Soft FORTH Reference Manual

Release Version: 2.4
Date: February 3, 1986

Preliminary

CONTENTS

CHAPTER 1
Introduction

1.1 *The Purpose of Soft FORTH*

The purpose of Soft FORTH is to allow the creation of soft-configured ROM based applications written wholly or partially in FORTH. The programs and procedures provide a great deal of flexibility in the creation of FORTH-based custom ROM applications with a minimum of hassle. Soft FORTH does have some limitations. Additional steps may be necessary to ensure that applications will correctly compile and run.

The use of Soft FORTH eliminates the need for the FORTH/Assembler ROM to be present for the application to run. Any number of these Soft FORTH-based applications are able to co-exist in the HP-71.

Each FORTH-based application must contain a version of the Soft FORTH kernel. This kernel is provided on a 16K byte chip which contains the Soft FORTH system, Soft FORTH initialization, and poll handling. This then limits the size of FORTH-based applications to 48K in a 64K custom ROM.

1.2 *Current Soft FORTH version*

The current version of Soft FORTH as of %G% is version E (VER$=SFTH:E).

CHAPTER 2
The Development Process

2.1 *Developing the Application*

The desired FORTH-based application is developed using the (hard-addressed) FORTH/Assembler ROM development system, as well as any LEX, BASIC, or BIN files that are to be included in the final application ROM.

2.1.1 New FORTH Words

Some additional FORTH words are necessary to correctly compile a Soft FORTH application. For example, CFA's (Code Field Addresses) of words compiled into a colon definition via "," (COMMA) must instead use a new word ",REL" (COMMA-RELATIVE.) This ensures that the offset to the desired word is compiled. The standard "," (COMMA) is still used to compile absolute values (i.e. a literal like 5) into a definition.

Since the application is developed originally in the FORTH/Assembler ROM environment, versions of these new words are required. The definitions to be used with the FORTH/Assembler ROM are listed in the last chapter of this document. Using these words throughout the development of the application simplifies the transition to the Soft FORTH environment and avoids the error-prone process of changing the application code to move to Soft FORTH.

2.1.2 Soft FORTH Application Development

The application is written and tested using the FORTH/Assembler ROM and the Soft FORTH word definitions listed in the last chapter of this document.

The Soft FORTH Custom ROM process requires a 64K plug-in RAM module for the Custom ROM image, the Soft FORTH software, and the FORTH/Assembler ROM. 48K byte FORTH applications need an additional 16K bytes of RAM which can be separated in an IRAM.

The Soft FORTH software includes the following files:

SFTFORTH48: Soft FORTH kernel, debug version, for 32K byte or smaller FORTH applications.

SFTFORTH64: Soft FORTH kernel, debug version, for 48K byte FORTH

applications.

ROMFORTH: Soft FORTH kernel, same as final ROM kernel.

FORTHEAD16, FORTHEAD32: This file contains all of the word headers for the words contained in the Soft FORTH kernel. FORTHEAD16 is loaded into the last 16K of the 64K RAM for applications of less than 32K bytes, or in the 16K RAM module for 48K applications. Headers for new words defined in the FORTH-based application are also compiled into FORTHEAD. If more than 16K bytes of headers are needed, FORTHEAD32 can be loaded into a 32K byte RAM instead of FORTHEAD16. These instructions assume use of FORTHEAD16 for simplicity. The final Soft FORTH image will be the same with either FORTHEAD file.

2.2 *Application Description File (ADF)*

All Soft FORTH-based applications are invoked through BASIC keywords. The keywords invoke FORTH words which do the actual processing.

The ADF is a TEXT file which contains a complete description of the application, including the information necessary to create the LEX file containing the BASIC keywords. The ADF can be created using the Text Editor in the FORTH/Assembler ROM.

The Soft FORTH procedure refers to specific lines of the ADF. The ADF is described in detail later in this document.

2.3 *Using the Soft FORTH system*

To use the Soft FORTH system, the 64K RAM must be in the HP-71 and set up as a single Independent RAM (IRAM) using the HP-71 FREE PORT command.

NOTE: For HP-71B with VER$=1BBBB, there is a bug may affect Soft FORTH users
It shows up only if there are IRAM(s) in the HP-71B which have a chip size of 32K bytes, the first IRAM containing 32K byte chips has two or more chips, and the system does NOT include a FORTH/Assembler Module.
Both the 64K and 96K RAM modules from Hand Held Products have a chip size of 32K bytes.
For Soft FORTH, there must be one IRAM of at least 64K bytes, so the FORTH/Assembler module must be present when using the 64K RAM module from HHP as an IRAM.
The 96K RAM module is usable without the FORTH/Assembler module if the PORT(5) is 32K bytes and PORT(5.01) is 64K bytes.

For applications which do not require the use of an additional 16K IRAM (less than 32K of FORTH code), the Soft FORTH kernel file (SFTFORTH48) should be copied into the 64K IRAM first. The headers file (FORTHEAD16) should be copied into the 64K IRAM after the kernel file.

If the application requires the use of the additional 16K IRAM (for a Custom ROM of 64K), the Soft FORTH kernel (SFTFORTH64) should be copied into the 64K IRAM. The headers file (FORTHEAD16) should be copied into a separate 16K IRAM.

The ADF should be created at this point (if it has not been created already).

When the IRAM(s) and the ADF are set up, the Soft FORTH system is ready for use. To enter the Soft FORTH environment:

```
>SFTFORTH
```

Soft FORTH signs on with the message:

```
HP-71 Soft FORTH A
```

Once in the Soft FORTH environment, type:

```
" <ADF name>" MAKEROM
```

MAKEROM first uses the ADF to create the LEX file and BASIC keywords that will invoke the FORTH-based application. MAKEROM then compiles the application into the 64K IRAM (the ROM image).

The portion of the application which is written in FORTH must be compiled in one invocation of MAKEROM.

