V7540-71-7

TRANSLATOR PAC
PROGRAMMER'S TOOLKIT

FOR THE HP-71B WITH THE TRANSLATOR PAC

USER'S MANUAL

BY WILLIAM C. WICKS

Translator Pac Programmer's Toolkit

For the HP-71 with the HP 82490A Translator Pac

User's Manual

by William C. Wickes

CONTENTS

Translator Pac Programmer's Toolkit

For the HP-71 with the HP 82490A Translator Pac

User's Manual

by William C. Wickes

1. INTRODUCTION

The Translator Pac Programmer's Toolkit, or TPPT for short, is a set of extensions to the HP 82490A HP-41 Translator Pac, that provides an enhanced programming capability for the HP-41 and FORTH languages. With the TPPT word set, you can:

- List (decompile) FORTH words and HP-41 programs.
- Single-step FORTH words and HP-41 programs.
- Set breakpoints to halt program/word execution at specified places.
- Run HP-41 programs in "trace" mode.
- Time program/word execution.
- Write HP-41 programs from within the HP-41 environment, bypassing the translation stage.

Once you have loaded the TPPT into your HP-71, you can use it as a permanent extension to the HP-41 and FORTH capabilities provided by the Translator Pac. Or you can use it only for program development--once an HP-41 program or set of FORTH words is perfected, you can load the new words or program into a FTH41RAM file without the TPPT.

1.1 Loading TPPT into the HP-71.

The TPPT file is an HP-71 file of type FORTH, that you use by simply making it the current FTH41RAM file:

From HP-IL mass storage:

```
COPY TPPT:TAPE TO FTH41RAM
```

From cards:

```
COPY CARD TO FTH41RAM
```

From memory:

RENAME TPPT TO FTH41RAM

If you already have a FTH41RAM file in memory, you will have to **PURGE** or **RENAME** it prior to executing any of these instructions.

The TPPT file is a precompiled FTH41RAM file, to which you can add your own HP-41 programs and/or FORTH words. With the TPPT file installed as the current FTH41RAM file, you can enter the FORTH or HP-41 environments as usual with the Translator Pac. The normal FTH41RAM dictionary structure is altered in the TPPT file, so that most of the file's functions (all except the real-time HP-41 program defining words **PRGM** and **END**) are available in both the HP-41 and the FORTH environments. This is accomplished by changing the links of the **FORTH** and **HP41V** words so that **HP41V** is placed ahead of the TPPT words in the search order.

1.2 FORTH Terminology

The Translator Pac was designed to provide HP-41 keyboard operation and program execution on the HP-71, without requiring you to understand the low-level design of the translator or anything of the FORTH language that underlies the HP-41 environment. Similarly, you can use the HP-41 programming tools included in TPPT without knowing FORTH. However, if you wish to make your own customizations to the HP-41 emulator, or to use the FORTH tools, you should have a working knowledge of FORTH programming.

On many occasions in this manual, we will mix certain FORTH and HP-41 terminology. If you're an HP-41 programmer unfamiliar with FORTH (or vice-versa), here's a brief summary of the concepts we'll be using.

In HP-41 programming, the terms *function*, *program*, and *program line* are used frequently:

- A *function* is a built-in (or plug-in) keyword that the HP-41 recognizes--when included in a program, it is listed by name, with no **XEQ** prefix. Most HP-41 functions are *postfix*, meaning that they expect any arguments to be present on the floating-point stack, the alpha register, etc., before the functions are executed. Postfix functions are all "one-byte" or "one-part" functions in the HP-41. HP-41 multi-byte functions are *prefix* functions, where the function name (**RCL**, **STO**, **XEQ""**, **SF**, etc.) is augmented by an argument--a register number, label number, or alpha string--that must follow the function name.

- A *program line* consists of a single function that has been "compiled" into program memory. (Number entry lines have no explicit function associated with them, but the numbers can be viewed as the arguments to an invisible number entry function.) When you view a program line in the HP-41, a line number is displayed in front of the function name, but the line number is not part of the program line--it is re-computed each time you switch to program mode.

- A *program* is an ordered collection of program lines that is intended to be executed as a unit. Programs are invoked by name using the **XEQ""** or **GTO""** functions. The HP-41 maintains a "program pointer," that indicates the memory address of the next program line to be executed. The program pointer is updated after each program line while a program is running. Also, you can set the program pointer to any program line by using **GTO** or **GTO..**

In FORTH there is no distinction between programs and functions. All programming consists of creating "words," which are named, individually executable functions. A word can be written in assembly language, like HP-41 functions, or it can be defined as a collection, called a "secondary," of previously defined words, analogous to an HP-41 program. But there is no analog to the HP-41 program line--once a word is compiled, its internal structure is normally inaccessible.

FORTH program memory is a linked list of words called the "dictionary.". The dictionary can be subdivided into "vocabularies", which are independent linked lists. In the Translator Pac dictionary, HP-41 words, and all user-added programs, are collected in the **HP41V** vocabulary.

The FORTH equivalent of the HP-41 program pointer is the "inner interpreter pointer." The inner interpreter is the "engine" that keeps the FORTH system running. The inner interpreter pointer points to the next FORTH word to be executed, which may also be considered as the first word in a list of pending words. Each word can itself be a list of words (i.e., it is a secondary)--the FORTH return stack saves values of the interpreter pointer when execution is nested down any number of levels into secondaries of secondaries of ....

In this manual, we will use the term *word* as a generic term that includes HP-41 functions and ordinary FORTH words. We will use the terms *function* and *program* when we want to refer specifically to HP-41 programming. An HP-41 program in the emulator is structured very like a FORTH secondary--it is a list of words to be executed successively by the inner interpreter. The inner interpreter pointer serves as the HP-41 program pointer when a program is executing; when the program halts, the current value of the pointer is saved (in a variable we will also refer to as the HP-41 program pointer) so that the inner interpreter can continue ordinary FORTH execution.

1.3 Nomenclature Conventions

Throughout this manual, we will use a few simple conventions:

1. FORTH words are normally described by specifying their use of the FORTH stack, also called the data stack, parameter stack, or integer stack. We will specify stack use in the following format:

   SAMPLE [*itemn* ... *item2 item1* -> *item'm* ... *item'2 item'1*]

   where **SAMPLE** is the FORTH word being defined, and *item1* through *itemn* are the objects required on the stack (*item1* on top) before execution of **SAMPLE**. *item'1* through *item'm* are left on the stack after execution.

2. Certain FORTH and TPPT words use a prefix syntax, that is, they require input from the keyboard to follow the word name. For example, the word **CREATE** takes the name of a new dictionary entry from the keyboard. We will indicate this kind of syntax by enclosing the required keyboard input items in braces { }. **CREATE**, then, would be listed with the the syntax **CREATE** {*wordname*}, to indicate that **CREATE** takes the next string from the keyboard input stream as the name for the new dictionary entry. If a single item needs more than one word to represent it, we will join the words with hyphens to avoid confusion with multiple-item lists.

3. The typeface used in specifying input and output indicates the fixed or variable nature of the input or output:

   - **Boldface** print specifies functions, words, commands, etc., where the form of the input or output is fixed.

   - *Italics* indicates variable input and output, where an item specified in italics is replaced in actual use by a specific value or text. For example, the a single line of an HP-41 program listing, for example, would be described in the format

     address function

     which indicates that there are two items in each line of output: first, the memory address, then the name of the function.

Thus in the example **CREATE** {*wordname*}, **CREATE** is a specific FORTH word, and is shown in boldface, whereas *wordname* represents a varying user input.

4. HP-71 keys are indicated by square brackets around the key name in boldface, e.g., **[ENDLINE]**.

5. HP-71 memory addresses will usually be listed in hexadecimal (base 16), for a more compact representation (5 digits instead of 6 in decimal). In HP-41 program listings, however, we will use decimal, since that is the normal base for the HP-41 environment.

1.4 Controlling Output Display Duration

When a FORTH word sends text or numerical results to the HP-71 display, the result remains in the display until it is superceded by another display (such as the **OK { *n* }** message). When words display many successive results, the displays may flash by too rapidly to be read. For this reason, any TPPT words that output more than one line of displayed results execute the TPPT word **WAIT** (analogous to BASIC **WAIT**) after each new display, which, besides extending the duration of the display by a user-specified amount, allows you to suspend execution and single-step through each successive display.

You can set the duration of the delay produced by **WAIT** by using the word **PAUSE** {*time*}, where *time* is in milliseconds. For example, **PAUSE 1000** causes each TPPT word to display each of its results for one second, **PAUSE 500** for one half second, etc. In addition to the effect of **WAIT**, the duration and scrolling of displays longer than 22 characters are affected by the current HP-71 delay setting (through the BASIC keyword **DELAY**, which you can execute from the FORTH or HP41 environments using **BASICX**).

During the paused display produced by **WAIT**, you can suspend further execution by pressing any HP-71 key. Then the keyboard is active as follows:

- **[ATTN]** aborts the current word and returns to the normal FORTH/HP-41 keyboard mode.

- The left- and right-cursor keys scroll the display.

- The down-cursor key resumes execution of the current word, but insures that execution will halt again at the next output display. Successive use of down-cursor thus allows you to "single-step" through a multi-display word.

- Any other key resumes normal execution.

1.5 **Printer Output**

All TPPT words that return visible results normally direct their output to the LCD display and the current HP-71 DISPLAY IS device. By using the TPPT word **PRINT**, however, you can cause all display output of any FORTH word to be printed on an HP-IL printer connected to the HP-71.

PRINT is used in the form **PRINT** *{display-word}*, where *display-word* is any FORTH word that produces displayed output. During execution of *display-word*, the current HP-71 **PRINTER IS** device is also set as the **DISPLAY IS** device. When *display-word* has finished executing, the **DISPLAY IS** setting is returned to its original setting. For example,

PRINT F.

will print the current contents of the floating-point X-register, and

PRINT UN: ROOM

will print the decompiled (see Section 2.2) listing of the word **ROOM.**

If any error occurs during execution of **PRINT**, or you press the **[ATTN]** key, the HP-71 will display **Halted**. You can press **[ERRM]** to read the error message associated with the error.

[Note: **PRINT** sets **ONERR** to point to a recovery routine that will restore the **DISPLAY IS** device in case an error occurs during execution of *display-word*. If you have defined a *display-word* that itself resets **ONERR**, you can include the TPPT word **DBACK** in the associated error handling word to also restore the display. Or you can execute **DBACK** from the keyboard (note: **DBACK** also stores 0 in **ONERR**).]

