Rev. J. K. Horn, O.Praem.
1042 Star Route
Orange, CA 92667

HEWLETT PACKARD

# HP 82441A
# FORTH/Assembler ROM

# Owner's Manual

**For the HP-71**

**April 1984**

MODULE BY BOB MILLER
(primarily)

82441-90001

Printed in Singapore

- X^2 returns $x^2$.
- 10^X returns $10^x$.
- SIN returns the sine of $x$.
- COS returns the cosine of $x$.
- TAN returns the tangent of $x$.
- E^X returns $e^x$.
- 1/X returns the reciprocal of $x$.
- SQRT returns the square root of $x$.
- Y^X returns $y^x$.
- LGT returns $\log_{10}$ of $x$.
- LN returns the natural log of $x$.
- ATAN returns the arc tangent of $x$.
- ASIN returns the arc sine of $x$
- ACOS returns the arc cosine of $x$.
- RDN rolls down the stack ("down" in the HP-RPN sense).
- RUP rolls up the stack ("up" in the HP-RPN sense).
- X<>Y swaps $x$ and $y$.
- X, Y, Z, T, and L return the address of the corresponding floating-point register.
- LASTX pushes the contents of the LAST X register onto the floating-point stack.
- FENTER pushes the contents of the X-register onto the floating-point stack.
- RCL fetches a floating-point number from the address on top of the data stack and pushes it onto the floating-point stack.
- STO stores $x$ into the address on top of the data stack.
- F. displays $x$ without altering the floating-point stack.
- FVARIABLE creates a floating-point variable in the FORTH dictionary.
- FCONSTANT creates a floating-point constant in the FORTH dictionary.
- X=0?, X>Y?, X<Y?, X=Y?, X#Y?, X<=Y?, and X>=Y? perform the specified test and, if true, push a true flag (−1) onto data stack; or if false, push a false flag (0) onto data stack.
- DEGREES sets the active angular mode to degrees.
- RADIANS sets the active angular mode to radians.
- STD, FIX, ENG, and SCI set the display format.

IP : INTEGER PART OF X → X ; LASTX.

FP : FRACTIONAL PART.

The Copy command permits you to copy one or more lines from one place in the file to another place in the file. You can also copy part of another file into your edit file. Copy always inserts the copied text before the current line. The Move command is similar to the Copy command but deletes the text in the original location.

If no filename is specified, the indicated lines come from the edit file. If a filename is specified, the indicated lines come from the specified file. You can't copy or move a block of text that includes the current line, unless the current line is the first or last line of the block of text.

The `Working...` message is displayed when you copy or move text.

Here are some examples of the Copy and Move commands:

| | |
|---|---|
| `C` | Duplicate the current line. |
| `5 C` | Copy line 5 and insert it before the current line. |
| `3 9 M` | Move lines 3 through 9 from within the edit file and insert them before the current line, then delete the original lines 3 through 9. |
| `C CAT` | Copy the file `CAT` and insert the lines before the current line. |
| `20 C ABC` | Copy lines 20 through the last line of the file ABC and insert the lines before the current line in the edit file. |

## The Delete (D) Command

[*beginning line number* [*ending line number*]] `D` [*filename* [+]]

Default values: *beginning line number* = current line
*ending line number* = beginning line number

The Delete command deletes one or more lines from the edit file. You can place the deleted lines into a new file or, using the + option, append the lines to an existing file. When you execute Delete with line number parameters specifying more than one line, the message `OK to delete? Y/N:` will appear. You must answer `Y` before the editor will complete the deletion. If you answer `N`, the Command Prompt returns.

The `Working...` message is displayed when you use Delete.

The following examples show some uses of the Delete command:

| | |
|---|---|
| `D` | Delete the current line. |
| `12 32D` | Delete lines 12 through 32. |
| `4 9 D CACHE` | Delete lines 4 through 9 and store them in a new file called CACHE. |
| `2 21D ARCHV+` | Delete lines 2 through 21 and append them to the end of a file called ARCHV. |

You can not purge a file while you are in the editor, but you can delete all of the text and leave an empty file. Refer to section 6 of the *HP-71 Owner's Manual* for instructions on how to purge a file.

COMMANDS CANNOT BE CONCATENATED AFTER THE "+" OPTION.

