROM
The number of labels + headers in the ROM exceeds 64.

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.

7.6.9 Errors in HP-41 Program

WARNING: Unresolved XEQ "<label>" on program line #<line#>
An XEQ references an alpha 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#>
A GTO references an alpha 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 pgm line #<line#> > 127 bytes (by <# bytes>), not compiled
A two-byte 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>,<yy> 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 GTO or XEQ references a local label that does not exist. Since this situation is also trapped in **READ41P**, this error should never occur.

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

7.6.10 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. Section 8.4 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> defined more than once
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 label defined more than once 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.

7.7 Fatal BUILD 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 disc 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 disc,

regardless of where the actual ROM image files are being created).

- Default disc is out of disc 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.

8. 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:

1. **Definition of new keywords.** A microcode file can add a new function to the HP-41.
2. **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 disc #1. Privacy and key processing are special cases of microcode files. Information on creating your own microcode files is contained in sections 13.3.1 and 13.3.6.

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.

8.1 Type-2 Microcode Files

Most of the type-2 microcode files implement popular functions that are already available in the Extended Functions ROM and the HP-41CX. By including these functions in your ROM, however, you make them available for your application on any version of the HP-41.

WARNING

> When using one of these functions, it is important that your program contain an XEQ, not an XROM reference. For example, if using the [ALENG] function in your application, your program must contain **XEQ "ALENG"**, *not* **XROM 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 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.

8.1.1 AIP (Alpha Integer Part)

Not from the Extended Functions ROM.

[AIP] 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.

8.1.2 ALENG (Alpha Length)

This function exists in the Extended Functions ROM and the HP-41CX.

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

8.1.3 ANUM (Alpha Number)

This function exists in the Extended Functions ROM and the HP-41CX.

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):

TABLE 4. Effect of flags 28 and 29 on ANUM function

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.

For more detailed information on the operation of ANUM, see volume 2 of the *HP-41CX Owner's Manual.*

8.1.4 AROT (Alpha Rotate)

This function exists in the Extended Functions ROM and the HP-41CX.

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).

8.1.5 ATOX (Alpha to X)

This function exists in the Extended Functions ROM and the HP-41CX.

ATOX shifts the leftmost character out of the Alpha register and returns its character code in the X-register. If the Alpha register is empty, the function returns 0.

8.1.6 BININ (Binary Input)

Not from the Extended Functions ROM.

BININ is the BINary INput function, in which the HP-41 keyboard is used to input up to a 10-bit number in binary format. Keys 2 through 9 are inactive, and digit entry terminates when a non-digit key is pressed. Upon termination of digit entry, the entered number is converted into the internal HP-41 floating-point format.

8.1.7 BINVIEW (Binary View)

Not from the Extended Functions ROM.

BINVIEW , the BINary VIEW function, displays X in binary format, ignoring the sign and the fractional part of X. BINVIEW uses flag 29 and the current display setting to determine placement of the digit separators in the displayed value. If $|X|>1023$, BINVIEW will report an OUT OF RANGE error.

8.1.8 CLKEYS (Clear Keys)

This function exists in the Extended Functions ROM and the HP-41CX.

CLKEYS clears all USER key assignments.

8.1.9 CLRGX (Clear Registers by X)

Not from the Extended Functions ROM. This function is in the HP-41CX.

CLRGX clears a block of registers. The X-register contains a control number *bbb.eeeii*, where:

- R_{bbb} (*begin*) is the first (smallest-addressed) register to be cleared;
- R_{eee} (*end*) is the last (largest-addressed) register to be cleared;
- *ii* is the *increment* if you want only every ii^{th} register cleared. If you don't specify *ii* (i.e., it is zero), the computer assumes $ii = 01$.

R_{bbb} is cleared even if $bbb>eee$ or $bbb+ii>eee$. The sign of the control number and any excess fractional digits are ignored.

If R_{bbb} or R_{eee} is non-existent, no registers are cleared and a NONEXISTENT error occurs.

8.1.10 ENROM1 (Enable ROM 1)

Not from the Extended Functions ROM.

ENROM1 is used to enable ROM 1 when a 12K (bank-switching) ROM is being used. It should be placed *only* in a non-bank-switching ROM core. For more information, see chapter 11.

8.1.11 ENROM2 (Enable ROM 2)

Not from the Extended Functions ROM.

ENROM2 is used to enable ROM 2 when a 12K (bank-switching) ROM is being used. It should be placed *only* in a non-bank-switching ROM core. For more information, see chapter 11.

8.1.12 GETKEY (Get Key)

This function exists in the Extended Functions ROM and the HP-41CX.

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.

8.1.13 HEXIN (Hexadecimal Input)

Not from the Extended Functions ROM.

[HEXIN] is the HEXadecimal INput function, in which the HP-41 keyboard is used to input up to a 32-bit number in hexadecimal format. During hex input, the A through F keys are active as digit entry keys, and digit entry terminates when a non-digit-entry key is pressed. Upon termination of digit entry, the entered number is converted into the internal HP-41 floating-point format.

8.1.14 HEXVIEW (Hex View)

Not from the Extended Functions ROM.

[HEXVIEW] , the HEXadecimal VIEW function, displays X in hex format, ignoring the sign and the fractional part of X. [HEXVIEW] uses flag 29 and the current display setting to determine placement of the digit separators in the displayed value. If $|X|>4,294,967,295$ (2^{32}-1), [HEXVIEW] will report an OUT OF RANGE error.

8.1.15 OCTIN (Octal Input)

Not from the Extended Functions ROM.

[OCTIN] is the OCTal INput function, in which the HP-41 keyboard is used to input up to a 30-bit number in octal format. Keys 8 and 9 are inactive, and digit entry terminates when a non-digit key is pressed. Upon termination of digit entry, the entered number is converted into the internal HP-41 floating-point format.

