Software Development Kit

For the HP-41

Release 6

PROGRAMMER'S MANUAL

http://www.hp41.org

CONTENTS

Introduction

The Software Development Kit for HP-41 (SDK41) package contains all the software needed to develop custom HP-41 ROM images on MS-DOS machines. The following basic setup is recommended:

- HP-41CV/CX
- IBM PC compatible computer; hard disk recommended
- MLDL (Machine Language Development Lab) with at least 8K
- HP82160 HP-IL module for HP-41
- Either HP8297A HP-IL interface card (for IBM compatible) or HP-IL <=> RS232 converter

The mnemonics in this manual use the Zencode instruction set.

SDS is the software used to develop ROM images by Hewlett-Packard. It does many of the same things as SDK41, but it was never released to the public. Many of the files produced by SDK41 are compatible.

Below you can see an overview of the various pieces that comprise the full SDK41.

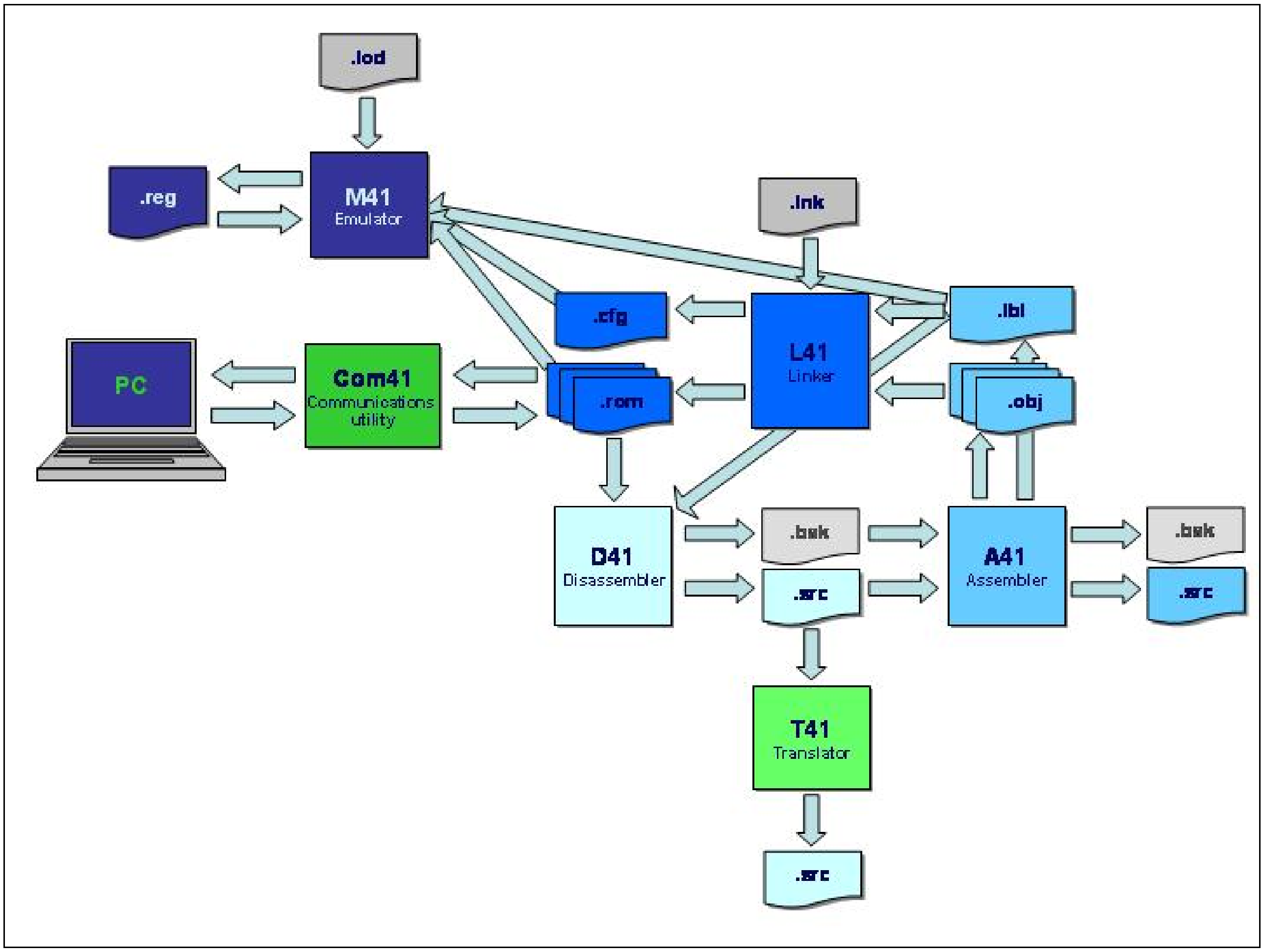


Figure 1 Overview of SDK41 components

SECTION 2 - ***Features***

Assembler *A41*

- Assembles MCODE instructions from any one of the three instruction sets: HP, Zencode or Jacobs/De Arras
- Contains built-in symbol table of the HP-41's mainframe labels which allows easy referencing to operating system entry points
- Allows generation of internal and external symbolic labels
- Assembles FAT pseudo-instructions for easy ROM image building
- Produces symbol cross reference table, if desired

Linker *L41*

- Resolves external label references and creates a ROM image file suitable for loading into an EPROM or an MLDL
- Can link more than one ROM image at a time
- Produces symbol cross reference table, if desired

Disassembler *D41*

- Disassembles MCODE instructions into any one of the three instruction sets
- Contains built-in symbol table to the HP-41's mainframe labels
- Disassembles FAT pseudo-instructions for easy ROM image decoding
- The source file generated by the *D41* can be immediately re-assembled by the assembler

Emulator *M41*

- Reads ROM images and allows single stepping and debugging of the instructions which may be in any one of the three instruction sets
- Executes the HP-41 operating system (the HP-41 operating system ROMs are not included and must be loaded by the user with the communications utility)
- Provides Full screen display of the HP-41's internal registers, display and RAM and allows the user to set and clear breakpoints, jump to any address and preload registers
- Incorporates mainframe labels into pages 0-2 and can read and incorporate labels from user-specified label files
- Automatic edit-assembly-link-reload sequencer for changing ROM code
- Support for the full HP-41 Halfnut display character set and 43 and 60 line mode in EGA and VGA
- Supports a mouse interface for the simulated keyboard. Cursor keys may also be used.

Instruction Set Translator *T41*

- Translates the instructions in source files from any of the three sets to any other set

Communications *COM41*

- Upload and download ROM image files from an HP-41 through HP-IL to a PC (requires basic setup)

SECTION 3 - *Definitions*

Note: The mnemonics in this manual use the Zencode instruction set.

SDS This is the software used to develop ROM images by Hewlett-Packard. It does many of the same things as SDK41, but it was never released to the public. Many of the files produced by SDK41 are compatible.

<address>	Any value in the range 0000-FFFF (hex)
<value>	Any value in the range 0000-FFFF (hex). This is the same as an <address> but is used in a more generalized sense.
<page>	Any value in the range 0 to F (hex). A page is also 4096 words.
<bank>	Any value in the range 1 to 4 (dec). The HP-41 supports bankswitching with special ROM devices.
<+disp>	Any value in the range +0 to +63 (dec).
<-disp>	Any value in the range -1 to -64 (dec).
(local label>)	A label that is delimited with parenthesis. Example: *(FOOBAR).* Local labels are local to the file that they are defined in and are not "visible" outside that file. Any characters except spaces may be contained in the label definition but for convention labels should be limited to uppercase characters and the underscore. If a global label is the same as a local, there will be no conflict, but this should not be done to avoid confusion. Local machine code labels should not be confused with local User Code labels as they are completely different.
[global label]	A label that is delimited with brackets. Example: *[FOOBAR]*. Global labels are "visible" to all files that are linked together. Any characters except spaces may be contained in the label definition but for convention labels should be limited to uppercase characters and the underscore. Global machine code labels should not be confused with global User Code labels as they are completely different.
<label>	Either a <local label> or a <global label>. This is similar to a <symbol> but is used in the more restrictive sense that labels are not defined with the .EQU directive.
<symbol>	Same as a <label> but used in a more generalized sense to include all symbols including those defined with the .EQU directive.
<operand>	The argument that is given after the instruction. Example: XS is the operand for A=B XS.

Table 1: Definitions of expressions used in this manual

SECTION 4 - *Utilities*

A41 ASSEMBLER

Syntax

A41 [options] *<file>*

Description

A41 assembles HP-41 MCODE mnemonics from one of three instruction sets and produces an object file that is linkable into a ROM image. A41 expects to find *<file.src>* that contains the code to assemble and produces <*file.obj*>. A41 also has the ability to reformat the source file and insert error messages and object data into it.

Files

File name	***Purpose***	***Action***
<file.src>	source file	(read from and written to)
<file.bak>	backup file	(written to)
<file.obj>	object file	(written to)
<file.lbl>	label file	(written to)

Options

Option	***Description***
/S	If this option is not specified, the source file is read and not modified. Otherwise, the source file is read and formatted and various useful information such as error messages, object data and cross references are optionally inserted into it. A backup file is created that contains the original source code.
/R	Does the same as /S and also produces a complete symbol cross reference table at the end of the source file.
/O	This option causes the generation of the <*address*> and <*data*> fields into the source file when the /S or /R option is specified. These fields are for the user's benefit and are ignored by the assembler upon subsequent assemblies. This option does nothing if /S or /R is not specified.
/L	Generates *<file.lbl>* that contains all local labels in the source file, but no global labels. See Section 6 for label file format.
/E	Erases internal mainframe label table. The internal mainframe label table is a data table in the assembler that holds all of the HP-41's operating system entry points as specified by the VASM listings. This option supports custom HP-41 operating systems.

Assembler Directives

Directive	*Operand*	*Description*
.TITLE	*"title"*	Gives a title to the object code. It this title is longer than 80 characters it will be truncated.
.HP		Specifies the HP instruction set and must be given before the code starts. One source file may contain several instruction set directives and they may be different.
.ZENCODE		Specifies the Zencode instruction set and must be given before the code starts.
.JDA		Specifies the Jacobs/De Arras set and must be given before the code starts.
.FILLTO	*<address>*	Fills from current address to the specified address with zeros. If the .ORG directive is specified, this directive will fill from the current address to the specified address PLUS the origin address. The addresses are considered occupied and are not open for linking any other object code into.
.BSS	*<number>*	Fills the next *<number>* words with zeros, where *<number>* is a positive decimal number. The addresses are considered occupied and are not open for linking any other object code into.
.NAME	*"name"*	Macro for defining function names. This is used for construction of the FAT. It converts each character to its LCD equivalent and adds 80 hex to the last character. The order of the characters is automatically reversed as required by the HP-41.
.MESSL	*"string"*	Similar to the NAME directive, the MESSL directive puts the string into the format required for output via the [MESSL] entry point. Each character is converted to its LCD equivalent and the 20 hex is added to the last character.
.ORG	*<address>*	Specifies an absolute address in the range 0000-FFFF (hex) to originate the code at. All symbols defined in a file that contains this directive are absolute and cannot be relocated by the linker. This directive can be specified only once in each source file.
.EQU	*<global symbol>* *<value>* *OR* *<local symbol>* *<value>*	Equates a symbol with a value in the range 0000-FFFF (hex). This symbol functions just like any label, but is NOT relocatable since it represents an absolute constant.
#000-#3FF		This is not a directive but allows any literal to be entered directly into ROM. This is similar to the CON pseudo-instruction except that only literal values in the range 000-3FF (hex) can be entered and not symbols

Error Messages

Error Message	*Description*
FATAL ERROR (A01) *Code runs past FFF (hex)*	The assembler cannot assemble more than FFF (hex) words into one object module since that is the maximum length of a ROM image.
FATAL ERROR (A02) *Source File Is Empty! Check backup file.*	The assembler found the source file to be empty which could have been caused by interruption of the assembler when it was running previously. The backup file will contain the original source file.
FATAL ERROR (A03) *Failure to rename <file x> to <file y>*	This error will result if for some reason the operating system prevents the assembler from renaming <file x> to <file y>. Check the file access on the files.
FATAL ERROR (A04) *Out of memory!*	This error occurs when the system's dynamic memory has been all used up. If this occurs, shorten source file.
ERROR (A05) *Jump address not in same 4K page*	The address specified for the quad relative jump is not in the same page as the instruction itself. The instruction is assembled anyway.
ERROR (A06) *Illegal label definition: <label>*	The label is greater than 13 characters or is not delimited by brackets or parenthesis.
ERROR (A07) *Operand not recognized: <operand>*	The operand is not valid for the instruction given. The assembler will still generate code.
ERROR (A08) *Illegal .EQU definition*	The symbol specified is not a legal symbol since it is greater than 13 characters or is not delimited by brackets or parenthesis or the value is not in the range 0000-FFFF (hex).
ERROR (A09) *FAT names cannot be greater than 11 Characters.*	The HP-41 does not support FAT names greater than 11 characters long. No data will be generated.
ERROR (A10) *Illegal address: <address>*	The specified address is not in the range 0000-FFFF (hex).
ERROR (A11) *Illegal number: <number>*	The specified number is not a valid positive number.
ERROR (A12) *.ORG must be specified before code starts*	The .ORG directive was specified after the code started and was ignored.
ERROR (A13) *Illegal .ORG definition*	The address specified for the .ORG directive is not in the range 0000-FFFF (hex).
ERROR (A14) *.ORG specified more than once*	The .ORG directive cannot be specified more than once in each source file.
ERROR (A15) *Unknown directive: <directive>*	The directive is not valid.
ERROR (A16) *Reference to external local not permitted: <local symbol>*	A reference to a local symbol (delimited by parenthesis) was made and that symbol is not in the current source file.
ERROR (A17) *Illegal instruction: "INSTRUCTION"*	The instruction given is not in the current instruction set.
ERROR (A18) *Missing Operand*	An operand is required.

Error Message (cont.)	***Description (cont.)***
ERROR (A19) *Illegal characters in NAME string*	There were characters found in the FAT function name that are not allowed by the HP-41. The assembler supports all possible characters for NAME strings.
ERROR (A20) *Illegal characters in MESSL string*	There were characters found in the MESSL string that are not mapped to the HP-41. There are characters that can be displayed on the HP-41 that the assembler does not support. These are any characters with punctuation bits set or the extra characters that only the halfnut LCD can display. The MESSL directive cannot be used to encode these; they must be entered manually as literal data using the # token.
ERROR (A21) *Illegal displacement: <displacement> (dec)*	A displacement was specified that was out of the range -64 to +63 (dec).
ERROR (A22) *Literal Address in relocatable object module*	This occurs when a source file does not have the .ORG directive in it and short jump instructions have operands that are literal address. (Such as JNC).
WARNING (A50) *Operand greater than FFF (hex)*	The LC3 macro instruction expected an operand in the range 000-FFF (hex). The operand was truncated to 12 bits.
WARNING (A51) *Operand greater than 3FF (hex)*	The CON pseudo-instruction expected an operand in the range 000-3FF (hex). The operand was truncated to 10 bits.
WARNING (A52) *Duplicate symbol: <symbol> Address not used <address>*	A symbol was defined more than once in the same source file. Only the first occurrence is used,and the others are ignored. The address of all later occurrences are listed in the warning message.