1.6 **Further Reading**

- In Section 2, we begin the discussion of the major TPPT features by describing the methods of listing HP-41 programs and decompiling FORTH words
- Section 3 describes traced execution (single-stepping and breakpoint execution) and a method of timing HP-41 programs and FORTH words.

- In Section 4, we introduce some methods of extending the HP-41 language through adding new HP-41 functions to the emulator capability set. We also describe a method of writing HP-41 programs without using **TRANS41**. You will need at least a rudimentary knowledge of FORTH programming techniques to use this material.

- Section 5 is an alphabetical listing of the words added to the Translator Pac's FORTH/HP-41 dictionary by the TPPT. Each word is listed with its stack use and other arguments, plus a short description of its operation and a section number reference for further explanation.

2. PROGRAM LISTING AND DECOMPILATION

HP-41 programs and FORTH words in the FORTH dictionary are "compiled," that is, user-readable program "source" code has been replaced with FORTH execution addresses and data. The TPPT contains several FORTH words that will "decompile" programs and words, converting the stored addresses and data to readable text that will closely resemble the original program source code.

We will begin by describing the method of obtaining HP-41 program listings in section 2.1. In section 2.2, we will describe the words for decompilation of various types of FORTH words.

2.1 Listing HP-41 Programs

The TPPT provides three HP-41 program listing words. **PRP** and **LIST** are modeled on their HP-41 pointerparts **PRP** and **LIST**. The third word, **LISTN**, is just a postfix version of **LIST**. They are used in the following forms:

PRP {*alpha-label-name*}**"**

LIST {*number-of-lines*}

LISTN [*number-of-lines* ->]

(A space is required between **PRP** and the first character of the label name. The final **"** is required only if additional words follow in the command line.) In addition, the words **LBLS** and **ALBLS** list the current contents of the HP-41 labels buffer.

2.1.1 *Program Listing Format*

PRP { *label* } lists the entire program containing the specified alpha *label*. **LIST** { *number* } (or *number* **LISTN**) lists *number* lines, starting at the current HP-41 program pointer, stopping at the program **END** if it is encountered before the full number of lines is listed. **LIST 0** will list from the current program pointer to the **END** of the program. (You can move the program pointer to the start of a program by using **RTN** or **END**; to a program label using **GTO** or **GTO"""**; or to an arbitrary program line using the TPPT function **GTO.** described in Section 3.2.1.)

The output of **PRP** or **LIST** is very similar to HP-41 printer program listings. The principal difference is the substitution of HP-71 memory addresses for HP-41 program line numbers. For listing and debugging purposes, the memory addresses can be used just like HP-41 program line numbers; the memory addresses have the additional advantage of allowing you to use FORTH words such as @ and ! for viewing or replacing compiled code.

Numbers in number entry lines are listed in the current floating-point display format. **STD** format is probably the best format for program listing, since you will see all the significant digits of a number.

One other difference arises from the fact that program labels are stored separately from the programs. This results in program listings showing the labels without "line numbers" (addresses). Note that when you execute **GTO** *label* or **GTO"***label*", the program pointer is positioned to the program line that follows the label.

Numeric and local alpha **GTO** (the following applies to **XEQ** also) lines are listed in either of these formats:

GTO *label*

or

GTO *label* **at** *address*

where *label* is the number or letter of the target label. When a **GTO** line is executed in a running program, the compiled label number is replaced by the label address. If you list a program after it has been executed, the **GTO** line is shown with the **at** *address* extension, which shows the precise destination of the **GTO**.

When you read listings obtained with **PRP**, you should remember that the translator program **TRANS41** makes certain changes to an HP-41 program during translation. These changes show up as differences between your original program text file version of the program and the listing of the program you obtain with **PRP**. Specifically:

- HP-41 program text lines are translated as the FORTH word **A"** followed by a counted string representing the text (similar to the FORTH word **."**). The program line **"TEXT"** is thus listed by **PRP** as **A" TEXT"**.

- The functions **ENTER^**, **CLX**, **S+** and **S-**, which disable stack lift in the HP-41, are replaced by appropriate functions that produce the same effects as stack lift disable. The choice of a replacement for one of these functions depends on whether the immediately following program line contains a function that normally raises the stack. For example, if **CLX** is followed by **RCL 1**, it is replaced in the translated program by **RDN**, whereas if it is followed by **SIN**, it is left as **CLX**, which is actually equivalent to **RDN 0**. A third version of **CLX, named CLXR**, is required when the next line is

RCL with a stack argument, such as **RCL X, RCL T,** or **RCL IND Z.** Table I lists the replacements for each stack-lift disable function for each of the three possible program situations.

Table I. Substitutions for Stack-Lift Disable Functions

Original Function		Next Line:	
	Stack Raise	No Stack Raise	RCL X,Y,Z,T,L or RCL IND X,Y,Z,T,L
CLX	**RDN**	**CLX**	**CLXR**
ENTER^	**NOP**	**ENTER^**	**ENTER^R**
S+	**S+D**	**S+**	**S+R**
S-	**S-D**	**S-**	**S-R**

Here is a sample output from **PRP:**

```
*LBL"SUMF"
210883 A" COUNTING"
210912 AVIEW
210917 1.1
210938 0
*LBL 1
210959 1
210980 RCL Z
210990 INT
210995 +
211000 LASTX
211005 *
211010 +
211015 ISG Y
211035 GTO 1 at 210959
211045 BEEP
211050 A" SUM = "
211069 FIX 0
211079 ARCL X
211089 PROMPT
211094 END to 210883
```

The final entry shows that the **END**, when executed (with an empty HP-41 return stack), will put the program pointer at the start of the program, address **210883**.

2.1.2 *Listing Program Labels.*

HP41 program labels are stored in a special "labels buffer" at the end of the FTH41RAM file. You can list the contents of the buffer using **LBLS** or **ALBLS**. **LBLS** lists all of the local labels and **END**'s. Each label is displayed in the format

LBL *number* *address*

where *number* is the label number (or letter for local alpha labels), and *address* is the program address corresponding to the label. The labels and **END**'s are listed in order of increasing program address.

ALBLS lists the alpha labels in catalog order, i.e., in order of decreasing program address. Each is listed in the format

LBL"*text***"** *address*

2.2 **Decompiling FORTH Words.**

The FORTH decompiler words provided in the TPPT are **UN:** {*wordname*}, **UN:A**, and **UN:C**. The latter two are subfunctions of **UN:**, so a discussion of **UN:** will also cover the other two functions.

The basic purpose of **UN:** is to provide you with a "map" of a dictionary word, that shows you not only the memory location and structure of the word, but also allows you to determine the original source code definition of the word. **UN:** is most useful for secondaries (hence the name "uncolon," implying a reversal of the colon compiling process), providing, in effect, a "program" listing, for reference during debugging and single stepping.

When you execute **UN:** {*wordname*}, the following output is sent to the display (the duration of each successive display is determined by the latest **PAUSE** value--see Section 1.4):

LFA: *addr* **Link:** *previous-NFA* **to** *previous-wordname*
NFA: *addr* *nibbles*
CFA: *addr* *prologue-addr* *prologue-name* *parameters*

where the items in italics are derived from the word being decompiled. Following these three lines is additional output that depends on the type of word being decompiled.

2.2.1 *The Link Field*

The first line of output from **UN:** is

LFA: *addr* **Link:** *previous-NFA* **to** *previous-wordname*

where *addr* is the memory address of the link field, and *previous-NFA* is the contents of the link field, a pointer to the name field of the preceding word in the same vocabulary. *Previous-wordname* is the name of that preceding word.

2.2.2 *The Name Field*

Following the link field, the next line output by **UN:** is the name field, in the format

NFA: *addr* *count* *wordname*

where *addr* is the address of the name field, *count* is the count byte (displayed in hexadecimal), and *wordname* is the text of the name. The 3 most significant bits of the count byte of the name field are used as follows:

--The first is always set to value 1 to indicate the start of the name field.

--The second is 1 for an immediate word, 0 otherwise.

--The third is the so-called smudge bit, which is set during compilation of a word to prevent it from being compiled within its own definition, allowing redefinition of a word using the same name.

The 5 least significant bits of the count byte encode the number of characters in the word name. The last character byte of the name field also has its most significant bit set to 1 to indicate the end of the name field. The most significant bit set on the first and last bytes allows the name field to be traversed in either direction.

2.2.3 *The Code Field*

The code field is displayed after the name field, in the format

CFA: *addr* *prologue-name* *parameters*

addr is the code field address. *Prologue-name* is the name of the "prologue," i.e., the assembly language code that constitutes the

CPU-executable portion of the word. The code field contains the address of the prologue. *Parameters* consists of extra information that varies according to the prologue type.

In HP-71 FORTH with the TPPT extensions, there are four categories of prologue addresses:

1. If the prologue address stored in the code field of a word is the same as the parameter field address, then the word is an assembly language primitive. **UN:** will return the prologue name "**Primitive**" (*parameters* will be blank).

2. If a word was created by a new defining word containing the **CREATE...DOES>...** sequence, the prologue address is the address in the defining word just after the **DOES>**. In this case, **UN:** will return a prologue name "**DOES>**", and *parameters* will be "**at** *run-time-addr*." *Run-time-addr* points to the run-time code for the decompiled word, which is contained in the defining word after the **DOES>**.

3. If the word is a headerless FORTH system word that has been assigned a name by the TPPT defining word **HEAD** (see Section 4.2), the *prologue-name* is given as **Remote CFA**, and *parameters* will be the code field address of the headerless word.

4. If a word was created by one of the built-in FORTH defining words, the name of the original defining word will be given as the prologue name. Table II gives a list of the defining words, and any associated *parameters*. If the word is either a string or string-array word, *parameters* shows the string dimension(s) in brackets [ ]. For simple strings, the dimension is the maximum number of characters the string can hold. For string arrays, the dimensions are shown in the form [*n x m*], where *n* is the number of elements in the array, and *m* is the maximum number of characters in each element.

5. If the prologue does not belong to one of the preceding four categories, **UN:** will report the prologue name "**Unknown.**"

2.2.4 *The Parameter Field*

For all word types except vocabulary words and secondaries, the last entry in the decompiled output has the form

End at *address*.

Address is the address of the first nibble of the link field of the word immediately following in the dictionary.