2.4 *Debugging with Soft FORTH*

Two problems likely to be encountered while developing a Soft FORTH-based application are the need to distinguish between RAM and ROM space, and between relative and absolute addresses. Most FORTH systems, including the FORTH system used in the FORTH/Assembler ROM, intermix the code and data structures. For ROM-based Soft FORTH, any data structures which need to be changed must be placed in RAM, separate from the FORTH code. In addition, code addresses are no longer absolute. The soft-configured ROM address can change whenever the HP-71 is turned on. This forces addresses to be relative to the current IP (Instruction Pointer).

The application should be tested carefully after creating the IRAM ROM image to verify that it performs correctly for all paths through the code. Particular care should be taken to test code that uses data structures other than the FORTH stacks.

2.4.1 The Debug Version

To ease the transition between the FORTH/Assembler ROM and Soft FORTH, a debug version of the Soft FORTH kernel is included. All FORTH-based applications are compiled using this version of the Soft FORTH kernel. This version includes a check in each word which writes to memory. Any attempts to write to the IRAM containing the application code are flagged and information to help track the error is displayed. The debug version of the kernel is slightly slower than the ROM version, but is suitable for most testing.

A version of the kernel which is identical to the actual ROM chip is also included (ROMFORTH). This version can be used for testing of time-critical code. To use the ROM version, use the SFTFORTH to create the LEX file. Copy the application LEX file after executing ENDFORTH. CLAIM the port (CLAIM PORT), then FREE it again to clear it out. Copy ROMFORTH to the port, followed by the application LEX file.

2.4.2 Debug Concerns

An application may modify the contents of the IRAM during compilation (i.e. during the execution of the MAKEROM word). The debug version of the kernel does not flag writes to IRAM during compilation. Checks for writing to IRAM become active when the compilation is completed.

2.5 *Copying the ROM Image*

When the application works as desired in the soft-configured form, and any additional BASIC, BIN, DATA, LEX, or TEXT files have been copied into the 64K IRAM, the ROMCOPY keyword is used to copy the code to an external device. ROMCOPY inserts the proper checksum values while copying the file.

To find the locations of the checksums (required by ROMCOPY), use the CHECKSUMS word in SFTFORTH before executing ENDFORTH (see the chapter on Soft FORTH words for a description of CHECKSUMS).

2.6 *Sample Session*

This sample uses the debug version of the kernel, and illustrates a 48K byte FORTH-based application.

NOTE: If any error occurs during this process after MAKEROM has started, it is necessary to restart the entire process. This includes CLAIMing the RAM module(s) and FREEing them again. This is necessary because MAKEROM modifies the SFTFORTH kernel.

1. Free the RAMs to create the necessary IRAM environment:

   >FREE PORT(5) ! Free the RAM module for FORTHEAD

   >FREE PORT(5.01) ! Free the 64K RAM module

   (Where 5 and 5.01 are the port numbers of the RAM)

2. Purge FORTHSYS, if it exists now, as well as any file with the allocated name specified in the ADF.

3. Copy the FORTH kernel and the keyword creation code into PORT(5.01).

   >COPY SFTFORTH64:TAPE TO :PORT(5.01) ! Copy kernel into 64K IRAM

   SFTFORTH is a 64K byte file. It contains the 16K Soft FORTH kernel, 16K of filler, code for the SFTFORTH keyword, and more filler. The filler ensures that the header file goes at the right address if it is copied into the 64K IRAM, and maintains file system integrity while the FORTH code is being generated. The filler is overwritten by the application's FORTH code. It may take several minutes for SFTFORTH to be copied into IRAM.

4. Copy the headers into the IRAM (if this application was less than 32K bytes, the headers would go to PORT(5.01), and the kernel file SFTFORTH48 would be used).

   >COPY FORTHEAD16:TAPE TO :PORT(5)

5. Enter the Soft FORTH environment from BASIC:

   >SFTFORTH

   This keyword searchs for the base address of FORTHEAD and SFTFORTH (the FORTH kernel) and stores the addresses into FORTHSYS (FORTHSYS is created here by SFTFORTH).

6. Compile the application code:

   " MYROM:TAPE" MAKEROM (MYROM is the name of the ADF)

7. The headers are still available when the compilation is complete. This allows testing of FORTH words directly.

   **NOTE: The kernel version in IRAM at this point is the debug kernel which identifies and prevents stores to the IRAM.

8. Exit the Soft FORTH environment by typing BYE or by turning off the HP-71. This zeroes out the locations in FORTHSYS which point to the header file and to the FORTH kernel.

9. Thoroughly debug the application using the newly created BASIC keywords to invoke the FORTH-based application.

10. If the application attempts to write to the ROM image, SFTFORTH displays the address of the offending write and displays the contents of the return stack at that point.

11. If a problem is discovered, reenter the Soft FORTH environment:

```
>SFTFORTH
```

This resets the FORTHSYS locations which point to the headers and the Soft FORTH kernel. This allows testing of FORTH words directly.

After identifying and fixing problems in the application source code, **restart the entire process from the beginning.**

12. When satisfied that everything is correct, enter the Soft FORTH environment once more to get the checksum addresses:

```
>SFTFORTH          ! Only if currently in BASIC
DECIMAL CHECKSUMS ( Next line is a sample result )
101 32768 65567 0  OK ( 0 )
```

These numbers are the offsets from the beginning of the ROM to the checksums that are needed for the ROMCOPY statement.

The addresses are displayed in the current number base. They are the offset of the one byte checksum fields left by Soft FORTH in the IRAM. In this example, the application is between 32K+1 and 48K bytes long. The first checksum (101) is for the kernel. These values are used later by the ROMCOPY statement.

13. Once the application code is correct and the checksums are known, finish the Soft FORTH conversion:

```
ENDFORTH  ( Remove the headers, shrink the file, disable SFTFORTH )
```

14. Copy any BASIC, BIN, DATA, LEX, or TEXT files that are to be included in the application ROM into the IRAM (using the HP-71 COPY statement). Note that if any file copied into the IRAM crosses a 16K byte boundary, an additional checksum is required. See the ROMCOPY documentation for assistance in selecting a location for this new checksum.