L41 LINKER

Syntax

L41 [options] <file>

Description

The linker links the object files together to create one or more ROM files that contain one ROM image each. The commands that direct the linker are contained in <file.lnk>. See Section 6 for more information on the link file. L41 can link SDS format .41O files. (Any use of SDS object files requires the linking of the SDS file MFENTRY.41O since the SDS assembler does not resolve mainframe entry points)

Files

File name	***Purpose***	***Action***
<file.lnk>	link file	Read from
<????.obj>	object file(s)	Read from. With ???? being specified in the link file
<????.lbl>	label file	Read from. With ???? being specified in the link file
<file.lbl>	label file	Written to
<file.cfg>	config file	Written to
<file.rom>	ROM file	Written to
<file#.rom>	ROM file(s)	Written to. Where # is a decimal number from 1 to the maximum number of ROM images that are linked *MINUS* one.

Options

Option	***Description***
/L	Creates a label file containing all global labels in all object modules. This file has the same name as the link file
/LL	Same as /L but also copies all label files with the same name as the object files into the one label file. This consolidates all local and global labels for one application into one file
/R	Creates symbol cross reference table for the global labels and places it in the config file.
/A	Assembles object modules that are out of date. This option causes A41 to be called if source file is newer than object file. Any letters that follow the /A are passed to A41 as options.

Example: /ARO calls A41 with the /R and /O options. A41 is also called if /AL is specified and the source file is newer than the label file.

Error Messages

Error Message	***Description***
FATAL ERROR (L02): *Object code runs past end of page*	Attempted link for code that runs past the end of the current ROM image.
FATAL ERROR (L03): *<object file> is corrupt*	The object file is corrupt and cannot be read by the linker because it contains unexpected data.
FATAL ERROR (L04): *Out of memory!*	This error occurs when the system's dynamic memory has been all used up.
FATAL ERROR (L05): *Cannot create the same page twice*	The link file has more then one page command with the same parameters.
FATAL ERROR (L06): *Illegal Parameter in $PAGE command*	The parameters specified for the page command are not valid.
FATAL ERROR (L07): *$PAGE Command not given before first object file name*	A page command must be specified before the first object file name is given.
ERROR (L08): *Jump address not in same 4K page*	The address specified for the quad relative jump was not in the same page as the instruction itself.
ERROR (L09): *Illegal displacement: <displacement> (dec)*	The short jump instruction is referenced to a symbol that is out of its range of -64 to +63 (dec).
ERROR (L10): *Symbol not defined: <symbol>*	A reference to a non-existent symbol was made. The address used to link is 0000.
ERROR (L11): *<object file> written to space occupied by <object file>*	If the current object file maps to the space that another object file has already been linked to this error results and the new object file will be loaded over the old object file
ERROR (L12): *<number> Error(s) in assembly of <source file>*	The assembler returned errors from the assembly of a source file. This message appears just before the linker terminates so if there are errors in the assembly they will be more obvious.
ERROR (L15): *Illegal $OFFSET command in: <label file> - Line <line>*	The address specified for the $OFFSET command is not in the range 0000-FFFF (hex).
ERROR (L16): *Illegal label definition: <label> in: <label file> - Line <line>*	The label is greater than 13 characters or is not delimited by brackets or parenthesis.
WARNING (L50): *Reference from bank <bank x> to bank <bank y> at address <address>*	This warning results when a reference from one bank to another is made. The reference is resolved as if the banks were the same which means unpredictable results are possible at run time.
WARNING (L51): *Object file <object> originates at <origin page> but has been forced into page <current page>*	This results when the address specified for the ORG directive when the file is assembled is not in the same page as the current page. The linker forces the object file into the current page and continues linking.
WARNING (L52): *Duplicate symbol: <symbol> Address not used <address> from object file <object file>*	A symbol was defined more than once. Only the first occurrence is used, and the others are ignored. The address and object file of all later occurrences is listed in the warning message.
WARNING (L53): *$LOC value <value> has been forced into page <page>*	The page specified for the $LOC command is not the same as the current page. The linker forces the <value> into the current page.

D41 DISASSEMBLER

Syntax

D41 [options] <file>

Description

The D41 disassembler is capable of disassembling ROM image files into one of the three instruction sets. It expects to find <file.rom> and it produces <file.src>. If the ROM file contains User Code, the User Code instructions are represented as literal data in the range #000 to #3FF.

Files

File name	***Purpose***	***Action***
<file.rom>	ROM file	read from
<file.lbl>	label file	read from
<file.src>	source file	written to

Options

Option	***Description***
/H	Disassembles into the HP set (default). This option may not be given in conjunction with /Z or /J
/Z	Disassembles into the Zencode set. This option may not be given in conjunction with /H or /J.
/J	Disassembles into Jacobs/De Arras set. This option may not be given in conjunction with /H or /Z.
/Pn	Maps the ROM into page n, where n is 0 to F (hex). This is used for producing a listing for a ROM that is hard-configured such as one of the operating system pages. See Section 5 for more information on the use of this option. This option does not actually change any code. If this option is not specified, the default is page 8.
/F	Causes the ROM image to be disassembled with a FAT. If this is not specified, the ROM is assumed to have no FAT.
/E	Erases the internal mainframe label table so that the disassembly does not place the HP-41 operating system entry labels into the source file. This option is only useful if the ROM image maps into pages 0-2 but is not part of the HP-41's operating system. This option supports nonstandard operating systems.
/L:<label file>	Specifies a label file containing labels which are incorporated into the source listing just as the mainframe labels are. See Section 6 for label file format. This option may be used multiple times to specify all label files desired.

Defaults

- HP instruction set
- Page 8
- Does not disassemble with a FAT
- Mainframe labels active
- No external label files

Error Messages

Error Message	***Description***
FATAL ERROR (D04): *Out of memory!*	This error occurs when the system's dynamic memory has been all used up.
ERROR (D05): *Illegal label definition: <label> in: <label file>*	The label is greater than 13 characters or is not delimited by brackets or parenthesis.
ERROR (D08): *Illegal $OFFSET command in label file: <label file>*	The address specified for to the $OFFSET command is not in the range 0000-FFFF (hex).
ERROR (D09): *FAT entry not recognized*	The data in the FAT entry differs from the standard HP-41 FAT pseudo-instructions.
ERROR (D10): *FAT not followed by two NOP instructions*	The HP-41 requires that the FAT be followed by two NOP instructions.
ERROR (D11): *Data at ROM address 1 (number of FAT entries) is incorrect*	The data in address 1 of the ROM differs from the actual number of FAT entries disassembled.
WARNING (D50): *Halfnut character found in NAME string*	The NAME string contains a character that is only displayable by an HP-41 with a halfnut display. The disassembler supports all possible non-halfnut characters for NAME strings. A tilde (~) character is displayed instead.
WARNING (D51): *Halfnut character found in MESSL string*	The MESSL string contains one or more characters that are only displayable by an HP-41 with a halfnut display. The disassembler supports all possible non-halfnut characters.
WARNING (D52): *Punctuation bits set in MESSL string*	The MESSL string contains one or more characters that have punctuation bits set and causes the punctuation fields on the display to light up. The .MESSL line will not show these, but the line by line literal disassembly will.

M41 EMULATOR

Syntax

M41 [options] [<load file>]

Description

M41 is a powerful and useful tool for testing and developing custom MCODE programs. It allows single stepping of MCODE programs and also supports a continuous run mode that executes the HP-41 operating system ROMs when they have been loaded with 41COM.

Operation

After loading the operating system ROMs, the 'U' command may be executed and the emulator will mimic an HP-41. There are several things to know when the emulator is in this mode. If a *?KEY* instruction is encountered, the emulator will check the PC's keyboard to see if a key is being pressed. If one is not, it will go on without interrupting the execution. If a key is pressed, it will be translated and loaded into the KEY register and will continue to be loaded for a certain amount of clock cycles. Also, the registers and instructions will not be updated on the screen but the display will be. If the 'R' command is executed, the emulator will also run at full speed, but will stop for key and powoff trap conditions and breakpoints.

The screen is divided into three areas; the instructions, the registers and the display. The pointer at the right hand side of the instructions indicates which instruction will be executed next. After every instruction is executed, it is rewritten to the display. This sometimes causes the instruction to change. An example of this is when a *REG=C 3* instruction changes to a *WRAB6L* instruction. This occurs because no peripheral was selected when the entire screen was disassembled, but a *PERSLCT FD* was executed after this and that changed the active peripheral.

The display and register parts of the screen are mostly self-explanatory with the following notes. "BK" stands for BANK and is 1 to 4. "KEY" is the KEY register which contains the keycode, while ?KEY is the keydown register and is either 1 or 0. "PERPH" is the selected peripheral. "HEX" is 1 if the CPU is in hex mode and 0 if in decimal mode. There are three registers that have different names depending on the active instruction set: SB is also ST, XSB is XST and F is T (This is shown in Appendix H). The block of RAM registers at the bottom left of the screen is the active chip and RAM is the RAM address selected. If chip 0 is selected, the alternate (USERCODE) names will be displayed also. If one or more registers in the chip is non-existent, dashes are displayed. If an Extended Functions ROM is loaded into page 3 the emulator automatically makes all extended memory available. "CODE" is the hex code of the instruction just executed.

Files

File name	*Purpose*	*Action*
<file.lod>	load file	read from. If <load file> is not given on the command line, DEFAULT.LOD is used instead
<????.rom>	ROM file(s)	read from and written to. ???? are file names specified in the load file
<file.lbl>	label file(s)	read from
<file.reg>	register file	read from and written to
<file.cfg>	config file	read from

Options

Option	*Description*
/H	Displays instructions using the HP set (default). This option may not be given in conjunction with /Z or /J
/Z	Displays instructions using the Zencode set. This option may not be given in conjunction with /H or /J.
/J	Displays instructions using Jacobs/De Arras set. This option may not be given in conjunction with /H or /Z.
/BW	Disables color for LCD displays that are based on a color adapter.
/TD	Sets the text direct video mode. This writes directly to screen memory and may not work with all systems. If this option or /GD is not given, the text bios video mode is used. Text bios uses the system bios and works with all video adapters but is slower.
/GD	Sets the graphics direct video mode. This can be used only on EGA or VGA and writes directly to screen memory. This supports all HP-41 display characters and also 25,43 and 60 line fonts. The default is 25 lines.
/43	Sets the 43 line font so that 43 lines are displayed. This only works with /GD and an EGA or VGA adapter.
/60	Sets the 60 line font so that 60 lines are displayed. This only works with /GD and an VGA adapter.

Defaults

- HP instruction set
- Color enabled
- Starting address = 0000
- Text bios video mode
- 25 lines

Commands

Command	*Operand*	*Description*
A		Display alpha register.
BC	*<address>*	Clear the breakpoint at <address> if there is one.
BC	*<label>*	Clear the breakpoint at <label> if there is one.
BC		Clear the breakpoint at the current address if there is one.
BC ALL		Clear all breakpoints
BS	*<address>*	Set breakpoint at <address>.
BS	*<label>*	Set a breakpoint at <label> if it exists.
BS		Set breakpoint at the current address.
BL		List all breakpoints.
BD		Disable all breakpoints.
BE		Enable all breakpoints.
C	*<chip>*	Display the specified RAM chip where <chip> is 000 to 3FF (hex) with exceptions as noted in Appendix F.
D		DOS shell (Type EXIT to return). COMMAND.COM must be in the path.

Command (cont.)	***Operand (cont.)***	***Description (cont.)***
E <instruction>	*<operand>*	Executes any valid non-pseudo-instruction. Instructions must be in the current set and must have the following syntax. The operand can be any legal A41 operand and can be in upper or lower case. (Example: C=C+1 PT)
FS	*<reg file>*	Saves all registers to a register file.
FL	*<reg file>*	Loads all register from a register file.
G	*<address>*	Goto the address specified.
G	*<label>*	Goto the label specified, if it exists.
G	*<-displacement>*	
G	*<+displacement>*	Goto address plus or minus displacement. Displacement is any valid value from -65536 to +65535 (dec).
I		Invert the display.
H		Change to the HP instruction set.
J		Change to the Jacobs/De Arras instruction set.
K	*<key>*	Enter a key to the KEY register and set the keydown register. This maps the PC keyboard to the HP-41 keyboard, just as for the 'U' run mode
L	*<register>* *<value>*	Load *<register>* with *<value>*, where *<register>* is A,B,C,N or N and *<value>* is up to 14 hex digits long.
LR	*<RAM register>* *<value>*	Load *<RAM register>* with *<value>* where a RAM register may be any existing register from 000 to 3FF (hex) and the value is up to 14 hex digits long.
M		Display this menu of commands.
QY		Quit the emulator.
R		Run at full speed until a breakpoint, keyboard, POWOFF or instruction limit trap occurs.
S		Switch between disassembly format 1 and 2. Default is format 1. Format 1: [<label>] <instruction> [<operands>] Format 2: <address> <code> <instruction> [<operands>]
T	*<value>*	This sets the instruction trap limit for any value 1 to ???. When the specified number of instructions are executed under 'R' mode, execution is halted unless one of the other trap conditions occurs first.
U		Execute the operating system ROMs (that you loaded) and check the keyboard when a ?KEY instruction is detected (see KEYBOARD MAPPING below). The ESC key will cause a break out of run mode, but it may take several seconds for it to be processed. The +/- in the upper left corner is the busy indicator. It is a green plus when a POWOFF is encountered and a red minus when the emulator is busy executing code.
V		Modify video configuration. The video mode may be set to text or graphics and to direct or BIOS screen memory writes. There is also an option for displaying B&W on color adapters (for LCD screens).
X		Modify ROM configuration. This allows the loading and removal of ROM images from the configuration. Press ESC to exit this mode.

Command (cont.)	*Operand (cont.)*	*Description (cont.)*
Y		Edit ROM code. This causes the emulator to read the .CFG file with the same name as the .ROM file at the PC. The .CFG file is used to determine which .OBJ file the PC is currently at. Then, the emulator executes the editor name given with the $EDITOR command in the .LOD file. After the editor terminates, the emulator executes the linker and the linker executes the assembler to rebuild the .ROM image(s). Then the emulator reloads ALL ROM images as it was before the edit process
Z		Change to the Zencode instruction set.
SPACE BAR		Single step the emulator.

Note: The config file must be created with the /R option so the emulator can read the reference data

Keyboard Mapping

To simulate the HP-41 in USERCODE mode (execute the 'U' command) the PC's keyboard has been mapped to the following specifications. This is not a key for keycode mapping in all cases. For instance, the letters A-Z may only be entered if the emulator sees that ALPHA mode is set (user flag 48). Also, certain keys on the HP-41 keyboard are shifted, but may be entered directly on the PC keyboard *without* first entering a SHIFT on the emulator because the emulator will set shift mode first. An example of this is the % function. The shift key on the PC keyboard does not map to the shift key on the HP-41.

Emulator in Any Mode

PC Key	*HP-41 meaning*	*MCODE keycode value*
F1	ON	18 (hex)
F2	USER	C6
F3	PRGM	C5
F4	ALPHA	C5
F5	SHIFT	C4
F6	SST	C2
F7	<- (BACKARROW)	C3
F8	R/S	87
F9	FUNCTION (described below)	
F10	KEYCODE (described below)	
SHIFT F1	XEQ	32
SHIFT F2	ENTER^	13
SHIFT F3	CHS	73
SHIFT F4	EEX	83

Emulator is not in ALPHA mode:

PC Key	*HP-41 meaning*	*MCODE keycode value*
.	.	77
0 to 9	0 to 9	

Emulator is in ALPHA mode:

PC Key	*HP-41 meaning*	*MCODE keycode value*
A to Z	A to Z	
A to e	a to e	10, 30, 70, 80, C0
S	Sigma	11
N	Not equal	71
X	Append	32
G	Angle symbol	73
% < > ^ $	% < > ^ $	31, 81, C1, 83
- + * /	- + * /	14, 15, 16, 17
Space , .	Space , .	37, 77, 77
0 to 9	0 to 9	

Functions

All HP-41 keys that are marked with something other than a single digit symbol may be entered with the FUNCTION key (F9). The emulator will prompt for the name of the command to enter. The commands are not case sensitive. The tables below lists valid functions for ALPHA and non-ALPHA mode:

Alpha Mode

ON	USER	PRGM	ALPHA	
				BST
	APEND	ASTO	ARCL	SST
				AVIEW
				R?S

Not in Alpha Mode

ON	USER	PRGM	ALPHA	
∑-	Y^X	X^2	10^x	e^x
∑+	1/X	SQRT	LOG	LN
CL∑	%	ASIN	ACOS	ATAN
X<>Y	RDN	SIN	COS	TAN
	ASN	LBL	GTO	BST
SHIFT	XEQ	STO	RCL	SST
CATALOG		ISG	RTN	CLX/A
ENTER^		CHS	EEX	

X=Y?	SF	CF	FS?
X<=Y?	BEEP	P->R	R->P
X>Y?	FIX	SCI	ENG
X=0?	PI	LASTX	VIEW
			R/S

Keycodes

It is also possible to enter hex keycodes directly by using the F10 key. This will enter any hex value from 00 to FF into the KEY register. If in 'U' mode, F10 toggles keycode entry. See Appendix E for the keycode table.