Table II. Translator Pac Prologues

Prologue Addr	Defining Word	Parameters
E77B0	**:**	
E77C5	**CONSTANT**	*value*
E77D9	**FCONSTANT**	*value*
E7803	**VARIABLE** or **CREATE**	
E78F6	**VOCABULARY**	
E7816	**STRING**	*dimension*
E748D	**STRING-ARRAY**	*dimensions*

2.2.4.1 *Vocabularies*

The parameter field of a vocabulary word contains a pointer to the name field of the latest word added to the corresponding vocabulary, and a vocabulary link field. All vocabularies are linked together by their vocabulary link fields. The contents of each such field points to the same field in the chronologically previous vocabulary word. **UN:** decompiles the parameter field of a vocabulary word like this:

addr1 **Latest:** *latest-word-NFA latest-word-name*
addr2 **Prev. Voc.:** *previous-voc.-link previous-voc.-name*

Addr1 and *addr2* are the addresses of each entry in the vocabulary word's parameter field. *Latest-word-NFA* is the NFA of the latest word created in that vocabulary; *latest-word-name* is its name. *Previous-voc.-name* is the name of the last vocabulary word that was created prior to the vocabulary word being decompiled; *previous-voc.-link* is the address of the prior vocabulary's vocabulary link field.

2.2.4.2 *Secondaries*

For secondaries, the parameter field is decompiled as a list of the words that comprise the secondary word's definition. Each entry in the list has the form

addr cfa wordname

where *addr* is the address in the decompiled word's parameter field, and *cfa* is the contents of that address. Each *cfa* is the code field address of one of the words that comprises the decompiled word's definition. *Wordname* identifies by name the word corresponding to *cfa*.

For most words, *wordname* is just the dictionary name of the compiled word. There are two types of exceptions: immediate words that are compiled into a secondary along with data, and branch words. For the immediate words, the name is supplemented or replaced by an explicit display of the data, as shown in Table III.

Table III. Wordnames for Decompiled Immediate Words

Form in Definition	Compiled *cfa*	*wordname*	Example
(literal) *nnnnn*	E2083	*nnnnn*	12345
(floating point literal) *sm.mmmmmmmmmmmmEeee*	E2AB3	*sm.mmmmmmmmmmmmEeee* (in current display format)	1.234E25
" *text*"	E069B	" *text*"	" Hello"
." *text*"	E0FAF	." *text*"	." There"
ABORT" *text*"	E612D	ABORT" *text*"	ABORT" Error"

There are many system utility words contained in the Translator Pac ROM, plus several additional included in the TPPT file, that are compiled without name and link fields so that they are not available for ordinary programming. If you decompile a ROM word or a TPPT word that contains one of these "headerless" words in its definition, *wordname* is replaced by **Headerless**.

Branch words, which cause the interpreter pointer to jump to a new location, are compiled with the jump distance compiled immediately following the cfa. The *wordname* for branch words consists of the name of the word followed by **to** *destination*, where *destination* is the address where execution will resume after the branch.

The branch words are of three types, flag conditional, loop conditional, and unconditional branches. The flag conditional branch words, which test a flag on the stack to determine whether to branch or not, are **OF**, **IF**, **WHILE**, and **UNTIL**. The compiled *cfa* for all four words is **E6628**, so the decompiler must determine which wordname to report by examining the context of the branch. This latter process is not foolproof; when program structures are nested in certain ways, a wordname may be reported that differs from the original word definition. Moreover, **OF** is decompiled as the sequence

```
OVER
=
IF to destination
DROP
```

Nevertheless, the execution flow described by the decompilation will be unambiguous.

LOOP and **+LOOP** are the loop conditional branch words, which branch or not depending on the loop parameters stored on the return stack. The decompilation of these words is unambiguous.

The unconditional branch words are **ELSE**, **ENDOF**, **LEAVE** and **REPEAT**, which compile as word address **E663B**. As in the case of the flag conditional branch words, the decompiler determines which wordname to report from the context of the branch. Unfortunately, it is not straightforward to distinguish **ENDOF** from **ELSE**, so **ENDOF** is decompiled as **ELSE**. Like **OF**, **LEAVE** is actually compiled as four words, and is decompiled as

```
R>
R>
2DROP
ELSE to destination.
```

Finally, the words **BEGIN** and **THEN** have no compiled representation at all, and so are not detected during decompilation.

2.2.5 *FORTH Decompiler Words*

Here are the definitions of the FORTH decompile words:

- **UN:** {*wordname*}. Decompile the link field, name field, and code field of a dictionary word identified by *wordname*. For secondaries, decompile the parameter field, stopping at semicolon. For vocabularies, display the vocabulary linking parameters. For other words, report the address of the end of the word, i.e., the link field of the next word in the dictionary.

- **UN:A** [*addr* ->]. Decompile part of the parameter field of a secondary, starting at a specified address. If *addr* does not contain a valid code field address, return the error message **Invalid Address**.

- **UN:C** [*cfa* ->]. Decompile the code field of a dictionary word identified by its *cfa*. For secondaries, also decompile the parameter field, stopping at semicolon.

To illustrate the output of the decompiler words, consider a sample word compiled as follows:

```
: WORD IF ." YES" ELSE " NO" THEN 12345 0 DO LEAVE LOOP ;
```

The command **UN: WORD** would then produce this output:

```
LFA: 33690  Link: 3367C
NFA: 33695  84 WORD
CFA: 3369F  E77B0 :
 336A4  E6628 IF to 336C5
 336AE  E0FAF ." YES"
 336BB  E663B ELSE to 336D2
 336C5  E069B " NO"
 336D2  E2083 12345
 336DC  E02EB 0
 336E1  E2096 DO
 336E6  E0AA5 R>
 336EB  E0AA5 R>
 336F0  E3FA7 2DROP
 336F5  E663B ELSE to 33709
 336FF  E4925 LOOP to 336E6
 33709  E797E ;
```

(If we had executed **' WORD UN:C**, we would have the same output, starting with the CFA line. Or to decompile, for example, only from 336E6 to 33709, we could have executed **336E6 UN:A**.) The starting address 33690 is arbitrarily chosen for the example. The following points illustrate various features of the decompiler:

- **IF to 336C5** is the decompilation of the consecutive 5-nibble quantities **E6628 0001C** compiled at address **336A4**. The former is the execution address of **IF**; the latter is the offset to the **THEN** (measured from the location of the offset).

- The literal **12345** is compiled as **E2083 12345**.

- The display string **." YES"**, shown as **E0FAF ." YES"**, is compiled as **E0FAF** followed by the counted string: 1 byte (3) for the length then 3 character bytes.

- The string **" NO"** is compiled as **E069B** followed by 1 byte for maximum length (2), 1 byte for current length (2), and 3 character bytes. Note that this is a normal string variable format, so it is possible to use string words like **S!**, **S<&**, and **S>&** to change the contents of the compiled string.

- **LEAVE** is compiled as **R> R> 2DROP ELSE**. The decompilation of **ELSE** shows the jump will be to **33709**, to the next word after **LOOP**.

- **LOOP to 336E6** shows that the jump caused by **LOOP** will be to **336E6**, the first word after the **DO**.

If you have a printer available, you can direct the output of the decompiler words to the printer using **PRINT** (see Section 1.5), as in

```
PRINT UN: WORD
336E6 PRINT UN:A
```

3. TRACED EXECUTION OF PROGRAMS AND WORDS

The Translator Pac was designed primarily for execution of existing HP-41 programs, not for HP-41 program development. It does not provide any means of tracing program execution, or any other form of debugging tools. Similarly, the underlying FORTH system does not include any such tools; you can find coding errors only through trial-and-error execution, or source code review.

The TPPT supplies a set of words that enable "traced execution" of both FORTH words and HP-41 programs. Before explaining the methods of traced execution in the following sections, we need to introduce several definitions:

- *Traced execution* --a collective term meaning either single-step or break-step execution, where execution of compiled words and programs can begin and end at user-specified points in the code that differ from normal FORTH and HP-41 entry and exit points.
- *Test-word* --the FORTH secondary or HP41 program whose operation we are probing with traced execution.
- *Single-step* --execute the next word in the definition of a test-word.
- *Break-step* --execute continuously one or more words in the definition of a test-word, up to a breakpoint or to the end of the test-word.
- *Breakpoint* --the address at which execution of a test-word is to halt.
- *Next-word* --the "next" word to be executed when traced execution is resumed. The *next-word pointer* is a stored variable that indicates where traced execution is to resume.
- *Trace-word* --the FORTH word, typically a display word, that is automatically executed at each halt in traced execution.

3.1 Setting the Trace Word

All of the traced execution words complete their operation by executing (unless instructed not to) a *trace-word* that you can specify. The trace-word is usually a word that displays data or status information. The trace-word thus provides you with an automatic means of checking the progress of the traced execution of a test-word.

You can select any FORTH word to be the trace-word by using the TPPT word **TRACE** {*trace-word*}. For example, if you wish to monitor the X-register during traced execution, you can type

TRACE F. [ENDLINE]

When you first load the TPPT FTH41RAM file, the trace-word is **S.**, which is a TPPT word that displays the contents of the FORTH data stack.

Special cases of **TRACE** are:

TRACE XONLY --display the X-register after each step.

TRACE A,X --make **A/X** the trace-word.

TRACE STACK --make **ST.** the trace-word.

The word **TRACEOFF** suppresses execution of the trace-word; **TRACEON** restores it.

The traced execution words **STEP** and **SST** precede a single-step execution by displaying the word (or program step) about to be executed. **TRACEOFF** suppresses this preliminary display as well as the trace-word. If you want to display the word to be executed, but do not wish the trace-word to follow after each step, you can execute **TRACE NOP TRACEON.** This situation is often desirable when you are working in the HP-41 environment, where the outer interpreter also executes a display word.