15. Copy the ROMCOPY LEX file from the mass storage device to the HP-71:

```
>COPY ROMCOPY:TAPE
```

and use the ROMCOPY statement to copy the ROM image from the 64K IRAM to the mass storage device (see the ROMCOPY documentation for the syntax of the ROMCOPY statement).

CHAPTER 3
The End Product

The end product is a Custom ROM containing 32, 48 or 64K bytes of code consisting of the 16K byte soft-configured FORTH kernel, the FORTH-based application along with the BASIC keywords which invoke it, and any LEX, BIN, or BASIC files that were copied into the IRAM.

Soft FORTH-based applications do not require that the FORTH/Assembler ROM be present in order to run. Any number of Soft FORTH-based applications may co-exist in the HP-71.

3.1 *Calling BASIC*

If the application makes use of the FORTH to BASIC capability (specifically BASICX) then those statements which invoke BASICX must be non-programmable. This is because the FORTH and BASIC environments are not reentrant; the application cannot be started from BASIC, exercise FORTH, have FORTH exercise the BASIC system through BASICX, return to FORTH and then return to a running BASIC program. This restriction also applies in the FORTH/Assembler ROM: FORTHX (a BASIC keyword) may not invoke BASICX from within FORTH.

3.2 *RAM Files*

A Soft FORTH-based application interacts with 2 FORTH-type RAM files. The first, called FORTHSYS, takes the place of FORTHRAM. It contains the FORTH system variables, the Terminal Input Buffer, the Data and Return stacks, the floating point stack, and the Mass Storage buffers. It resides as the first file in memory just as FORTHRAM does, and it requires approximately 1100 bytes of memory. FORTH-based application ROMs must not make any assumptions about the contents of user variables (such as BASE or SECONDARY) since other FORTH-based application ROMs may have used FORTHSYS and left the variables with different values.

The second file is unique to each application; it contains the variables and data structures (defined by the application) needed to run the application. To avoid a conflict between two different applications, the name of this file is an allocated resource (just as are LEX ID and token numbers). See the section on Allocated Resources in the next chapter for more details.

3.3 *Application FORTH-Type RAM File*

The Soft FORTH kernel contains the code to create and maintain a FORTH-type RAM file for a particular FORTH-based application. However, it is the responsibility of the application to initialize any user variables it requires before it begins to run.

CHAPTER 4
Implementation Details

4.1 *The 64K RAM module*

This module must be configured as independent RAM. This creates a contiguous 64K block of RAM into which the application is compiled. The soft-configured FORTH system (either SFTFORTH or ROMFORTH) is copied from the provided cassette or disc into the first part of this IRAM. A second file (FORTHEAD) contains header information for the system words which are called by an application. For an application of less than 32K bytes, FORTHEAD is copied into the last 16K of this IRAM.

4.2 *ADF Format*

The parameters for the MAKEROM word are contained in a TEXT file (created using the HP-71 Text Editor). If there is a mistake in the file (e.g. an invalid parameter), the Soft FORTH program displays an error message and exits. If an error is generated by Soft FORTH, correct the ADF and start again at the beginning. All numeric values in the ADF are base 10 (decimal). The format of the file is:

Line	Contents	
1	Name of the unique FORTH-type RAM file	*
2	Name of the application file to LOADF	
3	Name of the LEX file which will contain the BASIC keywords which invoke the application	
4	LEX ID for the LEX file specified in line 3 followed by the VER$ response	*
5	Token number for the first BASIC keyword: Soft FORTH automatically increments by one for subsequent keywords	*
6	Description of the first BASIC keyword	*
7	Description of the second BASIC keyword	*
.	.	
.	.	
.	.	
N	Description of the last BASIC keyword	*

* Resources allocated by HP

4.2.1 ADF Details

The first line of the ADF is the name of the unique FORTH-type file which has been allocated by HP for this application.

The second line is the name of the file to LOADF when MAKEROM is invoked. If multiple files are to be LOADed, create a dummy file containing a LOADF for each file, and put the dummy file's name here. This file may reside on a mass storage device.

The third line is the name to be given to the LEX file in the ROM image.

There are no restrictions on the filenames in lines two and three other than that they be legal HP-71 filenames.

The fourth line contains the LEX ID allocated by HP and the VER$ response for the application.

The fifth line is the beginning token number allocated by HP.

4.2.2 BASIC Keyword Compilation

Line six and all following lines of the ADF describe the BASIC keywords. Each line contains the text which invokes the keyword allocated by HP, its characterization nibble in decimal, and the FORTH word associated with the keyword (FORTHword).

The keywords MUST be in alphabetical order, as specified in the HP-71 Software IDS, Volume I, section 6.1.

If the keyword is a function, the kind of function (Ftype) and the number of parameters in decimal (numpar) are included next. The line ends with the appropriate number of parameter descriptions (ptype).

FUNCTIONS:
keyword charactization FORTHword Ftype numpar ptype ptype ...

ALL OTHER KEYWORDS:
keyword charactization FORTHword ptype ptype ...

Both the kind of function (Ftype) and parameter description (ptype) are defined as one of:

F (for floating point)
I (for integer)
S (for string)

If the application makes use of the FORTH to BASIC interface, BASICX, it is the responsibility of the application to be sure that the keyword which invokes this application is non-programmable. This is done by

providing an appropriate characterization nibble (see the HP-71 Software IDS, Volume I, Chapter 6).

The HP-71 operating system allows functions to have a variable number of parameters and to have several possible types for each parameter. The Soft FORTH system does not support this capability.

4.2.3 ADF Example

As an example, here is an ADF for a file with two keywords.

The FORTH-type RAM file name allocated to this application is "ALLOCATD".

The file containing the FORTH code is named "FTHCODE".