8.1.16 OCTVIEW Octal View)

Not from the Extended Functions ROM.

[OCTVIEW] , the OCTal VIEW function, displays X in octal format, ignoring the sign and the fractional part of X. [OCTVIEW] uses flag 29 and the current display setting to determine placement of the digit separators in the displayed value. If $|X|>1,073,741,823$ (2^{30}-1), [OCTVIEW] will report an OUT OF RANGE error.

8.1.17 PASN (Programmable Assign)

This function exists in the Extended Functions ROM and the HP-41CX.

[PASN] allows you to make key assignments under program control. To make an assignment:

1. Enter the function name or global label into the Alpha register.
2. Enter the keycode of the key to be redefined (using the keycodes shown in appendix B) in the X-register.
3. Execute [PASN] .

8.1.18 PCLPS (Programmable Clear Programs)

This function exists in the Extended Functions ROM and the HP-41CX.

PCLPS clears one or more of the programs in main memory. To clear a program and all subsequent programs in program memory:

1. Place any global label from the program in the Alpha register.
2. Execute PCLPS.

Executing PCLPS when the Alpha register is empty clears the current program and all subsequent programs.

If the current program is removed by PCLPS, execution stops immediately.

8.1.19 POSA (Position in Alpha)

This function exists in the Extended Functions ROM and the HP-41CX.

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, the function returns -1.

8.1.20 PSIZE (Programmable Size)

This function exists in the Extended Functions ROM and the HP-41CX.

PSIZE works like the SIZE function provided with the calculator except that it can be executed from within a program. This 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.

8.1.21 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. After executing RCLFLAG, the display is not 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 flag numbers *bb* through *ee* from the data in the Y-register.

8.1.22 REGMVSWP (Register Move/Swap)

These functions exist in the Extended Functions ROM.

This file provides two functions: [REGMOVE] and [REGSWAP].

Both functions take an argument of the form *sss.dddnnn* in the X-register.

[REGMOVE] copies a block of *nnn* registers beginning at register R_{sss} to a block of the same length beginning at register R_{ddd}. Any data that was already in the destination block is lost. For example, to move ten registers of data from registers 2 through 11 to registers 20 through 29, place 2.020010 in the X-register and execute [REGMOVE].

[REGSWAP] exchanges the contents of a block of *nnn* registers beginning at register R_{sss}, with the contents of a block of the same length beginning at register R_{ddd}. Executing [REGSWAP] with 2.020010 in the X-register will exchange registers 2 through 11 with registers 20 through 29.

8.1.23 SIZE (Determine Current SIZE)

This function exists in the Extended Functions ROM and the HP-41CX.

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.

8.1.24 XTOA (X to Alpha)

This function exists in the Extended Functions ROM and the HP-41CX.

[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] can take 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*.

8.1.25 XF (X Exchange Flags)

This function exists in the Extended Functions ROM and the HP-41CX.

The microcode file XF provides the function [X<>F]. [X<>F] uses the number in the X-register to set flags 00 through 07. 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:

$$\sum_{i=0}^{i=7} x_i, \text{ where } \begin{cases} x_i = 0, \text{ if flag } i \text{ is clear} \\ x_i = 2^i, \text{ if flag } i \text{ is clear} \end{cases}$$

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:

TABLE 5. Computing numeric equivalents of flags

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 00 through 07 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 00 through 07 in a register. For example, to check the status of an extended flag, recall the flag status code into 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.

8.2 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 the files containing those functions 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.

TABLE 6. Labels Defined in Type-1 Microcode Files

Microcode File	Labels Defined
ALEN	ALEN,CNTBYT,FAHED
ALNAM2	ALNAM2
BIND	BIN_D
CSKBD	BININ0,BININ1
PNCTUA	PNCTUA
XB	X_256,X_999
XVIEW	HEXVU0,REGHEX

8.3 Type-0 Microcode Files

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

8.3.1 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.

8.3.2 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 (the MCODE table is explained below).

8.3.3 KEYASN

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

8.3.4 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. MCODE's length is variable and dependent on the ROM's configuration:

- If the ROM core is bank-switching, MCODE begins at ROM address 4014 (0FAEH) and ends at 4083 (0FF3H).

- Otherwise, MCODE begins at ROM address 4020 (0FB4H) and ends at 4083 (0FF3H).

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 section 13.3.6.

8.4 Microcode Library File Requirements

For each file in the microcode library, the following table 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.

TABLE 7. ROM usage and dependencies of microcode files

Microcode File	Type	Bytes Required	Dependencies
AIP	2	29	
ALEN	1	81	
ALENG	2	13	ALEN BIND
ALNAM2	1	98	
ANUM	2	114	ALEN
AROT	2	49	ALEN BIND
ATOX	2	23	ALEN BIND
AUTOST	0	98	(MCODE)
BIND	1	28	
BININ	2	11	CSKBD PNCTUA
BINVIEW	2	47	XVIEW PNCTUA
CLKEYS	2	55	
CLRGX	2	107	
CSKBD	1	405	PNCTUA
ENROM1	2	9	
ENROM2	2	9	
GETKEY	2	60	
HEXIN	2	12	CSKBD PNCTUA
HEXVIEW	2	18	XVIEW PNCTUA
KEYASN	0	see	(MCODE)

Microcode File	Type	Bytes Required	Dependencies
		text	
MCODE	0	see text	
OCTIN	2	11	CSKBD PNCTUA
OCTVIEW	2	49	XVIEW PNCTUA
PASN	2	83	ALNAM2
PCLPS	2	127	ALNAM2
PNCTUA	1	58	
POSA	2	84	ALEN BIND XB
PRIVACY	0	13	(MCODE)
PSIZE	2	95	XB
RCLSTFLG	2	108	
REGMVSWP	2	121	
SIZE	2	19	BIND
XB	1	29	
XF	2	42	BIND XB
XTOA	2	18	XB
XVIEW	1	99	PNCTUA