## The Search (S) and Replace (R) Commands

[*beginning line number* [ending line number]][?] S/*string1*[/]

Default values: *beginning line number* = current line + 1
*ending line number* = last line

[*beginning line number* [*ending line number*]][?] R/*string1*/*string2*[/]

Default values: *beginning line number* = current line
*ending line number* = beginning line

The Search and Replace commands allow you to search through a file for a certain string of characters *string1*. If you use a Search command, the first line containing *string1* becomes the current line. If you use a Replace command, all occurrences of *string1* are replaced by *string2*, and the last line containing *string1* becomes the current line. If either command can't find *string1*, it displays `Not Found`.

These commands search the specified lines in the edit file for the string indicated between the slashes (/). These slashes act as *delimiters*, marking the string's boundaries. If you need / as a normal character in your search string, you can use any other character (except a blank space) as the delimiter. The first non-blank character after the command S or R is the delimiter. The last delimiter is optional unless another command follows this command. *

Search and Replace can distinguish between uppercase and lowercase letters. For example, a search for the string `jack` will not find the string `Jack`.

The following examples show some Search commands and Replace commands with parameters:

| | |
|---|---|
| `S/Jack` | From the next line through the end of the file, search for the first occurrence of the string "Jack." |
| `3 7 S/Jill` | From line 3 through line 7, search for the string "Jill." |
| `R/cat/dog/` | Replace all occurences of "cat" with "dog" on the current line. |
| `4 7R/cat/dog` | On lines 4 through 7, replace all occurences of "cat" with "dog." |
| `R‡3/4‡3/8` | On the current line, replace all occurences of "3/4" with "3/8." The character ‡ is used as the delimiter so that slashes may occur in the strings. |
| `.#R/meet//` | From the current line to the end of the file, replace "meet" with the null string (that is, delete "meet"). |

If the replacement *string2* causes the line to be longer than 96 characters, the editor will redimension variables, causing a slight delay.

*COMMANDS ARE CONCATENATED WITH SEMICOLONS, e.g.

1S/FRED/;T

SEARCHES FOR "FRED" THEN ENTERS TEXT MODE.

Section 4

# The Assembler

The FORTH/Assembler ROM contains an assembler that enables you to write assembly language extensions to the FORTH system or to the BASIC operating system. The assembler provides access to the complete HP-71 CPU instruction set through source code mnemonics that are nearly identical to those of the assembler used to produce the HP-71 BASIC operating system, as listed in the *HP-71 IDS*.

The assembler is invoked from the FORTH environment by the word ASSEMBLE, which is preceded by a string specifying the name of the assembler source file. The source file is an HP-71 text file, which you can create using the editor described in section 3. The output of the assembler can be either new FORTH words, which are placed directly into the FORTH dictionary, or HP-71 language extension (LEX) or binary (BIN) files, which are loaded automatically into the HP-71 file chain.* The type of assembler output is specified by pseudo-ops included in the source file. The assembler can also produce an optional assembly listing, which is directed to an HP-71 file or to a listing device on HP-IL.

This section gives the rules for using the assembler, describes the HP-71 CPU, shows some sample source files for the three types of assembly, and finally describes the assembler's mnemonics and pseudo-ops.

## Using the Assembler

### Running the Assembler

The assembler is run while in the FORTH system by typing:

```
" source-file specifier" ASSEMBLE
```

The source-file specifier can include a mass storage device specifier. You can't run the assembler from BASIC (using FORTHX) because the assembler uses BASICX.

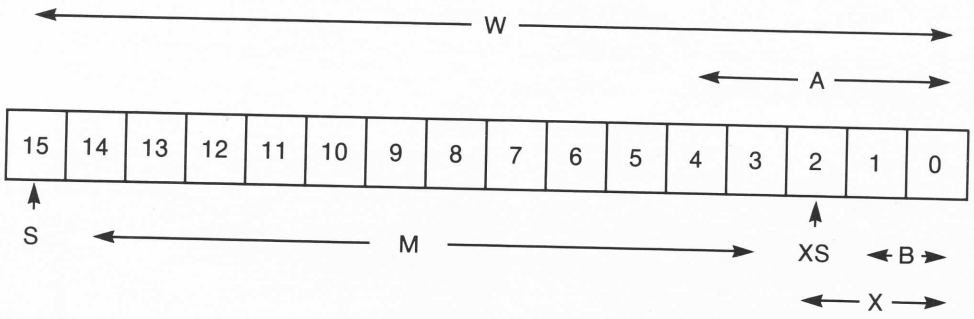
There is no intermediate link operation. The assembler acts as a loader, creating absolute modules that are ready to execute. New FORTH primitives go directly into the FORTHRAM dictionary. LEX and BIN files go directly into the file chain in RAM.*

While the assembler is running, the display will show PASS 1 . . . . or PASS 2 . . . . to indicate the assembly's progress. A dot . is added to the display as each source line is processed. If you press [ATTN] while the assembler is active, the assembler will halt and prompt you with the message ABORT [Y/N] ? If you now press [Y], the assembly will terminate, and the message assembler aborted will be displayed. If you press any other key, the assembly will resume.

