D+ (d1 d2 -- d1+d2 , add double numbers)

Double precision numbers are 64 bits wide.

```
HEX  12345678.55443322  19283746.98765432 D+ D.
( prints  1B4C8DAEEDBA8754 )
```

D- (d1 d2 -- d1+d2 , subtract double numbers)

D. (d -- , print double number)

D.R (d width -- , print D in field WIDTH wide)

This can be used to print tables of right justified double precision numbers.

D2* (d -- d*2 , fast multiply by 2)

D2/ (d -- d/2 , fast divide by 2)

DDUP (d1 -- d1 d1 , duplicate double number)

Same as 2DUP.

DECIMAL (-- , set the numeric base to decimal 10)

```
HEX  9 1 + .  ( base 16 numbers )
10 BASE !    9 1 + .  ( Oops! Still in HEX cuz $10 = 16 )
DECIMAL  9 1 + .   ( Now we are really in base 10 )
```

DEFER (<name> --)

Define a deferred word which can be changed later. See the section on deferred words elsewhere in this manual. Here is a brief example of their use:

```
DEFER  FOO
WHAT'S FOO >NAME ID.   ( default is QUIT )
: HI ." Hello" cr ;
' HI IS FOO   ( set FOO to be HI )
FOO  ( says Hello)
```

Related Words: IS WHAT'S >NAME EXECUTE

DEPTH (-- n , number of items on stack)

HMSL monitors the stack depth using this word. If something accidentally leaves numbers on the stack, or takes them away, when in HMSL is running, it will stop and print a message. This prevents the stack from completely over- or underflowing. Check your stack diagrams if you have problems with this.

```
0SP 123 99 7654 DEPTH . ( PRINTS 3 )
```

DIALOG.A ($string --)

Host independant routine to display a message until user acknowledges.

DIALOG.A/B ($string -- flag)

Host independant routine to display a message until user acknowledges. Returns TRUE if user selects "Continue". Returns FALSE if user hits "Cancel".

DIALOG.Y/N/C ($string -- flag)

Host independant routine to display a message until user acknowledges. Returns 1 if user selects "Yes". Returns 2 if user hits "No". Returns 3 if user selects "Cancel".

DIGIT (char base -- n true | char false)

Convert a character to number N and a TRUE. Returns a FALSE and the CHAR if the conversion fails.

```
ASCII 8 10 DIGIT  SWAP . . ( print 8 and -1 , TRUE = -1 )
ASCII W 10 DIGIT  SWAP EMIT  SPACE . ( print W and 0 )
```

DO (limit start -- , begin DO LOOP)

Start a DO LOOP. Set the loop index I equal to START and resume execution. When LOOP is reached, I is incremented. If I is less than LIMIT, then jump back to DO. Otherwise continue execution. Here is an example that prints "Hello" 10 times.

```
: 10HI  10 0 DO ." Hello" CR LOOP ;
```

Here is an example that prints the numbers 0 through 5. Note that 6 numbers are printed.

```
: P-0-5    6 0 DO I . LOOP ;
```

Note that the LIMIT is one greater than the highest index we want to use.

Here is another DO LOOP example. N factorial is the product of the integers 1 to N. To calculate we can use a DO LOOP with START = 1. We use a LIMIT = N+1. so that the highest index will be N.

```
: NFACT  ( N -- N! , calculate N factorial )
   1 SWAP 1+ 1  ( 1 N+1 1 on stack )
   DO  I  *
   LOOP
;
 4 NFACT .  ( prints 24 )
```

In HMSL, DO first checks to to see if LIMIT is greater than START. If not, then it skips to LOOP and doesn't execute the inside of the LOOP. Some Forths will execute the contents of the loop even if START equals LIMIT. Notice how DO and LOOP are indented the same amount. This is considered by many to be good Forth style.

Note: DO LOOP uses the Return Stack so do not put DO or LOOP between >R R@ and R>. Use local variables or BEGIN UNTIL instead.

```
: BAD-BAD-BAD
   20 >R     5 0 DO R@ . LOOP  R> DROP
;   ( Don't expect to print 20. Try it. )
```

Notice how DO and LOOP were not indented to the same level. Putting DO and LOOP on the same line is considered lazy programming but it does save trees when you print manuals.

