MESA FORTH for Atari 8Bit Handbook#

Table of Contents

Chapter 1#

Introduction#

MesaForth is based on the 6502 fig-Forth model. The major difference from the model is in the size of the screen on disk (512 bytes instead of 1024 bytes). You should refer to fig-Forth documentation for a complete description of Forth and the standard words provided in fig-Forth. This document will describe the special features provided in MesaForth.

A major difference between MesaForth and other Forth's available for ATARI is that MesaForth runs under ATARI DOS 2.0S. Forth source files and data files are written on standard DOS formatted diskettes. This means that Forth files can be interchanged with any other program that uses DOS files (ATARI BASIC, Assembler/Editor, Music Composer, ATARI Word Processor, etc.). This can make MesaForth a powerful tool in producing efficient software on your ATARI.

Your MesaForth system diskette contains more than the fig-Forth model. It also contains some additional software:

1. Complete CIO interface to ATARI Operating System and DOS
2. Support of Graphics/Sound/Joystick routines in O.S.
3. Complete support of character string manipulation
4. Interface to Disk Handler routines in O.S.
5. Screen editor for modification of Forth files
6. Text formatter for use with on-line documentation
7. Turnkey support to create application programs

Chapter 2#

Booting MesaForth#

Before you use your MesaForth disk for the first time, you should make sure it is write-protected so that you can't accidentally destroy it. It is suggested that you duplicate this disk and store the original and use it only as a master disk. You should use another disk when you are running MesaForth. To boot MesaForth, follow these steps:

1. Turn on disk drive and insert MesaForth diskette
2. Turn on any other peripherals
3. Turn on computer

DOS will boot up first. It will execute the AUTORUN.SYS file on the system disk. This will load and execute files with extensions of .IN0, .IN1, .IN2, .IN3, and .IN4 until it finds no files with these extensions. If you have an Interface Module (850), you will need the file INTRFACE.IN0 on your system disk. If you have any other special boot files, they should be loaded next. Then the MesaForth object file will be loaded and executed last (FORTH.IN1 on standard system disk). You will then see a message (fig-FORTH x.x) which announces that you are now in Forth.

At this point, you can enter any existing Forth word or enter a colon definition.

Defining system words#

When you first boot up your MesaForth, you will have the minimum subset of Forth words defined. Before you can do anything else, you will probably need to add the system extensions. These words are contained in the SYSTEM.4TH file. This file also contains in screens 4 through 6 the system error messages (you should usually leave DR0 selected to SYSTEM.4TH). This file defines some useful extensions to Forth such as character string manipulation and the :SELECT word. These will be described later.

To load these words, type:

```
1 LOAD
```

MesaForth will then go out to the disk and load the words from SYSTEM.4TH.

After it is finished loading, it will type OK. This indicates that it is finished with the previous command(s) and is ready to accept new instructions.

Defining the screen editor words#

Now that you have loaded the system words, you can load the MesaForth editor. To do this, type:

```
LOAD-ED
```

This will select EDITOR.4TH as DR1 and execute a LOAD for that file.

Defining the DOS words#

Often you will want to perform some standard DOS operations on your disk. You can do these by loading the DOS words. This can be done by typing:

```
LOAD-DOS
```

This will load the words from DOS.4TH. The words available will allow you to do directories, deletes, renames, locks, unlocks, and some copying. These words are designed to be forgotten when you are finished with them. This will release the memory used by these functions so that you can use it for you program. To forget DOS, type:

```
FORGET DOS
```

Chapter 3#

MesaForth file format#

Some differences have been made in the Forth file structure in MesaForth. First, the screen size has been modified to 512 bytes. This change is primarily due to the problem of displaying an entire Forth screen at one time. Fig-Forth suggests using a screen size of 1024 bytes (16 lines of 64 characters each). Since the ATARI line width is at most 40 characters, using 64 character lines would cause some confusion. In MesaForth, the line size is 32 characters. Each screen still has 16 lines, producing a screen size of 512 bytes.

Another major change is having Forth run under the ATARI DOS. Using DOS gives you access to files used by other ATARI DOS-supported programs. But, running under DOS adds some additional changes to Forth. First, DOS must be booted before Forth. DOS requires some memory; you lose this memory for use by your Forth programs. Also, Forth usually performs random accesses to screens on disk. But under ATARI DOS, files are normally accessed sequentially. MesaForth will simulate the random access, by creating an internal screen list showing the position of each screen within an ATARI file. After the list has been made (it is made automatically), then accesses to individual screens will be very fast (random access).