9. Emulating ROMs

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

9.1 EPROM Box ROM Emulation

A number of EPROM boxes are commercially available for emulating HP-41 ROMs. Because they are using commercial EPROMs with 8-bit words to emulate the HP-41 10-bit words, these products usually rely on one of two common schemes:

- One EPROM contains the lower 8 bits of each word, another EPROM contains the upper 2 bits of each word (packed 4 words/byte). Products employing this scheme include the EPROM boxes from ERAMCO and HandHeld Products.
- Each word of HP-41 ROM is contained in two words of EPROM: the first byte contains the lower 8 bits, the second byte contains the upper two bits. This scheme is used in the EPROM boxes from CMT.[5]

Chapter 10 explains how the SDS-II tools can generate EPROM patterns for both of these schemes.

Of the various EPROM products available, we recommend the ERAMCO 16K EPROM box because of its ability to emulate the bank-switching properties of the HP 1LG9 12K ROM. Specifics on using the ERAMCO 16K box are provided in chapter 10.

9.2 RAM Box Emulation

A number of RAM boxes are commercially available for emulating HP-41 ROMs. Typically, these boxes must be programmed by the HP-41 using specialized software. For example, the ERAMCO 16K RAM box can be programmed from a ROM image file (on a LIF medium) through use of the GETROM keyword in the ERAMCO MLDL operating system. For more information on using RAM boxes, see section 13.2.

5. Corvallis MicroTechnology, Inc., 33815 Eastgate Circle, Corvallis, OR 97333.

10. Using EPROM Boxes

This chapter addresses two topics:

1. How to burn EPROMs.
2. How to use EPROM boxes.

While this chapter will concentrate on the EPROM products recommended earlier in this document, the material is applicable (in some degree) to any EPROM emulation setup.

10.1 Connecting EPROM Burners to the Host Computer

Chapter 4 discussed how to connect the Data I/O 21A EPROM programmer to your MS-DOS system. Assuming you have now completed that task with your 21A (or whatever EPROM programmer you are using), you must configure your system to communicate properly.

The specifics of this will depend on your EPROM burner. For the Data I/O unit, communications can be performed at up to 4800 baud without any hardware handshaking. A good choice of protocol would be 8-bit, no parity. To select these options on the 21A, press the following keys:

SELECT A 480 Δ 34 SET

The RS-232 interface on the host computer must be set to a matching protocol and baud rate, and communications can proceed — once a data format has been selected.

The Data I/O unit allows many data formats; to select the Intel format, use the following sequence:

SELECT 9 0 Δ 30 SET

10.2 The EPROM Utility

The **EPROM** utility generates data from ROM image files for EPROM programmers. The utility generates the data in one of two popular formats: Intel and Motorola. By redirecting the output to an RS-232 port, you can directly communicate with an EPROM programmer. The syntax is:

EPROM [-lhci] <filename> [<filename>...]

Options:

-i Output data in Intel format. If not specified, data is output in Motorola format.

-l Output low 8 bits of each word (default).

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

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

Following the options is a list of ROM image files — the extension ".41R" is automatically appended by **EPROM**. This list supports MS-DOS wild-carding.

EXAMPLE 1: Load the lower-8 data for file **TEST0.41R** and **TEST1.41R**, in Intel format, to an EPROM programmer connected to device COM2:

EPROM -il TEST0 TEST1 >COM2

EXAMPLE 2: Load the data for all **.41R** files in this directory, in Intel format, to an EPROM programmer connected to device COM1. Use the CMT data lay-out:

EPROM -ic * >COM1

10.3 Using Generic BURN Programs

If you have a generic BURN program that you would prefer to use over the **EPROM** utility, the **HIGHLOW** utility can be used to re-group the ROM image data into useful files.

For each ROM image file specified, **HIGHLOW** creates two new files:

- A 4K low-bits file, containing the lower 8 bits of each word, and
- A 1K high-bits file, containing the upper 2 bits of each word, packed 4 words/byte.

These two files conform to the data format required in most EPROM boxes. To invoke:

HIGHLOW <filename> [<filename>...]

For each **.41R** file named on the command line[6], **HIGHLOW** will create a **.41L** file containing the low bits, and a **.41U** file containing the high bits. You can then program your EPROM burner using a generic BURN utility.

HIGHLOW is found on disc #2.

10.4 Using the ERAMCO 16K EPROM Box

The ERAMCO 16K EPROM box provides two independently addressable 4K pseudo-ROMs, each consisting of two banks. The rotary and slide switches in the upper left-hand corner of the circuit board are used to address and enable/disable the pseudo-ROMs according to the following scheme:

- The left-hand rotary switch sets the page address of the "lower" pseudo-ROM.
- The left-hand slide switch enables (slide to the left) and disables (slide to the right) the "lower" pseudo-ROM.
- The right-hand rotary switch sets the page address of the "upper" pseudo-ROM.
- The right-hand slide switch enables and disables the "upper" pseudo-ROM.

Each pseudo-ROM consists of two 4K banks. When the box first receives power, bank 1 is enabled. When an ENROM2 instruction is executed from within either pseudo-ROM, bank 2 is enabled. Similarly, executing an ENROM1 will reenable bank 1. For more information on bank-switching, see chapter 11.