* BUT LEX FILES ARE NOT ADDED TO THE LEX CHAIN AUTOMATICALLY! YOU MUST TURN THE 71 OFF & BACK ON TO CONFIGURE THEM.

Subfields of the working registers may be manipulated by field selection. The possible field selections range from the entire register to any single nibble of the register. Certain subfields are designed for use in BCD calculations; others are used for data access or general data manipulation. The following diagram shows the seven fixed fields within a 16-nibble working register.

**Fixed Fields within a Working Register**




There is a one-nibble CPU pointer (the P register, described under "Control Registers") that can indicate any nibble in a working register. This allows two variable fields to be defined: the indicated nibble alone, or that nibble along with all lower nibbles (to the right). This makes a total of nine fields, listed below.

**Fields within a Working Register**

| Name | Nibbles | Description | digits (d) |
|---|---|---|---|
| B | 1-0 | Exponent or byte. | 2 |
| X | 2-0 | Exponent and sign. | 3 |
| XS | 2 | Exponent sign. | 1 |
| A | 4-0 | Address. | 5 |
| W | 15-0 | Full word. | 16 |
| M | 14-3 | Mantissa. | 12 |
| S | 15 | Sign. | 1 |
| P | P | At pointer. | 1 |
| WP | P-0 | Word through pointer. | (P) |

**Sample Binary Program.** This binary program displays HELLO.

```
        BIN     'HELLO'  Declare an assembly of a binary file called "HELLO."
        CHAIN   -1       There are no subprograms in this file. Binary
         *                subprograms are described in the HP-71 IDS.
BF2DSP  EQU     #01C0E   Define a label for the entry point of a system
         *                routine. The system routine displays the string in
         *                memory that starts at DAT1 and ends with a
         *                character #FF.
ENDBIN  EQU     #0764B   Define a label for the entry point of the system
         *                routine that ends a binary program.
         *               The code immediately follows the pseudo-ops.
        GOSUB   POP      This line, combined with C=RSTK (labeled POP), puts
         *                the address of the following string into the A
         *                field of register C.
        NIBASC  'HELLO'  The string HELLO.
        NIBHEX  D0A0FF   Carriage return, line feed, end-of-string mark.
POP     C=RSTK           Pop the return address (which is the address of the
         *                preceding string) into the A field of register C.
        D1=C             Copy the string's address to D1.
        GOSBVL  BF2DSP   Call the system routine to display the string
         *                pointed to by D1.
        GOVLNG  ENDBIN   The correct way to exit a binary program.
        END              Mark the end of the source file (optional).
```

# Assembler Mnemonics

The assembler mnemonics are listed below in condensed form, grouped by function. A list of all mnemonics (listed in ASCII order) with their opcodes and cycle times appears in the *HP-71 Software IDS*.

## Branching Mnemonics

**GOTO Mnemonics.** In the following mnemonics,

- *offset* is the distance in nibbles to the specified label.

| | | |
|---|---|---|
| GOTO *label* | Short goto ($-2047 \leq$ *offset* $\leq 2048$). | 6aaa (6300 = NOP) |
| GOC *label* | Short goto if carry ($-127 \leq$ *offset* $\leq 128$). | 4aa (420 = NOP) |
| GONC *label* | Short goto if no carry ($-127 \leq$ *offset* $\leq 128$). | 5aa (520 = NOP) |
| GOLONG *label* | Long goto ($-32766 \leq$ *offset* $\leq 32769$). | 8Caaaa (8C4000 = NOP) |
| GOVLNG *label* | Very long goto (to absolute address). | 8Daaaaa |
| GOYES *label* | Short goto if test true ($-128 \leq$ *offset* $\leq 127$). (Used only with test mnemonics.) | — |

cycles

T (F)

opcode

**GOSUB Mnemonics.** In the following mnemonics,

- *offset* is the distance in nibbles to the specified label.

| | |
|---|---|
| GOSUB *label* | Short gosub ($-2044 \leqslant$ *offset* $\leqslant 2051$). |
| GOSUBL *label* | Long gosub ($-32762 \leqslant$ *offset* $\leqslant 32773$). |
| GOSBVL *label* | Very long gosub (to absolute address). |

**Return Mnemonics.**

| | |
|---|---|
| RTN | Return. |
| RTNSC | Return and set carry. |
| RTNCC | Return and clear carry. |
| RTNSXM | Return and set External Module Missing bit. |
| RTI | Return from interrupt (enable interrupts). |
| RTNC | Return if carry set. |
| RTNNC | Return if no carry set. |
| RTNYES | Return if test true. (Used only with test mnemonics.) |

## Test Mnemonics

Each test mnemonic must be followed with a GOYES or RTNYES mnemonic. The test mnemonic and the GOYES or RTNYES mnemonic combine to generate a single opcode. Each test will set the carry flag if true, or clear the carry flag if false.