Note that since MesaForth supports all CIO functions, including the DOS file accesses, you will normally use Forth files for only your Forth sources (word definitions). Data files will usually by normal DOS files.

Chapter 4#

Character strings#

A useful extension to Forth in MesaForth is the support of character string operations. Since Forth is stack-oriented for its numeric operations, it is natural to add character string words as stack-oriented operations. A special string stack is set up separate from the numeric stack. It has its own stack pointer ($SP). The size of the string stack is defined by the variable

- $*. The string stack and words are established when you load

SYSTEM.4TH. You can change the size of the stack, or change/add words by changing SYSTEM.4TH.

Refer to Appendix A for a complete list of the string operations. For efficiency, no special checks are performed on the string stack. If you exceed its size, or pop too many elements off of the string stack, your program will probably crash. Remember, you can add these checks by modifying the words in SYSTEM.4TH. The following sections will describe the string operations provided in MesaForth.

Stack Manipulaton#

```
" text"                 ( -$> $ )
```

This operation pushes a quoted text string on top of the string stack. The string must be terminated by a double quote ("). NOTE that there "must" be a space between the first quote mark (") and the first character in the text string. Examples:

```
" This is a text string"
" D1:EDITOR.4TH"
```

```
""                      ( -$> $ )
```

This word pushes an empty (zero-length) string on top of the string stack.

```
$DROP                   ( $2 $1 -$> $2 )
```

This is like the DROP word, except it drops the top string off the string stack.

```
$DUP                    ( $1 -$> $1 $1 )
```

This is like the DUP word, except that it duplicates the top string on the stack.

```
$FILL                   ( n c --> ) ( -$> $ )
```

This operation, takes the number of characters (n) and the character (c) off the numeric stack, and produces a string at the top of the string stack which contains the indicated number of characters c. Example:

```
20 BL $FILL
```

This will put a string of 20 blanks on top of the string stack.

```
$SWAP                   ( $1 $2 -$> $2 $1 )
```

Just like the SWAP word, except that it swaps the top two strings on the string stack.

String operations#

```
$+                      ( $1 $2 -$> $1+$2 )
```

This operation is similar to the + word; it concatenates the top string on the stack onto the end of the second string on the string stack. Example:

```
" a" " b" $+
```

(this produces "ab" on the string stack)

```
$.                      ( $ -$> )
```

This prints the top string on the string stack.

```
$<                      ( --> f )  ( $1 $2 -$> )
```

This operation compares the top two string on the string stack. If $1 is less than $2, then the result (f) will be true(non-zero), otherwise the result will be false (zero).

```
$=                      ( --> f )  ( $1 $2 -$> )
```

Similar to $<, except the result will be true if $1 is equal to $2.

```
$COMPARE                ( --> n )  ( $1 $2 -$> )
```

This word is the internal comparison function. It is called by $< and $= to perform the actual comparison. It compares the top two strings on the string stack and returns -1 if $1<$2, 0 if $1=$2, and 1 if $1>$2.

```
$FETCH          ( addr len --> )  ( -$> $ )
```

This word is used to fetch strings not stored in string variables. It takes a string starting at the address (addr), and pushes it on top of the string stack. Its length will len.

```
$LEN            ( --> n )
```

This returns the length of the top string on the string stack.

```
$P!              ( --> )
```

This word resets the string stack pointer (makes the string stack empty). This is automatically done whenever a warm-restart (or cold-start) is performed. You can also do this yourself anytime, but be warned that anything that was on the string stack will be lost.

```
$P@             ( --> n )
```

This word returns the value of the string stack pointer (which is pointing to the address of the top string on the string stack).