Related Words: J BEGIN UNTIL WHILE REPEAT

DOES> (-- addr , of CREATEd word)

This is used with CREATE to make intelligent data structures. It is used extensively, for example, by ODE to make classes and objects work. DOES> provides Forth with abilities not commonly found in other languages.

In a CREATE DOES> word, the CREATE code defines a word and its initial data values and allocates space using ALLOT or , or W, , etc. When the newly defined word is executed, the DOES> code runs. It is passed the address of the data which it can manipulate in any way desired. You can pass parameters to CREATE or DOES> code if needed.

Here is an example of a CREATE DOES> word that creates self incrementing data structures, or "counters". This counter will keep a running total of how many somethings have been counted and return the current total.

```
: COUNTER  ( <name> -- , define counter )
   CREATE  0 ,  ( initialize value to 0 )
   DOES>  ( #MORE addr-N -- N+1 )
      TUCK (  -- addr-N #MORE addr-N )
      +! @
;
COUNTER TOADS
COUNTER FROGS
3 FROGS . ( PRINT 3 )
2 FROGS . ( PRINT 5 )
4 TOADS . ( PRINT 4 )
7 FROGS . ( PRINT 11 )
```

Related Words: CREATE , :STRUCT ..@ IMMEDIATE VALUE CONSTANT VARIABLE LITERAL

DP (-- address , of dictionary pointer)

This is the address at which new code or data will be compiled into the dictionary. The value of DP is normally fetched using HERE. The word ALLOT increments the dictionary pointer to allot memory for data. If you allot an odd number of bytes you must call ALIGN so that DP will always be word aligned (even).

Related Words: HERE , ALLOT ALIGN

DPL (-- addr)

A variable that holds the number of places to the right of the decimal point in a double number. If there is no decimal point, DPL = -1.

```
635.8613  DPL @ .  ( prints 4 )
```

DROP (n -- , remove top number from stack)

The number is discarded.

111 222 333 DROP .S (prints: 111 222)

Related words: 0SP RDROP XDROP 2DROP OVER SWAP ROT DUP

DST (addr <structure-type> --)

Diagnostic debugging tool. Prints the data at ADDR assuming it is a structure of the specified type. The structure member names and their contents will be printed. Optionally loaded.

```
INCLUDE  HSYS:DUMP_STRUCT
RECT MyRect
60 MyRect ..! RECT_LEFT
MyRect DST RECT
```

Related words: DUMP :STRUCT ..@

DUMP (addr #bytes -- , hex dump of memory)

Print the contents of #BYTES bytes of memory starting at ADDR. Print as hex numbers and as ASCII text. To pause the output of DUMP, hit the space bar.

```
VARIABLE VAR1
$ 1BA5EF77 VAR1 !
VAR1 20 DUMP  ( see 1BA5Ef77 plus other mysterious stuff )
```

DUP (n -- n n , DUPlicate the top item on the stack)

23 DUP .S (prints 23 23)

Related words: 2DUP OVER TUCK

DUP>R (n -- n) (-r- n)

Push a copy of n onto the return stack. Faster than DUP >R separately but does the same thing.

ELSE (--)

Provides an "otherwise" clause to IF THEN.

```
:  SHOE?  ( shoe-size  -- )
   11 =
   IF  ." The shoe fits!" CR
   ELSE ." Ouch! Find another shoe." CR
   THEN
;
```

Notice that IF ELSE and THEN are indented the same amount. This makes the code more readable.

Related words: IF THEN BEGIN WHILE REPEAT

EMIT (char -- , print ASCII char)

Low level output word. Note that in most Forths, TYPE calls EMIT. In HMSL, EMIT calls TYPE for speed reasons.

```
ASCII W EMIT  ( print W )
: ALPHABET  ( -- , print alphabet )
   $ 7B  $ 5A
   DO I EMIT
   LOOP
;
```

EMIT is deferred.

```
WHAT'S EMIT >NAME ID.
```

EMIT-TO-COLUMN (char column# --)

This will print multiple CHARs out to column COLUMN#. Useful for making tables, etc.

```
CR 123456 . ASCII * 20 EMIT-TO-COLUMN  ( prints)
123456 *************
```

ENDCASE (n --)

Terminate a CASE statement. See CASE.

ENDOF (--)

Terminate an OF ENDOF statement. See CASE.

EOL (-- char)

End Of Line character. This is a $0D, a carriage return, on a Macintosh. On normal computers it is $0A, a line feed. When parsing files, look for this character to mark the end of a line.