Limitations And Warnings

Some of the TEF instructions have undefined behaviors if the emulator is in decimal mode and there is a value greater than 9 in the register being added or subtracted.

Sometimes the emulator displays code from ROM images that are really not present. This code may be random "garbage" or a duplicate of another ROM. As long as there is no attempt to write to the non-existant pages there should be no problem.

Error Messages

Error Message	***Description***
FATAL ERROR (M04): *Out of memory! - Too many ROM files*	This error occurs when the system's dynamic memory has been all used up.
FATAL ERROR (M05): *Cannot load the same page twice*	The load file has more then one page command with the same parameters.
FATAL ERROR (M06) *Illegal Parameter in $PAGE command*	The parameters specified for the page command are not valid.
FATAL ERROR (M07) *ROM file name not specified in $PAGE command*	The $PAGE command must have a ROM file name specified after it to load.
FATAL ERROR (M08) *No ROM images loaded*	There were not ROM images loaded.
FATAL ERROR (M09) *Label file is empty: <label file>*	The label file specified in the load file was empty.
ERROR (M20) *Illegal $OFFSET command in: <label file>*	The address specified for to the $OFFSET command is not in the range 0000-FFFF (hex).
ERROR (M21) *Illegal label definition: <label> in: <label file>*	The label is greater than 13 characters or is not delimited by brackets or parenthesis.
ERROR (M30) *Unknown result*	A peripheral I/O instruction was executed that has an unknown result for the current peripheral selected.
ERROR (M31) *Unknown Peripheral*	The peripheral select code is undocumented.
ERROR (M32) *Timer not implemented*	The timer I/O instructions are not supported.
ERROR (M33) *Card reader not implemented*	The card reader I/O instructions are not supported.
ERROR (M34) *Printer not implemented*	The printer I/O instructions are not supported.
ERROR (M35): Instruction is not used	

T41 INSTRUCTION SET TRANSLATOR

Syntax

T41 <file>

Description

The instruction set translator translates the mnemonics in a source file from one instruction set to any of the three. (It is possible to translate from Zencode to Zencode, for instance.) The translator reads the source file until it finds an instruction set directive which specifies the current instruction set. Then, it prompts for the new set with self-explanatory messages. The old source file becomes the backup file. Any errors will be flagged and stored in the new source file. If there is more than one instruction set directive in the file, T41 will prompt for a new instruction set each time it finds one.

Files

File name	***Purpose***	***Action***
<file.src>	source file	read from and written to
<file.bak>	backup file	written to

Options

None

Error Messages

Error Message	***Description***
FATAL ERROR (T02): Source File Is Empty! Check backup file	The translator found the source file to be empty. The backup file should contain the original source file.
FATAL ERROR (T03): Failure to rename *<file x>* to *<file y>*	This error will result if for some reason the system prevents the translator from renaming <file x> to <file y>. Check the file access on the files.
FATAL ERROR (T04): Out of memory!	This error occurs when the system's dynamic memory has been all used up.
ERROR (T05): Instruction not given	An instruction was expected on this line and none was found.
ERROR (T06): Illegal label definition: <label>	The label is greater than 13 characters or is not delimited by brackets or parenthesis.
ERROR (T07): Unknown directive: *<directive>*	The specified directive is not valid.
ERROR (T08): Illegal instruction: *"<instruction>"*	The instruction given is not in the current instruction set.

41COM COMMUNICATIONS UTILITY

Syntax

41COM <file>

Description

This utility transmits and receives 4K ROM image files to and from the HP-41 over the HP-IL loop. It requires two programs on the HP-41 called "ROMIN" and "ROMOUT". For information on loading these programs, see the file "BOOT.SRC". On-line help is available by typing H after running 41COM. This utility requires an HP82973A HP-IL interface card for the PC and an HP82160 HP-IL module for the HP-41.

It is also possible to use an HP-IL module in the HP-41 with an HP-IL <=> RS232 convertor and use the RS232 serial port of the PC to send and receive ROM files. The 41COM utility is not needed in this case since the PC can simply send and receive the ROM file over its serial port using any serial upload/download program that supports 8-bit ASCII. The software on the HP-41 end is the same either way.

Files

File name	***Purpose***	***Action***
<file.rom>	ROM file	read from or written to

Options

None

Error Messages

Error Message	***Description***
ERROR: *EOT received before all data sent*	The HP-41 sent an EOT signal before it sent all 8192 bytes.
ERROR: *Time out*	The HP-41 prematurely terminated its transmission and the PC timed out or the HP-41 failed to respond in time to data sent by the PC. The timeout factor is approximately two seconds.

SECTION 5 - *Linking And Disassembly*

Background Information

Because the smallest unit of ROM memory in HP-41 is one page, the linker builds pages. Each HP-41 page is 4096 words long. Each word is 10 bits long. The HP-41 can address a total of 16 pages which are numbered page 0 through page F (hex). Some of these pages are already hard-wired to the HP-41 operating system (See Appendix B). Other pages are left open for the user's plug-in modules and these map to the four I/O ports. Each port has two 4K pages associated with it so a plug-in module can use either one or two pages. The actual hardware of the module determines which page(s) it uses. Most 4K modules use the lower page (although some use the upper) while all 8K modules use both. These configurations are known as port-configured since the actual pages that the module maps into are dependant on which port it is plugged into. Most of the commercially available ROM equipment is of this variety.

12K and 16K modules must be bank-switched as described later in this section.

It is possible to have a module that does not occupy the pages associated with the port it is plugged into. The ROM image(s) in this type of module are hard-configured to always map into the same page(s) regardless of which port it is plugged into. This is the case for several HP-41 peripherals or special modules such as the Mass Storage ROM. For instance, regardless of which port the Mass Storage ROM is plugged into, it will always occupy page 7. (The Mass Storage ROM is inside the HP-IL module.) Since most accessory hardware for the HP-41 on the market today does not support hard-configured ROM images, this is not an alternative to most MCODE programmers. One way to get around this is to pretend that a piece of hardware is hard-configured and always plug it into the same I/O port. This distinction is important since some jump instructions are not position independent.

The linker can link many possible configurations including port- and hard-configurations and bank switching.

Linking Port-Configured ROMs

Most types of user-created ROMs will be of this type. The best way to link these is to start with a $PAGE 8 1 command for the lower ROM. If there is an upper ROM, it would be linked with a command of $PAGE 9 1. If the upper page is bank switched, the hidden page would be linked with a command of PAGE 9 2. The only reason the linker needs to know a page number is to resolve references that are relative to each other. It would work just as well to specify $PAGE 5 for the lower ROM and $PAGE 6 for the upper.

An example of this type of link is the Advantage ROM. It is a 12K bankswitched port-configured module. It could be linked with $PAGE 8 1, $PAGE 9 1 and $PAGE 9 2. If the link file were named ADV.LNK, The ROMs would be named ADV0.ROM, ADV1.ROM and ADV2.ROM, respectively.

Linking Hard-Configured ROMs

To link a ROM of this type, the $PAGE command would be specified with the page that the ROM is to be located at. If there is more than one ROM to be linked, simply specify exactly which page and bank each one is to be located at. The hardware will insure that each ROM is mapped to its proper location. For example, the operating system of the HP-41 is 12K of hard-configured non bank switched ROMs. It could be linked with $PAGE 0, $PAGE 1, $PAGE 2. If the link file were named NUT.LNK, the ROMs would be named NUT0.ROM, NUT1.ROM and NUT2.ROM, respectively.

Disassembly Of Port-Configured ROMs

To disassemble a port-configured ROM, it is best to start at page 8 for the first ROM image and if there is a second, disassemble to page 9. For example, to disassemble an 8K port-configured ROM, do not specify the /Pn option (the default is 8) for the first page and specify /P9 for the second.

Disassembly Of Hard-Configured ROMs

To disassemble a hard-configured ROM, the page that the ROM was taken from must be specified with the /Pn option. For example, to disassemble the operating system of the HP-41, specify /P0 for the first page and /P1 and /P2 for the other two. If the operating system is disassembled into pages 0-2, the mainframe labels will appear properly at the locations they are defined at.

SECTION 6 - *File Types*

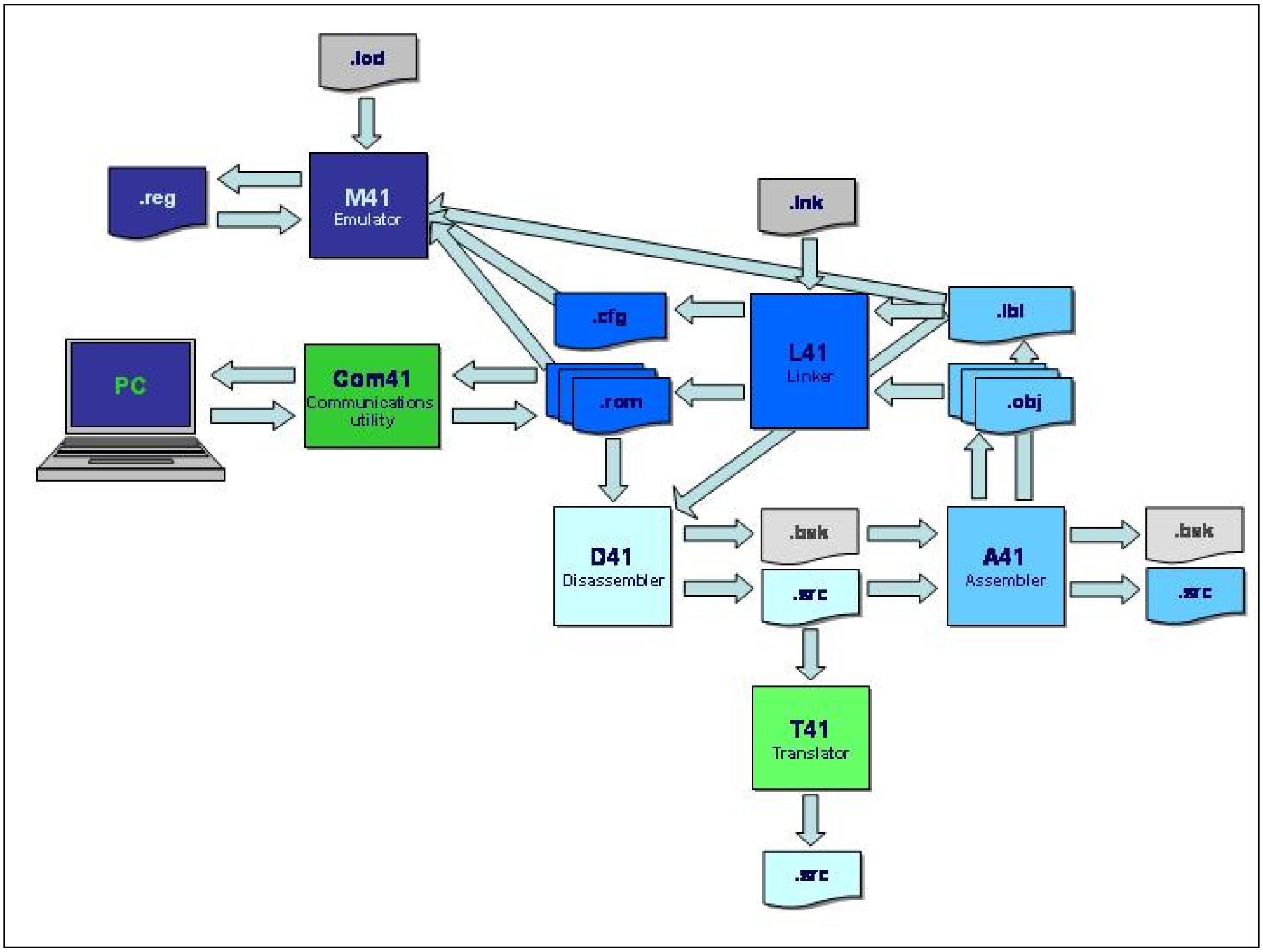


Figure 2 Filesystem and interaction with components of SDK41

SOURCE FILE FORMAT

File Extension

.SRC All source files must have a .SRC extension.

File Type

Normal MS-DOS text file

Line Formats

Directive instruction lines may have comments at the end if the comments are preceded by a semicolon.

.<directive>	Directive lines begin with a period and are followed by the directive which must be in all caps. *Example:* `.NAME "ROMIN"`
; *Comments*	Any line that begins with a semicolon is ignored.

* Error
* Macro
	Error and macro lines begin with an asterisk and are removed by the assembler when they are found in the source file. This type of line may contain anything after the asterisk. They are placed in the source file by the utilities to denote errors in code or to denote the code generated by macro expansions.
	Example:
	`*000A 012               #012        ; "R"`
	`*000B 00F               #00F        ; "O"`
	`*000C 00D               #00D        ; "M"`
	`*000D 009               #009        ; "I"`
	`*000E 00E               #00E        ; "N"`
	`*000F 220               #220        ; " "`

Instruction lines	Any line that is not one of the above is an instruction line and must follow the syntax diagram below:
	<address> <data> <label> INSTRUCTION <operands> ;comments CR
	Example:
	`  0010   37903C03C            NCXQREL [GET_PAGE] 003C`
	`  0013   04E                  C=0   ALL`
	
	`  0014   130017               LDI   017`
	`  0016   1BC                  RCR   11`
	`  0017   158                  M=C`
	`  0018   3751C8   (Start)     NCXQ  72DD              ;loop setup`

Instruction lines have optional fields that may or may not affect assembly.

- The *address* and *data* fields are ignored by the assembler but are rewritten if the source file is rewritten. The user does not place the address and data fields in the source file.
- The *address field* is 4 hexadecimal digits long and the data field is 3, 6 or 9 hexadecimal digits long.
- The *label field* is next and is where a local or global label is defined. Labels without instructions on the same line will cause an error.
- Depending on which *instruction* is used, there may be from zero to three operand fields after the instruction but the assembler uses only the first and the rest are ignored and placed for the user's benefit.
- Any characters after that must be preceded by a semicolon and are treated as a *comment*.

Used By

A41	The assembler reads the old source file and writes a listing to the new source file if the /S or /R option is given; otherwise it does not modify the source file.
D41	The disassembler writes its disassembly listing to a source file. If there is a source file with the same name as the one the disassembler is about to disassemble into, the original one is made into a backup file.
T41	The instruction set translator reads the old instructions from the source file and writes the translated instructions to a new source file of the same name.

File Safety

If the assembler or translator are interrupted, it is possible to lose the source file, but it is not possible to lose both the backup and source files.

Compatibility With SDS

None. There are significant differences in SDS .41A source files and SDK41 .SRC files.