By restricting the box to certain configurations, it can be used to emulate many of the configurations achievable with the 1LG9 ROM. The following table shows some configurations that can be used to emulate different 1LG9 options (default **BUILD** configurations assumed):

6. Wild-carding is supported.

TABLE 8. Typical configurations with the ERAMCO 16K EPROM box

ROM Size	Port #	LH Rotary Switch	RH Rotary Switch	Notes
4K	1	8	(disabled)	Same data in both banks
4K	2	A	(disabled)	Same data in both banks
4K	3	C	(disabled)	Same data in both banks
4K	4	E	(disabled)	Same data in both banks
8K	1	8	9	Same data in both banks, both pseudo-ROMs
8K	2	A	B	Same data in both banks, both pseudo-ROMs
8K	3	C	D	Same data in both banks, both pseudo-ROMs
8K	4	E	F	Same data in both banks, both pseudo-ROMs
12K	1	8	9	Same data in both banks, lower pseudo-ROM
12K	2	A	B	Same data in both banks, lower pseudo-ROM
12K	3	C	D	Same data in both banks, lower pseudo-ROM
12K	4	E	F	Same data in both banks, lower pseudo-ROM

The data itself is contained in four 27C64 EPROMs at the bottom of the PC board. These parts *must* be CMOS for the box to work properly. The two *lower* EPROMs contain the *lower* pseudo-ROM (controlled by the left-hand address and enable switches), while the two *upper* EPROMs contain the *upper* pseudo-ROM (controlled by the right-hand address and enable switches). The following table illustrates the lay-out:

TABLE 9. EPROM Lay-out in the ERAMCO 16K EPROM Box

	Left Side		Right Side		
	0000-0FFF	1000-1FFF	0000-17FF	1800-1BFF	1C00-1FFF
Upper Row	Upper Page Low Bits Bank 1	Upper Page Low Bits Bank 2	don't care	Upper Page High Bits Bank 1	Upper Page High Bits Bank 2
Lower Row	Lower Page Low Bits Bank 1	Lower Page Low Bits Bank 2	don't care	Lower Page High Bits Bank 1	Lower Page High Bits Bank 2

Notice that there is no bank 0. To achieve a bank 0, the same data should be written in both banks 1 and 2 in a given page.

EXAMPLE

Using the command sequences for the Data I/O 21A, here is how to program **Fujitsu** 27C64 EPROMs to emulate a default 12K configuration[7] on the ERAMCO 16K EPROM box. Assume that the ROM image files are named **TEST0.41T**, **TEST1.41T** and **TEST2.41T**.

1. Select the ROM type, the data communications protocol, and the data format (Intel):

   ROMTYPE F64 SET (specify Fujitsu part)[8]

7. Default configurations are shown in appendix F.
8. This operation sets all 21A defaults, including the begin and end addresses for the programming operation: 0-1FFF.

SELECT Λ 480 Δ 34 SET (4800 baud, no parity)
SELECT 9 0 Δ 30 SET (Intel format, no offset)

2. Assuming the RS-232 interface at COM2 has been set up for 4800 baud, 1 stop bit, no parity, and the appropriate cable has been hooked up, communications may proceed. First, we will program the left-hand EPROMs. The lower left EPROM contains the lower 8 bits of the bank-0 image in the lower page. To program it, first set up the 21A to receive the data:

   SELECT 6 SET (ready to receive)

3. Typing quickly (the 21A has a data timeout), send the data from the host computer:[9]

   EPROM -il TEST0 TEST0 >COM2

4. After the data transfer is complete, load a blank EPROM into the socket and begin the programming:

   DEVICE F SET (specify B.P.R.[10] operation)
   DEVICE SET (begin operation)

5. Remove the EPROM, place it in the lower left socket, and place another blank EPROM in the programmer.

6. To load the lower-8 data for the upper pseudo-ROM, again set the 21A to receive:

   SELECT 6 SET (ready to receive)

   send the data:

   EPROM -il TEST1 TEST2 >COM2

   and program the EPROM:

   DEVICE SET (begin operation)

7. Remove the EPROM, place it in the upper left socket, and place another blank EPROM in the programmer.

8. To load the upper-2 data for the upper pseudo-ROM, set the 21A to receive:

   SELECT 6 SET (ready to receive)

   and send the data:

   EPROM -ih TEST0 TEST0 >COM2

9. Once the data has been received, it must be moved to the proper address (given in the EPROM lay-out table, above):

   EDIT 4 0 Δ 1800 Δ 800 SET (move data to 1800H)

10. Set the program start address to 1800H to avoid programming addresses 0-17FF:

    SELECT 2 1800 SET

11. Perform the programming:

    DEVICE SET (begin operation)

---

9. Because the lower-half of the port is in bank 0, the same data (TEST0.41R) is loaded into the bank-1 and bank-2 areas.
10. B.P.R. means: perform a blank-check, program the EPROM, then read (verify) its contents. Now that it has been specified, it need not be specified again this session.

12. Remove the EPROM, place it in the lower right socket, and place a blank EPROM in the 21A. To load the upper-2 data for the upper pseudo-ROM, again set the 21A to receive:[11]

    SELECT 6 SET (ready to receive)

    and send the data:

    EPROM -ih TEST1 TEST2 >COM2

13. Again, move the data to the proper location:

    EDIT 4 0 Δ 1800 Δ 800 SET (move data to 1800H)

14. Program the EPROM (the start address is still 1800H from the earlier operation):

    DEVICE SET (begin operation)

15. Place this EPROM in the upper right socket, set the address switches for the desired port, and the box will be ready to use.

11. This operation is not affected by the setting of the program start address, above.

11. Bank-Switching

The HP-41 address space lay-out 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 one or two (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 two to one for an 8K plug-in, but also 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.