ERASE (addr count --)

Set COUNT bytes to zero starting at ADDR.

Related words: FILL ALLOT

EVEN-UP (n -- n')

If N is odd, add one so that it becomes even. Otherwise do nothing.

Related words: ALIGN

EXECUTE (cfa --)

Execute the word whose Code Field Address is on the stack. This allows you to pass a function address as a parameter to another function.

```
: HI ." Hello" cr ;
HI  ( prints hello )
' HI EXECUTE   ( prints hello )
```

Related words: ' 'C 'N NAME> FIND @EXECUTE IF.EXECUTE

EXISTS? (<name> -- flag)

Returns TRUE if a word with the given name is defined in the dictionary. Usually used with .IF.

EXISTS? ASM .IF ." Assembler loaded!" .THEN

Related words: .NEED .IF .THEN FIND FINDNFA

EXIT (--)

Immediately exit from a colon definition. This is not a smart word. It basically just compiles an RTS (Return from Subroutine) instruction. Don't use it from inside a DO LOOP. Use RETURN instead. This word should not be used unless absolutely necessary. It is difficult to port to other Forths and is often a source of mysterious problems. Try to use conditionals instead.

EXPECT (addr maxchars --)

Accept characters from the keyboard and save them starting at address ADDR. Stop when the user hits <Return> or you get MAXCHARS characters. Look in the variable SPAN to see how many characters EXPECT received. EXPECT will take care of BACKSPACE keys. EXPECT is deferred. The Command Line History system changes EXPECT so that it uses the arow keys and remembers previously entered lines.

```
: ASK.INPUT  ( -- $string )
   PAD  1+ 128 EXPECT   ( get characters )
   SPAN @ PAD C!    ( set string length )
   PAD   ( return address )
;
: PASSWORD  ( -- flag )
   ." Enter password: "
   ask.input  " rasputin" $=
;
```

FALSE (-- 0 , Boolean constant)

Related words: TRUE NOT IF

FCLOSE (file-refnum -- , closes an open file)

See section on File I/O.

FENCE (-- addr)

Variable containing an address below which Forth will not FORGET. You can use this to protect the dictionary from accidentally FORGETTING. This is usually set by calling FREEZE.

FERROR (-- addr , of last file error number)

Look in this variable to see what the last file error was.

```
FOPEN  NONEXISTENT    ( returns 0 )
FERROR @ REPORT.MAC.ERROR  ( convert number to message)
```

FILE? (<name> --)

Tells you what file a word was compiled from then offers to show the source code in the editor. This does not work for words defined in the kernel.

```
FILE? HMSL.START
```

Related words: WORDS.LIKE

FILEWORD (<filename> -- addr)

Scans the input stream for a filename. If the first character is a double-quote, then it scans until the next double-quote. This allows you to specify filenames that might have spaces in it.

```
: FOPEN  FILEWORD $FOPEN ;
FOPEN  "bigdisk:music stuff:my hmsl:piece 1"
```

It is recommend that you *do not* use spaces in your filenames. It can cause problems when using command line systems like MPW or HMSL. It also makes it difficult to move files to other computers. Also avoid using '/' in filenames as most computers use this as a separator, eg. /tmp/file1.

FILL (addr n byte -- , fill memory)

Fill N bytes of memory with BYTE starting at ADDR. Useful for initializing tables.

```
PAD 20  $CC FILL
PAD 20 DUMP
```

Related words: CMOVE MOVE ERASE C!

FIND ($name -- $name 0 | cfa 1 | cfa -1)

Search the dictionary for the given name. Returns the CFA if found or the original NAME if not. A -1 implies that the word is IMMEDIATE.

```
: STAT  ( <name> -- , give info )
   BL WORD  FIND
   CASE
      0 OF $TYPE ."  not found." ENDOF
      1 OF ." Found at: " .HEX ENDOF
      -1 OF ." IMMEDIATE - Found at: " .HEX ENDOF
   ENDCASE  CR
;
```

Related words: FINDNFA

FINDNFA ($name -- $name 0 | nfa 1 | nfa -1)

Search the dictionary for the given name. Returns the NFA if found or the original NAME if not. A -1 implies that the word is IMMEDIATE. This is handy when you really want the NFA. This is much faster then using FIND and >NAME. The LFA and SFA are all next to the NFA. NAME> is fast but >NAME must search the headers to find the NFA that corresponds to the CFA. See the section on HForth dictionary structure.