Example

See any of the .SRC files on the SDK41 disk, e.g. *ROMIN.src* or the *SKWID1A.src* example in Appendix I

BACKUP FILE FORMAT

File Extension

.BAK	All backup files must have a .BAK extension.

File Type

Normal MS-DOS text file

Purpose

Backup files are maintained automatically by the utilities as a measure of file safety. It is not possible to interrupt one of the utilities and loose both the source and backup files.

Line Formats

Same as for source files.

Used By

A41	The assembler will make the old source file into the backup file if /S or /R is given. If a backup file already exists, it is overwritten. If /S is not given the backup file will not be modified.
D41	When the disassembler first executes, it checks for a source file with the same name as the one it is about to disassemble into. If it finds one that already exists, it makes it into a backup file. If a backup file by that same name also exists, it is overwritten. If it does not find a source file it does not do anything to the backup file.
T41	The old source file becomes the backup file every time that the translator is executed. Any preexisting backup file of the same name is overwritten.

ROM FILE FORMAT

File Extension

.ROM All ROM files must have a .ROM extension.

File Type

Binary data file

Data Format

HIGH 2, LOW 8	o Each 10-bit HP-41 word is stored in two 8-bit bytes in a ROM file. o The high 2 bits of the 10-bit word are stored in the low 2 bits of the first byte and the low 8 bits are stored in the second byte. o Since there are 4096 10-bit bytes in each ROM image, all ROM files must be exactly 8192 bytes long.

Used By

L41	The linker writes the final, linked code to one or more ROM files.
D41	The disassembler reads the code from the ROM file and disassembles it.
M41	The emulator can read and execute the ROM image files.
COM41	The communications utility can transmit .ROM files (???)

Compatibility with SDS

???

OBJECT FILE FORMAT

File Extension

.OBJ All object files must have a .OBJ extension.

File Type

Binary data file

Used By

A41	The assembler writes internal global label definitions (entry points), unresolved external references, relocation fixups, and the object code into each object file along with some other information used by the linker.
L41	The linker reads the object files, locates the code in the ROM image, resolves external references and fixes up all references.

Compatibility With SDS

Good. SDK41 .OBJ files may be renamed .41O and used with the SDS LINK41 utility. SDS .41O files may be renamed .OBJ and used with the SDK41 L41 utility but the SDS file MFENTRY.41O must be renamed MFENTRY.OBJ and linked in also.

LINK FILE FORMAT

File Extension

.LNK	All link files must have a .LNK extension.

File Type

Normal MS-DOS text file

Purpose

Link files are used to direct the linker in the creation of the ROM files.

Line Formats

<object file name>	This is the file name of the object file to load given *without* the .OBJ extension. The object data is loaded at the current load address in the current ROM image. The load address is then incremented so the next object file will load immediately following the previous.
$PAGE <*page*> $PAGE <*page*> <*bank*> $PAGE <*page*> <*ROM name*> $PAGE <*page*> <*bank*> <*ROM name*>	This defines the current page where <*page*> is 0 to F (hex) and <*bank*> is 1 to 4. If <*bank*> is not given, the default is 1. This command opens a new ROM image at the specified page and bank. All object files specified after this are linked into this ROM image until another $PAGE command is given or the link file ends. When there is only one ROM image it will be named the name of the link file (with a .ROM extension). If there is more than one ROM image, a number from 0 to the number of ROM images MINUS one is appended to this name. If <*ROM name*> is specified, that name will be used to name the ROM image instead.

Line Formats (cont.)

$LOC <address>	Where <*address*> is a value 0000 to FFFF (hex). This command changes the load address so the object files following it are loaded consecutively starting at this address. If <*address*> is not in the same page as the page given with the current $PAGE command, a warning message will result and the linker will force <*address*> into the current page. If this command is not given, the initial load address is p000 where p is the current page.
$CH	This causes the checksum to be computed and placed in location FFF (hex) for the current ROM image. This command must be given for each page where a checksum is desired. If location FFF is occupied by any object code, an error message is generated and the checksum is written anyway.
$LABELS *<label file>*	The GLOBAL labels in the *<label file>* will be read and used to link the ROM image(s). Local labels are ignored.
; anything * anything * blank line	Any line that begins with a semicolon, asterisk or is blank is ignored.

Used By

L41	Link files are used to direct the linking of object files.

CONFIGURATION FILE FORMAT

File Extension

.CFG All configuration files must have a .CFG extension.

File Type

Normal MS-DOS text file

Purpose

This type of file is only written by the linker and is used to show which ROM is mapped to which page as specified by the link file. The configuration file may optionally have the symbol cross reference table written to it.

Line Formats

$PAGE <*page*> <*bank*> *<rom file>*	This documents the mapping of <rom file> to the specified page and bank where <*page*> is 0 to F (hex) and <*bank*> is 1 to 4.
; anything * anything blank line	Any line that begins with a semicolon, asterisk or is blank is ignored.

Used By

L41	Config files are written by the linker. The linker will write the ROM image mapping and optionally a symbol cross reference table.
M41	The emulator reads the config file if the 'Y' command is given. The emulator first tries to read the config file with the same name as the ROM image that the PC is currently in. If this fails and there is a number on the end of the name, it will remove the number and try again. *Example:* If the PC is in page 1 and the ROM name is NUT1, the emulator will look for NUT1.CFG and NUT.CFG.

Example

After executing the demo on the SDK41 disk, a configuration file with the name PCCOM.CFG will be created.

LOAD FILE FORMAT

File Extension

.LOD All load files must have a .LOD extension.

File Type

Normal MS-DOS text file

Purpose

This file is very similar to the config file format. It is only used by the emulator to load the ROM images and label files.

Line Formats

$PAGE *<page>* *<bank>* *<rom file>*	The ROM file will be loaded into the specified page and bank where *<page>* is 0 to F (hex) and *<bank>* is 1 to 4.
$EDITOR <editor name> <option1> <option2>	Defines which editor to use when modifying code with the 'Y' command. If given, the two options are passed on the command line after the source file name. The call will look like: *<editor name> <source file name> <option1> <option2>*. The default editor is "SEE" with no options.
; anything * anything blank line	Any line that begins with a semicolon, asterisk or is blank is ignored.
$LABELS *<label file>*	The labels in the *<label file>* will be read and incorporated into the disassembly listing of the emulator.
$REG *<reg file>* $RUN	This causes the emulator to go immediately into 'U' mode and not stop without ending the entire program. Since it is not possible to do things like change the video mode once the emulator starts, the $REG command can be used to load a pre- configured system.

Used By

M41	Load files are read by the emulator and specify the ROM images to read in. The file *DEFAULT.LOD* is special since the emulator will read it if it exists and a load file name is not specified on the command line

LABEL FILE FORMAT

File Extension

.LBL All label files must have a .LBL extension.

File Type

Normal MS-DOS text file

Line Formats

<*label*> <*address*>	Where <*label*> is any global or local label and <*address*> is a value 0000 to FFFF (hex). Labels do not have to be in any particular order. If the label file has more than about fifty labels in it, they should NOT be placed in alphabetical or reverse alphabetical order.
$OFFSET <*address*>	Where <*address*> is a value 0000 to FFFF (hex). This address is simply added to each of the addresses of the labels affected. The labels affected are those that appear *after* each $OFFSET command and *before the next* $OFFSET command or the end of file. This command may be given more than once in the same file and if not given at all, the default is 0000.
; anything * anything blank line	Any line that begins with a semicolon, asterisk or is blank is ignored.

Used By

A41	When the /L option is given, the assembler writes all local labels to a label file that has the same name as the source file. It does not write global labels.
L41	When the /L or /LL option is given, the linker writes all global labels read from all object files to one label file that has the same name as the link file. If the /LL option is given, the labels stored in the label files with names corresponding to object files are read and incorporated into the one label file.
D41	For each /L:<label file> option specified on the command line of the disassembler, all local and global labels are read from the specified file and incorporated into the source listing just as for the mainframe labels. The addresses specified by each label are used exactly as they appear unless they are offset by an $OFFSET command in the label file.
M41	The emulator reads label files that are specified in the load file and incorporates them into its disassembly listings.

Example

```
[FOOBAR_A] 0023      ; interpreted as 0023
(FOOBAR_D) 32C2      ; interpreted as 32C2
$OFFSET B000
[FOOBAR_B] 04A3      ; interpreted as B4A3
$OFFSET 0200
(FOOBAR_C) 032B      ; interpreted as 052B
```

REGISTER FILE FORMAT

File Extension

.REG All REG files must have a .REG extension.

File Type

Binary data file

Purpose

Register files contain all RAM register data, breakpoints, video configuration and other data necessary to preserve the state of the emulator.

Used By

M41	The emulator reads and writes the CPU and RAM registers into a register file.

SECTION 7. – *Loading the Operating System ROMs*

The following .rom files are ROM images found in HP-41C, HP-41CV and HP-41CX series programmable calculators.

Reproduction of HP-41 ROM images by permission of Hewlett-Packard.
Hewlett-Packard Company makes no warranty as to the accuracy or completeness of the foregoing information and hereby disclaims any responsibility therefore.

HP 41C/CV			
Page	*Bank*	*Name*	*Description*
0	1	NUT0.ROM	HP-41C operating system
1	1	NUT1.ROM	HP-41C operating system
2	1	NUT2.ROM	HP-41C operating system
HP 41CX			
Page	*Bank*	*Name*	*Description*
0	1	XNUT0.ROM	HP-41CX operating system
1	1	XNUT1.ROM	HP-41CX operating system
2	1	XNUT2.ROM	HP-41CX operating system
3	1	CXFUNS0.ROM	HP-41CX extended functions (built in)
5	2	CXFUNS1.ROM	HP-41CX extended functions (built in)
ROM			
Page	*Bank*	*Name*	*Description*
5	1	TIMER.ROM	Timer functions (built in to CX)
6	1	PRINTER.ROM	82143 Thermal Printer functions
n	1	ADV0.ROM	Advantage Pac
n+1	1	ADV1.ROM	Advantage Pac
n+1	2	ADV2.ROM	Advantage Pac

*Where n can be 8,A,C or E

Alternatively, one can create these images from a working HP 41 C/CV/CX via the following procedure:

1. The HP-41 operating system ROMS must be loaded in order to run the Emulator.
2. Using a machine code editor (like the Zenrom MCED) manually enter the *BOOT.SRC* MCODE program into your MLDL (Machine Language Development Lab). Specific instructions are in the *BOOT.SRC* file.
3. Connect your HP-41 to your PC through the HP-IL interface loop module and card.
4. Type *41COM PCCOM* on the PC.
5. *XEQ "BOOTCOM"* on your HP-41. This will load the PCCOM.ROM communications program onto your MLDL.
6. Type *41COM NUT0.ROM* on the PC.
7. *XEQ "ROMOUT"* on your HP-41 and enter '*0*'. This will store the first page of the HP-41's operating system in the file *NUT0.ROM* on your PC.
8. Repeat this procedure for pages 1 and 2, storing them into *NUT1.ROM* and *NUT2.ROM*, respectively.
9. If you have an HP-41CX, you will also need page 4. Store this *CXFUNS0.ROM*.

SECTION 8 - *Reference Instruction Formats*

SHORT JUMPS

Instructions

JNC <*operand*>	Jump on no carry, otherwise do nothing
JC <*operand*>	Jump on carry, otherwise do nothing

Description

The short jump instruction jumps -64 to +63 words from where the instruction is. This type of instruction encodes only a relative displacement in words in its bits.

Formats

JNC <+*disp*> JNC <-*disp*>	Where <+*disp*> is +0 to +63 and <-*disp*> is -1 to -64 (dec). The actual address that is jumped to is relative to where the short jump instruction is located after it is assembled and linked and the ROM is plugged into the HP-41. Regardless of where the instruction ends up, it always jumps the same number of words forward or backward.
JNC <*address*>	Where <*address*> is 0000 to FFFF (hex). The assembler resolves this type by taking the specified address and subtracting the jump instruction's ASSEMBLE TIME ADDRESS to get a positive or negative displacement. This should *only be used* if the object module is assembled with the .ORG directive so that the assemble time address will be the SAME as the run time address. This is a literal specification and the linker does not relocate anything. It is possible to specify an address that is too far from the jump, causing A41 to report an error.
JNC <*label*>	Where <*label*> is any local or global label. The address that will be jumped to is whatever the value of the label is, assuming it is within range. If the label is external, the linker must resolve the reference, otherwise it is completely resolved during assembly.

LONG JUMPS

Instructions

NCGO <*operand*>	Goto on no carry, otherwise do nothing
CGO <*operand*>	Goto on carry, otherwise do nothing
NCXQ <*operand*>	Execute on no carry, otherwise do nothing
CXQ <*operand*>	Execute on carry, otherwise do nothing

Description

These instructions occupy two words each and can jump to anywhere in the HP-41 address space. The address that they jump to is encoded in the instruction's bits and regardless of where the instruction is located, it always jumps to the same location. The GOTO-types of long jumps simply cause an immediate jump depending on the carry flag. The EXECUTE-types do the same but also push a return address onto the HP-41's return stack. The value that is pushed is the address of the word that follows the long jump.

Formats

NCXQ <*address*>	Where <*address*> is 0000 to FFFF (hex). The assembler resolves this type of reference literally and the linker does not change it.
NCXQ <*label*>	Where <*label*> is any local or global label. If the label is internal, it is resolved but will be fixed up (resolved again) at link time. If the label is a mainframe label, the assembler resolves it immediately without any relocation possible. If it is not a mainframe label or an internal, then the label is external and the linker must resolve the reference after it relocates all of the object files. If the /E option is specified, the mainframe label table is erased and all references to mainframe labels are treated as references to externals.

OTHER REFERENCE INSTRUCTIONS

The rest of the jump and reference instructions use a general format described below:

INSTRUCTION <*value*>	Where INSTRUCTION is one of the other reference "instructions" described below and <*value*> is some number usually from 0000 to FFFF (hex). This value is used exactly as it appears and is not relocated.
INSTRUCTION <*symbol*>	Where <*symbol*> is any local or global symbol. The symbol is relocated by the linker if it is a relative label but is not relocated if it is an absolute symbol.

For the descriptions below assume [FOOBAR] resolves to the value 06D2 (hex).