**Register Tests.** In the following mnemonics,

- (*r*, *s*) = (A, B), (A, C), (B, A), (B, C), (C, A), (C, B), (C, D), or (D, C).
- *fs* = A, P, WP, XS, X, S, M, B, or W.

| | |
|---|---|
| ?r=s *fs* | Is *fs* field of *r* equal to *fs* field of *s*? |
| ?r#s *fs* | Is *fs* field of *r* not equal to *fs* field of *s*? |
| ?r=0 *fs* | Is *fs* field of *r* equal to zero? |
| ?r#0 *fs* | Is *fs* field of *r* not equal to zero? |
| ?r>s *fs* | Is *fs* field of *r* greater than *fs* field of *s*? |
| ?r<s *fs* | Is *fs* field of *r* less than *fs* field of *s*? |
| ?r>=s *fs* | Is *fs* field of *r* greater than or equal to *fs* field of *s*? |
| ?r<=s *fs* | Is *fs* field of *r* less than or equal to *fs* field of *s*? |

**Pointer Tests.** In the following mnemonics,

- *n* is an expression whose hex value is from 0 through F.

| | |
|---|---|
| ?P= *n* | Is P register equal to *n*? |
| ?P# *n* | Is P register not equal to *n*? |

**Program-Status Tests.** In the following mnemonics,

- *n* is an expression whose hex value is from 0 through F.

| | |
|---|---|
| ?ST=0 *n* | Is bit *n* in ST equal to 0? |
| ?ST=1 *n* | Is bit *n* in ST equal to 1? |
| ?ST#0 *n* | Is bit *n* in ST not equal to 0? |
| ?ST#1 *n* | Is bit *n* in ST not equal to 1? |

**Hardware-Status Tests.**

| | |
|---|---|
| ?XM=0 | Is the External Module Missing bit clear? |
| ?SB=0 | Is the Sticky bit clear? |
| ?SR=0 | Is the Service Request bit clear? |
| ?MP=0 | Is the Module Pulled bit clear? |

## P Register Mnemonics

In the following mnemonics,

- *n* is an expression whose hex value is from 0 through F.

Note that the C register is the only working register used with the P register. During those operations that involve a calculation, the carry flag is set if the calculation overflows or borrows; otherwise the carry flag is cleared.

| | |
|---|---|
| P= *n* | Set P register to *n*. |
| P=P+1 | Increment P register. |
| P=P-1 | Decrement P register. |
| C+P+1 | Add P register plus one to A field in C. Arithmetic is hexadecimal. |
| CPEX *n* | Exchange P register with nibble *n* in C. |
| P=C *n* | Copy nibble *n* in C to P register. |
| C=P *n* | Copy P register to nibble *n* in C. |

## Status Mnemonics

In the following mnemonics,

- *n* is an expression whose hex value is from 0 through F.

| | | |
|---|---|---|
| 4 | ST=0 *n* | Set bit *n* in ST to 0. 84n |
| 4 | ST=1 *n* | Set bit *n* in ST to 1. 85n |
| 6 | CSTEX | Exchange X field in C and bits 0 through 11 in ST. 0B |
| 6 | C=ST | Copy bits 0 through 11 in ST into X field in C. 09 |
| 6 | ST=C | Copy X field in C into bits 0 through 11 in ST. 04 |
| 6 | CLRST | Clear bits 0 through 11 in ST. 08 |
| 3 | SB=0 | Clear Sticky bit (SB). 822 |
| 3 | SR=0 | Clear Service Request (SR) bit. 824 |
| 3 | MP=0 | Clear Module Pulled (MP) bit. 828 |
| 3 | XM=0 | Clear External Module Missing (XM) bit. 821 |
| 3 | CLRHST | Clear SB, SR, MP, and XM bits. 82F → 82n { 1 = XM, 2 = SB, 4 = SR, 8 = MP |