FL/ (n divisor -- quotient , floored division)

Rounds toward negative infinity instead of toward zero.

```
-5 2 / . ( print -2 )
-5 2 FL/ . ( print -3 )
```

FL/MOD (n divisor -- remainder quotient , floored division)

Rounds quotient toward negative infinity instead of toward zero. Gives positive remainder which is sometimes handy.

```
-14 5 /mod swap . .  ( print -4 -2 )
-14 5 fl/mod swap . .  ( print 1 -3 )
```

FOPEN (<name> -- file-refnum | 0 , open file)

Always check for zero in case the file could not be opened. See section on file I/O.

FORGET (<name> --)

Remove the named word from the dictionary. FORGET also removes all words defined after the named word. To automatically FORGET words when recompiling, see ANEW. To execute cleanup code in case you FORGET some code, use IF.FORGOTTEN. To modify the way FORGET works, see [FORGET]

```
: W1 ." Howdy!" CR ;)
: W2 ." Har'ya?" CR ;
FORGET W1  ( removes W1 AND W2 )
```

Related words: IF.FORGOTTEN [FORGET] ANEW

FREAD (file-refnum addr maxbytes -- nbytes | -1)

Read a maximum of MAXBYTES from the specified file. Place the data or text read at ADDR. Return the actual number of bytes read. This might be less than MAXBYTES if you hit the end of the file. If there is an error, it returns -1. You can then check FERROR to see what went wrong. See the section on file I/O.

FREEZE (--)

Set FENCE to the top of the current so that FORGET will not accidentally forget below HERE. This is called by SAVE-FORTH.

Related words: SAVE-FORTH FENCE FORGET

FSEEK (file-refnum offset mode -- prev-position | -1)

Move the file cursor in the specified file. The next FREAD or FWRITE will start from the new position. Useful for rewinding a file, going to the end, or doing random access. See section on file I/O.

FWRITE (file-refnum addr nbytes -- bytes-written | -1)

Write NBYTES from memory starting from ADDR to the specified file. If there is an error, it returns -1. See the section on File I/O.

GR.xxx Graphics word.

See section on Macintosh Graphics.

GOTOXY (xcol yrow -- , move text cursor)

Move the cursor in the HMSL text window to XCOL,YROW. This is useful for making formatted tables or running status displays.

```
: RANDHI  ( -- , fill screen randomly )
   CLS  \ clear screen
   BEGIN  50 CHOOSE 10 CHOOSE GOTOXY
      ." HMSL"  ?TERMINAL
   UNTIL
;
```

HASH.ON (-- , turn ON dictionary hashing)

The dictionary is hashed which means that words are looked up using a numeric code. This make compilation much faster. Hashing is ON by default.

HASH.OFF (-- , turn OFF dictionary hashing)

HEADERS-BASE (-- addr)

Variable containing address of start of headers memory area. HForth has "separate headers" which can be easily detached when doing TURNKEY.

HEADERS-BASE @ 100 DUMP (dump lowest names)

Related words: CODE-BASE HEADERS-SIZE SAVE-FORTH

HEADERS-SIZE (-- addr)

Variable containing size of headers memory area. See SAVE-FORTH and section on HForth internals.

Related words: CODE-SIZE HEADERS-BASE SAVE-FORTH

HERE (-- addr , current dictionary location)

This is the address at which new code or data will be put. HERE returns the contents of the DP variable. Here is an example that defines a data structure and uses HERE to calculate how many entries are in the data structure. If you add items, NUM_DATA will automatically stay correct.

```
CREATE  MYDATA HERE  ( start simple array )
999 , 6543 ,  1234 ,
HERE SWAP - CELL/ CONSTANT NUM_DATA
```

HEX (--)

Set numeric base to 16. This applies to all numeric input and output.

```
HEX 9 1 + .  ( print A )
```

HOLD (char --)

Add this character to the number being built.

```
: .PHONE ( n -- , print phone number )
   S->D  <# # # # # ascii - HOLD # # # #>
   TYPE SPACE
```

```
;
5552312  .PHONE ( print "555-2312" )
```

HMSL.xxx - See the HMSL glossary in the main manual.