The LEX ID allocated to the LEX file is 5C (hex) = 92 (decimal), and the tokens allocated to the LEX file are 7 and 8.

The VER$ response of this application is "EXM:A".

The first keyword is a statement (COMPUTE) with three parameters: a string, an integer, and a floating point number, respectively. It invokes the FORTH word DOCOMPUTE. The second keyword (STRIP$) is a string function with 2 parameters: a string and an integer. The FORTH word for STRIP$ is DOSTRIP.

The name of this LEX file is to be "LEXDEMO".

The ADF looks like this:

```
ALLOCATD
FTHCODE
LEXDEMO
92 EXM:A
7
COMPUTE 13 DOCOMPUTE S I F
STRIP$ 15 DOSTRIP S 2 S I
```

4.3 *Word Headers*

In FORTH, each word has a header.

FORTH Header	Size
Link field	5 nibbles
Name field	2 + Length of word in nibbles
Code address field	5 nibbles

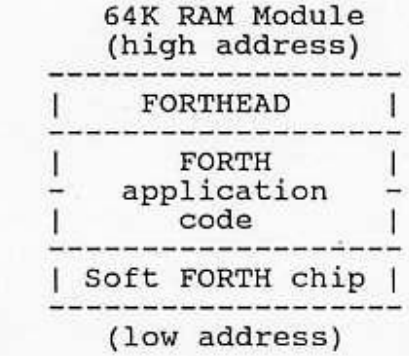
The link field points to the next word in the dictionary. The name field contains the word name, length nibble, smudge bit, and immediate execute bit. The code address field points to executable code for the word in question.

The header is used to compile new words and to find the executable code for words typed in during execute mode. It takes a minimum of 7 bytes of code space for a header, and the header is 11 bytes for a 5 character word.

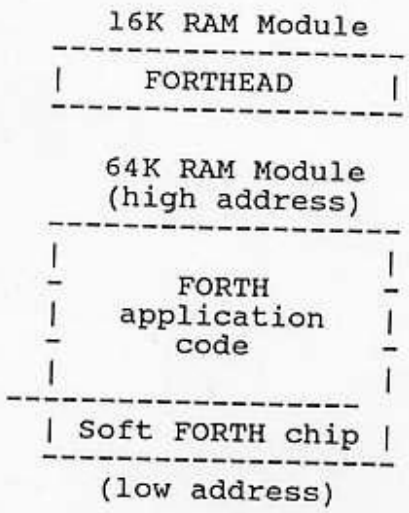
Headers are usually linked in a serial list, with new words added at the head. When searching for a word, all words which were defined after that word are examined (regardless of their length) before the desired word is found. This can significantly slow down the compilation of new definitions.

Soft FORTH separates the headers from the code during the compilation process.

Small Application without separate 16K RAM
(<32K bytes of code)



Large Application with 16K RAM for FORTHEAD
(>32K bytes of code)



Headers are located via a jump table with 32 entries (legal word lengths are 0 to 31). Each entry points to a linked list of words of that particular length. New entries are added to the head of the appropriate list and the jump table is updated. In this fashion user defined words are found before system defined words.

The advantages of the Soft FORTH header scheme are

1) it significantly speeds up compilation of the application, and
2) the headers do not take code space from the application.

The headers remain in the IRAM until the ENDFORTH word is used so that they are available for debugging purposes.

The Soft FORTH code address field is relative to the start of the kernel file.

Sample Soft FORTH Headers

```
                                                        (Field)
          REL(5)   =!PREV
!BASE     CON(2)   #84                  BASE            (Link)
          NIBASC   \BAS\                                (Length&bits)
          CON(2)   \E\+#80                            (Name)
          CON(5)   (=BASE)-(offset)                     (Code offset)
*
          REL(5)   =!BASE
!THEN     CON(2)   #C4                  THEN (IMMEDIATE)(Link)
          NIBASC   \THE\                                (Length&bits)
          CON(2)   \N\+#80                            (Name)
          CON(5)   (=THEN)-(offset)                     (Code offset)
*
```

The Soft FORTH system uses 6647 nibbles of the 16K bytes allocated for headers, including the FORTHEAD file header, the jump table, and the words defined by Soft FORTH (the FORTH/Assembler words plus the unique to Soft FORTH words). This leaves 26121 nibbles for user defined words. If the average word header takes 20 nibbles (4 chars/word), an application can define about 1306 words before running out of header space. If the FORTHEAD32 header file is used, the space available for user defined words is 58889 nibbles, or about 2944 words.

4.4 *Allocated Resources*

LEX ID's, token numbers, keywords, and unique FORTH-type RAM file names are resources which are allocated by HP. For information about obtaining allocated resources, see the HP-71 IDS, Volume I, Chapter 18.

4.4.1 LEX ID's and Token Numbers

The application must have token numbers allocated for each of the BASIC keywords which invoke the FORTH-based application software. The token numbers must be contiguous. The starting token number is conveyed to the Soft FORTH system via the ADF. The ADF also contains the information needed to create the BASIC keywords which are used to invoke the FORTH-based applications.

4.4.2 The Unique FORTH-Type RAM File

All variables and data structures created during the compilation of the application exist (at runtime) in a FORTH-type RAM file. The name for this file is conveyed to the Soft FORTH program via the ADF.

The name of the unique file is embedded in the application ROM by MAKEROM. The Soft FORTH kernel ensures that both FORTHSYS and the application's unique RAM file exist whenever the application is invoked.

If FORTHSYS does not exist, it is created and initialized. If the application's unique RAM file does not exist, it is created and initialized to zeros; the application code must ensure that the file is initialized upon entry to any keyword. This could be done by reserving a variable which is set to a non-zero value when the file is properly initialized by the application program.