11.1 A Word About Terminology

On the old 1LE9 ROM, the term "ROM" described a chip containing a single 4K piece of address space. Because a 1LG9 can contain up to three 4K pieces of address space, this document uses the term "ROM image" to denote code occupying a 4K piece of address space, and "ROM core" to denote the physical portion of a 1LG9 containing a ROM image.

When using **BUILD**, for example, the first **ROM#** command is used to define ROM image #1, the second **ROM#** command to define ROM image #2, and so on.

11.2 Basic Bank-Switching

Using SDS-II, you can create a single plug-in module containing up to 12K of ROM, or two plug-in modules containing up to 24K. Focusing on some simpler cases, consider single 4K, 8K and 12K plug-in modules. The following table illustrates where these ROM images are, by default, configured:[12]

TABLE 10. Default configurations for 4K through 12K ROMs

Size of ROM	ROM Image #	Where Addressed
4K	1	lower half
8K	1	lower half
	2	upper half
12K	1	lower half
	2	upper half, bank 1
	3	upper half, bank 2

The 4K and 8K cases are straightforward. For a 12K 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.

12. Other configurations can be chosen through the **CONFIG** clause on the **ROM#** command. A list of default configurations for all sizes from 4K to 24K is given in appendix F.

In effect, there are two *different* ROM images 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. Bank 1 is automatically enabled whenever a module is plugged into a machine.

Certain critical limitations apply to the ENROM1 and ENROM2 commands:

- ☞ These commands should be placed in non-bank-switching cores. Placing these commands in the bank-switching cores can cause unpredictable (and generally disastrous) results when they are executed.
- ☞ Programs using the ENROM1 and ENROM2 commands must reside in a non-bank-switching core. Placing such a program in a bank-switching core can cause unpredictable (and generally disastrous) results when the ENROM1 or ENROM2 instruction is executed.
- ☞ These commands will affect *only* the 1LG9 ROM in which they are resident. If, for example, the HP-41 contains two bank-switching plug-in ROMs in two different ports (say, ports 1 and 2), executing the ENROM2 keyword in port 1 will only affect the plug-in ROM in port 1.
- ☞ Because of the possibility that more than one plug-in ROM in the user's HP-41 will be bank-switching, and that ENROM1 and ENROM2 will subsequently be defined more than once, 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 a non-bank-switching ROM image, and only use ENROM1 and ENROM2 *within* your application to enable the banks containing your utilities.

11.3 Advanced Bank-Switching

This section contains important information if you are defining a bank-switching ROM using the advanced tools described in chapter 13. 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.

11.3.1 Using Bank-Switching

The 1LG9 has a number of configuration options, more fully explained in section 13.4. 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 section 11.2 (core 1=low/bank0, core 2=high/bank1, core 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 ENROM*x* instruction will only affect the 1LG9 out of which it is read. (Note that the ENROM*x* 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 ENROM*x* instruction be preceded by an instruction whose high bit is zero. This is customarily handled by placing a "GOTO $+1" instruction before the ENROM*x*.
- In general, an ENROM*x* instruction should not occur within a bank-switching ROM. It can, however, be done with careful planning. Keeping in mind that executing an ENROM*x* 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 its

next instruction from address F01H in bank 1.

- The HP-41CX self-check ROM requires that the data at address FFDH within any core contain a '1' in at least one of the upper two bits *if and only if* that core is a bank-selecting core. This practice, recommended but not required, is performed automatically by **BUILD**, but not by **LINK41**.
- Production-testing *requires* that all bank-switching cores contain the following data at the following addresses:

FC7	ENROM1
FC8	RTN
FC9	ENROM2
FCA	RTN

Any bank-switching ROMs submitted to the custom ROM program will be rejected if these test words do not appear in all bank-switching cores.

11.3.2 Placement of Global Labels in Bank-Switching Cores

The restrictions mentioned in section 11.2 about placement of major labels 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 ROM images. The third ROM image has a ROM ID of zero and, subsequently, an empty FAT table. It contains *only* 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 ROM images (and therefore two FATs) available for functions.

12. SDS-II Basic Utilities

SDS-II disc #1 contains the following utilities in addition to **READ41P**, **BUILD**, and **EPROM**.

12.1 CHECKSUM

The checksum utility can be used to verify the checksum of a ROM image file. The syntax 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.

12.2 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 HP-82161A cassette drive. To invoke:

LIFPACK <device>

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

LIFPACK C:

12.3 LISTFAT

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

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.

12.4 SDSCAT

The **SDSCAT** utility provides a catalog of HP-41 program files and MLDL-format files (created by **WRITMLDL**, explained in section 13.2.1) on an HP-41 mass-storage medium. Both catalogs are listed in alphabetical order, *not* the order the files are encountered on the disc. HP-41 program files are listed with the special characters substituted as described in appendix C.

Syntax:

SDSCAT <device>

For example:

SDSCAT C:

13. 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.

NOTICE

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, or by changes introduced to existing products.

13.1 Reading UCC Files

The **READ41P** utility can, in addition to reading HP-41 mass media, read the output of the HHP User Code Compiler (UCC). The expected file extension for the UCC file is "BIN". Syntax is:

READ41P -u <ucc_filename> [<filename>]

If <filename> is not specified, <ucc_filename> will be used with the extension ".41T". For example:

READ41P -u XYZZY

will read UCC output file **XYZZY.BIN**, creating a **READ41P** file named **XYZZY.41T**.

13.2 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 16K RAM Storage Unit, which contains the same bank-switching functionality as the 16K EPROM box.

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 disc #2 to support use of Q-ROM devices: **READMLDL** and **WRITMLDL**.

13.2.1 WRITMLDL

The **WRITMLDL** utility will copy a ROM image file created by **BUILD** or **LINK41** 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 syntax is:

WRITMLDL <ROMfile> <device>:<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 disc 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.