## System-Control and Keyscan Mnemonics

The first four mnemonics below are useful for most programmers. The remaining mnemonics are used by the system and have limited general use; for details, refer to the *HP-71 IDS* and the *HP-71 Hardware Specification*.

| | | |
|---|---|---|
| 3 | SETHEX | Set arithmetic mode to hexadecimal. 04 |
| 3 | SETDEC | Set arithmetic mode to decimal. 05 |
| 8 | C=RSTK | Pop return stack into A field in C. 07 |
| 8 | RSTK=C | Push A field in C onto return stack. 06 |
| 11 | CONFIG | Configure. 805 |
| 12 | UNCNFG | Unconfigure. 804 |
| 6 | RESET | Send Reset command to system bus. 80A |
| 6 | BUSCC | Send Bus Command C to system bus. 80B |
| 5 | SHUTDN | Stop here. 807 |
| 11 | C=ID | Request ID (A field in C). 806 |
| 7 | SREQ? | Sets service request bit if service has has been requested. Nibble 0 in C shows what bit(s) are pulled high. 80E |
| 5 | INTOFF | Disable interrupts (doesn't affect ON-key or module-pulled interrupts). 808F |
| 5 | INTON | Enable interrupts. 8080 |
| 6 | OUT=C | Copy X field in C into OUT. 801 |
| 4 | OUT=CS | Copy nibble 0 of C into OUT. 800 |
| 7 | A=IN | Copy IN into nibbles 0 through 3 in A. 802 |
| 7 | C=IN | Copy IN into nibbles 0 through 3 in C. 803 |
| | PC=(A) | used by HP-28S 808C |
| | RSI | " " " 8081 |

## Scratch Register Mnemonics

In the following mnemonics,

- *r* = A or C.
- *ss* = R0, R1, R2, R3, or R4.

| | |
|---|---|
| rssEX | Exchange *r* and *ss*. |
| r=ss | Copy *ss* into *r*. |
| ss=r | Copy *r* into *ss*. |

## Memory-Access Mnemonics

**Data-Pointer Mnemonics.** In the following mnemonics,

- *r* = A or C.
- *ss* = D0 or D1.
- *n* is an expression whose hex value is from 0 through F.
- *nnnnn* is an expression whose hex value is from 0 through FFFFF.

During those operations that involve a calculation, the carry flag is set if the calculation overflows or borrows; otherwise the carry flag is cleared.

| | |
|---|---|
| rssEX | Exchange A field in *r* with *ss*. |
| rssXS | Exchange nibbles 0 through 3 in *r* with *ss*. |
| ss=r | Copy A field in *r* into *ss*. |
| ss=rS | Copy nibbles 0 through 3 in *r* into *ss*. |
| ss=ss+ n | Increment *ss* by *n*. |
| ss=ss- n | Decrement *ss* by *n*. |
| ss=(2) nnnnn | Load *ss* with two nibbles from *nnnnn*. |
| ss=(4) nnnnn | Load *ss* with four nibbles from *nnnnn*. |
| ss=(5) nnnnn | Load *ss* with *nnnnn*. |

**Data-Transfer Mnemonics.** In the following mnemonics,

- *r* = A or C.
- *fs* = A, P, WP, XS, X, S, M, B, W (or a number *n* from 1 through 16).

| | |
|---|---|
| r=DAT0 fs | Copy data at address contained in D0 into *fs* field in *r* (or into nibble 0 through nibble *n* − 1 in *r*). |
| r=DAT1 fs | Copy data at address contained in D1 into *fs* field in *r* (or into nibble 0 through nibble *n* − 1 in *r*). |
| DAT0=r fs | Copy data in *fs* field in *r* (or in nibble 0 through nibble *n* − 1 in *r*) to address contained in D0. |
| DAT1=r fs | Copy data in *fs* field in *r* (or in nibble 0 through nibble *n* − 1 in *r*) to address contained in D1. |

## Load-Constants Mnemonics

In the following mnemonics,

- *h* is a hex digit.
- *i* is an integer from 1 through 5.
- *nnnnn* is an expression with hex value from 0 through FFFFF.
- *c* is an ASCII character.

3+d `LCHEX` *h . . . h* — Load up to 16 hex digits into C. The least significant digit is loaded at the pointer position; more significant digits are loaded into higher positions, wrapping around from nibble 15 to nibble 0. 3nhhh... , n=#digits −1

3+d `LC(i)` *nnnnn* — Load *i* hex digits from the value of *nnnnn* into C. The least significant digit is loaded at the pointer position; more significant digits are loaded into higher positions, wrapping around from nibble 15 to nibble 0.