Each time the application accesses data from this file, the starting address of the file must be known. The FORTHSYS file has a location which contains the address of the unique RAM file. APFILE is one of the Soft FORTH words which uses this location. The address is updated every time the file might have moved. The following Soft FORTH words can cause the file to move:

ADJUSTF	ASSEMBLE	BASICF
BASICI	BASICX	BASIC$
CREATEF	GROW	SHRINK

<the configure poll>

Space is allocated in this file by use of the NALLOT-RAM word. It is necessary to distinguish between RAM and ROM when allotting extra space for data structures.

4.5 *Primitives*

Soft FORTH uses the assembler in the FORTH/Assembler ROM (which must be present in the HP-71 during the compilation process). The process of invoking the assembler in the FORTH/Assembler is as follows:

1. If the space between the Terminal Input Buffer (TIB) and the bottom of the Soft FORTH return stack (which is a higher address than the top of the return stack) is not at least 475 nibbles then ASSEMBLE increases the space by 75 nibbles. The default size of this space is 400 nibbles. This allows 15 additional return stack levels which are needed because the MAKEROM word does a LOADF which in turn does an ASSEMBLE. The assembler checks to make sure that it is not extending into the TIB area. If it is extending into this area it will error with the "Invalid expression" error message (admittedly not the most informative of messages).

2. If the space between the Data Stack and PAD is less than 4564 nibbles (84 bytes for PAD, 60 items on Data Stack, plus 2K bytes dictionary), ASSEMBLE will GROW the FORTHSYS file to allow for a 2K byte dictionary.

3. Copy the FORTH vocabulary word (which hard FORTH expects as the first word in the RAM dictionary) into FORTHSYS at DICTST (#302ED).

```
*DICTST is here.    *DI~      s h~
            CON(5)   0            No previous RAM words
            CON(2)   #C5          immediate word, 5 chars long
            NIBASC   \FORT\
            NIBASC   \H\+#80
            CON(5)   #E7160       DOVOCabulary prologue
            CON(2)   #81          fake header retained for
            CON(2)   \ \+#80      historical purposes
            CON(5)   (=DICTST)+5  last word in this vocabulary,
*                                 points back to #C5 FORTH
            CON(5)   0            no other vocabularies
```

4. Save the runtime addresses for the vectored words in the SFTFORTH kernel, and set these locations with the default runtime addresses of the hard FORTH vectored words (obviously, this can only be done when the SFTFORTH kernel is in IRAM).

5. Set CONTEXT and CURRENT (which have no meaning in Soft FORTH) to point at the word link in the FORTH word (i.e. the CON(5) (=DICTST)+5).

6. Set the vocabulary link, VOC-LINK, to point to the vocabulary link in the FORTH word (i.e. the CON(5) 0 which follows the word link).

7. Rename FORTHSYS to FORTHRAM.

8. Save the current dictionary pointer (DP), the instruction pointer (I), and the ROM image base address on the FORTH return stack. The ROM image base address is normally stored in the user variable FENCE, which has no meaning in Soft FORTH.

9. Set up the 10 nibbles in the SFTFORTH kernel which follow the call to the hard addressed Assembler to look like a FORTH word sequence. For example, assume that the 10 nibbles in question start at 40000. When done, the 10 nibbles look like the following:

Address	Contents
40000	40005
40005	4000A
4000A	<code>

   Put the return address on the FORTH return stack. In this example the return address is 40000.

10. Set up the FORTH instruction pointer (I) to point to a fake word which essentially contains the execution address of the Assembler and the SEMI word.

At this point the hard FORTH assembler begins assembling the given file.

```
**************************************************************
* IF THE ASSEMBLER IS ABORTED WHILE ASSEMBLING, SOFT FORTH   *
* LOSES CONTROL TO THE FORTH/ASSEMBLER ROM. IF THIS OCCURS,  *
* IT IS NECESSARY TO RESTART THE ENTIRE SOFT FORTH PROCESS.  *
**************************************************************
```

When the assembler has finished, it returns to Soft FORTH via the following process:

1. Rename FORTHRAM to FORTHSYS.

2. Restore from the return stack the Soft FORTH DP, I and ROM image base address (the ROM image base address is stored in the user variable FENCE which has no meaning in Soft FORTH).

3. Restore the runtime addresses of the vectored words.

4. Enter a BEGIN WHILE REPEAT loop which moves the headers of any words defined by the assembler (it could have only been defining LEX files) into the appropriate place in the header file. The loop also sets the prologue of each primitive to CON(5) 5 (this is a relative offset to the parameter field from the code field).

5. Move all of the code generated by the assembler into the ROM image as one block, headers and all. It is necessary to waste the space for the headers because all relative jumps and GOSUBs include the space taken up by intervening headers. Removing them means it would be necessary to effectively re-assemble all the code.

6. Run the do-checksum word to allocate any needed checksums.

4.6 *Checksums*

The Soft FORTH kernel contains a word (CHECKSUMS) and two data structures to automatically reserve space for checksum bytes. The CHECKSUMS command is defined as:

```
: CHECKSUMS  0 . 0 . 0 . 0 . ;
```

This is the state of the CHECKSUMS word when the Soft FORTH kernel is first copied into the 64K IRAM. The CHECKSUMS word initially displays four zeros.

4.6.1 Implementation of CHECKSUMS

The location of each of the zeros is known internally, and is used by the code that actually allocates the checksum bytes.

Two locations are reserved within the kernel for processing the

checksums:

NIBi BSS 1 — This nibble determines how many checksums have been allocated. Zero means the checksum code needs to be initialized. NIBi is zero in the SFTFORTH kernel as distributed.

NIBADR BSS 5 — This location contains the last address at which a checksum was allocated, ANDed with #F8000. If no checksums have been allocated, this location contains zero.

Every time that the colon word ":" is executed, the internal word DOCHSM is executed.

The first time DOCHSM is executed, NIBi contains 0. This indicates that the checksum system needs to be initialized.

The current value of the dictionary pointer (DP) is ANDed with #F8000; this becomes the initial value of NIBADR.