[Note to FORTH programmers: The trace-word is identified by the cfa stored in a TPPT variable named **TRACEWORD.** That is, each traced execution word ends with the sequence **TRACEWORD @ EXECUTE.**]

3.2 Traced Execution of HP-41 Programs

Four words are provided in the TPPT for traced execution of HP-41 programs: **SST** (Single-STep), **SSTA** (Single-STep Alpha), **TRUN** {*breakpoint*} (Traced-RUN), and **TRUNA** {*breakpoint*} (Traced-RUN Alpha). **SSTA** and **TRUNA** are identical to the corresponding shorter-named words except that they begin by setting HP-41 ALPHA mode. So we will concentrate on the description of **SST** and **TRUN.**

For sake of illustration, we will use the following HP-41 program:

```
01 LBL"TEST"
02 FIX 3
03 1.005
04 STO 01
```

```
05 LBL 01
06 RCL 01
07 INT
08 "LATEST="
09 ARCL X
10 ISG 01
11 GTO 01
12 END
```

After translating and loading this program, we execute **PRP TEST** which gives us a listing with addresses replacing line numbers:

```
*LBL"TEST"
212320 FIX 3
212330 1.005
212351 STO 1
*LBL 1
212361 RCL 1
212371 INT
212376 A" LATEST="
212397 ARCL X
212407 AVIEW
212412 ISG 1
212432 GTO 1
212442 END to 212320
```

We will refer to this listing in the next sections. Remember that the addresses shown are typical; if you enter ths program into your HP-71, it will most likely start at a different address than that shown.

3.2.1 GTO. and PP

To support the HP-41 traced execution words, the TPPT includes the HP-41 keyboard function **GTO.** {*address*}, and a new function, **PP**. **GTO.** allows you to set the HP-41 program pointer to any point in a program, where *address* plays the role of the program line number. You can use **PRP** or **LIST** to list the program (test-word) with its address/line numbers.

PP displays the current value of the program pointer, which allows you to determine where program execution will begin when you press **[RUN]** or use any of the traced execution words.

3.2.2 *Single-Stepping*

SST operates in a manner very similar to its HP-41 counterpart. That is, **SST** executes the program line indicated by the program pointer, and advances the program pointer to the next line. If you have executed **TRACEON**, **SST** will also display the line before it is executed, the analog of holding the **[SST]** key down momentarily on the HP-41.

There are two separate displays that follow **SST**. First, after **SST** is executed, the current trace-word is executed (assuming that you have used **TRACEON**). Then, when any remaining words that follow **SST** in the command line have been executed, the normal HP-41 display word is executed. You can choose any words to be the trace-word and display word, (since both are FORTH vectored execution words):

- To set the trace-word, use **TRACE** {*trace-word*}.
- To suppress the trace-word display, while preserving the program line display, use **TRACE NOP**.
- To set the HP-41 display word, use

  ' *display-word* **195822 FTOI !**,

  or execute one of the built-in words **XONLY**, **STACK**, or **A,X**.

Let's try single-stepping the sample program. For the example, execute **TRACE NOP**, to suppress any trace-word display. Then put the program pointer at the start of the program with either **GTO"TEST"** or **GTO. 212320**. Then, the first **SST** gives the display

```
FIX 3          1.234
```

where 1.234 is the number that happened to be in the X-register when we started. Notice that although the program pointer was apparently pointing to the **LBL"TEST"** shown in the program listing, the label is not displayed when we single-step. This is because program labels are not stored with the program (the program listing functions show the labels just for completeness in the listings).

Another **SST** yields:

```
1.005          1.005
```

The first **1.005** is the program line; the second is the display of the resulting contents of the X-register. A series of **SST**'s gives these displays:

```
STO 1           1.005
RCL 1           1.005
          (Again, the LBL 1 is not displayed.)
INT             1.000
A" LATEST="     1.000
ARCL X          1.000
AVIEW           LATEST=1.000
```

AVIEW sets the HP-41 message flag (flag 50), which prevents the normal X-register display. If we were using a trace-word, its action would also be suppressed by the message flag.

Two more **SST**'s:

```
ISG 1           1.000
GTO 1           1.000
```

This brings us back to 212361, the first line after the **LBL 1**. In this example, we are assuming that the program has not been run before--the **GTO** is still compiled with the label number rather than the label address. If we single-step until we reach the **GTO** line again, the display will show:

```
GTO 1 at 212361 2.000
```

The **at 212361** indicates that the program now contains an label address rather than the label number, to reduce execution time. **SST** retrieves the label number for clarity in the single-step display.

If you single-step the HP-41 function **AON**, ALPHA mode will be activated, and the trace-word and HP-41 display word will not be executed. The display will show the current ALPHA register contents, and the ALPHA keys will be active. To single-step further, however, you will have to exit ALPHA mode so that you can type **SST [ENDLINE]**. But if the program requires ALPHA mode to be active while it is running, to give the correct results (it might, for example, make a program branch dependent on the value of the alpha flag 48), substitute **SSTA** for **SST**. **SSTA** turns on alpha mode, then executes ordinary **SST**.

The functions **CLXR, ENTER^R, S+R,** and **S-R,** are peculiar functions used by **TRANS41** to replace occurrences of **CLX, ENTER^, S+,** and **S-**, respectively, that follow **RCL** *stack* in a program (see Section 2.1.1 and Table I). *Stack* represents any of the stack register names (**IND**) **X, Y, Z, T,** or **L**. The functions are peculiar in that when they are executed, they automatically perform the subsequent **RCL**, then skip to the program line after the **RCL**. Although this effect is unnoticeable in normal program execution, you should be aware that single-stepping any of these functions will also perform the **RCL** in the same step, leaving the program pointer at the program line following the **RCL**.

3.2.3 *Break-step Execution with* **TRUN**.

TRUN {*breakpoint*} provides the analog of HP-41 operation with a printer set to TRACE mode. That is, you can run a program continuously, while obtaining a displayed or printed record of the effect of each program line as it executes. With **TRUN**, you can (optionally) specify a breakpoint, so that the program will halt at the specified line, which does not have to be a normal program halt word. You can duplicate all display output from **TRUN** on an HP-IL printer by preceding each use of **TRUN** (or **TRUNA**) with **PRINT**.

If **TRUN** halts at a breakpoint when alpha mode is active, the traceword display will supercede the normal alpha register display. Hit **[<-]** (back-arrow) to restore the alpha display. If you wish then to resume traced execution, execute **TRUNA**, which sets ALPHA mode then proceeds with ordinary **TRUN**.

If you have executed **TRACEON**, **TRUN** will cause each program line, and its corresponding trace-word output, to be displayed as it is executed. The duration of each display is determined by the value of *time* set by the most recent use of **PAUSE** {*time*}. If you press a key during the paused display (see Section 1.4), execution will be suspended and you can scroll the display using the left- and right-cursor keys. Any other key resumes normal execution of **TRUN**, except the down-cursor key, which causes **TRUN** to suspend after the next program line. The down-cursor then acts, in effect, as an **SST** key.

To obtain the same displayed results from our sample program as we did in the preceding section using **SST**, we can use

```
GTO"TEST" TRACEON TRACE XONLY TRUN.
```

Or we might want to halt just after **LBL 1** on each pass through the loop, without seeing each program line displayed:

```
GTO"TEST TRACEOFF TRUN 212361
```

With repeated use of **TRUN**, we see:

```
1.005
LATEST=1.000
1.000
LATEST=2.000
2.000
LATEST=3.000
3.000
LATEST=4.000
```

```
4.000
LATEST=5.000
5.000
```

3.3 **Traced Execution of FORTH Secondaries.**

As defined at the beginning of Section 3, traced execution implies executing a portion of the compiled definition of a FORTH secondary word. When you execute a secondary normally, the inner interpreter pointer is placed at the start of the word's parameter field. The interpreter then proceeds through the parameter field, usually eventually encountering the semicolon at the end of the parameter field, which ends execution of the word. The word, of course, may include other secondaries in its definition, so that the interpreter pointer threads its way up and down through various nested "levels" of execution. By "level" we mean a series of compiled word addresses that are executed at a constant return stack depth. To go "up" a level means to remove one address from the return stack; "down" a level means to add one address to the return stack. We will use the term "top-level" to refer to the actual contents of a secondary's parameter field, which correspond to the original source code definition of the word.

With traced execution, you can terminate execution of a test-word at points in its definition other than normal FORTH exits like semicolon. When traced execution halts, it saves the value of interpreter pointer, which indicates the address of the "next" word to execute, as the next-word pointer, so that you can resume execution of the test-word at a later time. It also saves the current return stack depth (both are saved within the variable **ISST**).

FORTH traced execution words are of two types: *begin* words and *continue* words. The begin words are **BREAK** and **STEP**; one of these must be used to start traced execution of a test-word. Continue words (**SST**, **SSTU**, **SSTD**, **SSTO**, **CONT**, **FINISH**) are used to continue traced execution of the test-word after **BREAK** or **STEP** has halted.

3.3.1 *Begin Words*

Traced execution of a FORTH secondary is initiated by either of the TPPT words **BREAK** {*test-word*} {*address*} or **STEP** {*test-word*}, where *test-word* is a secondary. Both words create a special environment (within the dictionary variable **ISST**) for traced execution, which contains a new FORTH return stack, the next-word pointer, and additional pointers. The alternate return stack used during traced execution allows you to single step words like **DO**, **R>**, etc, that put data other than return addresses on the return stack. If this data were left on the normal return stack, a system crash would result when traced execution halts.

After creating the trace environment, **STEP** single-steps the first word in the test-word's parameter field. Once you have executed **STEP**, you can use any of the continue words to further execution through the test-word.

BREAK {*word*} {*address*} also starts traced execution of a test-word at the first word in the test-word's parameter field. But rather than halting after the first word is finished, **BREAK** continues executing through the test-word until the (inner) interpreter pointer reaches *address*, or the end of the test-word. When the traced execution halts at the breakpoint, the trace-word is executed. Then you can use any of the continue words for further traced execution of the same test-word. You can omit the *address* entry following **BREAK**, as long as you follow *word* with **[ENDLINE]**. If you don't specify a breakpoint, **BREAK** will halt at the previous value for the breakpoint, or at the end of the test-word.

BREAK tests *address* to see if it is a valid breakpoint, i.e., that its contents are the code field address of a FORTH word. If not, **BREAK** aborts with the message **Invalid Address**. The breakpoint does not have to fall within the parameter field of the test-word. You can specify a breakpoint within any secondary that is executed during execution of the test-word. Then, for example, you can single step through a word that has been called by another, with the stacks set appropriately.

The breakpoint address is saved in the TPPT variable **BP**. You can also set the breakpoint by typing

address **BP ! [ENDLINE]**.

3.3.2 *Continue Words*

Continue words are intended to resume traced execution of a test-word that has been initiated by **BREAK** or **STEP**. Each of the words resumes at the word indicated by the next-word pointer. The various continue words differ from each other in the criteria they use to determine where to halt next. "Single-step" words check the return stack depth after each execution of the inner interpreter, and halt when that depth reaches a particular level. "Break-step" words halt when the intepreter pointer reaches the breakpoint, or the end of the test-word (recognized as a semicolon executed at the return stack level corresponding to the top level of the test-word), whichever comes first.