3+2c `LCASC 'c . . . c'` — Load up to eight ASCII characters into C. The least significant nibble of the low-order character is loaded at the pointer position; more significant nibbles are loaded into higher positions, wrapping around from nibble 15 to nibble 0. For example, `LCASC 'AB'` is equivalent to `LCHEX 4142`.

## Shift Mnemonics

In the following mnemonics,

- *r* = A, B, C, or D.
- *fs* = A, P, WP, XS, X, S, M, B, or W.

Non-circular shift operations shift in zeros. If any shift-right operation, circular or non-circular, moves a non-zero nibble or bit from the right end of a register or field, the Sticky bit SB is set. The Sticky bit is cleared only by a `SB=0` or `CLRHST` instruction.

*r*`SRB` — Shift *r* right by one bit. (MAY SET SB)

*r*`SLC` — Shift *r* left by one nibble (circular).

*r*`SRC` — Shift *r* right by one nibble (circular). (MAY SET SB)

3+d *r*`SL` *fs* — Shift *fs* field in *r* left by one nibble.

3+d *r*`SR` *fs* — Shift *fs* field in r right by one nibble. (MAY SET SB)

## Logical Mnemonics

These mnemonics are summarized below, using the following variables:

- (*r*, *s*) = (A, B), (A, C), (B, A), (B, C), (C, A), (C, B), (C, D), or (D, C).
- *fs* = A, P, WP, XS, X, S, M, B, or W.

*r*=*r*&*s* *fs* — *fs* field in *r* AND *fs* field in *s* into *fs* field in *r*.

*r*=*r*!*s* *fs* — *fs* field in *r* OR *fs* field in *s* into *fs* field in *r*.

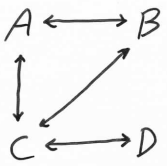
## Arithmetic Mnemonics

Arithmetic results depend on the current arithmetic mode. In hexadecimal mode (set by SETHEX), nibble values range from 0 through F. In decimal mode (set by SETDEC), nibble values range from 0 through 9, and arithmetic is BCD arithmetic.

There are two groups of arithmetic mnemonics. In the first group (general), almost all combinations of the four working registers are possible; in the second group (restricted), only a few combinations are possible. During those operations that involve a calculation, the carry flag is set if the calculation overflows or borrows; otherwise the carry flag is cleared.

**General Arithmetic Mnemonics.** In the following mnemonics,

- (*r*, *s*) = (A, B), (A, C), (B, A), (B, C), (C, A), (C, B), (C, D), or (D, C).
- *fs* = A, P, WP, XS, X, S, M, B, or W.




| | |
|---|---|
| r=0 fs | Set *fs* field in *r* to zero. |
| r=r+r fs | Double *fs* field in *r* (shift left by one bit). |
| r=r+1 fs | Increment *fs* field in *r* by 1. |
| r=r-1 fs | Decrement *fs* field in *r* by 1. |
| r=-r fs | Tens complement or twos complement, depending on arithmetic mode, of *fs* field in *r*. Clears Carry if *argument* =0; sets Carry otherwise. |
| r=-r-1 fs | Nines complement or ones complement, depending on arithmetic mode, of *fs* field in *r*. Clears Carry. |
| r=r+s fs | Sum *fs* field in *r* and *fs* field in *s* into *fs* field in *r*. |
| s=r+s fs | Sum *fs* field in *r* and *fs* field in *s* into *fs* field in *s*. |
| r=s fs | Copy *fs* field in *s* into *fs* field in *r*. |
| s=r fs | Copy *fs* field in *r* into *fs* field in *s*. |
| rsEX fs | Exchange *fs* field in *r* and *fs* field in *s*. |

**Restricted Arithmetic Mnemonics.** In the following mnemonics,

- (*r*, *s*) = (A, B), (B, C), (C, A), or (D, C).
- *fs* = A, P, WP, XS, X, S, M, B, or W.

| | |
|---|---|
| r=r-s fs | Difference of *fs* field in *r* and *fs* field in *s* into *fs* field in *r*. |
| r=s-r fs | Difference of *fs* field in *s* and *fs* field in *r* into *fs* field in *r*. |
| s=s-r fs | Difference of *fs* field in *s* and *fs* field in *r* into *fs* field in *s*. |

## No-op Mnemonics

| | | |
|---|---|---|
| NOP3 | Three-nibble no-op. | 420 |
| NOP4 | Four-nibble no-op. | 6300 |
| NOP5 | Five-nibble no-op. | 64000 |

## Macro-Expansion Pseudo-ops for LEX Files

`LEX 'name'` — Assemble a new LEX file called *name*. This pseudo-op must be the first line in the source file. The LEX file will have the correct header. The intial data for this file is defined by the `ID`, `MSG`, and `POLL` pseudo-ops, which must be present in that order.