The first checksum offset is set to point to a location in the Soft FORTH kernel which is reserved for a checksum.

The second checksum offset is set to the current value of DP, since the first user definition must be in the second ROM chip (the first ROM chip is the Soft FORTH kernel). The DP is incremented by 2 before returning to colon to finish processing the ":".

Each of the four checksums in CHECKSUMS is compiled as:

```
REL(5)  =LIT          (LITERAL)
CON(5)  0
REL(5)  =DOT
```

Subsequent executions of DOCHSM progress as follows:

NIBi is non-zero: no initialization is needed.

The current DP value is ANDed with #F8000 and compared with the contents of NIBADR. If they match, no further action is needed, and DOCHSM returns to colon. If NIBADR and (DP AND #F8000) are not equal, NIBADR is set to (DP AND #F8000), NIBi is incremented and the checksum is allocated as above.

The address for the Nth checksum offset is computed as:

(address of the first offset in CHECKSUMS) + (15 * (N-1)),

where N is the contents of NIBi.

4.6.2 Using CHECKSUMS

After the application has been compiled, the CHECKSUMS word shows how many checksums have been allocated and the offset from the beginning of the IRAM to each checksum.

DOCHSM is available as the Soft FORTH primitive DO-CHECKSUM for use by user-created defining words.

4.6.3 Limitations and Precautions of CHECKSUMS

1. In order to eliminate problems arising from creating a definition before executing the MAKEROM word, NIBi, NIBADR and all 4 checksum location holders in CHECKSUMS are zeroed whenever MAKEROM is executed.

2. The DOCHSM word has also been added to the Assembler, but the DOCHSM happens only *after* any code generated is moved into the IRAM.

3. This scheme assumes that an application does not create a word which is longer than 16K bytes.

CHAPTER 5
Limitations in Soft FORTH

5.1 *Calling BASIC*

If the applications make use of the FORTH to BASIC capability (specifically BASICX) then these statements must be non-programmable. This is because the FORTH and BASIC environments are not reentrant; the application cannot start in BASIC, exercise FORTH, have FORTH exercise the BASIC system through BASICX, return to FORTH and then return to a running BASIC program. This restriction also applies in the FORTH/Assembler ROM: FORTHX (a BASIC keyword) may not invoke BASICX from within FORTH.

5.2 *Calling Other FORTH-based Applications*

The restriction that words which call BASIC must be non-programmable also applies when one FORTH-based application calls another FORTH-based application. Since FORTH-based applications are invoked through BASIC keywords, the only way one can call another is through BASICX. This means that if one application keyword calls a second application, the first application keyword must be nonprogrammable.

Special efforts are necessary to return from calling another FORTH-based application. Since both applications share the same FORTHSYS file, the called application clears the active flag in FORTHSYS and the calling application does not regain control. To avoid this, the calling application must rename the FORTHSYS file to something else during the call, and restore it before the return:

```
" RENAME FORTHSYS TO FS @ DOFORTH @ PURGE FORTHSYS @
RENAME FS TO FORTHSYS" BASICX
```

5.3 *Calls from the Forth/Assembler ROM*

FORTH based applications created via Soft FORTH may be called from FORTH (the FORTH/Assembler ROM) via the FORTH to BASIC words. As an example, suppose a FORTH based application is exercised via the keyword MYWORD. Enter the hard addressed FORTH environment and execute

```
" MYWORD" BASICX ( or BASICI,BASICF,BASIC$ )
```

MYWORD executes, but control returns to BASIC, not FORTH.

When MYWORD is executed, the FORTHSYS file is created (or moved) so that it is the first file in memory (displacing FORTHRAM). When the line containing MYWORD finishes, the HP-71 issues the main loop poll. When the FORTH/Assembler ROM sees the poll, it sees that FORTHRAM is not the first file in memory and decides that FORTH is not active.

One way to execute MYWORD and return to FORTH is

```
" MYWORD @ PURGE FORTHSYS" BASICX
```

This restores FORTHRAM as the first file in memory and allows FORTH to regain control at the main loop poll.

This will work only for Soft FORTH words which do NOT call BASIC through BASICX.

CHAPTER 6
Soft FORTH Words

6.1 *Words not present in Soft FORTH*

These words which are in the FORTH/Assembler ROM are not in Soft FORTH:

DEFINITIONS
FORGET
FORTH
FORTH$
FORTHF
FORTHI
FORTHX
VOCABULARY

DEFINITIONS, FORGET, FORTH, FORTHX, and VOCABULARY are not meaningful in the Soft FORTH environment.

FORTH$, FORTHF, and FORTHI have been replaced by SFORTH$, SFORTHF, and SFORTHI, respectively.

Soft FORTH does *not* include any of the keywords contained in EDLEX, DBGLEX, or KEYBOARD. These words are:

EDLEX	*KEYBOARD*	*DBGLEX*
DELETE#	ESCAPE	DEBUG
EDPARSE$	KEYBOARD IS	
EDTEXT	RESET ESCAPE	
FILESZR		
INSERT#		
MSG$		
REPLACE#		
SCROLL		
SEARCH		

6.2 *New Words present only in Soft FORTH*

These words are similar in function to the FORTHx words:

SFORTH$ SFORTHF SFORTHI

These words are not in the FORTH/Assembler ROM, but have been added because Soft FORTH requires them:

,REL	APFILE	CHECK	CHECKSUMS
CREATE-RAM	DEBUG	DO-CHECKSUM	ENDFORTH
MAKEAP	MAKEROM	NALLOT-RAM	NXTHED
NXTVAR	ROMADD	STRING-ROM	UNDEBUG

,REL ,REL (n ---) converts an absolute address (of a CFA) into a offset from the beginning of the Soft FORTH kernel and compiles it into the ROM image.

APFILE APFILE (n --- address) finds the address of the start of data in the unique FORTH type RAM file and adds n to it, producing the address of the desired data structure. N is the offset into the unique FORTH type RAM file.