CON	Insert constant into Rom	This is not an instruction at all but is used to directly enter a 10 bit value into the ROM image. This value could represent data, or even an instruction. CON 123 would make the object data contain the value 123 while CON [FOOBAR] will be 6D2. If the high 6 bits are not all zero a warning message will be displayed and the assembler will only use the low 10 bits.
DEFP4K DEFR4K DEFR8K U4KDEF U8KDEF	Not used, kept for SDS comp. Def 4k MCODE FAT entry Def 8k MCODE FAT entry Def 4k User Code FAT entry Def 4k User Code FAT entry	These FAT entry pseudo-instructions are used to define the FAT and should only appear in the first part of the ROM image.
LC3		This is a macro instruction that expands into three LC instructions containing the low three nybbles of the address or symbol. *LC3 E2A4* expands to LC 2, LC A, LC 4 *LC3 [FOOBAR]* expands to LC 6, LC D, LC 2. If the high 4 bits are not all zero a warning message will be displayed and the assembler will only use the low 12 bits.
NCGOREL, NCXQREL		The quad relative goto and execute pseudo-instructions are used to jump from one address to another within a port-configured ROM image. They call special mainframe routines that do the actual jump so they are slower and use 1 or 2 return stack levels. (See Section 5 for port-configured ROM images)

APPENDIX A - *Common XRom Id's*

1	Math	HP-41Z			
2	Statistics	David Assembler	Sandmath		
3	Surveying	MC TEST	Sandmath		
4	Finance	ES-41	ML EPROM		
5	Standard	Zenrom			
6	Circuit Analysis	ES-41	Alpha ROM	David Assembler Mainframe	TOMS Rom
7	Structures	HEPAX			
8	Stress Analysis	Sandbox			
9	Home Management	CCD	Zengrange Programmer		
10	Games	Auto/Dup	PPC ROM		
11	Real Estate	Eramco	CCD	Paname	
12	Machine Design	Melbourne Rom	Toulouse Rom		
13	Thermal	Sandbox	Toolbox		
14	Navigation				
15	Petroleum	Mountain Computer	MC-EPROM by PPC		
16	Petroleum	MM EPROM			
17	Plotter	NFCROM	BLDRom		
18	Plotter	AECRom			
19	Securities	Structures	Clinical Lab	Aviation	HP-IL Diag
20	PPC ROM				
21	Data Acquisition	Assembler 3	ML Rom	Profiset MCODE	
22	HP-IL Development	Advantage			
23	Extended I/O				
24	HP-IL Development	Advantage	MLDL OS		
25	Extended Functions				
26	Time				
27	Wand	Extended IL	Profiset Tools	Profiset OS	
28	Mass Storage				
29	Printer				
30	Card Reader				
31	Data Acquisition	Astro ROM I & II	Profiset OS	W&W Rambox	

APPENDIX B – *ROM and RAM Memory Maps*

HP-41 Rom Memory Map

Page	Physical Location	Bank 1	Bank 2
F	Port 4	Upper Page	
E	Port 4	Lower Page	
D	Port 3	Upper Page	
C	Port 3	Lower Page	
B	Port 2	Upper Page	
A	Port 2	Lower Page	
9	Port 1	Upper Page	
8	Port 1	Lower Page	
7	Internal	HP-IL Mass Storage	not used
6	Internal	Printer ROM	not used
5	Internal	Timer ROM	X-Func Rom (CX)
4		*Reserved by HP for Service*	
3	Internal	X-Func Rom (CX)	not used
2	System	OS ROM 2	not used
1	System	OS Rom 1	not used
0	System	OS Rom 0	not used

HP-41 Ram Memory Map

Address	***RAM***
3FF	
	Extended Memory #2
300	
2FF	
	Extended Memory #1
200	
1FF	Top of Main Memory
	------------------ data register 00 --------------------
	top of User programs
	------------------------ .END. -------------------------
	I/O Buffer Area
0C0	Key Assignements
0BF	Top of X-Function X-Memory
040	Bottom of X-Function X-Memory

Nonexistent registers
VOID

00F	Status Registers
	T-Z-Y-X-L, M-N-O-P, Q,k, a -b-c -d -e
000	*0-1-2-3-4, 5 -6 -7-8, 9,A, B,C,D,E,F)*

APPENDIX C – *Format for ROM with FAT*

p is the number of the page that the ROM maps to. This format is used for ROMs at pages 3 and 5 to F.

Address	***Content Description / Value***	***Example***	
p000	ROM ID Number	8000	001 ;XROM ID=1
p001	Number of Functions (n)	8001	002 ; 1 header, 1 func
------------------ FAT -------------------------			
p002	Address of First Function	8002	000 ;address of first executable
p003	" "	8003	08C ;for first func (often Cat 2 name)
.		8004	000 ;address of first executable
.		8005	091 ;for 2nd function (Y<>X function)
p(2n)	Address of Last Function	8006	000 ;end of FAT
p(2n+1)	" "	8007	000 ;end of FAT
p(2n+2)	FAT Terminator (must be loaded with 000)		
p(2n+3)	000		
------------------ CODE ------------------------			
p(n2+4)		8084	081 ;”A”, last letter of Cat 2 name (SKWID 1A)
.		808C	3E0 ;RTN, first executable instruction
.		808D	09A ;”Z”, last letter of Y<>Z function
pFF3		8091	0B8 ;Read 2(Y)
------------------ POLLING VECTORS -------------			
pFF4	Pause Loop		
pFF5	Main Running Loop		
pFF6	Deep Sleep Wake Up With No Key Down		
pFF7	Power Off		
pFF8	I/O Service		
pFF9	Deep Sleep Wake Up		
pFFA	Cold Start		
------------------ ROM TERMINATOR --------------			
pFFB	Revision Level Characters (optional)		
pFFC	" "		
pFFD	" "		
pFFE	" "		
pFFF	Checksum (optional)		

APPENDIX D - *Character Translation Table*

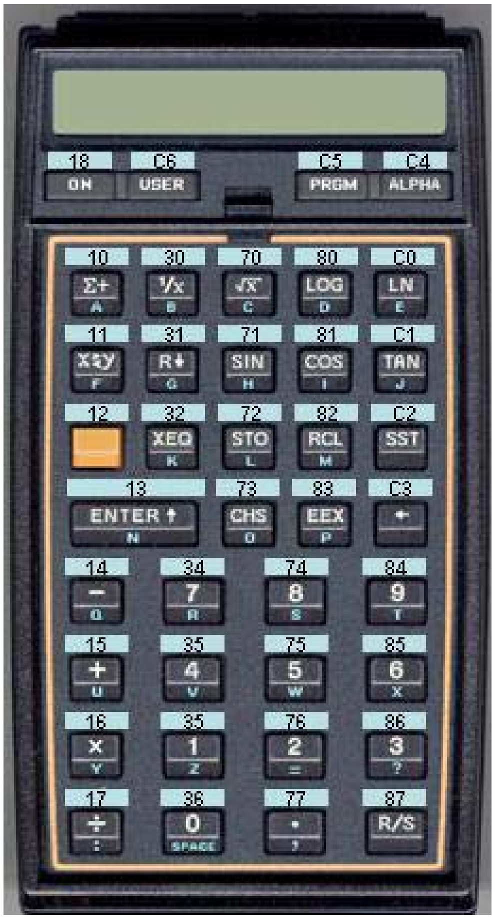
This translation does not apply to the emulator when it is in GRAPHICS DIRECT video mode. GRAPHICS DIRECT mode supports all HP-41 halfnut characters. (The halfnut display is the one with the rounded edges). The halfnut display contrast adjustment is not supported.

HP Code	*IBM Code*	*Char*
00	40	@
01	41	**A**
02	42	**B**
03	43	**C**
04	44	**D**
05	45	**E**
06	46	**F**
07	47	**G**
08	48	**H**
09	49	**I**
0A	4A	**J**
0B	4B	**K**
0C	4C	**L**
0D	4D	**M**
0E	4E	**N**
0F	4F	**O**
10	50	**P**
11	51	**Q**
12	52	**R**
13	53	**S**
14	54	**T**
15	55	**U**
16	56	**V**
17	57	**W**
18	58	**X**
19	59	**Y**
1A	5A	**Z**
1B	5B	**[**
1C	5C	\
1D	5D	**]**
1E	5E	**^**
1F	5F	**_**

HP Code	*IBM Code*	*Char*
20	20	**Space**
21	21	**!**
22	22	**"**
23	23	**#**
24	24	**$**
25	25	**%**
26	26	**&**
27	27	**'**
28	28	**(**
29	29	**)**
2A	2A	*****
2B	2B	{
2C	2C	**-**
2D	2D	}
2E	2E	**/**
2F	2F	**0**
30	30	**1**
31	31	**2**
32	32	**3**
33	33	**4**
34	34	**5**
35	35	**6**
36	36	**7**
37	37	**8**
38	38	**9**
39	39	~
3A	3A	**;**
3B	3B	**<**
3C	3C	**=**
3D	3D	**>**
3E	3E	**?**
3F	3F	

HP Code	*IBM Code*	*HP Char*	*IBM Char*
100	2B	Left Goose	**{**
101	2D	Right Goose	**}**
102	39	Boxed Star	**~**
103		Comma character	**;**
104		Append	**x**
105		Overbar	**o**
106		Single Quote	**`**
107		one leg hangman	**t**
108		two leg hangman	**u**
109		one arm hangman	**v**
10A		Full hangman	**w**
10B		micro	**m**
10C		Not equal to	**n**
10D		Sigma	**s**
10E		Angle symbol	**g**
10F			

APPENDIX E - *Keycode Table*



APPENDIX F - *RAM Configuration Data*

M41 emulates extended memory to support the HP-41CX. If a ROM is loaded into page 3 when M41 is run, it assumes all possible RAM registers are available.

	Any HP41	*HP41CV + X/F +2XM* *HP41 CX*
	no ROM in page 3	ROM in page 3
000 - 00F	☑	☑
040 - 0BF	☒	☑
0C0 - 1FF	☑	☑
201 - 2EF	☒	☑
301 - 3EF	☒	☑

APPENDIX G - *CPU Special Cases*

1. The bankswitching scheme has several special cases that are not well documented. If the PC is in page 3 or page 5, and ENBANK2 is executed, page 5 bank 2 is enabled. If an ENBANK is executed in pages 8 to F, it affects both banks of the current port: 8-9 A-B, etc.
2. If a NCXQ instruction is executed and there is no ROM at that address, the PC is only incremented by 2.
3. The HEPAX ROM uses ENBANK3 and ENBANK4. These are emulated for pages 5 to F.
4. When a POWOFF is executed, an ENBANK1 is effectively executed, the PC is set to 0000 and the carry is set according to the state of the display. If the display is ON, the CPU goes into light sleep and the carry is cleared. If the display is OFF, the CPU goes into deep sleep and will not wake up unless the ON key is pressed.

APPENDIX H – *CPU Registers and Structure*

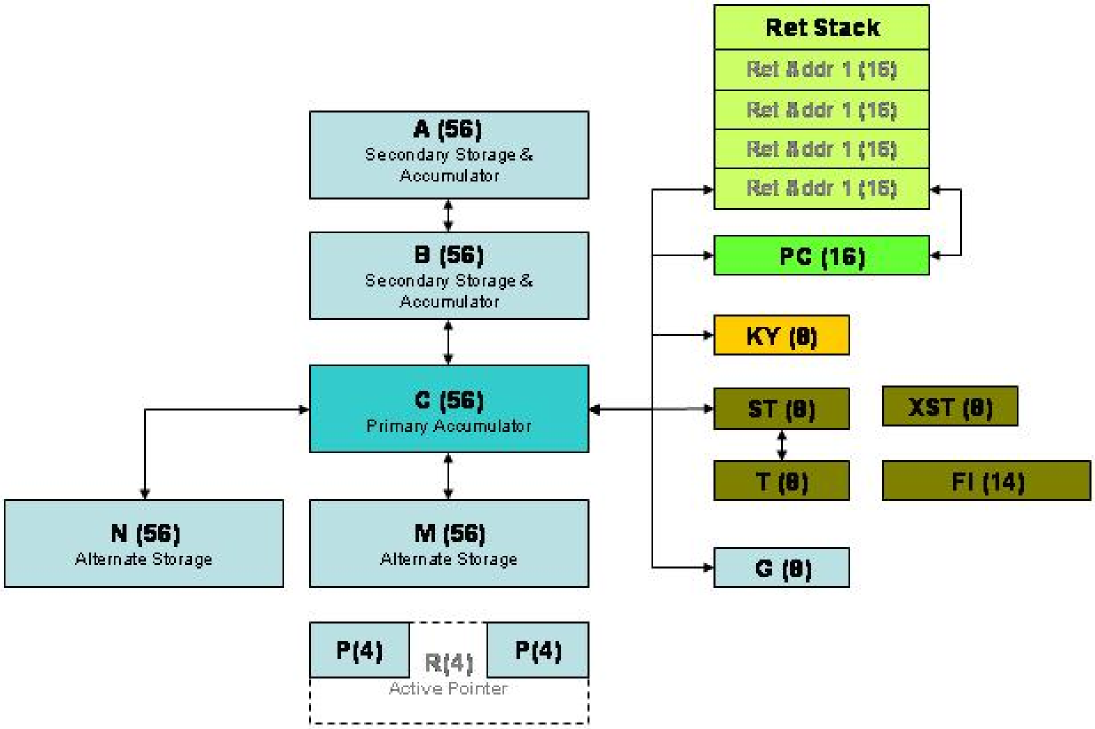


Figure 3 CPU Registers and communication paths

CPU Registers

HP	*Zencode*	*Jacobs/De Arras*	*Bits*	*Description*
C	C	C	56	Primary accumulator
A	A	A	56	Secondary storage and accumulator
B	B	B	56	Secondary storage and accumulator
M	M	M	56	Alternate storage
N	N	N	56	Alternate storage
P	P	P	4	Nybble Pointer
Q	Q	Q	4	Nybble Pointer
PT	PT	R	4	Either P or Q, whichever is the active pointer
G	G	G	8	Alternate storage (often flags)
F	F	T	8	Beeper register
Status Bits	ST	ST	8	Flag register. Lower CPU flags 7-0. Read write via C
Status Bits	XST	XST	6	Upper CPU flags 13-8. No access via other registers
		FI	14	Peripheral flags. No access via other registers. Set by peripheral
Carry Flag	Carry Flag	Carry Flag	1	Set by some instructions; cleared after all others
Keydown Flag	Keydown Flag	Keydown Flag	1	Set when key register has data ready
KY	KY	KY	8	Key register. Contains the keycode entered by pressing a key
STK	STK	ADR	16	The first address on the return stack
Ret Stack	Ret Stack	Ret Stack	16x4	Return Stack. Contains 4 16-bit words that hold return addresses
PC	PC	PC	16	Program counter. Points to next instruction to execute

56 Bit Register Format

```
Nybble: 13 : 12 : 11 : 10 :  9 :  8 :  7 :  6 :  5 :  4 :  3 :  2 :  1 :  0
Name     S    M    M    M    M    M    M    M    M    M    X    X    X    X
                                                                 XS
```

Nybble 13 is the most significant; 0 is the least
Notation: A[x:y] means all nybbles in A REG from x to y; A[x] is just nybble x

APPENDIX H - *Instruction set cross reference table*

This document contains all of the instructions supported by SDK41 and is intended as a reference guide to programming the HP-41.

Note: When the Jacobs/De Arras set failed to contain instructions (such as the display instructions) HP mnemonics were substituted. The mnemonics NCXQREL and NCGOREL were created for SDK41.