The continue words, with their halt criteria, are:

- **SST** (Single step): Halt when the return stack level equals the level saved when traced execution last halted. Repeated use of **SST** effectively single-steps through a definition at a constant level. Note: when a word like **R>** or **DO**, that affects the return stack depth, is single-stepped, the saved depth is incremented or decremented prior to the actual single step. This preserves the sense of single stepping through a word's definition essentially in the order you would expect from the original source code.

- **SSTD** (Single step down): Assuming that the next-word is a secondary, execute only the first word in the definition of the next-word. Then subsequent **SST**'s will step through the remainder of the definition, one return stack level deeper than prior to the **SSTD**. If the next-word is not a secondary, display the reminder message **(Primary)** and then execute a normal **SST**.

- **SSTU** (Single step up): If the next-word is in the top level definition of the test-word, display the reminder message **(Top)**, then do a normal **SST**. But if the next-word is down one or more levels, in the definition of a secondary executed by the test-word, complete execution of that secondary and return to the word that called the secondary for subsequent single-stepping.

- **SST0** (Single step to level 0): Complete execution of all secondaries below the top-level of the test-word, so that subsequent **SST**'s will step through the remainder of the test-word's top-level definition.

- **CONT** {*address*} (Continue): Resume continuous traced execution, halting at the breakpoint *address*. If *address* is omitted, use the current breakpoint.

- **FINISH**: Complete execution of the test-word through its final semicolon. **FINISH** sets the current breakpoint to address 0, which is equivalent to no breakpoint, since the inner interpreter should never reach address 0.

To illustrate the operation of the traced execution words, let us create the following sample words:

```
: WORD2.2 WORD2.2.1 WORD2.2.2 WORD 2.2.3 ;

: WORD2 WORD2.1 WORD2.2 WORD2.3 ;

: TEST WORD1 WORD2 WORD3 ;
```

UN: would then decompile these words as follows:

```
LFA : 33E7E Link: 33E68 to WORD3
NFA: 33E83 84 TEST
CFA: 33E8D E77B0 :
33E92 33E2F WORD1
33E97 3EE4A WORD2
33E9C 33E74 WORD3
33EA1 E797E ;

LFA: 33E39 Link: 33E23 to WORD1
NFA: 33E3E 85 WORD2
CFA 33E4A E77B0 :
33E4F 33DC7 WORD2.1
33E54 33DE6 WORD2.2
33E59 33E14 WORD2.3
33E5E E797E ;

LFA: 33DD1 Link: 33DB7 to WORD2.1
NFA: 33DD6 87 WORD2.2
CFA: 33DE6 E77B0 :
33DEB 33D43 WORD2.2.1
33DF0 33D66 WORD2.2.2
33DF5 33D89 WORD2.2.3
33DFA E797E ;
```

Here, of course, the addresses (shown in hexadecimal) are typical, but will vary according to actual memory contents.

We will take **TEST** as our test-word. Before single-stepping, we will execute

TRACE S. TRACEON 1 OKFLG !

to make **S.** the active trace-word, and to suppress the **OK** { *n* } message. **S.** displays the current stack contents between square brackets [ ]. Since our sample words don't have a specified effect on the stack, we will use the notation [*word*] to represent the stack contents after *word* is executed.

3.3.3 *Single-Step Examples*

To single-step **TEST**, we start by executing

STEP TEST

The HP-71 responds with the display

WORD1 [*WORD1*]

We have executed **WORD1**, the first word in **TEST**'s parameter field, leaving the quantities represented by *WORD1* on the stack. The next-word pointer has the value 33E97 (which you can check by executing **ISST ?**), which points to the second entry in **TEST**'s definition.

If we now execute **SST**, the interpreter runs until the return stack is at the same level as it was when **WORD1** was executed. That is, all of **WORD2** is executed, making the display

```
WORD2            [ WORD2 ]
```

Two more **SST**'s then yield the displays

```
WORD3            [ WORD3 ]

           ; Word End
```

The last display indicates that we have come to a semicolon at the current return level, which in this case is the end of the test-word. Any additional **SST**'s will just repeat the **; Word End** message.

Let's suppose now that instead of single-stepping through **TEST** at its "top" level, we wish to investigate the behavior of **WORD2**, that it exhibits when it is executed by **TEST**. Then rather than using **SST** after the initial **STEP TEST**, we can execute **SSTD, which gives the display**

```
WORD2            [ WORD1 ]
```

Notice that **WORD1's** output is still on the stack--we haven't actually begun to execute **WORD2** yet. The next-word pointer now has the value 33E4F, pointing to the first entry (**WORD2.1**) in **WORD2**'s parameter field. Now **SST** executes **WORD2.1**

```
WORD2.1          [ WORD2.1 ]
```

If we **SSTD** again:

```
WORD2.2          [ WORD2.1 ]
```

and the next-word pointer points to the start of the parameter field of **WORD2.2**, which in turn contains the word **WORD2.2.1**. Successive **SST**'s from here will work through **WORD2.2.1**, **WORD2.2.2**, and **WORD2.2.3**, resulting in the **Word End** message at the semicolon terminating **WORD2.2**. On the other hand, at any point during the single-stepping of **WORD2.2** (including when the next-word pointer indicates the final semicolon), we could execute **SSTU**. This would have the effect of decrementing the stored return level, so that execution will proceed through the completion of **WORD2.2**, leaving [*WORD2.2*] on the stack, and the next-word pointer at 33E59, at the **WORD2.3** entry in **WORD2**'s definition. Another **SSTU** completes execution of **WORD2**, returning back to the top level of **TEST**.

Another option is the use of **SSTO**, which resets the return level to the original test-word and thus completes execution of all nested secondaries at levels below the top level of the test-word.

Here's an example of single-stepping an actual word. **ROOM** is a (ROM dictionary) word that returns the number of nibbles currently available in the user dictionary. We start with **STEP ROOM**, then execute a series of **SST**'s to produce the outputs (**S.** is still the trace-word, and we are in hex mode):

```
              Display                          Remarks

SP0          [ 2FB11 ]          Address of stack bottom pointer
@            [ 3419E ]          Address of stack bottom
HERE         [ 3419E 33EDA ]    End of user dictionary
-            [ 2C4 ]            Dictionary space
1CA          [ 2C4 1CA ]        Unavailable space
-            [ FA ]             Net available space
; Word End
```

3.3.4 *Break-Step Examples*

In the preceding section, we looked at the process of single-stepping through a test-word, at various return stack levels. Now let's see how we can execute arbitrary portions of a word, without regard to return stack level. Suppose, for example, that we want to examine the stack as it appears during execution of our test-word **TEST**, at the point where **WORD2** has completed and **WORD3** is about to execute. We can use **BREAK** to halt execution at that point, without having to single-step from the start of **TEST**.

As described earlier, **BREAK** is used in the form **BREAK** {*word*} {*breakpoint*}. So we execute **BREAK TEST 33E9C**, which halts execution of **TEST** at **33E9C**, just prior to the **WORD3** entry in **TEST**'s definition. **BREAK** doesn't display the next-word name when traced execution begins (as the single-step words do), so we see only the display

[*WORD2*]

From here, we can use any of the single-step words to probe subsequent portions of the test-word. Or, we can use **CONT** {*breakpoint*} to resume break-step execution up to a new breakpoint. **FINISH** also resumes execution, but sets the breakpoint to address zero, which ensures that the remainder of the test-word will be executed up to its final semicolon.

Note that the breakpoint set with **BREAK** or **CONT** does not have to be in the top-level definition of the test-word. If we want, for example, to halt **WORD** at the **WORD2.2.2** entry in **WORD2.2**, we use **BREAK TEST 33DF0**. Then a subsequent **SST** will execute **WORD2.2.2**.

If you don't specify *{breakpoint}* with **CONT** (or with **BREAK**), the last breakpoint set remains active. Thus, for example, you can halt at the same point on successive passes through repeated code contained within a **DO...LOOP** construct, by specifying a breakpoint within the loop with **BREAK**, then using **CONT** each time you want to advance once through the loop.

Break-step execution runs at about 2/3 the speed of normal FORTH execution. The decrease in speed is due to the extra steps in the inner interpreter, which check for the breakpoint and for the end of the test word.

> Warning: *Do not attempt traced execution of any word that contains, at any level of definition, any of the dictionary resizing words* **GROW, SHRINK, DSIZE,** *or* **XSIZE.** When the dictionary size is changed by these words, the normal return stack area moves. If this occurs during traced execution, the stored values of the return stack pointers, which are needed for restoration of normal execution, will be invalid--**Memory Lost** is the inevitable result. Since the resizing words are primarily used from the keyboard (no other built-in word calls any of the four), this is not a serious limitation on the utility of traced execution.

3.4 Timing Execution

You can determine the execution time of any FORTH word or HP-41 program by using the TPPT word **TIMED** *{word}*. **TIMED** reads the HP-71 system clock before and after the execution of *word*, then subtracts the two clock readings and displays the difference, in seconds. (The result remains in the X-register.) The result is accurate to about 0.01 seconds, which is the accuracy with which the system clock can be read.

If you are timing an HP-41 program, or any FORTH word that uses the floating-point stack, you should be aware that **TIMED** lifts, then drops the stack once on input, so that the contents of the T-register are lost before *word* begins to execute. Similarly, when *word* is done, **TIMED** lifts the stack twice and drops it once, so that the X-, Y-, and Z-register outputs of *word* are raised to the Y-, Z-, and T-registers, respectively, and any T-register output is lost.

Examples:

FORTH:

```
DECIMAL : EMPTYLOOP 1000 0 DO LOOP ;
```

TIMED EMPTYLOOP gives the result **0.59** seconds.

HP41:

```
GTO"TEST" TIMED RUN,
```

where **TEST** is the example program used in the previous section, shows the result 1.15 seconds.

4. EXTENDING THE HP-41 LANGUAGE

(In this section, it will be presumed that you have a working knowledge of FORTH programming.)

An important advantage of the Translator Pac's use of a FORTH language system to implement HP-41 emulation on the HP-71 is that you can add new HP-41 functions to the emulator by writing them in FORTH. Once an new function has been added to the HP-41 function set, it is indistinguishable in execution style from any of the built-in functions. No analog to **XEQ""** is required to use a new function.