CHECK CHECK (address --- address) tests the target address of FORTH words which write to memory to ensure no writes access the IRAM (ROM image). If the DEBUG flag is not set, CHECK does no checking. The target address is compared with the address of the Soft FORTH kernel. If the target address is within the range starting at the address of the kernel and extending for 64K bytes, CHECK prints out the items on the FORTH return stack and goes to QUIT. CHECK leaves the system in interactive Soft FORTH, where the contents of the data stack can be printed out. It is assumed that the CHECK code is only exercised when headers are available and the development environment is active.

In the ROM version of Soft FORTH (ROMFORTH), CHECK returns immediately.

CHECKSUMS CHECKSUMS (---) displays the offset from the beginning of the IRAM to the one byte fields allotted for use as checksums. One field is allotted for each 16K byte chip. If no code has been written into a 16K byte chip the CHECKSUMS word returns a 0 in that position.

CREATE-RAM CREATE-RAM (---) creates a dictionary entry which points into the RAM file.

DEBUG DEBUG (---) sets a flag (a nibble in FORTHSYS: oDEBUG) signifying "debug on" and saves the current dictionary pointer in FORTHSYS variable oROMDP, resetting DP into FORTHSYS (at DICTST).

DEBUG is a no-op in the ROM version of Soft FORTH (ROMFORTH).

DO-CHECKSUMS DO-CHECKSUMS (---) causes a checksum byte to be allocated if the DP has crossed into a new ROM chip since the last checksum was allocated. DO-CHECKSUMS is called by ":", and is included in Soft FORTH for use in new defining words.

ENDFORTH — ENDFORTH (---) finishes the process started by MAKEROM. ENDFORTH collapses the application LEX file to the minimum size needed, and removes the header file. It also changes the SFTFORTH kernel file so that the SFTFORTH statement is no longer recognized.

MAKEAP — MAKEAP (---) creates the unique FORTH-type RAM file. MAKEAP assumes that the information preceding the text table in the application LEX file has been set up. It takes the file size parameter and the file name parameter and creates the file (if it doesn't already exist).

Since this is done automatically each time a FORTH based application is executed there should not be any reason to use this word, but it is here for any exceptional cases.

The information in the application LEX file required by MAKEAP is set up by the MAKEROM word. The information is used to prepare the unique FORTH-type RAM file before an application keyword is executed.

MAKEROM — MAKEROM (str ---) creates a FORTH-based application, using the information in the ADF (str). The SFTFORTH kernel file is shrunk to 16K bytes when the application is completely loaded.

NALLOT-RAM — NALLOT-RAM (n ---) is used to allot additional space in the FORTH type RAM file. NALLOT-RAM allots n nibbles.

As an example, allot 25 nibbles for a data structure called MYDATA (Note that VARIABLE allots 5 nibbles):

HP-71 FORTH: VARIABLE MYDATA 20 NALLOT

Soft FORTH: VARIABLE MYDATA 20 NALLOT-RAM

See the Soft FORTH definition of VARIABLE for another example of how NALLOT-RAM is used.

NXTHED — NXTHED (--- n) returns the address of a variable which contains the address of the next free nibble in the 16K byte block containing the headers.

NXTVAR — NXTVAR (--- n) returns the address of a location (the 5 nibble field which precedes the text table in the application LEX file) which contains an offset to the next free nibble in the user's FORTH type RAM file.

*** CAUTION ***

NXTHED and NXTVAR should be used with great care. They are

provided to make it possible to create new defining words (i.e. VARIABLE is a defining word). They have no counterpart in HP-71 FORTH. NXTVAR is used by CREATE-RAM.

ROMADD — ROMADD (n --- n+base) finds the starting address of the Soft FORTH kernel and adds this to n, producing the current absolute address. N is assumed to be an offset from the beginning of the Soft FORTH kernel.

ROMADD is typically used when dealing with the headers, because the CFA offset is stored with each header. ROMADD is used to compute the current absolute CFA for an application that directly manipulates the headers.

STRING-ROM — STRING-ROM (n ---) works like STRING, except that the space is allocated in the application LEX file instead of the user RAM file.

UNDEBUG — UNDEBUG (---) tests and clears the debug flag in FORTHSYS. If the debug flag was set, reset DP to the value saved in oROMDP by DEBUG. See DEBUG for more information.

UNDEBUG is a no-op in the ROM version of Soft FORTH (ROMFORTH).

6.3 *Words Which Changed in Soft FORTH*

The following words have changed with respect to the FORTH/Assembler ROM:

```
'
[']
ABORT
ABORT"
DOES>
FIND
GROW
INTERPRET
SHRINK
STRING
STRING-ARRAY
VARIABLE
(ONERR)
```

6.3.1 ' (TIC)

TIC returns one of two results, depending on the current mode. Execute mode TIC returns the absolute address of the word specified, while compile mode TIC returns the offset from the start of the ROM to the

word.

6.3.2 ['] (BRACKET TIC)

BRACKET TIC compiles the offset from the start of the ROM to the word.

6.3.3 ABORT

ABORT and ABORT" exit the FORTH environment after executing. This is absolutely necessary, since there is no FORTH outer loop available to a FORTH based application.

If the ONERR location is set, ABORT and ABORT" will use its contents as a CFA. See "(ONERR)" below for more details.

The FORTH/Assembler module manual indicates they do not use ONERR, but the FORTH/Assembler code does look at ONERR.

6.3.4 DOES>

DODOES has been removed from the 16K Kernel and placed in FORTHSYS. The defining word DOES> compiles <;CODE> followed by a GOSBVL =DODOES. This GOSBVL is necessary because of the way <;CODE>, DODOES and DOES> interact. A GOSUBL is not acceptable because only words defined within 16K bytes of the 16K kernel could have successfully used DODOES due to the range limitations of GOSUBL. FORTHSYS represents the only fixed address space available to Soft FORTH, so DODOES has been added to it.