General Purpose Instructions

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
004	ST=0	CF	CLRF	0 to 13 (dec)	clear flag
008	ST=1	SF	SETF	0 to 13 (dec)	set flag
00C	ST=1?	?FS	?FSET	0 to 13 (dec)	set carry if flag set
3C4	CLRST	ST=0	ST=0		clear lower CPU flags (7-0)
358	ST=C	ST=C	ST=C		copy C[1:0] into ST
398	C=ST	C=ST	C=ST		copy ST into C[1:0]
3D8	CSTEX	C<>ST	C<>ST		exchange C[1:0] and ST

Load Constants

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
010	LC	LC	LD@R	0 to F (hex)	load constant to C[PT], then decrement pointer
010,010, 010	LC3	LC3	LD@R3	000 to FFF (hex)	Do three LC instructions
130,000	LDI	LDI	LDIS&X	000 to 3FF (hex)	load next word in ROM into C[2:0]
000	CON	CON	CON	000 to 3FF (hex)	enter hex constant into ROM
000	FCNS	FCNS	FCNS	0 to 64 (dec)	decimal constant same as CON
000	XROM	XROM	XROM	1 to 31 (dec)	decimal constant same as CON

Pointer

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
014	?PT=	?PT=	?R=	0 to 13 (dec)	set carry if active pointer equal
01C	PT=	PT=	R=	0 to 13 (dec)	set active pointer
0A0	SELP	PT=P	SLCTP		make P the active pointer
0E0	SELQ	PT=Q	SLCTQ		make Q the active pointer
120	?P=Q	?P=Q	?P=Q		set carry if P=Q
3D4	DECPT	-PT	R=R-1		decrement the active pointer
3DC	INCPT	+PT	R=R+1		increment the active pointer

Ram Accessing

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
270	DADD=C	RAMSLCT	RAMSLCT		select the RAM chip addressed in C[2:0]
2F0	DATA=C	WDATA	WRITDATA		writes C[13:0] to selected RAM register
038	C=DATA	RDATA	READDATA		writes selected RAM register to C[13:0]
028	REGN=C	REG=C	WRIT	0 to F (hex)	write to RAM register in selected chip
038	C=REGN	C=REG	READ	0 to F (hex)	read from RAM register in selected chip

Rom Accessing

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
330	CXISA	RDROM	FETCHS&X		copies the ROM data addressed in C[6:3] into C[2:0]
040	WMLDL	WMLDL	WROM		write C[2:0] to address C[6:3]
100	ENROM1	ENBANK1	ENROM1		enable ROM bank 1 for current ROM device
180	ENROM2	ENBANK2	ENROM2		enable ROM bank 2 for current ROM device
140	ENROM3	ENBANK3	ENROM3		enable ROM bank 3 for current ROM device
1C0	ENROM4	ENBANK4	ENROM4		enable ROM bank 4 for current ROM device

Keyboard

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Description*
220	C=KEYS	C=KEY	C=KEY	copy key register to C[4:3]
230	GOKEYS	GTOKEY	GTOKEY	copy key register into the low byte of the PC
3C8	RSTKB	CLRKEY	CLRKEY	clear keydown flag if no key pressed and clear key register
3CC	CHKKB	?KEY	?KEY	set carry if keydown flag is set

Mode Setting

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
060,000	POWOFF	POWOFF	POWOFF		halt the CPU
260	SETHEX	SETHEX	SETHEX		set hexadecimal mode
2A0	SETDEC	SETDEC	SETDEC		set decimal mode
2E0	DISOFF	DISOFF	DSPOFF		turn display off
320	DISTOG	DISTOG	DSPTOG		toggle state of display

M, N, G, F Registers

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
070	N=C	N=C	N=C		copy C[13:0] into N
0B0	C=N	C=N	C=N		copy N into C[13:0]
0F0	CNEX	C<>N	C<>N		exchange C[13:0] and N
158	M=C	M=C	M=C		copy C[13:0] into M
198	C=M	C=M	C=M		copy M[13:0] into C
1D8	CMEX	C<>M	C<>M		exchange C[13:0] and M
058	G=C	G=C	G=C		copy C[PT+1:PT] into G (if PT= 13, high byte is undefined)
098	C=G	C=G	C=G		copy G into C[PT+1:PT] (if PT= 13, high byte is undefined)
0D8	CGEX	C<>G	C<>G		exchange C[PT+1:PT] and G (if PT= 13, high byte is undefined)
258	F=SB	F=ST	T=ST		copy ST into beeper register
298	SB=F	ST=F	ST=T		copy beeper register into ST
2D8	FEXSB	ST<>F	ST<>T		exchange ST and beeper register

Other

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
000	NOP	NOP	NOP		no operation
160	?LLD	?BAT	?LOWBAT		set carry if battery is low
03C	RCR	RCR	RCR	0 to 13 (dec)	rotate C reg right by the nybble
370	C=CORA	C=CORA	C=CORA		C[13:0] = C bitwise or A
3B0	C=C&A	C=CANDA	C=CANDA		C[13:0] = C bitwise and A
1A0	CLRABC	ABC=0	A=B=C=0		clear all nybbles of A,B,C registers

Time Enable Field Instructions

Nybble:	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	**S**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**X**	**X**	**X**	**X**
												XS		

HP	*Zencode*	*Jacobs/DeArras*	*Nybbles*	*Description*
PT	PT	@R	[PT]	Nybble pointed to by active pointer
X	X	S&X	[2:0]	Sign and exponent
WPT	WPT	R<	[PT:0]	Nybbles pointed to by active pointer through nybble 0
W	ALL	ALL	[13:0]	Entire register
PQ	PQ	P-Q	[Q:P]	Nybbles from pointer Q to pointer P subject to: if P<=Q then [Q:P]; if P>Q then [13:P]
XS	XS	XS	[2]	Exponent sign
M	M	M	[12:3]	Mantissa
S	S	MS	[13]	Mantissa sign

All of the instructions in the following group work on the above Time Enable Fields (TEF). C[TEF] is the C register with whatever field is selected from above. Ex: A[PT], A[XS] etc. Any of the arithmetic TEF instructions set the carry flag if either an overflow or underflow occurs.

Time Enabled Instructions – 1 byte

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
002	A=0	A=0	A=0	TEF	clear A[TEF]
022	B=0	B=0	B=0	TEF	clear B[TEF]
042	C=0	C=0	C=0	TEF	clear C[TEF]
062	ABEX	A<>B	A<>B	TEF	exchange A[TEF] and B[TEF]
082	B=A	B=A	B=A	TEF	copy A[TEF] into B[TEF]
0A2	ACEX	A<>C	A<>C	TEF	exchange A[TEF] and C[TEF]
0C2	C=B	C=B	C=B	TEF	copy B[TEF] into C[TEF]
0E2	BCEX	B<>C	B<>C	TEF	exchange B[TEF] and C[TEF]
102	A=C	A=C	A=C	TEF	copy C[TEF] into A[TEF]
122	A=A+B	A=A+B	A=A+B	TEF	add B[TEF] to A[TEF]
142	A=A+C	A=A+C	A=A+C	TEF	add C[TEF] to A[TEF]
162	A=A+1	A=A+1	A=A+1	TEF	add one to A[TEF]
182	A=A-B	A=A-B	A=A-B	TEF	subtract B[TEF] from A[TEF]
1A2	A=A-1	A=A-1	A=A-1	TEF	subtract one from A[TEF]
1C2	A=A-C	A=A-C	A=A-C	TEF	subtract C[TEF] from A[TEF]
1E2	C=C+C	C=C+C	C=C+C	TEF	double C[TEF]
202	C=A+C	C=A+C	C=C+A	TEF	add A[TEF] to C[TEF]
222	C=C+1	C=C+1	C=C+1	TEF	add one to C[TEF]
242	C=A-C	C=A-C	C=A-C	TEF	subtract C[TEF] from A[TEF] store in C[TEF]
262	C=C-1	C=C-1	C=C-1	TEF	subtract one from C[TEF]
282	C=-C	C=-C	C=0-C	TEF	16's complement of C if in hex mode; 10's complement if dec mode
2A2	C=-C-1	C=-C-1	C=-C-1	TEF	15's complement if hex mode; 9's complement if dec mode
2C2	?B#0	?B#0	?B#0	TEF	set carry if B[TEF] is not equal to 0
2E2	?C#0	?C#0	?C#0	TEF	set carry if C[TEF] is not equal to 0
302	?A<C	?A<C	?A<C	TEF	set carry if A[TEF] is less than C[TEF]
322	?A<B	?A<B	?A<B	TEF	set carry if A[TEF] is less than B[TEF]
342	?A#0	?A#0	?A#0	TEF	set carry if A[TEF] is not equal to 0
362	?A#C	?A#C	?A#C	TEF	set carry if A[TEF] is not equal to C[TEF]
382	ASR	ASR	RSHFA	TEF	shift A right by one nybble (leftmost byte set to 0)
3A2	BSR	BSR	RSHFB	TEF	shift B right by one nybble (leftmost byte set to 0)
3C2	CSR	CSR	RSHFC	TEF	shift C right by one nybble (leftmost byte set to 0)
3E2	ASL	ASL	LSHFA	TEF	shift A left by one nybble (rightmost byte set to 0)

Time Enabled Instructions – 2 byte

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
062,082	A=B	A=B	A=B	TEF	copy B[TEF] into A[TEF]
0E2,0C2	B=C	B=C	B=C	TEF	copy C[TEF] into B[TEF]
0A2,102	C=A	C=A	C=A	TEF	copy A[TEF] into C[TEF]

Jumping Instructions

There are duplicate mnemonics for three of the HP jump instructions. GONC is the same as GOTO, GSUBNC is the same as GOSUB, and GOLNC is the same as GOLONG. HP's assemblers will check the instruction proceeding a GOTO, GOSUB or GOLONG to be sure that it cannot set the carry. This insures that these mnemonics cause an unconditional jump and not a just a jump on no carry. SDK41 does not do this and assembles the duplicates exactly the same.

Short Jumps					
Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
007	GOC	JC	JC	-64 to +63 (dec)	short relative jump on carry
003	GONC	JNC	JNC	-64 to +63 (dec)	short relative jump on no carry
Long jumps					
Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
001,000	GSUBNC	NCXQ	?NCXQ	ADDRESS	execute on no carry
001,001	GSUBC	CXQ	?CXQ	ADDRESS	execute on carry
001,002	GOLNC	NCGO	?NCGO	ADDRESS	goto on no carry
001,003	GOLC	CGO	?CGO	ADDRESS	goto on carry
Quad Relative Jumps					
Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
349,08C,000	GSB41C	NCXQREL	?NCXQREL	ADDRESS	execute relative to current quad
341,08C,000	GOL41C	NCGOREL	?NCGOREL	ADDRESS	goto relative to current quad

The HP instructions GSBSAM and GOLSAM are three byte jumps just like GSB41C and GOL41C, but they are limited to jumping into the current 1K quad. SDK41 does not implement GSBSAM and GOLSAM since GSB41C and GOL41C are assembled intelligently by SDK41 to use the same-quad mainframe routines if they can or else use the quad-specific mainframe routines. The following tables show which addresses are used to decide which quad relative routines to assemble to.

Quad Relative Branch Instruction Bytes					
	Quad 0	*Quad 1*	*Quad 2*	*Quad 3*	*Same Quad*
	0-3FF	400-7FF	800-BFF	C00-FFF	
NCXQREL	349 08C	36D 08C	391 08C	3B5 08C	379 03C
NCGOREL	341 08C	365 08C	389 08C	3AD 08C	369 03C
** These are followed by the third byte containing the low 10 bits of the address to jump to.*					
Actual Instruction for Quad Relative Branches					
	Quad 0	*Quad 1*	*Quad 2*	*Quad 3*	*Same Quad*
NCXQ (Address)	[GOSUB0] 23D2	[GOSUB1] 23DB	[GOSUB2] 23E4	[GOSUB3] 23ED	[GOSUB] 0FDE
NCXQ (Address)	[GOL0] 23D0	[GOL1] 23D9	[GOL2] 23E2	[GOL3] 23EB	[GOLONG] 0FDA
Actual Instruction in HP mnemonics					
	Quad 0	*Quad 1*	*Quad 2*	*Quad 3*	*Same Quad*
GOSUB	[GOSUB0]	[GOSUB1]	[GOSUB2]	[GOSUB3]	[GOSUB]
GOSUB	[GOL0]	[GOL1]	[GOL2]	[GOL3]	[GOLONG]

Note that the actual instruction is always NCXQ to the appropriate label and the mainframe routine at the label determines if a return will be pushed or not making either a NCXQREL or a NCGOREL. You can't use a NCGO or a CGO to jump to one of these routines because these instructions do not push the return address, which is needed to know which page the jump was in.

It is possible to make a three byte jump that jumps on carry using CXQ [FOOBAR]. These are not really useful since the carry must always be set or the program will execute the third byte of the jump instruction after skipping the first two.

Likewise, never set the carry before a NCGOREL or NCXQREL. The address that is pushed on the return stack for a three byte jump is the address to the THIRD word, so if the jump crosses a quad boundary, it will jump into the quad that contains the third word.

FAT definition

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
000,100	DEFP4K	DEFP4K	DEFP4K	ADDRESS	Obsolete
000,000	DEFR4K	DEFR4K	DEFR4K	ADDRESS	define MCODE function in same 4K ROM
000,000	DEFR8K	DEFR8K	DEFR8K	ADDRESS	define MCODE function in next 8K ROM
200,000	U4KDEF	U4KDEF	U4KDEF	ADDRESS	define USER CODE function in same 4K ROM
200,000	U8KDEF	U8KDEF	U8KDEF	ADDRESS	define USER CODE function in next 8K ROM

Return Stack And Returns

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
1E0	GOTOC	GTOC	GOTOADR		jump to the address in C[6:3]
170	STK=C	STK=C	PUSHADR		push C[6:3] onto the return stack
1B0	C=STK	C=STK	POPADR		pop the return stack into C[6:3]; put 0 in last location of stack
020	SPOPND	CLRRTN	XQ>GO		pop first address off return stack
360	RTNC	CRTN	?CRTN		return if carry set
3A0	RTNNC	NCRTN	?NCRTN		return if carry not set
3E0	RTN	RTN	RTN		unconditional return

Peripheral instructions

Peripheral addresses for PERSLCT	
00	No peripheral enabled
FB	Timer
FC	Card Reader
FD	Display
FE	Wand
10	Special display for halfnut versions

Peripheral Accessing

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
3F0	PFAD=C	PERSLCT	PRPHSLCT		select the peripheral addressed in C[1:0]
024	SELPF	PERTCT	SELPF	0 to F (hex)	allow peripheral to take control

Peripheral Flags

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
02C	FLG=1?	?PF	?FI=	0 to 13 (dec)	set carry if peripheral flag set
02C	?F3=1	?PF 3	?FI= 3		set carry if peripheral flag 3 set
06C	?F4=1	?PF 4	?FI= 4		set carry if peripheral flag 4 set
0AC	?F5=1	?EDAV	?FI= 5		set carry if peripheral flag 5 set
0EC	?F10=1	?ORAV	?FI= 10		set carry if HP-IL output register available
12C	?F8=1	?FRAV	?FI= 8		set carry if HP-IL frame available
16C	?F6=1	?IFCR	?FI= 6		set carry if HP-IL interface clear received
1AC	?F11=1	?TFAIL	?FI= 11		set carry if timer clock access failure
22C	?F2=1	?WNDB	?FI= 2		set carry if wand has data in wand buffer
26C	?F9=1	?FRNS	?FI= 9		set carry if HP-IL frame not received as sent
2AC	?F7=1	?SRQR	?FI= 7		set carry if service request received
2EC	?F13=1	?SERV	?FI= 13		set carry if service request
32C	?F1=1	?CRDR	?FI= 1		set carry if card reader flag set
36C	?F12=1	?ALM	?FI= 12		set carry if alarm due
3AC	?F0=1	?PBSY	?FI= 0		set carry if peripheral flag 0 set

Display instructions

Zencode display instructions:			
WR/RD	*A/B/C*	*1/4/6/12*	*R/L*
Write/Read	to/from display registers A/B/C	1/4/6/12 characters to/from	Right/Left of display

Writing instructions cause the new characters to be pushed on the specified side which pushes the characters on the other end off into oblivion. Reading instructions cause the characters to be taken off the specified side and pushed on the other side.