`ID byte` — Define the LEX ID of this LEX file. The byte is placed in the appropriate data field. This pseudo-op is required when the `LEX` pseudo-op is used.

`MSG label` — Define the beginning of this LEX file's message table. `MSG` will place *label* in the appropriate field. This pseudo-op is required when the `LEX` pseudo-op is used. If there is no message table, enter `MSG 0`.

`POLL label` — Define the beginning of this LEX file's poll-handling routine. `POLL` will place *label* in the appropriate field. This pseudo-op is required when the `LEX` pseudo-op is used. If there is no poll-handling routine, enter `POLL 0`.

`ENTRY label` — Begin the definition of a BASIC keyword. Each keyword requires four pseudo-ops: `ENTRY`, `CHAR`, `KEY`, and `TOKEN`.

Because of the structure of the LEX file's keyword tables, these pseudo-ops require a particular order. For example, the following assembly-language header defines two keywords, `KEY1` and `KEY2`.

| | | |
|---|---|---|
| 1 | `ENTRY label1` | The code for the first keyword begins at *label1*. |
| | `CHAR 5` | The first keyword is legal for keyboard execution and after `THEN...ELSE`. |
| 2 | `ENTRY label2` | The code for the second keyword begins at *label2*. |
| | `CHAR #F` | The second keyword is a function. |

| | | |
|---|---|---|
| 1 | `KEY 'KEY1'` | The first keyword is invoked with "KEY1" in BASIC. |
| | `TOKEN 1` | The first keyword has token 1. |
| 2 | `KEY 'KEY2'` | The second keyword is invoked with "KEY2" in BASIC. |
| | `TOKEN 2` | The second keyword has token 2. |
| | `ENDTXT` | Mark the end of the keyword tables. |

`CHAR h` — Describe the type of BASIC keyword. Each `ENTRY` requires a corresponding `CHAR`, which places a "characterization nibble" in the keyword tables. The characterization nibble defines BASIC keywords as follows.

# DELETE#

Deletes one record from a text file.

| | |
|---|---|
| ■ Statement | ■ Keyboard Execution |
| □ Function | □ CALC Mode |
| □ Operator | ■ IF...THEN...ELSE |

DELETE# *channel number , record number*

## Example

```
DELETE# 5,14
```

Deletes the 14th record from the text file currently assigned to channel #5.

## Input Parameters

| Item | Description | Restrictions |
|---|---|---|
| channel number | Numeric expression rounded to an integer. | 1 through 255. |
| record number | Numeric expression rounded to an integer. | 0 through EOF |

## Operation

The DELETE# keyword deletes the specified record from the text file assigned to the specified channel number. Record numbers always begin at 0, so line number 1 is record number 0.

The channel number and the record number can be expressions. DELETE# rounds each of the resulting values to an integer.

DELETE# returns an error message if the assigned file is external, protected, or not a text file.

## Related Keywords

ASSIGN#, INSERT#, REPLACE#, FILESZR

# EDTEXT

Invokes the text editor.

| | |
|---|---|
| ■ Statement | ■ Keyboard Execution |
| □ Function | □ CALC Mode |
| □ Operator | ■ IF...THEN...ELSE |

```
EDTEXT file specifier[, command string]
```

## Examples

| | |
|---|---|
| `EDTEXT SCREEN` | Runs the editor program, with `SCREEN` as the edit file. |
| `EDTEXT SCREEN, L` | Runs the editor program, with `SCREEN` as the edit file. Begins by listing the file to the display device. |

## Input Parameters

| Item | Description | Restrictions |
|---|---|---|
| file specifier | String expression or unquoted string. | File must be in RAM or IRAM. |
| command string | String or unquoted string<br>See description of editor command strings in section 3. | |

## Operation

The `EDTEXT` keyword starts the editor program. The optional command string permits you to have the editor begin immediate execution of editor commands that appear in the command string.

An error can cause the editor program to terminate without going through its normal exit path. If you are running the editor from another BASIC program, or from the FORTH environment, you can check for this situation by using `DISP$` to read the display contents. If the result is other than `Done: <filename>`, then you will know that the editor has encountered a fatal error, the edit file may be in a corrupt state, and the editor key assignments may still be active. For example, from the FORTH environment, you can type the sequence

```
" EDTEXT SCREEN" BASICX " DISP$" BASICX DROP @ -102588 =
```

to edit the file `SCREEN`. When the editor terminates, a true flag will be pushed on the stack if the editor terminated normally (here we are checking the numerical equivalent of the first three characters on the display to see if they match "Don", which translates to −102588).