As described in the Translator Pac Owner's Manual, pages 47-49, any number of postfix HP-41 functions, or FORTH words, can be combined into new functions using a standard FORTH secondary colon definition:

: *NAME word1 word2 ... wordN ;*

Here *word1 ... wordn*, and hence *NAME*, are presumed to be "simple" postfix functions (i.e., they do not take a register or label argument as do the prefix functions like **RCL** *n* or **LBL** *n*). There are restrictions on the characters in *NAME* that arise from conventions expected by the translator program **TRANS41:**

- *NAME* can't contain spaces, quotes, or question marks, which **TRANS41** assumes to indicate prefix functions, text functions, and conditionals, respectively.
- If the first character of *NAME* is ^, **TRANS41** assumes that execution of *NAME* lifts the floating-point stack.
- *NAME* can't start with **E+, E-, E** followed by a number digit, **ISG** , or **DSE.**
- *NAME* should not end with **"**, which identifies a standard HP-41 text function.

The reason that postfix functions should not generally be included in new function definitions is that the resulting functions will not execute properly from the keyboard. All of the Translator Pac's built-in prefix functions check the program running flag 52 to determine whether to obtain their arguments from the keyboard (flag 52 clear) or from the compiled program (flag 52 set). Suppose we want to create a new function ^RCL1 that combines **RCL 1** into a single word:

```
: ^RCL1 RCL 1 ;
```

If ^**RCL1** is compiled into a program it will work correctly when the program is run. But if you use it from the keyboard, the **RCL** takes the next string from the keyboard as the register number. If that doesn't cause an error (i.e., the next string is a valid register number), the interpreter pointer then points at the compiled register number (1) following the **RCL**. What happens next depends on the contents of memory address 00001--in this case there is no apparent effect, but for arbitrary register numbers, anything, up to memory loss, can result.

4.1 Vocabularies

As described in the Translator Pac manual, the FTH41RAM file is (initially) organized into two vocabularies. The **FORTH** vocabulary contains the built-in FORTH system words, and any HP-41 words that can be considered as ordinary FORTH words, i.e., they are postfix words that do not depend on any special memory structures other than the stacks. The **HP41V** vocabulary contains the remaining HP-41 words, especially the prefix words, and any user programs.

The **HP41V** vocabulary word is contained in the **FORTH** vocabulary, so that most built-in FORTH words are available even when **HP41V** is the context vocabulary. The only **FORTH** words that are not available are the words that have identically-named entries in the **HP41V** vocabulary (like **STO, RCL, 0, 1, 2, 3, X,** etc.) that are found first in a dictionary search.

When you create new HP-41 words, you should enter them into the **HP41V** vocabulary by executing **HP41V DEFINITIONS** before defining the new word. However, matters are complicated by the duplication of word names in the two vocabularies, so that you may have to change context after the word definition has begun to insure that the desired version of a duplicated word is compiled.

Even changing contexts requires care. In the **HP41V** vocabulary, there is a version of the word **FORTH** that is intended for use to exit the HP-41 environment. Among other things, this version of **FORTH**, which is not an immediate word, executes **FORTH DEFINITIONS**, where the latter **FORTH** is the original vocabulary word. You should not execute the **HP41V** version of **FORTH** during compilation of an **HP41V** vocabulary word, since an error during compilation will then leave the dictionary in a corrupt state. For this reason, the TPPT includes the word **FORTHV**, which is immediate, that you can use to set the context to **FORTH** during compilation. For example,

```
HP41V DEFINITIONS : ANUM0 FORTHV 0 HP41V ANUM ;
```

compiles an **HP41V** word that puts 0 on the data stack, then executes **ANUM**.

In the following sections, we will describe how to create new prefix words and other types of HP41 functions. But first, we will introduce a new TPPT word, **HEAD**, that provides a convenient, code efficient method of including headerless Translator Pac words in your new function definitions.

4.2 Headerless Words and HEAD.

There is a large number of "headerless" utility words contained in the Translator Pac built-in dictionary. That is, the link fields and name fields of these words are absent, so that you can not include the words by name in secondary definitions. You can, assuming that you know the code field address *cfa* of a headerless word, include it in a definition with either

cfa **EXECUTE**

or

[*cfa* ,]

These constructions lead to rather cryptic source code, so if you use a particular headerless word often, it is convenient to give it a *name*:

: *name cfa* **EXECUTE** ;

or

: *name* [*cfa* ,] ;

The TPPT defining word **HEAD** provides a third method of assigning a name to a headerless word, that is more code efficient and faster executing than these two. **HEAD** uses the syntax

HEAD { *cfa name* }

HEAD creates a dictionary entry with *name* in the name field, and a parameter field consisting only of *cfa*. The code field points to a special prologue that adds one level of indirection to ordinary FORTH execution. That is, the prologue moves the CPU program pointer to the prologue of the headerless word, indicated by the contents of the *cfa* which itself is stored in the original word's parameter field.

For example, the headerless word that returns an HP-41 register address (after checking for its existence) has its code field at **EB177**.

HEAD EB177 NTHREG

creates the word **NTHREG** [*n* -> *addr*], which takes a register number *n* from the data stack and returns the memory address of the register (or the **Nonexistent** error if the *n*th register does not exist.) The stack registers L, X, Y, Z, and T correspond to *n*= 10000, 10001, 10002, 10003, and 10004 (decimal), respectively. *n*<0 indicates indirect register access, i.e., the *addr* returned is the address of the register whose number is stored in register *n-1*.

By using **NTHREG**, we can create the **^RCL1** function described at the start of Section 4:

: ^RCL1 FORTHV 1 HP41V NTHREG FORTHV RCL ;

We have to go back and forth between context vocabularies, using **FORTHV** and **HP41V**, because the versions of **1** and **RCL** we want are in the **FORTH** vocabulary, whereas **NTHREG** should be in the **HP41V** vocabulary.

4.3 Creating New Prefix Register Functions.

All Translator Pac prefix register functions are defined according to this model:

: *name* **GETNUM NTHREG ... ; IMMEDIATE**

where **NTHREG** is the headerless word described in the preceding section. The ... indicates the remainder of the definition; the key point is that all the prefix register functions are immediate secondary words whose definitions start with the two words **GETNUM NTHREG**. **GETNUM** is another headerless word:

HEAD EBCE1 GETNUM

For example, the (HP-41) RCL is defined:

: RCL GETNUM NTHREG FORTHV RCL ; IMMEDIATE

where the **RCL** inside the definition is the postfix version contained in the **FORTH** vocabulary, that takes an address from the data stack and pushes the 8 byte quantity stored at that address onto the floating-point stack.

GETNUM is the key to all prefix functions. It executes as follows:

- In program running mode (flag 52 set), return to the data stack the number from the program address on the top of the return stack, and increment that return address by 5.
- In execute mode (flag 52 clear), get the next string delimited by spaces from the keyboard.
    - If the string is null, drop one address from the return stack (to terminate execution of the word that called **GETNUM**) and exit.
    - If the string is not null (if it is "IND", set a flag to indicate indirection, and get the next string from the keyboard):
        - If it is a letter L, X, Y, Z, or T, return to the data stack the register number 10000, 10001, 10002, 10003, or 10004 (decimal), respectively.
        - If it is a letter in the range A - J or a - e, return the register number

          10000 + ASCII value of letter
        - If it is not a letter L, X, Y, Z, T, A - J, or a - e, use **NUMBER** to return the numeric value of the string (or error if not a number).
        - If the IND flag was set, negate the register number on the stack and subtract 1.
    - If compile mode is active, compile the word that called **GETNUM**, followed by the register number that is on the stack.

To create a new HP-41 prefix register function, you should follow the above model. Use **GETNUM NTHREG** to put the register address on the data stack, then any number of additional words to manipulate the data at that address. Remember that such prefix functions should only be used in HP-41 programs (at top level), not as arbitrary FORTH words.

Table IV lists code field addresses for some postfix register words included in the Translator Pac:

The headerless words at these addresses perform the same operation as their prefix counterparts, but use a register address already on the data stack rather than a register number from the keyboard or program. Note that named postfix versions of **STO** and **RCL** are present in the **FORTH** vocabulary.

Table IV. Postfix Register Words

Address	Prefix Equivalent
EA634	**ST***
EA538	**ST+**
EA60E	**ST-**
EA65A	**ST/**
EB6BD	**X<>**

For example, suppose we wish to define a function **Z<>** that works like the standard function **X<>**, except that data is exchanged between a number register and the Z-register instead of the X-register. Here is a suitable definition:

```
HEAD EB6BD POSTX<>

: Z<> GETNUM NTHREG FORTHV Z DUP
  HP41V POSTX<> SWAP DUP FORTHV X HP41V <>
  IF POSTX<> POSTX<>
  ELSE 2DROP
  THEN
; IMMEDIATE
```

where we have used **HEAD** to give the name **POSTX<>** to the headerless prefix version of **X<>**.

4.4 Creating New Conditional Functions

TRANS41 assumes that an HP-41 conditional function, which it recognizes by the presence of a **?** in the function name, puts a true/false flag on the data stack. **TRANS41** adds **XBR** *jump* immediately after the conditional function in the intermediate program file. The compiled form of **XBR** tests the flag, and advances the interpreter pointer by *jump* nibbles if the flag is false.

Any FORTH word you define to return a flag to the data stack can act as an HP-41 conditional, if you include a **?** in its name to signal **TRANS41**. To make a new conditional display **YES** or **NO** when executed from the keyboard, you can include the headerless word

```
HEAD E7DC9 TRUTH
```

TRUTH will remove a flag from the stack and display **YES** or **NO** if:

- The HP-41 environment is active, and
- The program running flag 52 is clear

Otherwise it does nothing, leaving the flag on the stack.

Example: Define a conditional function that tests whether the alpha register is empty:

: ALPHAEMPTY? 2FC82 C@ 0= TRUTH ;

2FC82 is the address of the count byte in the alpha register string variable at **2FC80**.

Table V lists TPPT flag words that you will find useful in writing new HP-41 functions. These words are present in the Translator Pac as headerless words; they are given names in the TPPT by use of **HEAD**.

Table V. TPPT Flag Words

Name	ROM CFA	Stack Use	Operation
CFL	**ED511**	[*n* ->]	Clear flag *n*
SFL	**E24B3**	[*n* ->]	Set flag *n*
FL?	**ED537**	[*n* -> *flag*]	Test flag *n*: return true if set, false if clear.

n positive means an HP-41 flag; *n* negative means an HP-71 system flag.