Display reading

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
038	FLLDA	RDA12L	FLLDA		
078	FLLDB	RDB12L	FLLDB		
0B8	FLLDC	RDC12L	FLLDC		
0F8	FLLDAB	RDAB6L	FLLDAB		
138	FLLABC	RDABC4L	FLLABC		
178	READEN	READAN	READEN		copy annunciators into C[2:0]
1B8	FLSDC	RDC1L	FLSDC		
1F8	FRSDA	RDA1R	FRSDA		
238	FRSDB	RDB1R	FRSDB		
278	FRSDC	RDC1R	FRSDC		
2B8	FLSDA	RDA1L	FLSDA		
2F8	FLSDB	RDB1L	FLSDB		
338	FRSDAB	RDAB1R	FRSDAB		
378	FLSDAB	RDAB1L	FLSDAB		
3B8	RABCR	RDABC1R	RABCR		
3F8	RABCL	RDABC1L	RABCL		

Display writing

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
028	SRLDA	WRA12L	SRLDA		
068	SRLDB	WRB12L	SRLDB		
0A8	SRLDC	WRC12L	SRLDC		
0E8	SRLDAB	WRAB6L	SRLDAB		
128	SRLABC	WRABC4L	SRLABC		
168	SLLDAB	WRAB6R	SLLDAB		
1A8	SLLABC	WRABC4R	SLLABC		
1E8	SRSDA	WRA1L	SRSDA		
228	SRSDB	WRB1L	SRSDB		
268	SRSDC	WRC1L	SRSDC		
2A8	SLSDA	WRA1R	SLSDA		
2E8	SLSDB	WRB1R	SLSDB		
328	SRSDAB	WRAB1L	SRSDAB		
368	SLSDAB	WRAB1R	SLSDAB		
3A8	SRSABC	WRABC1L	SRSABC		
3E8	SLSABC	WRABC1R	SLSABC		
2F0	WRTEN	WRITAN	WRTEN		copy bits from C[2:0] into annunciators

Time module writing

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
028	WRTIME	WTIME	WRTIME		
068	WDTIME	WTIME-	WDTIME		
0A8	WRALM	WALM	WRALM		
0E8	WRSTS	WSTS	WRSTS		
128	WRSCR	WSCR	WRSCR		
168	WSINT	WINTST	WSINT		
1E8	STPINT	STPINT	STPINT		
228	DSWKUP	WKUPOFF	DSWKUP		
268	ENWKUP	WKUPON	ENWKUP		
2A8	DSALM	ALMOFF	DSALM		
2E8	ENALM	ALMON	ENALM		
328	STOPC	STOPC	STOPC		
368	STARTC	STARTC	STARTC		
3A8	PT=B	TIMER=A	PT=B		
3E8	PT=A	TIMER=B	PT=A		

Time module reading

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
038	RDTIME	RTIME	RDTIME		
078	RCTIME	RTIMEST	RCTIME		
0B8	RDALM	RALM	RDALM		
0F8	RDSTS	RSTS	RDSTS		
138	RDSCR	RSCR	RDSCR		
178	RDINT	RINT	RDINT		

Card reader

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
028	ENWRIT	ENDWRIT	ENWRIT		
068	STWRIT	STWRIT	STWRIT		
0A8	ENREAD	ENDREAD	ENREAD		
0E8	STREAD	STREAD	STREAD		
168	CRDWPF	CRDWPF	CRDWPF		
1E8	CRDOHF	CRDOHF	CRDOHF		
268	CRDINF	CRDINF	CRDINF		
2E8	TSTBUF	TSTBUF	TSTBUF		
328	TRPCRD	SETCTF	TRPCRD		
368	TCLCRD	TCLCTF	TCLCRD		
3E8	CRDFLG	CRDEXF	CRDFLG		

Printer

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
003	BUSY?	BUSY?	BUSY?		set carry if printer busy
083	ERROR?	ERROR?	ERROR?		set carry if printer error
043	POWON?	POWON?	POWON?		set carry if printer is on
007	PRINT	PRINT	PRINT		add C[1:0] to print buffer
03A	STATUS	STATUS	STATUS		copy printer status to C[13:10]
005	RTNCPU	RTNCPU	RTNCPU		return from PERTCT (only necessary for STATUS)

Intelligent peripheral

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
200	HPIL=C	HPIL=C	HPIL=C	0 to 7	copy C[1:0] to HP-IL register

Variations

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
062	BAEX	B<>A	B<>A	TEF	
0A2	CAEX	C<>A	C<>A	TEF	
0E2	CBEX	C<>B	C<>B	TEF	
202	C=C+A	C=C+A	C=A+C	TEF	
1D8	MCEX	M<>C	M<>C		
0F0	NCEX	N<>C	N<>C		

Variations And Duplicates

Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
2E2	C#0?	?C#0	?C#0	TEF	
2C2	B#0?	?B#0	?B#0	TEF	
302	A<C?	?A<C	?A<C	TEF	
322	A<B?	?A<B	?A<B	TEF	
342	A#0?	?A#0	?A#0	TEF	
362	A#C?	?A#C	?A#C	TEF	
370	C=C!A	C=CORA	C=CORA		
3B0	C=C.A	C=CANDA	C=CANDA		
3B8	FRSABC	RDABC1R	FRSABC		
0EC	ORAV?	?ORAV	?FI= 10		
12C	FRAV?	?FRAV	FRAV?		
16C	IFCR?	?IFCR	?FI= 6		
26C	FRNS?	?FRNS	?FI= 9		
2AC	SRQR?	?SRQR	SRQR?		
36C	ALARM?	?ALM	?FI= 12		
160	LLD?	?BAT	?LOWBAT		
120	P=Q?	?P=Q	?P=Q		
014	PT=?	?PT=	?R=	0 to 13 (dec)	
001,000	GOSUB	NCXQ	?NCXQ	ADDRESS	
001,002	GOLONG	NCGO	?NCGO	ADDRESS	
003	GOTO	JNC	JNC	-64 to +63 (dec)	
024	HPL=CH	PERTCT	SELPF	0 to F (hex)	
384	S0= 0	CF 0	CLRF 0		
304	S1= 0	CF 1	CLRF 1		
204	S2= 0	CF 2	CLRF 2		
004	S3= 0	CF 3	CLRF 3		
044	S4= 0	CF 4	CLRF 4		
084	S5= 0	CF 5	CLRF 5		
144	S6= 0	CF 6	CLRF 6		
284	S7= 0	CF 7	CLRF 7		
104	S8= 0	CF 8	CLRF 8		
244	S9= 0	CF 9	CLRF 9		
0C4	S10= 0	CF 10	CLRF 10		
184	S11= 0	CF 11	CLRF 11		
344	S12= 0	CF 12	CLRF 12		
2C4	S13= 0	CF 13	CLRF 13		
388	S0= 1	SF 0	SETF 0		
308	S1= 1	SF 1	SETF 1		

Variations And Duplicates (Cont.)					
Code	*HP*	*Zencode*	*Jacobs/DeArras*	*Operand*	*Description*
008	S3= 1	SF 3	SETF 3		
048	S4= 1	SF 4	SETF 4		
088	S5= 1	SF 5	SETF 5		
148	S6= 1	SF 6	SETF 6		
288	S7= 1	SF 7	SETF 7		
108	S8= 1	SF 8	SETF 8		
248	S9= 1	SF 9	SETF 9		
0C8	S10= 1	SF 10	SETF 10		
188	S11= 1	SF 11	SETF 11		
348	S12= 1	SF 12	SETF 12		
2C8	S13= 1	SF 13	SETF 13		
38C	?S0=1	?FS 0	?FSET 0		
30C	?S1=1	?FS 1	?FSET 1		
20C	?S2=1	?FS 2	?FSET 2		
00C	?S3=1	?FS 3	?FSET 3		
04C	?S4=1	?FS 4	?FSET 4		
08C	?S5=1	?FS 5	?FSET 5		
14C	?S6=1	?FS 6	?FSET 6		
28C	?S7=1	?FS 7	?FSET 7		
10C	?S8=1	?FS 8	?FSET 8		
24C	?S9=1	?FS 9	?FSET 9		
0CC	?S10=1	?FS 10	?FSET 10		
18C	?S11=1	?FS 11	?FSET 11		
34C	?S12=1	?FS 12	?FSET 12		
2CC	?S13=1	?FS 13	?FSET 13		

References

- HP-41 MCODE for Beginners, Ken Emory, Synthetix, 1985
- The ZENROM Programmer's Manual, Zengrange Ltd., 1984
- The HP-41 VASM listings, Hewlett Packard, 1985
- Software Development System II Manual, Hewlett Packard, 1986

APPENDIX I – *Step by Step Example*

The following is a step by step example based of the first few functions from the SKWID 1a ROM from Ken Emery's book 'MCODE for Beginners'. Following these steps we will

1. Create a .txt file which is the code as one would type it in directly from the book. We will use the powerful features of A41 to make use of labels, creation of the FAT, etc
2. We will copy the .txt file into a .src file and assemble it with A41. In my experience, it has been useful to keep an 'unadulterated' file with no extraneous information filled in from the assembler, at least until the assembly process is completed with 0 errors.
3. We will then link the file to a particular page (page 8)
4. Then we will load the so created Rom image into the emulator M41 and follow some of the code step by step to see how it changes the registers.
5. Following our analysis we will add a second .src file – the sREG function from the ZENROM manual – to this ROM image
6. Lastly we will translate the sREG.src file from ZENROM code into JDA code

Create raw SKWID.txt file

First we will create a raw MCODE file which will look as close as possible to the listing in Ken Emery's book on pages 40ff, 42ff,50ff and 54ff.

Important Note: Ken Emery lists jump distances in hex distances (e.g. 80DF JNC -0D) while A41 expect them in decimal format!

A completed file should look something like the listing below. Note that the '.ORG 8000' directive hard-maps the code to page 8, so that the .src file that A41 produces closely resembles the listings published in Ken's book. However this is not practical if you want to link more than one .src file together (see 'Linking two files together')

```
;***************** Start of SKWID sample file ****************
.TITLE  "SKWID"
.JDA
.ORG    8000
;**********************************
;*  FAT for SKWID 1A ROM          *
;**********************************
       XROM    1       ;XROM number
       FCNS    6       ;Header + 1 function
       DEFR4K  [Header] ;first executable of header
       DEFR4K  [Y<>Z]  ;first executable of header
       DEFR4K  [GE]    ;first executable of header
       DEFR4K  [COUNT] ;first executable of header
       DEFR4K  [MA]    ;first executable of header
       DEFR4K  [AM]    ;first executable of header

       NOP             ;FAT termination
       NOP             ;FAT termination
;***********************************
.FILLTO 0081


;******** Start of Code ************
;**** Header
.NAME   "SKWID 1A"
[Header] RTN

;**** Y<>Z Function
.NAME   "Y<>Z"
[Y<>Z]  READ    2(Y)    ;Load Y Reg
```

```
        A=C     ALL     ;Store in A
        READ    1(Z)    ;Load Z
        WRIT    2(Y)    ;Copy to Y Reg
        A<>C    ALL     ;Retrieve Y Reg
        WRIT    1(Z)    ;Store in Z
RTN

;**** GE Function
.NAME   "GE"
[GE]    READ 13(c)
        C=0     M
        R=      3
        LD@R    3
        CLRF    10
        SETF    13
        WRIT    12(b)
 RTN

;**** COUNT Function
.NAME   "COUNT"
[COUNT] SETDEC
        C=0     ALL
        C=C+1   M
        ?KEY
        JNC     -2
        LDIS&X  009
        R=      12
        A=C     M
        ?A#0    @R
        JC      +4
        C=C-1   S&X
        LSHFA   M
        JNC     -4
        CLRKEY
        ?KEY
        JC      -2
        A<>C    M
        WRIT    3(X)
RTN

;**** AM & MA Function
.NAME   "MA"
[MA]    SETF    9
        JNC     +4
.NAME   "AM"
[AM]    CLRF    9
        READ    13(c)
        RCR     3
        A=C     S&X
        LDIS&X  1FD
        ?A<C    S&X
        JC      +4
        C=0     ALL
        WRIT    3(X)
RTN
        R=      0
        LDIS&X  005
        ?FSET   9
        JNC     +2
        A<>C    S&X
        RAMSLCT
        C=C+1   S&X
        C<>B    S&X
        READDATA
        A<>C    ALL
        WRITDATA
        A=A+1   S&X
        R=R+1
        C<>B    S&X
        ?R=     4
        ?CRTN
        JNC     -13
```

Assembling the SKWID.txt example

Next we will assemble this file with A41 and instruct the assembler to create a .src file which includes all sorts of helpful information.

1. Open a DOS window
2. Navigate to the directory with the SDK41 and your SKWID.TXT file
3. Copy the .txt file into a SRC file
    a. *copy skwid.txt skwid.src*
    b. You might have to confirm the overwriting with ‘y’
4. Assemble the source file with the option of creating a .bak file, append a table with all symbols and insert useful comments into the .src file
    a. *a41 /r /o /l skwid*
5. If there are errors, open skwid.txt in a text editor of your choice (e.g. notepad) and skwid.src in a second instance (or second window) of the text editor
    a. The .src file will have lots of information filled in from A41. Don’t be discouraged if this looks overwhelming in th beginning. It will become much clearer over time.
    b. Focus on the lines which have an error. Identify the correct spelling/command etc and correct in your .txt file
    c. Overtime you can correct things directly in the .src file and ignore the .txt file. I found it useful in the beginning to have a less information rich file to work with and concentrate on the code
    d. Some typical sources of error
        i. No tab/space between the command and the operand.
            1. E.g. R=3 instead of R= 3
        ii. Extraneous space in the command.
            1. E.g. READ DATA instead of READDATA
            2. LDI S&X instead of LDIS&X
6. Repeat Steps 3-5 until there are no errors
7. You should now have the following files
    a. SWID.TXT – the raw file which has only the MCODE instructions and symbols, very much like it is listed in Ken Emery’s book
    b. SKWID.SRC – the source file with all the resolutions and comments and formatting from A41 included
    c. SKWID.OBJ – the object file created by A41, ready to be linked.