HISTORY (-- , print previously entered lines)

The command line history system stores old lines for editing or reentry. They can be accessed and edited using the arrow keys in HForth. HISTORY will print those lines. You can enter HISTORY in the editor then hit the <Enter> key, which will print the command line history to the file for saving.

HISTORY# (-- , HISTORY with line numbers)

HISTORY.OFF (-- , turn OFF command line history)

HISTORY.ON (-- , turn ON command line history)

I (-- index)

Return loop index of current DO LOOP.

```
: TESTI  20 10  DO I . LOOP ;
TESTI  ( print 10 to 19 )
```

Related words: J DO LOOP

ID. (<name> --)

Print name of the word whose name field address is specified.

IF (flag --)

Continuing executing the following code if the flag is true (non-zero). If the flag if false, skip to the next ELSE or THEN.

```
: TEST  ( n -- )
   0<  ( is it less than zero )
   IF  ." Negative!"
   ELSE ." Positive"
   THEN
;
```

IF is an IMMEDIATE word that compiles code to test and branch.

Related Words: ELSE THEN BEGIN WHILE REPEAT UNTIL =

IF.FORGOTTEN (<name> --)

Execute the specified word if the dictionary is forgotten below the current location. This is useful if you want to automatically cleanup some code if it is forgotten. Make sure that your cleanup word can be called twice safely. If you free memory, check first to see that it is still allocated. If you close a file, check first to make sure that it is still open.

```
VARIABLE MY-FILE
: CLEANUP
   my-file @ ?dup  \ is file open
   IF fclose
      my-file off  \ make sure we don't do this twice!
   THEN
;
IF.FORGOTTEN CLEANUP
```

Compile the above, then:

```
FOPEN filename MY-FILE !
FORGET MY-FILE   \ automatically closes it
```

Related words: FORGET AUTO.TERM

IMMEDIATE (--)

Make the word just compiled into an IMMEDIATE word. This means that it will execute during compilation when referenced. This is a powerful tool that allows you to extend the HForth compiler.

```
: SHOW.STATE  ( -- , print value of STATE )
   ." State = " STATE @ . CR
;
: FOO  123 456 SHOW.STATE + . ;  ( executes now! )
```

If you want to compile a reference to an IMMEDIATE word, you must use [COMPILE] to prevent it from executing.

```
: SS ( -- ) [COMPILE] SHOW.STATE ;
```

Related words: LITERAL STATE [COMPILE] IMMEDIATE?

IMMEDIATE? (nfa -- flag)

Return true if a word is IMMEDIATE.

```
'N SWAP IMMEDIATE? .  ( print 0 )
'N IF IMMEDIATE? .  ( print -1 )
```

INCLUDE (<filename> -- , compile file named using ASSIGN)

Open the file named and execute the contents as if it was being read from the keyboard. You can execute words or compile code. We recommend that code placed in files be restricted to that needed for compilation only. Do not execute test routines directly from the file because that could cause various problems. Files can be nested. If you have several files that go together you can use INCLUDE? to make sure they all get loaded. It is also handy to have a "load" file that consists solely of INCLUDE? commands to load what is needed. See HH:LOAD_HMSL. Also see ANEW. Don't forget you can also compile files directly from the Text Editor. Here is an example of compiling the Debugger.

```
INCLUDE  HSYS:DEBUGGER
```

Related words: ANEW ASSIGN FOPEN INCLUDE?

INCLUDE? (word <filename> -- , compile if needed)

```
INCLUDE? DEBUG HSYS:DEBUGGER
```

Related words: ANEW and INCLUDE

INDEX ($string char -- address_char true | false)

Search for character in string. If found return address and true, otherwise false.

INTERPRET (--)

Interpret the string in the TIB, Terminal Input Buffer, as Forth code. Deferred.

INSTANTIATE (<class> | object | 0)

Instantiate, create, an object of the given class. See ODE chapter.

IS (cfa <name> --)

Set deferred word to execute CFA. See section on DEFER.

ISBLACK (char -- flag , dark printing character?)

ISDIGIT (char -- flag , 0-9 ?)

ISLETTER (char -- flag , a-z or A-Z ?)

ISLOWER (char -- flag , lower case letter?)

ISSPACE (char -- flag , tab space or CR?)

ISPRINT (char -- flag , printable? not control character?)

ISUPPER (char -- flag , upper case letter?)

IV=> (value <ivar-name> --)

Set the value of an instance variable. See ODE chapter.