## Related Keywords

`ASSIGN#, DELETE#, REPLACE#, FILESZR`

# ESCAPE

Adds or modifies an escape-sequence key specification in the current KEYBOARD IS key map buffer.

| | |
|---|---|
| ■ Statement | ■ Keyboard Execution |
| □ Function | □ CALC Mode |
| □ Operator | ■ IF...THEN...ELSE |
| | ■ Device Operation |

ESCAPE *string , key number*

## Example

```
ESCAPE "A",43
```

Specifies that the escape sequence (ESC)A received from the KEYBOARD IS device will be changed to key code 43.

```
ESCAPE "A",0
```

Cancels the (ESC)A assignment.

## Input Parameters

| Item | Description | Restrictions |
|---|---|---|
| string | String expression. | Only the first character is used. |
| key number | Keycode. | 0 through 168. |

## Operation

ESCAPE specifies that a particular one-character escape sequence (the escape character ASCII 27 followed by a single character) received by the HP-71 from the current KEYBOARD IS device will be replaced by an HP-71 keycode in the key buffer input. ESCAPE requires two parameters, a one-character string and a numeric keycode. The string specifies the escape sequence; the number indicates the corresponding keycode.

The first execution of ESCAPE creates a special HP-71 buffer that specifies the mapping of escape sequences received from a KEYBOARD IS device to HP-71 keycodes. Each subsequent use of ESCAPE will add a new character/key code mapping, or modify an existing one, in the buffer. You can clear the buffer completely by executing RESET ESCAPE. The buffer will be cleared if you turn on the HP-71 when the FORTH/Assembler ROM is not installed.

MAPKBD CHR$(3), 43

↳ TURN CTRL-C to ATTN ; requires KEYBOARD LEX FILE (on disk)

# SCROLL

Scrolls the display to a position and waits for a key to be pressed.

| | |
|---|---|
| ■ Statement | ■ Keyboard Execution |
| □ Function | □ CALC Mode |
| □ Operator | ■ IF...THEN...ELSE |

```
SCROLL position
```

## Example

```
DISP "Hello there" @ SCROLL 4
```

Display the string "Hello there," with the fourth character in the string as the first character in the display, so that the display shows "lo there."

## Input Parameters

| Item | Description | Restrictions |
|---|---|---|
| position | Numeric expression rounded to an integer. | 1 through 96. |

## Operation

The SCROLL keyword enables you to display a string, under program control, that can be scrolled from the keyboard. Execution of SCROLL causes the current display string to shift so that the character in the position specified by the numeric expression is the leftmost character in the display. Execution halts, so that a user can press the left- and right-arrow keys to scroll the display. Execution resumes when any other key is pressed (the pressed keycode is placed in the key buffer). The number input with SCROLL must be greater than zero.

# SEARCH

Finds a string in a text file.

| | |
|---|---|
| ☐ Statement | ■ Keyboard Execution |
| ■ Function | ☐ CALC Mode |
| ☐ Operator | ■ IF...THEN...ELSE |

```
SEARCH(search string, column number, begin line, end line, channel)
                                      RECORD    RECORD
```

## Example

```
X=SEARCH("Hello",5,1,99,2)
```

Searches the file assigned to channel #2 for the string "Hello." The search starts in column 5, line 1, and extends through line 99.

## Input Parameters

| Item | Description | Restrictions |
|---|---|---|
| search string | String expression. | ~~1 through 9999~~ |
| column number | Numeric expression rounded to an integer. | 1 through 9999 |
| begin line | Numeric expression rounded to an integer. | ~~0~~1 through 9999 |
| end line | Numeric expression rounded to an integer. | ~~0~~1 through 9999 |
| channel | Numeric expression rounded to an integer. | 1 through 255 |

## Operation

The SEARCH keyword enables you to determine the location of a specified string within an HP-71 text file. If the search is successful, SEARCH returns a value in the format *nnn.ccclll*, where *nnn* is the record number, *ccc* is the column number, and *lll* is the length of the matched string. If the search is unsuccessful, zero is returned.

The search string can be any string expression, and the other parameters can be any numeric expression. Each input value is rounded to an integer. A zero is returned for an empty file.

## Related Keywords

INSERT#, DELETE#, REPLACE#

The search string may contain "\" wildcard characters, in which case the "lll" part is useful.