The flag words listed in Table V work on HP-71 system flags -64 through -1, and HP-41 flags 0 through 62. Note that HP-41 flags 0 through 7 are identical to HP-71 user flags 0 through 7.

4.5 Creating New Display Functions

You can create new HP-41 display functions that respect the print/halt conventions of **VIEW** and **AVIEW**, as controlled by the printer flags 21 and 55, by using the headerless Translator Pac word we will call ***VIEW**:

HEAD ED0D1 *VIEW

***VIEW** [*string* ->] takes a string from the data stack and **TYPE**s it to the display, setting the message flag 50. It will halt program execution or print to the current printer, according to flags 21 and 55. *Note: because* ***VIEW** *can alter the return stack, it should only be used in the top level definition of HP-41 functions that are intended for use in HP-41 programs.*

Another headerless word that is useful in conjunction with ***VIEW** is

HEAD EBDEF REGSTR$

REGSTR$ [*address* -> *string*] converts the floating-point number at *address* into a *string* (at the FORTH pad), according to the current floating-point display mode.

Example: Define **DISPX+2**, which displays and prints the contents of the X-register followed by the contents of Register 2.

```
: DISPX+2 FORTHV 2 X HP41V REGSTR$ TYPE
  SPACE NTHREG REGSTR$ *VIEW
;
```

If we replace the ***VIEW** with **TYPE**, we would then have a version of **DISPX+2** that can be used as an ordinary FORTH word at any level of definition.

4.6 Extending TRANS41

The translation program **TRANS41** is designed to allow translation of new functions as well as those included in the Translator Pac HP-41 function set. TRANS41 makes as few changes as possible to each line from a program-text file as it is moved into an intermediate file--most lines are copied unchanged. To determine whether a line needs modification, TRANS41 applies these criteria, in order (when any criterion is met, TRANS41 handles the line accordingly and skips the rest of the tests):

After removing comments enclosed in parentheses, and stripping leading and trailing spaces--

1. If the line ends with **"**, it is assumed to contain alpha text and is rewritten in keyboard form (no extraneous spaces) for **""**, **LBL""**, **XEQ""**, or **GTO""**.

2. If the line is a number entry line, it is rewritten into FORTH floating-point entry form.

3. If the line contains a **?** or begins with **ISG** or **DSE**, it is identified as a conditional, and **XBR** *jump* is added.

4. If the line contains a space, it is identified as a register function or a local label. The line is left unchanged, but the correct compiled line length is computed.

5. If the line is a stack lift disable function **ENTER^**, **CLX**, **Σ+**, or **Σ-**, it is rewritten according to whether the following function lifts the stack.

6. All other functions are left unchanged, and are presumed to be postfix functions of compiled length 5 nibbles. If the function name starts with the character ^, the function is presumed to lift the floating-point stack.

In most cases, therefore, when you add new functions to the HP-41 emulator vocabulary, you need only insure that the function is named appropriately for **TRANS41** to handle it properly. However, **TRANS41** does allow you to add any number of additional tests and changes to its its program line translation. Before processing any program line, **TRANS41** calls a BASIC subprogram named **SPEC41(A$,K)**. If **SPEC41** is not present, as might usually be the case for ordinary translation, control returns to **TRANS41** which then translates the line. If it is present, **SPEC41** can either translate the line itself, or return it to **TRANS41** for translation.

When **TRANS41** calls **SPEC41(A$,K)**, the input variables are:

A$ = current program line string

K = 0.

When **SPEC41** returns to **TRANS41**, the variables should be as follows:

A$ = (un)modified program line string

K = 0 if **SPEC41** does not modify this string

= -1 to cause **TRANS41** to halt with an error message

= *jump* if **SPEC41** does modify this string.

If **K** = *jump* <> 0, then *jump* is the length, in nibbles, of the compiled code corresponding to **A$**. In addition, if **K** <> 0, **SPEC41** should return with

- Flag 1 clear
- Flag 0 set if the function corresponding to **A$** will raise the floating-point stack, clear otherwise.

TRANS41 does not correctly translate HP-41 number entry lines with no exponent digits, i.e., of the form *m...m* **E**, where *m...m* are one or more mantissa digits. As an example of the use of **SPEC41**, here is a version that extends **TRANS41** to handle this type of program line:

```
10 SUB SPEC41(A$,K)
20 N=NUM(A$) @ L=LEN(A$)
30 IF A$[L]<>"E" OR NOT (N=45 OR N<58
   AND N>47) THEN END
40 A$=A$[1,L-2]
50 IF POS(A$,".")=0 THEN A$=A$&"."
60 K=21
70 CFLAG 1 @ SFLAG 0
80 END
```

Line 30 tests **A$** to see if it starts with a number or a **-** sign, and ends with **E** (i.e., no exponent digits). If not, **SPEC41** returns to **TRANS41**, which will process **A$**, since **K** will still be 0. Line 50 adds a decimal point if it is absent. Line 60 sets **K** = 21, the compiled length of a floating-point number. In line 70, we set flag 0, since number entry lines raise the floating-point stack.

4.7 Real-Time Creation of HP-41 Programs

Although use of the Translator Pac text editor and **TRANS41** is a general, safe method of creating HP-41 programs in the HP-71, that method can be a little tedious and slow if you just want to create a quick and dirty HP-41 test program. The TPPT words **PRGM** and **END** allow you to create HP-41 programs directly from the HP-41 environment, bypassing the translation stage. **PRGM** and **END** act as HP-41 analogs to the FORTH words **:** and **;**, respectively, which mark the beginning and end of FORTH secondary compilation.

The general format for use of these words is

PRGM {*word1 word2* ... *wordn*} **END**

where *word1 word 2* ... *wordn* is the sequence of HP-41 words that define your program. You can intersperse any number of **[ENDLINE]**'s among the words as you enter them; however, **[ENDLINE]** will not clear the display, but will only turn the cursor off. The next key you hit will clear the display as it begins a new line of input. If you accidentally press the **[RUN]** key during program entry, it will act as **[ENDLINE]** if you are in the first line of the program (i.e., **PRGM** is still in the edit line). In subsequent lines, **[RUN]** will have no effect.

The price you must pay for the convenience of omitting translation is that you must input the program words in a less flexible format than is permitted by **TRANS41**. Furthermore, you have to take over the dictionary management performed automatically by **TRANS41**--you will have to estimate the compiled size of the program, then use **XSIZE** to open enough room in the dictionary to hold the program and prevent the **Dictionary Full** error.

Here are the guidelines you must follow to include the various types of HP-41 words in a program:

- You should include at least one alpha label in your program for subsequent reference. **PRGM** automatically includes an invisible dummy label **LBL"@@"** in each program, which you can use with **GTO""**, **XEQ""**, or **CLP""** if you neglect to include your own label.
- **GTO"***label***"** and **XEQ"***label***"** must be entered with no spaces between the **GTO** or **XEQ** and the leading **"**.
- ALPHA text entered in the form **"***text***"** must not include a leading space in the text. (A **"** by itself would be interpreted as the FORTH **"**, so that the following text would be compiled as a FORTH string.) If your text has a leading space, use the form **A"** *text***"**
- Number entry lines must be entered in FORTH floating-point form. They must contain a decimal point, and no spaces between the last mantissa digit and the **E** of the exponent, if any.
- Register and local label functions must be followed by their register numbers (or **IND** *number*) on the same command line, without any intervening **[ENDLINE]**.
- Conditionals must be followed by the word **XBR** *jump*, where *jump* is the compiled length, in nibbles, of the next program line, plus 5 nibbles. Table VI lists the lengths of the various HP-41 function types.

To illustrate these guidelines, we will enter the following HP-41 program:

```
01 LBL"EXAMPLE"
02 FS? 10
03 GTO"BLAZES"
04 "TEXT"
05 XEQ"SPIES"
06 1123
07 RCL IND 99
08 PI
```

Table VI. Compiled Lengths of HP-41 Functions

Function	Length (nibbles)
Alpha labels	5
Local labels	5
END	5
1-byte functions	10
2-byte functions	15
Conditionals	25
GTO"*label***"**	10 + 2*(*n*+1)
XEQ"*label***"**	10 + 2*(*n*+1)
"*text***"**	10 + 2*(*n*+1)

Here *n* is the number of characters in *label* or *text*.

```
09 LBL B
10 END
```

We enter the program like this:

PRGM LBL"EXAMPLE" FS? 10 XBR 24 [ENDLINE]

GTO"BLAZES" "TEXT" XEQ"SPIES" 1123. RCL IND 99 [ENDLINE]

PI LBL B END [ENDLINE]

The placement of the **[ENDLINE]**'s is arbitrary, as long as they don't break up multi-part functions onto separate lines. Notice that **FS? 10** is entered followed by **XBR 24**. The **24** is derived from the following **GTO"BLAZES"**. The number *n* of alpha characters in "BLAZES" is 6, so 10 + 2*(n+1) = 24.

5. TPPT DICTIONARY

This section contains an alphabetical list of the words added to the FORTH dictionary by the TPPT. Sorted by category, the words are:

Input/Output	*Variables*	*Defining Words*
DSAVE	**BP**	**HEAD**
GTO.	**FSCRATCH** *	**PRGM**
PAUSE	**ISST**	**END**
PC	**PAUSELEN**	
PRINT	**TEMP$** *	
S. *	**TRACEWORD**	
STACKS *		
TRACE		
TRACEOFF		
TRACEON		
VLIST *		
WAIT		

FORTH Decompile	*HP-41 Program Listing*	*HP-41 Flags*
'NAME$ *	**ALBLS**	**CFL**
BREAK	**LBLS**	**FL?**
CONT	**LIST**	**SFL**
FINISH	**LISTN**	
NAME *	**PRP**	
NAME$ *	**SST**	
SST	**SSTA**	
SSTO	**TRUN**	
SSTD	**TRUNA**	
SSTU		
UN:		
UN:A		
UN:C		

Words marked with an asterisk * are TPPT utility words included in this dictionary, that are not described anywhere else in this manual.

Postfix words are listed in the format

Name [*stack* -> *stack'*]

where *stack* represents the arguments *Name* takes from the FORTH integer data stack, and *stack'* represents the items returns to the stack by *Word.* All stack arguments and returned stack values for TPPT words are single length integers, except those items identified by *string*, which indicates a FORTH string (address and character count).

Prefix words that expect following input from the keyboard are listed in the format

Word { inputs }

where *inputs* is one or more items that *Word* takes from the keyboard.