13.2.2 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.

Syntax:

READMLDL <device>:<filename> <ROMfile>

13.3 Other Advanced Utilities

This section assumes prior knowledge of HP-41 assembly language, and the HP-41 architecture and operating system[13].

13.3.1 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 non-HP products. The following subsections list the mnemonics and their opcodes, which should facilitate translation from other assembly languages.

13.3.1.1 Command Line Syntax

To invoke the assembler:

ASSEMB41 [-els8] [-o <outputfile>] <inputfile>

---

13. An excellent source of information about these topics is the ZENROM manual, from ZENGRANGE Ltd., Greenfield Road, Leeds, LS9 8DB, England.

Command line options:

-e Send error messages to screen in addition to standard output.

-l Produce a source code listing on standard output.

-s Print a symbol table to standard output.

-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**.

ASSEMB41 sends its output to standard output. If the -l or -s option is specified, the output is formatted for a printer, and should be redirected to one using the '>' command line feature of MS-DOS.

13.3.1.2 Assembler Syntax Conventions

Following are the general syntax rules for **ASSEMB41**.

13.3.1.2.1 Comments

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

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

13.3.1.2.2 Fields

A line consists of three fields: label, instruction, 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 instruction 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 instruction.
- The operand, if required for the instruction, must be separated from the instruction 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.

13.3.1.2.3 Expressions

Most instructions 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 section 13.3.1.2.4. 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.

13.3.1.2.4 Global Labels

ASSEMB41 supports global references for the following instructions:

- 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 instruction (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 instructions 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.

13.3.1.3 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 instruction). The type-0 instructions are presented both in alphabetical and numeric order, to facilitate understanding this set of mnemonics.

For the instructions 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.

13.3.1.3.1 Type-0 Instructions — Alphabetical Order

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

TABLE 11. Type-0 assembly-language instructions — alphabetical order

MNEMONIC	OPERAND	CODE
?F0=1		3ACH
?F10=1		0ECH
?F11=1		1ACH
?F12=1		36CH
?F13=1		2ECH
?F1=1		32CH
?F2=1		22CH
?F3=1		02CH
?F4=1		06CH
?F5=1		0ACH
?F6=1		16CH
?F7=1		2ACH
?F8=1		12CH
?F9=1		26CH
?LLD		160H
?P=Q		120H
?PT=	0-13	014H
?S0=1		38CH
?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
ENWRIT		028H
F=SB		258H
FEXSB		2D8H
FLG=1?	0-13	02CH
FLLABC		138H
FLLDA		038H
FLLDAB		0F8H
FLLDB		078H
FLLDC		0B8H
FLSDA		2B8H
FLSDAB		378H
FLSDB		2F8H
FLSDC		1B8H
FRAV?		12CH
FRNS?		26CH
FRSABC		3B8H
FRSDA		1F8H
FRSDAB		338H
FRSDB		238H
FRSDC		278H
G=C		058H
GOKEYS		230H
GOTOC		1E0H
HPIL=C	0-7	200H
HPL=CH	0-7	024H
IFCR?		16CH
INCPT		3DCH
LC	0-15	010H
*LC3	<expr>	010H,010H,010H

MNEMONIC	OPERAND	CODE
LDI		130H
LLD?		160H
M=C		158H
MCEX		1D8H
N=C		070H
NCEX		0F0H
NOP		000H
ORAV?		0ECH
P=Q?		120H
PFAD=C		3F0H
POWOFF		060H,000H
PT=	0-13	01CH
PT=?	0-13	014H
PT=A		3E8H
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

MNEMONIC	OPERAND	CODE
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
SRSDAB		328H
SRSDB		228H
SRSDC		268H
ST=0	0-13	004H
ST=1	0-13	008H
ST=1?	0-13	00CH
ST=C		358H
STARTC		368H
STK=C		170H
STOPC		328H
STPINT		1E8H
STREAD		0E8H
STWRIT		068H
TCLCRD		368H
TRPCRD		328H
TSTBUF		2E8H
WDTIME		068H
WMLDL		040H
WRALM		0A8H
WRSCR		128H
WRSTS		0E8H
WRTEN		2F0H
WRTIME		028H
WSINT		168H

The CON instruction compiles into the low 10 bits of the expression in the operand field. Unlike most

of these instructions, it does not perform a range check on the operand to require that it be in the range 0 through 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 through 0FFFH.

13.3.1.3.2 Type-0 Instructions — Numeric Order

TABLE 12. Type-0 assembly-language instructions — numeric order

MNEMONIC	OPERAND	CODE
CON	<expr>	000H
NOP		000H
S3=	0-1	004H
ST=0	0-13	004H
ST=1	0-13	008H
?S3=1		00CH
ST=1?	*0-13*	*00CH*
LC	0-15	010H
LC3	<expr>	010H,010H,010H
?PT=	0-13	014H
PT=?	0-13	014H
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

MNEMONIC	OPERAND	CODE
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
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

MNEMONIC	OPERAND	CODE
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
C=STK		1B0H
FLSDC		1B8H
CMEX		1D8H
MCEX		1D8H
GOTOC		1E0H
CRDOHF		1E8H
SRSDA		1E8H
STPINT		1E8H
FRSDA		1F8H
HPIL=C	0-7	200H
S2=	0-1	204H
?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

MNEMONIC	OPERAND	CODE
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
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