Assembled SKWID.SRC file

Below is the .src file that A41 has created. As you can see, it contains a host of information around the code. This is very useful but can also be difficult to compare to say a listing in a book or a PPC article. This is why I like to keep a raw .txt file which has only the MCODE instructions.

```
* SKWID.SRC
* Assembled by A41
* Sun Apr 20 21:47:33 2008
;***************** Start of SKWID sample file ***************
                              .TITLE "SKWID"
                              .JDA
                              .ORG   8000
;**********************************
;*  FAT for SKWID 1A ROM          *
;**********************************
 8000 001                     XROM   1       ;XROM number
 8001 006                     FCNS   6       ;Header + 1 function
 8002 00008A                  DEFR4K [Header] 808A  ;first executable of header
 8004 00008F                  DEFR4K  [Y<>Z] 808F    ;first executable of header
 8006 000098                  DEFR4K  [GE] 8098      ;first executable of header
 8008 0000A5                  DEFR4K  [COUNT] 80A5   ;first executable of header
 800A 0000BB                  DEFR4K  [MA] 80BB      ;first executable of header
 800C 0000BF                  DEFR4K  [AM] 80BF      ;first executable of header
 800E 000                     NOP            ;FAT termination
 800F 000                     NOP            ;FAT termination
;***********************************
                              .FILLTO 0081
;******** Start of Code ************
;**** Header
                              .NAME   "SKWID 1A"
*8082 081                     #081           ; "A"
*8083 031                     #031           ; "1"
*8084 020                     #020           ; " "
*8085 004                     #004           ; "D"
*8086 009                     #009           ; "I"
*8087 017                     #017           ; "W"
*8088 00B                     #00B           ; "K"
*8089 013                     #013           ; "S"
 808A 3E0      [Header]       RTN
;**** Y<>Z Function
                              .NAME   "Y<>Z"
*808B 09A                     #09A           ; "Z"
*808C 03E                     #03E           ; ">"
*808D 03C                     #03C           ; "<"
*808E 019                     #019           ; "Y"
 808F 0B8      [Y<>Z]         READ    2(Y)   ;Load Y Reg
 8090 10E                     A=C     ALL    ;Store in A
 8091 078                     READ    1(Z)   ;Load Z
 8092 0A8                     WRIT    2(Y)   ;Copy to Y Reg
 8093 0AE                     A<>C    ALL    ;Retrieve Y Reg
 8094 068                     WRIT    1(Z)   ;Store in Z
 8095 3E0                     RTN
;**** GE Function
                              .NAME   "GE"
*8096 085                     #085           ; "E"
*8097 007                     #007           ; "G"
 8098 378      [GE]           READ    13(c)
 8099 05A                     C=0     M
 809A 01C                     R=      3
 809B 0D0                     LD@R    3
 809C 0C4                     CLRF    10
 809D 2C8                     SETF    13
 809E 328                     WRIT    12(b)
 809F 3E0                     RTN
;**** COUNT Function
                              .NAME   "COUNT"
*80A0 094                     #094           ; "T"
*80A1 00E                     #00E           ; "N"
*80A2 015                     #015           ; "U"
*80A3 00F                     #00F           ; "O"
*80A4 003                     #003           ; "C"
 80A5 2A0      [COUNT]        SETDEC
 80A6 04E                     C=0     ALL
 80A7 23A                     C=C+1   M
 80A8 3CC                     ?KEY
 80A9 3F3                     JNC     -2 80A7
 80AA 130009                  LDIS&X  009
 80AC 35C                     R=      12
 80AD 11A                     A=C     M
 80AE 342                     ?A#0    @R
 80AF 027                     JC      +4 80B3
 80B0 266                     C=C-1   S&X
 80B1 3FA                     LSHFA   M
 80B2 3E3                     JNC     -4 80AE
 80B3 3C8                     CLRKEY
 80B4 3CC                     ?KEY
 80B5 3F7                     JC      -2 80B3
 80B6 0BA                     A<>C    M
 80B7 0E8                     WRIT    3(X)
 80B8 3E0                     RTN
;**** AM & MA Function
                              .NAME   "MA"
*80B9 081                     #081           ; "A"
*80BA 00D                     #00D           ; "M"
 80BB 248      [MA]           SETF    9
 80BC 023                     JNC     +4 80C0
                              .NAME   "AM"
*80BD 08D                     #08D           ; "M"
*80BE 001                     #001           ; "A"
 80BF 244      [AM]           CLRF    9
 80C0 378                     READ    13(c)
 80C1 03C                     RCR     3
 80C2 106                     A=C     S&X
 80C3 1301FD                  LDIS&X  1FD
 80C5 306                     ?A<C    S&X
 80C6 027                     JC      +4 80CA
 80C7 04E                     C=0     ALL
 80C8 0E8                     WRIT    3(X)
 80C9 3E0                     RTN
 80CA 39C                     R=      0
 80CB 130005                  LDIS&X  005
 80CD 24C                     ?FSET   9
 80CE 013                     JNC     +2 80D0
 80CF 0A6                     A<>C    S&X
 80D0 270                     RAMSLCT
 80D1 226                     C=C+1   S&X
 80D2 0E6                     C<>B    S&X
 80D3 038                     READDATA
 80D4 0AE                     A<>C    ALL
 80D5 2F0                     WRITDATA
 80D6 166                     A=A+1   S&X
 80D7 3DC                     R=R+1
 80D8 0E6                     C<>B    S&X
 80D9 054                     ?R=     4
 80DA 360                     ?CRTN
 80DB 39B                     JNC     -13 80CE
*
* GLOBAL SYMBOLS
* SYMBOL        VALUE   TYPE    REFERENCES
* [AM]         80BF    ABS     800C
* [COUNT]      80A5    ABS     8008
* [GE]         8098    ABS     8006
* [Header]     808A    ABS     8002
* [MA]         80BB    ABS     800A
* [Y<>Z]       808F    ABS     8004
*
* LOCAL SYMBOLS
* SYMBOL        VALUE   TYPE    REFERENCES
*
* EXTERNAL REFERENCES
* SYMBOL        REFERENCED AT
*
* MAINFRAME REFERENCES
* SYMBOL        VALUE   REFERENCES
*
* A41:  0 WARNINGS(S)
* A41:  0 ERROR(S)
* END
```

Create SKWID.LNK file

Next we have to create the SKWID.lnk file which will tell the linker which .obj files to link together, what name to give the final rom and at what page to locate it. This file is again a simple text file so can be best created with a text-editor like notepad. For a description of the .lnk file check here
Note: There are no remarks after the link commands allow, but you can place separate remark lines in the link file. Your link file should look something like this:

```
;***************** Link file for SKWID1A example ************
;first, lets definte the page, bank and rom-name we want
$PAGE 8 1 SKWID1A

;second, lets tell the linker the name of the .obj file A41 has created
SKWID

;lastly, lets calculate and store the checksum in the rom-file as well
$CH
```

Make sure you store it in the same directory as your other files and SDK41

Link SKWID to create the SKWID1A.ROM file

Now we have everything in place to create our SKWID1A rom image with L41.

1. Click into your DOS window again. If you have closed it since last time, reopen a new DOS window
2. Navigate to the SDK41 directory (you might already be there if you have not closed your DOS window)
3. Call the linker with options that will reassemble the .src file is newer than the .obj file (see the description of L41) and creates a label file
    a. *L41 /aro /ll skwid*

Create SKWID.LOD file

Next we have to create an appropriate .lod file so that we can load the operating system and the SKWID1A rom into the emulator and then try out the new functions. This file is again a simple text file so can be best created with a text-editor like notepad. For a description on the .lod file check here.
Your file should look something like this:

```
;***************** Load file for SKWID1A example ************
;first, lets load the OS
$PAGE 0 1 XNUT0
$PAGE 1 1 XNUT1
$PAGE 2 1 XNUT2
$PAGE 3 1 CXFUNS0
$PAGE 5 1 TIMER
$PAGE 5 2 CXFUNS1

;second, lets load the SKWID rom as well
$PAGE 8 1 SKWID1A

;third lets include all labels so that we can move around easily
$LABELS skwid
```

Start emulator M41 and single step through function MA

Now we have everything in place to start the emulator and step through a function

1. Locate your DOS window again and make sure you are in the SDK41 directory
2. Launch the emulator in Jacobs/De Arras format
    a. Type at the prompt: *m41 /j skwid*
3. Bring up the help which shows all available commands
    a. Type: *m*
4. Make sure that we use the Jacobs/De Arras set
    a. Type: *j*
5. Select the format which shows both the address as well as the hex code
    a. Type: *s*
6. Goto to first address of GE function
    a. Type: *G 8098*
7. Select the format which shows labels again
    a. Type: *s*
    b. Confirm that we are at the [GE] label
8. Move to the start of the GE function
    a. Type: *G [MA]*
9. Set a breakpoint there
    a. Type: *B S [MA]* (or *B S 80BB*)
10. Start user-code and execute the command “MA”
    a. Type: *u* ;This starts the hp41 user code simulation
    b. Type: *<Shift F1> <F4> <Shift MA> <F4>* for XEQ Alpha MA Alpha
11. Single Step through the MCODE
    a. Press the *<Space>* bar for each step
12. When you are done with your analysis, you can leave the emulator
    a. Type: *q y*

Combine two separate .src files into one .rom image

Lets say we want to add the sREG function from the ZENROM manual to the SKWID1A rom we just created. The following sequence of steps will accomplish just that

1. Enter the sREG function from the ZENROM manual into a file *sREG.txt*
2. Copy the *sReg.txt* file to a *sReg.src* file and assemble that file with no errors or warnings. You don’t have change it to JDA format to be linked together with SKWID1a
3. Change the *SKWID.src* (or *SKWID.txt* file but remember to copy it over to the .src file) to a SKWID1b.src/.txt
    a. Exclude the *.ORG 8000* directive. This directive hard-maps the SKWID.OBJ file to the 8000 address space so we can not automatically link two .obj files together. Instead we will use the automatic sequential linking of L41 to link the .obj files together
    b. Add a DEFR4k directive to add the sREG function to the FAT
        i. If you have many separate functions to link together, it is advisable to have a separate .src file just for the FAT. See the note at the end of this section
4. Assemble the SKWID1b.src file into a SKWID1b.obj file
5. Assemble the sREG.src file into a sREG.obj file
6. Create a SKWID1B.lnk file to combine the two obj files
7. Link SKWID1B.obj and SREG.obj together into the SKWID1B.rom file
8. Create a SKWID1b.lod file with the new rom image name
9. Start the emulator M41 to see the new function sREG in the new SKWID1B rom image

Listing of sREG.txt (note that this is in ZENROM format and contains a global label)

```
;********************* sREG function from ZENROM manual p102ff
.ZENCODE
.NAME "sREG"
[SREG]  C=REG   13
        RCR     11
        A=C     X
        CF      0
        RAMSLCT
        RDATA
        A=C     M
        C=-C-1  M
        WDATA
        RDATA
        C=-C-1  M
        WDATA
        ?A#C    M
        *
        CGO     02E0
        ?FS     0
        JC      +8
        LDI     04
        A<>C    X
        M=C
        SF      0
        JNC     -19
        A<>C    X
        N=C
        RAMSLCT
        RDATA
        A=C     ALL
        C=M
        RAMSLCT
        C=C-1   X
        M=C
        A<>C    ALL
        WDATA
        C=N
        C=C-1   X
        JNC     -12
RTN
```

Copy the file to a .src file

- At the command prompt type: *copy sREG.txt sREG.src*

Assemble the sREG file

- At the command prompt type: *A41 /r /o /l sREG*
- Repeat the last two steps until A41 finishes with 0 errors and 0 warnings
- This will create the .obj file but also a label file

Listing of sREG.src file created by A41. Note the ZENROM notation was preserved.

```
* SREG.SRC
* Assembled by A41
* Wed Apr 23 21:48:36 2008
;*********************** sREG function from ZENROM manual p102ff
                                .ZENCODE
                                .NAME   "sREG"
*0000 087                       #087            ; "G"
*0001 005                       #005            ; "E"
*0002 012                       #012            ; "R"
*0003 04E                       #04E            ; "s"
 0004 378       [SREG]          C=REG   13
 0005 1BC                       RCR     11
 0006 106                       A=C     X
 0007 384                       CF      0
 0008 270                       RAMSLCT
 0009 038                       RDATA
 000A 11A                       A=C     M
 000B 2BA                       C=-C-1  M
 000C 2F0                       WDATA
 000D 038                       RDATA
 000E 2BA                       C=-C-1  M
 000F 2F0                       WDATA
 0010 37A                       ?A#C    M
 0011 38100B                    CGO     02E0
 0013 38C                       ?FS     0
 0014 047                       JC      +8 001C
 0015 130004                    LDI     04
 0017 0A6                       A<>C    X
 0018 158                       M=C
 0019 388                       SF      0
 001A 36B                       JNC     -19 0007
 001B 0A6                       A<>C    X
 001C 070                       N=C
 001D 270                       RAMSLCT
 001E 038                       RDATA
 001F 10E                       A=C     ALL
 0020 198                       C=M
 0021 270                       RAMSLCT
 0022 266                       C=C-1   X
 0023 158                       M=C
 0024 0AE                       A<>C    ALL
 0025 2F0                       WDATA
 0026 0B0                       C=N
 0027 266                       C=C-1   X
 0028 3A3                       JNC     -12 001C
 0029 3E0                       RTN
*
* GLOBAL SYMBOLS
* SYMBOL        VALUE   TYPE    REFERENCES
* [SREG]        0004    REL
*
* LOCAL SYMBOLS
* SYMBOL        VALUE   TYPE    REFERENCES
*
* EXTERNAL REFERENCES
* SYMBOL        REFERENCED AT
*
* MAINFRAME REFERENCES
* SYMBOL        VALUE   REFERENCES
*
* A41:  0 WARNINGS(S)
* A41:  0 ERROR(S)
* END
```

Change the SKWID.txt file to SKWID1B.txt

- Remove the .ORG directive
- Add an additional entry for the sREG function into the FAT
- Copy to *SKWID1B.txt* to *SKWID1b.src*
- Assemble with *a41 /r /o /l skwid1b*

The newly assembled *SKWID1b.src* should start like the below. Note that the address space is now not anymore fixed to 8000h but starts at 0000h. The linker and the $PAGE command will locate the final rom at page 8.

Listing of SKWID1B.src

```
* SKWID1B.SRC
* Assembled by A41
* Thu Apr 24 15:50:03 2008
;***************** Start of SKWID sample file ****************
                                .TITLE  "SKWID"
                                .JDA
;**********************************
;*  FAT for SKWID 1A ROM          *
;**********************************
 0000 001                       XROM    1       ;XROM number
 0001 006                       FCNS    6       ;Header + 1 function
 0002 00008A                    DEFR4K  [Header] 008A  ;first executable of header
 0004 00008F                    DEFR4K  [Y<>Z] 008F    ;first executable of function
 0006 000098                    DEFR4K  [GE] 0098      ;first executable of function
 0008 0000A5                  DEFR4K  [COUNT] 00A5   ;first executable of function
 000A 0000BB                  DEFR4K  [MA] 00BB      ;first executable of function
 000C 0000BF                  DEFR4K  [AM] 00BF      ;first executable of function
 000E 000000                  DEFR4K  [sREG] ;firsy executable of function
 0010 000                     NOP            ;FAT termination
 0011 000                     NOP            ;FAT termination
;************************************
                              .FILLTO 0081
;******** Start of Code ************
;**** Header
                              .NAME   "SKWID 1A"
*0082 081                     #081           ; "A"
*0083 031                     #031           ; "1"
*0084 020                     #020           ; " "
*0085 004                     #004           ; "D"
*0086 009                     #009           ; "I"
*0087 017                     #017           ; "W"
*0088 00B                     #00B           ; "K"
*0089 013                     #013           ; "S"
 008A 3E0      [Header]       RTN
;**** Y<>Z Function
                              .NAME   "Y<>Z"
*008B 09A                     #09A           ; "Z"
*008C 03E                     #03E           ; ">"
*008D 03C                     #03C           ; "<"
*008E 019                     #019           ; "Y"
 008F 0B8      [Y<>Z]         READ    2(Y)   ;Load Y Reg
 0090 10E                     A=C     ALL    ;Store in A
…
```

Now we can link the two files together into our new *Skwid1b rom*

- Change the *skwid1a.lnk* file to include the sReg file and save as *skwid1b.lnk*
- Link with *l41 /arol /l skwid1b*
- Note that the /l option makes sure that the [sReg] global label location is resolved

Listing of SKWID1b.LNK

```
;***************** Link file for SKWID1b example ************
;first, lets definte the page, bank and rom-name we want
$PAGE 8 1 SKWID1b

;second, lets tell the linker the name of the .obj file A41 has created
SKWID1b
sREG

;lastly, lets calculate and store the checksum in the rom-file as well
;$CH
```

Lastly, we can load the new rom into the emulator and confirm that we have all the functions including sReg there and also can use the [sREG] global label to position ourselves in the code.

- Change the *skwid1a.lod* file to use the skwid1b.rom file and save as *skwid1b.lod*
- Start the m41 emulator with *m41 skwid1b /j*
- Move to the [sREG] label with *G [sREG]*

Listing of SKWID1B.lod file should look like this

```
;***************** Load file for SKWID1B example ************
;first, lets load the OS
$PAGE 0 1 XNUT0
$PAGE 1 1 XNUT1
$PAGE 2 1 XNUT2
$PAGE 3 1 CXFUNS0
$PAGE 5 1 TIMER
$PAGE 5 2 CXFUNS1

;second, lets load the new SKWID rom
$PAGE 8 1 SKWID1b


;third lets include all labels
$LABELs skwid1b
```

Create separate FAT.src and function files for flexible rom building

The next step from here to facilitate the flexible building of roms is to separate the FAT from any particular function and assemble each function one would like to include separately. This allows for a very flexible way to build any custom rom

- Build a library of .src files with functions you might want to use
- Have a separate FAT.src file which you edit to only include the functions you would like and the header of the rom
- Create a .lnk file that tells the linker which .obj modules to combine together

This very flexible set-up is exactly what was done for the PCCOM rom example that comes with the SDK41 distribution files.

Hopefully this step-by-step example has made you comfortable and familiar with the enormous power and flexibility that SDK41 provides.

Have fun!