The headerless word PFA which was only used by <;CODE> has been removed.

6.3.5 FIND

FIND returns the current absolute CFA.
NOTE: FIND is ONLY useful during compilation since it requires the presence of the headers to be useful.

6.3.6 GROW and SHRINK

GROW and SHRINK grow and shrink the FORTHSYS file. SFTFORTH allows the minimum of space in FORTHSYS when it creates it.

Applications may grow and shrink the FORTHSYS file at run time. Since every FORTH based application uses the same FORTHSYS file, each application must use GROW and SHRINK in balance such that FORTHSYS is left with the same size.

6.3.7 INTERPRET

INTERPRET is also only useful during the compilation process.
NOTE: INTERPRET is ONLY useful during compilation since it requires the presence of the headers to be useful.

6.3.8 STRING

During compilation the following information is compiled into the application ROM image:

5 nibbles:	<DOSTR>	Relative offset to the string handling prologue
5 nibbles:	<offset>	Offset that, when added to the starting address of the unique FORTH-type file, points to this string
2 nibbles:	<max len>	Maximum number of bytes this string variable can contain.

In addition, (max+2)*2 nibbles are allotted (using NALLOT-RAM) in the application's unique FORTH-type file.

Whenever the string is accessed (i.e. the DOSTR prologue is executed) the <max len> compiled into the ROM image is written to the <max len> field in the RAM file. The string address and the <current len> are then pushed onto the data stack.

This is done because string operators require that strings have the form:

<maximum length> <current length> <string>

The unique FORTH-type RAM file may have just been created (i.e. the BASIC keyword which exercises the FORTH-based application may have just been executed). When the file is created, it is set to all zeros. Writing the maximum length to the RAM file each time ensures that other string words will function correctly.

6.3.9 STRING-ARRAY

STRING-ARRAY: During compilation the following information is compiled into the ROM image:

5 nibbles:	<DOSTRA>	Relative offset to the string array prologue
5 nibbles:	<offset>	Offset that, when added to the starting address of the unique FORTH-type file, points to this string array
2 nibbles:	<max len>	Maximum number of characters allowed in each array element
2 nibbles:	<dim>	Array dimension

In addition, (max+2)*dim*2 nibbles are allotted (using NALLOT-RAM) in the application's unique FORTH-type file

Whenever the string-array is accessed the DOSTRA code compares the <max len> compiled into the ROM image and the first <max len> field in the string-array.

If the <max len> fields are different, DOSTRA assumes the string-array has not been initialized, and initializes each <max len> field in the array with the <max len> value compiled into the ROM image.

The requested element is then found, and the address and current length of that element are returned on the data stack.

6.3.10 VARIABLE

The FORTH/Assembler ROM definition of VARIABLE is

```
: VARIABLE CREATE 5 NALLOT ;
```

The Soft FORTH definition of VARIABLE is

```
: VARIABLE CREATE-RAM 5 NALLOT-RAM ;
```

6.3.11 (ONERR)

The FORTH/Assembler interprets the contents of ONERR as a CFA, if non-zero. Soft FORTH examines the value first. If the value is less than 20000 Hex, the base address of the application Rom is added to the value, and the result is used as the CFA.

Soft FORTH has also changed the way BASIC's ON ERROR works with FORTH. First, a description of how the FORTH/Assembler module works. There are three different ways to trap an error:

1. A BASIC program calls FORTH via FORTHX with ON ERROR in effect.

2. A FORTH program calls BASIC via BASICX, BASIC$, BASICI, or BASICF. The FORTH ONERR location is set to the CFA of an error routine.

3. A FORTH program calls BASIC via BASICX, BASIC$, BASICI, or BASICF. The BASIC program traps errors via ON ERROR.

Both the FORTH/Assembler module and Soft FORTH are able to detect only two of the three cases. FORTH/Assembler handles cases 1 and 2; any use of case 3 will fail. Soft FORTH handles cases 2 and 3; any use of case 1 will fail. This change was made because Soft FORTH does not have FORTHX, and any keywords which use BASICX are not programmable (see the chapter on limitations in Soft FORTH). If an application uses BASIC$, BASICF, or BASICI, it must be sure to trap any possible errors if any calling program is to continue running.

6.4 *Words which check for writes to ROM*

Here is a list of the words which check for writes to ROM when DEBUG is active:

```
CMOVE   CMOVE>
N!      C!      !       +!
NFILL   FILL
NMOVE   NMOVE>
STO     ENTER
```

These checks are not done in the ROM version of Soft FORTH (ROMFORTH).

CHAPTER 7
FORTH/Assembler Definitions for Soft FORTH

This chapter provides definitions for those Soft FORTH words which are needed to develop an application for Soft FORTH, using the FORTH/Assembler ROM. These should be defined *before* compiling the application with the FORTH/Assembler ROM.

,REL	: ,REL , ;
APFILE	See note below
CHECK	: CHECK ;
CHECKSUMS	: CHECKSUMS 0 . 0 . 0 . 0 . ;
CREATE-RAM	: CREATE-RAM CREATE ;
DEBUG	: DEBUG ;
DO-CHECKSUMS	: DO-CHECKSUMS ;
MAKEAP	See note below
NALLOT-RAM	: NALLOT-RAM NALLOT ;
NXTHED	See note below
NXTVAR	See note below
ROMADD	: ROMADD ;
UNDEBUG	: UNDEBUG ;

*** Note: APFILE, MAKEAP, NXTHED, and NXTVAR have no counterpart in FORTH. They must be added to the application code when it is to be used with Soft FORTH.

NXTHED and NXTVAR are of use mostly to define new words. For NXTVAR, the most common usage is 'NXTVAR @ ,'. If the application needs to use NXTVAR, call it with another word, defined as follows:

```
: SETVAR NXTVAR @ , ; ( Soft FORTH definition )
: SETVAR ; ( FORTH/Assembler ROM definition )
```

and use SETVAR in new defining words.