MNEMONIC	OPERAND	CODE
CRDFLG		3E8H
PT=A		3E8H
SLSABC		3E8H
PFAD=C		3F0H
RABCL		3F8H

13.3.1.3.3 Arithmetic Instructions

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

TABLE 13. Arithmetic assembly-language instructions

MNEMONIC	OPERAND	CODE	MNEMONIC	OPERAND	CODE
?A#0	TE	342H	B#0?	TE	2C2H
?A#C	TE	362H	B=0	TE	022H
?A<B	TE	322H	B=A	TE	082H
?A<C	TE	302H	B=C	TE	0E2H,0C2H
?B#0	TE	2C2H	BAEX	TE	062H
?C#0	TE	2E2H	BCEX	TE	0E2H
A#0?	TE	342H	BSR	TE	3A2H
A#C?	TE	362H	C#0?	TE	2E2H
A<B?	TE	322H	C=-C	TE	282H
A<C?	TE	302H	C=-C-1	TE	2A2H
A=0	TE	002H	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			

13.3.1.3.4 Pseudo-Ops

The following pseudo-ops are used in ASSEMB41:

TABLE 14. Assembly-language pseudo-ops

MNEMONIC	OPERAND	CODE	MNEMONIC	OPERAND	CODE
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 the 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 **LINK41**. 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 Σ, ≠, and ∠ use the alternate representation explained in appendix C. The use of the LCDCHAR pseudo-op is illustrated in the examples.

LEGAL — Normally, the assembler complains if certain potentially erroneous combinations of instructions 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 command cannot be used as microcode library files.

SKIP — Same as EJECT.

SPACE — Skip the specified number of spaces in the output listing. The number of spaces is 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 command is encountered.

13.3.1.3.5 FAT Entries

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

TABLE 15. Assembly-language FAT entry pseudo-ops

MNEMONIC	OPERAND	CODE	MNEMONIC	OPERAND	CODE
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 This pseudo-op has always existed. Its purpose is lost to modern memory.

13.3.1.3.6 Branches

These instructions provide branching of various sorts:

TABLE 16. Assembly-language branching instructions

MNEMONIC	OPERAND	CODE	MNEMONIC	OPERAND	CODE
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 a "three-byte GOSUB" capable of reaching a target address anywhere within the current 4K block. (Uses mainframe routine GOSUB0, GOSUB1, GOSUB2, GOSUB3, or GOSUB, as appropriate.)

GSBSAM Compiles into a "three-byte GOSUB" capable of reaching a target address anywhere within the current 1K block. (Uses mainframe routine GOSUB.)

GOL41C Compiles into a "three-byte GOLONG" capable of reaching a target address anywhere within the current 4K block. (Uses mainframe routine GOL0, GOL1, GOL2, GOL3, or GOL, as appropriate.)

GOLSAM Compiles into a "three-byte GOLONG" capable of reaching a target address anywhere within the current 1K block. (Uses mainframe routine GOL.)

13.3.1.3.7 Peripheral Commands

TABLE 17. Assembly-language instructions for smart peripherals

MNEMONIC	OPERAND	CODE	MNEMONIC	OPERAND	CODE
?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

13.3.1.4 EXAMPLES

13.3.1.4.1 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
```

13.3.1.4.2 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
```

13.3.2 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".

NOTE

If you use LINK41 instead of BUILD to create your ROM image files, you must create your own configuration file. See appendices G and H for more details.

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

Syntax:

LINK41 [-e] [<command_file>]

The -e option specifies that error messages are sent to the display in addition to standard output.

If a <command_file> is specified, that file is opened and LINK41 commands are read from it. If no commands are accepted from the console (prompting is provided).