'NAME$ [*cfa* -> *name-string*]

Return a string containing the name of the dictionary entry identified by the code field address *cfa*. If there is no valid name field preceding *cfa*, return the string "**Headerless.**" If the identified code field contains 00000, return the string "**Empty.**"

ALBLS [->]

Display each HP-41 alpha label and its program address. Each display has the form

LBL"*text***"** *address*

Successive displays are separated by **WAIT**. Section 2.1.2.

BP [-> *address*]

Return the *address* of the variable containing the current breakpoint address. Section 3.3.1.

BREAK { *word* *breakpoint* }

Start traced execution of *word*, and continue until the inner interpreter reaches the *breakpoint* address, or the end of *word*. If no *breakpoint* is specified, use the current breakpoint. Section 3.3.1.

CFL [*number* ->]

Clear flag *number*. A positive *number* identifies an HP-41 flag; a negative *number* refers to an HP-71 system flag. Section 4.4.

CONT { *breakpoint* }

Continue traced-execution, starting at the address stored in the next-word pointer. Halt at the *breakpoint* address, or the end of the test-word, whichever comes first. If no *breakpoint* is specified, use the current breakpoint. Section 3.3.2.

DBACK

Restore the previous **DISPLAY IS** device. **DBACK** should only be used when preceded by **PRINT** { *word* }, where *word* has overwritten the **ONERR** entry used by **PRINT**, and an error has occurred. **DBACK** can be included in *word*'s error-handling word. Section 1.5.

END

In compile mode, terminate compilation of an HP-41 program. When not in compile mode, act as a normal HP-41 **END**. Section 4.7.

FINISH

Continue traced execution, starting at the address stored in the next-word pointer, and halting at the end of the test-word. Section 3.3.2.

FL? [*number* -> *flag*]

Test flag *number*. Return *true* (-1) if flag *number* is set; return *false* (0) otherwise. A positive *number* identifies an HP-41 flag; negative *number* refers to an HP-71 system flag. Section 4.4.

FSCRATCH [-> *address*]

Return the *address* of a floating point variable used by **TIMED** for temporary storage.

GTO. { *address* }

Place the HP-41 program pointer at *address*. If the contents of *address* are not a valid code field address of a valid FORTH word, return the **Invalid Address** message, and leave the program pointer unchanged. Section 3.2.1.

HEAD { *address name* }

Create a dictionary entry with *name* as the name field. When *name* is executed, the headerless word with its code field at *address* is executed. Section 4.2.

ISST [-> *address*]

Return the parameter field *address* of a variable containing the traced-execution pointers and return stack. The first entry at *address* is the next-word pointer. **ISST @** returns the value of the next-word pointer to the data stack. Section 3.3.1.

LBLS

Display the current HP-41 local labels and **END**'s. Each entry is displayed in the format

```
LBL number  address
        or
END         address
```

where *number* is the label number or letter, and *address* is the program address of the label or **END**. Entries are listed in order of increasing program address. Section 2.1.2.

LIST { *number* }

List an HP-41 program, starting at the current HP-41 program pointer. List *number* lines, or to the program **END**, whichever comes first. The program pointer advances to the next line after the last line listed. **LIST 0** lists from the program pointer to the end of the program. See also **LISTN**. Section 2.1.

LISTN [*number* ->]

List an HP-41 program, starting at the current HP-41 program pointer. List *number* lines, or to the program **END**, whichever comes first. The program pointer advances to the next line after the last line listed. **0 LISTN** lists from the program pointer to the end of the program. See also **LIST**. Section 2.1.

NAME [*address* ->]

Display the characters in the word name field at *address*. **LATEST NAME**, for example, displays the name of the most recent word created in the current vocabulary.

NAME$ [*address* -> *string*]

Return a string containing the characters in the word name field at *address*.

PAUSE { *time* }

Set *time* as the delay produced by **WAIT**, where *time* is expressed in milliseconds. Section 1.4.

PAUSELEN [-> *address*]

Return the address of the variable PAUSELEN, that holds the number of milliseconds that **WAIT** will pause execution. If **PAUSELEN** contains a negative value, **WAIT** will suspend execution.

PP

Display the current contents of the HP-41 program pointer variable at **2FC48**. Section 4.7.

PRGM

PRGM is used in the format

PRGM *word1 word2* ... *wordn* **END**

to define an HP-41 program containing the words *word1 word2* ... *wordn*. Section 4.7.

PRINT { *word* }

During execution of *word*, make the current **PRINTER IS** device the **DISPLAY IS** device, so that any display output from *word* is printed. Restore the original **DISPLAY IS** device when *word* is finished. Section 1.5.

PRP { *label* }

List the HP-41 program containing the designated alpha *label* from the first line of the program through the **END**. **A "** must follow the label name if any additional keyboard input precedes the next **[ENDLINE]**. Section 2.1.

S.

Display the current contents of the data stack, in the format

[*itemn* ... *item2 item1* **]**

where item1 is on the top of the stack (most recently entered).

SFL [*number* ->]

Set flag *number*. A positive *number* identifies an HP-41 flag; negative *number* refers to an HP-71 system flag. Section 4.4.

SST

In the FORTH environment: Single step the current test-word, starting at the address indicated by the next-word pointer, and continuing until the return-stack level matches the level stored at the last halt in traced-execution. Section 3.3.2.

In the HP-41 environment: Single step the HP-41 program line indicated by the HP-41 program pointer. Section 3.2.2.

SST0

Resume traced execution of the current test-word, starting at the value of the next-word pointer and halting when the return stack level corresponds to top-level execution of the test-word. Section 3.3.2.

SSTA

Set HP-41 alpha mode flag 48, then execute **SST**. Section 3.2.2.

SSTD

Resume traced-execution of the current test-word. If the next-word pointer indicates a secondary, halt with the next-word pointer pointing at the first word in the parameter field of the secondary. Otherwise, execute **SST**. Section 3.3.2.

SSTU

If the next-word pointer is below the top level of the current test-word, complete execution at the current return-stack level, then halt with the stored return stack-level decreased by one. If execution is already at the top level, display **(Top)**, and do **SST** instead. Section3.3.2.

STACKS

Display the contents of the data stack and the floating point stack in the format

[*data stack contents* **]** **{** *floating-point stack contents* **}**

The *data stack contents* are displayed in the same manner as by **S.**; the *floating-point stack contents* are displayed by the ROM word **ST.**.

TEMP$ [-> *string*]

Return the address and character count of the string contained in a the string variable **TEMP$**, which has a maximum length of 96 characters. **TEMP$** is used by the TPPT single-step and decompile words.

TRACE { *word* }

Make *word* the current trace-word to be executed at each halt in traced execution. Section 3.1.

TRACEOFF

Suspend automatic trace-word execution at traced-execution halts. Section 3.1.

TRACEON

Restore automatic trace-word execution at traced execution halts. Section 3.1.

TRACEWORD [-> *address*]

Return the address of the variable **TRACEWORD**, which hold the code field address of the current trace-word. Section 3.1.

TRUN { *breakpoint* }

Run an HP-41 program in trace mode, starting at the current HP-41 program pointer value and continuing up to the next program halt, or to the *breakpoint* address, whichever comes first. If **TRACEON** has been executed, display each program line and execute the current trace-word after each line. If no breakpoint is specified, use the current breakpoint. Section 3.2.3.

TRUNA { *breakpoint* }

Set HP-41 alpha mode flag 48, then execute **TRUN**. Section 3.2.3.

UN: { *word* }

Decompile *word*, including the link field, name field, code field, and, if possible, the parameter field. Section 2.2.5.

UN:A [*address* ->]

Decompile part of the parameter field of a secondary, starting at *address*. If *address* does not contain the code field address of a valid FORTH word, abort with the error message **Invalid Address.** Section 2.2.5.

UN:C [*address* ->]

Decompile the code field, and (if possible) the parameter field of the word whose code field is at *address*. Section 2.2.5.

VLIST

Display each word in the context vocabulary, starting with the most recently created word and running up through the built-in ROM-based dictionary.

WAIT

Pause execution for the amount of time stored (in milliseconds) in the variable **PAUSELEN.** If a key is hit during the pause, suspend execution to allow the display to be scrolled. See Section 1.4 for a list of the keys that are active during the halt.

---

6. Software Support

As the author of the TPPT, I wish to support this software at a level appropriate to the complexity and customer cost of the package. To this end, I invite purchasers of the TPPT to write to me directly if you discover bugs in the software or manual. I also welcome any comments you may have on the quality and usability of the TPPT. My address is:

William C. Wickes
4517 NW Queens Ave.
Corvallis OR 97330

The last page of this manual is a registration form. Please fill it out and send it to me with a self-addressed, stamped envelope, so that I can notify you if there are any changes necessary for the manual or the software.

I will try to respond to letters regarding bugs or other shortcomings. However, because of the power, range, and flexibility of the Translator Pac combined with the TPPT, and the number of potential purchasers of the TPPT, I can not guarantee to answer requests for programming tutorials, solutions to programming problems, tips, or answers to questions about the design of the Translator Pac. Correspondence of this nature is best directed to one of the HP calculator user clubs, where it will be given the maximum exposure to the user community. Two such clubs are:

PPC	CHHU
PO Box 9599	2545 W. Camden Pl.
Fountain Valley CA 92728-9599	Santa Ana CA 92704

Translator Pac Programmer's Toolkit

Bug Fix Procedure

Due to a compilation error, copies of the Translator Pac Programmer's Toolkit distributed prior to July 11, 1986, contain certain incorrect compiled addresses that prevent use of STEP, and proper error recovery when an HP41 program is single-stepped through a program line that causes an error.

You can permanently correct this defect by following the following procedure.

1. In the BASIC environment, type **PURGE FTH41RAM [ENDLINE]**.
2. Get a new copy of TPPT:
   - From disk or tape:

     COPY TPPT:TAPE TO FTH41RAM
   - From cards:

     COPY CARD TO FTH41RAM
3. Switch to the FORTH environment: **FORTH [ENDLINE]**.
4. Type:

   HEX 3152E 31633 ! 31658 DUP 31AFA ! 332BE ! [ENDLINE]

   31642 3163D ! 31651 3164C ! DECIMAL [ENDLINE]
5. Return to the BASIC environment: **BYE [ENDLINE]**.
6. Save FTH41RAM as your new copy of TPPT:

   COPY FTH41RAM TO TPPT:TAPE

   or

   COPY FTH41RAM TO CARD