LINK41 creates from one to six 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
?
```

The meanings of the **LINK41** commands are:

NEWROM Analogous to the **ROM#** command 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 disc #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 overridden either by the LOCATE command (above) or by use of the ORG command 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, it 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.

13.3.3 ASMBINFO

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

ASMBINFO [-d] <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 items of interest to **LINK41** and **BUILD**.

If the **-d** option is selected, the output will include a hex dump of the code in the file.

13.3.4 DISASM41

The **DISASM41** utility will disassemble an entire file, interpreting it as consisting of HP-41 assembly-language. The syntax 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 instruction's representation in hex, octal, decimal, ASCII, LCD characters, and finally, HP-41 instruction. 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 contains an odd number of bytes.

13.3.5 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 GOTOs and XROMs compiled, links resolved, etc.), and then incorporate that program into a ROM being developed with **ASSEMB41** and **LINK41**.

The syntax is:

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

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.

ROMfilename The name of 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 1, 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.

13.3.6 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 (**.41O**) 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 syntax is:

MUCODE <assemblyfilename> [<microcodefilename>]

The <assemblyfilename> will automatically have the extension ".41O" 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. Section 13.3.1.4.1 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.41O**, 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.

13.4 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 appendix F are adequate. However, it is possible to request alternate configurations. For each 4K core, the following options are available:

TABLE 18. 1LG9 ROM core configuration options

Enabled/Disabled		
Hard-Configured	Port-Configured	
Address	Lower Half	Upper Half
Bank 0/Bank 1/Bank 2		

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 ENROM*x* instruction affects only 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.

APPENDIX A. Contents of Discs

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

A.1 Disc 1

AIP.41T
ALEN.41T
ALENG.41T
ALNAM2.41T
ANUM.41T
AROT.41T
ATOX.41T
AUTOST.41T
BIND.41T
BININ.41T
BINVIEW.41T
CLKEYS.41T
CLRGX.41T
CSKBD.41T
ENROM1.41T
ENROM2.41T
GETKEY.41T
HEXIN.41T
HEXVIEW.41T
OCTIN.41T
OCTVIEW.41T
PASN.41T
PCLPS.41T
PNCTUA.41T
POSA.41T
PSIZE.41T
RCLSTFLG.41T
REGMVSWP.41T
SIZE.41T
XB.41T
XF.41T
XTOA.41T
XVIEW.41T
BUILD.EXE
CHECKSUM.EXE
EPROM.EXE
LIFPACK.EXE
LISTFAT.EXE
READ41P.EXE
SDSCAT.EXE

A.2 Disc 2

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

APPENDIX B. 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	

APPENDIX C. 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:

TABLE 19. SDS-II Substitutes for special HP-41 characters

HP-41 Character	Substitute	Usual HP-41 Representation
├	'x'	ASCII 127
μ	'm'	ASCII 12
≠	'n'	ASCII 29
Σ	's'	ASCII 126
∠	'g'	ASCII 13
_	'o'	ASCII 0

Only three of these characters, ≠, Σ, and ∠, 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 command line or in a DEFINE file, it is replaced with '.' as a placeholder.

EXAMPLE 1: To read in a program named "A≠B" from the HP-41 disc, 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 2: To read in a program named "A B" from the HP-41 disc, use:

READ41P C:A.B AB

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

APPENDIX D. Handling 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, in bytes, 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.

APPENDIX E. Command Syntax 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 [-d] <file.ext>

ASSEMB41 [-els8] [-o <object.41O>] <source.41A>

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

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

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

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

LIFPACK <device>
(<device> must include ':')

LINK41 [-e] [<command_file>]

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

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

READ41P <device>:<41_prog_name> <file.41T>
READ41P -u <ucc_file_name.BIN> [<file.41T>]

READMLDL <device>:<filename> <file.41R>

SDSCAT <device>
(<device> must include ':')

WRITMLDL <file.41R> <device>:<filename>

APPENDIX F. Default Chip Configuration

This table illustrates the default configurations used if the **CONFIG** option is not used in the DEFINE file. Table entries use the same format as the CONFIG option in the **ROM#** command.

TABLE 20. Default 1LG9 configurations for one to six ROM images

# ROMs	Default Configuration Options					
	ROM #1	ROM #2	ROM #3	ROM #4	ROM #5	ROM #6
1	1L0					
2	1L0	1U0				
3	1L0	1U1	1U2			
4	1L0	1U0	2L0	2U0		
5	1L0	1U1	1U2	2L0	2U0	
6	1L0	1U1	1U2	2L0	2U1	2U2

Notice that the 4-ROM configuration does not use bank-switching.

APPENDIX G. The CONFIGURATION File

A configuration file is automatically created by **BUILD** — *you* must create it for ROMs developed using the advanced tools. This file is used by Hewlett-Packard, upon receipt of your ROM image files, for three purposes:

- To verify that you have chosen a legal configuration,
- To determine whether your ROM images contain the proper test words in the proper locations (for bank-switching ROM cores), and
- To program the 1LG9 ROM's configuration.

The filename extension for the configuration file is always "41F". The file contains information about the exact one- or two-chip configuration you have chosen. For each ROM image, the file contains one text line consisting of five fields, separated from each other by a single blank. The fields, in the order they appear, are:

TABLE 21. Fields in the configuration file

Field	Possible Values	Meaning
Chip Number	**1,2**	Which 1LG9 this ROM image occupies (always 1 for a 1-chip ROM)
Filename	**<filename>.41R**	Name of file containing this ROM image
Address Mode	**P,H**	P=Port-configured; H=Hard-configured
Page	*If Port-Configured:* **L,U**	L=Lower half; U=Upper half
	If Hard-Configured: **0-F**	Hex page number
Bank	**0-2**	Bank number: 0 Non-bank-switching 1 Bank-switching; alternates w/bank 2 2 Bank-switching; alternates w/bank 1

EXAMPLES

If **BUILD** is used to create a 12K ROM using the default configuration, where "XYZZY" is the output file specified in the command line, the resulting configuration file will be named "XYZZY.41F", and will contain the following three lines:

```
1 XYZZY0.41R P L 0
1 XYZZY1.41R P U 1
1 XYZZY2.41R P U 2
```

If **BUILD** is used to create a default 16K configuration (8K non-bank-switched in each of two ROMs) the file will contain the following four lines:

```
1 XYZZY0.41R P L 0
1 XYZZY1.41R P U 0
2 XYZZY2.41R P L 0
2 XYZZY3.41R P U 0
```

APPENDIX H. Submitting ROM Images to Hewlett-Packard

Hewlett-Packard can accept ROM image files produced by SDS-II in one of two disc formats:

- 3½" micro-floppy, single- or double-sided, or
- 5¼" mini-floppy, 360K format *only*.[14]

The disc *must* contain the following files in the root directory:

1. A configuration file (extension "41F") specifying the configuration of the ROM(s) to be produced. If BUILD was used to create the ROM image files, this file is automatically created. Otherwise, you must create it yourself.
2. The ROM image files (extension "41R") created by BUILD or LINK41. The names of these files must be the same names referenced in the configuration file.

After receiving the disc, Hewlett-Packard will produce two listings:

1. A ROM catalog listing.
2. A configuration printout showing the 1LG9 programming configuration that will be used.

Unless you choose to waive this step, you must inspect and approve the listings before ROM mask generation and production can proceed.

14. 360K-formatted discs that have been written by a high-capacity (1.2M) drive are *not* acceptable. Such discs are readable by other high-capacity drives, but generally not by regular-capacity drives.

APPENDIX I. ROM ID Allocation

All possible ROM IDs have been used by HP and/or by custom products. Your choice of ROM should include consideration of possible conflicts that may occur with other modules likely to be used.

TABLE 22. ROM IDs 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 Clinical Lab Aviation
20	
21	Reserved for custom modules
22	HP-IL Development Advantage
23	Extended I/O
24	HP-IL Development Advantage
25	Extended Functions
26	Time
27	Wand
28	Mass Storage
29	Printer
30	Card Reader
31	Reserved for custom modules