- NOTE* that you will probably never need to use this word since all

of the string operations can be performed without knowing the address of the string stack. This word is used in defining the string operations like $DROP or $SWAP.

```
$P2              ( --> n )
```

This word returns the address of the second string on the string stack. *NOTE* that you will probably never need this word. It is used to define other string operations.

```
$STORE          ( addr max --> actlen )    ( $ -$> )
```

This word is used to store strings into non-string variable locations. The top string on the string stack is stored into memory at the address (addr). The maximum string size allowed for storage is max. This word will return the actual length of the data stored on top of the numeric stack (actlen).

```
$VARIABLE xxx                  ( len --> )
```

Similar to the VARIABLE word, except that the numeric value on top of the stack indicates how many characters to allocate to the string variable. This will limit the size of the string which may be stored in the string variable. But it does not mean that all strings need to be this size. The actual length is also kept for each string variable. Example:

```
              10 VARIABLE NAME ( define 10 character string variable
                                 called NAME )
              " Smith" NAME $! ( store 5 character string in NAME )
              NAME $VARLEN     ( this will return 5, since the actual
                                 length is only 5 )
              " Smith, Jonathan" ( this will only store the leftmost 10
                                   characters into NAME, since its max
                                   length is 10 )
```

```
$VARLEN                        ( vaddr --> len )
```

Returns the actual length of the string variable (vaddr).

```
$VARMAX                        ( vaddr --> max )
```

Returns the maximum length of the string variable (vaddr).

Chapter 5#

MesaForth Editor#

You will be using the MesaForth Editor to create and modify your Forth source files. This editor is a screen editor; it allows you to make changes to your source by using the cursor controls and making changes by typing on the screen without using special editing commands. The screen editor is similar to the functions provided by ATARI BASIC or the Assembler/Editor.

To use the editor on a source file, you must first select the file you wish to edit. This can be done using the $SETDR1 word.

For example:

```
" D1:FILE.4TH" $SETDR1 DR1
```

This will look for the file FILE.4TH on D1:. It will be used as DR1. DR1 is then selected as the active file. Now to invoke the editor, you use a special word:

```
EDIT                 ( screen --> )
```

This word invokes the editor on the indicated screen in the currently active file (DR0 or DR1). The television display will be cleared and the Forth screen will be displayed. All 16 lines of the screen will be shown, with a box around it (so that you can tell what is part of the screen, and what is information shown by the editor. The bottom of the display will show what special editor commands are available.

Standard editing commands:#

While you are in the editor, any character you type (except special control characters) while be entered into the screen at the current position of the cursor. It will replace whatever was in that position of the screen. The editor control characters are:

- arrows*

The arrows (control characters left of the RETURN key) will move the cursor the appropriate direction in the screen.

- DELETE/BACK-SPACE*

This key will delete the character preceeding the cursor in the screen.

- SHIFT DELETE/BACK-SPACE*

Holding the shift key down while hitting the DELETE/BACK-SPACE key will cause the entire line on which the cursor is positioned to be deleted. All lines below it on the screen will move up to fill in.

- SHIFT INSERT*

Holding the shift key down while hitting the INSERT key will insert a blank line at the line on which the cursor is positioned. All lines (including the line currently holding the cursor) will move down. Note that the last line in the screen will be moved outside of the screen. The editor will hold that line for you in case you accidentally inserted a line. To restore the screen to its original state, delete the line you just inserted.

- CTRL DELETE/BACK-SPACE*

Holding the ctrl key down while hitting the DELETE/BACK-SPACE key will delete the character currently under the cursor in the screen.

- CTRL INSERT*

Holding the ctrl key down while hitting the INSERT key will cause a space to be inserted at the cursor. All characters after the cursor (including the character under the cursor) will shift right one position. The editor will not allow you to shift a character off the end of the line. You will have to split the line up if it gets full and you still need to add something to it.

Special Editing commands#

There are some additional editing commands available. To enter these commands, first hit the ESCAPE key. Then enter one of the following commands:

- X* (eXit)

This command saves all of the screens that have been edited, and exits the editor mode, returning you to normal Forth command mode. You must enter this command before leaving the editor. Otherwise not all of the screens you have updated will get flushed to the disk.

- A* (Abort)

This command aborts editing of the current screen. It will save all of the other screens that have been updated though.

- C* (Copy)

This command allows you to move screens around within your file. It will ask you for the starting and ending screen numbers to be moved. It will also you ask you for the number of the first target screen. Enter each of these numbers, hitting a RETURN after each.

- E* (Erase)

This command erases the current screen. It fills the entire screen with blanks.

- S* (Search)

This command is not yet implemented, it would be a search through the file for a particular string of characters.

< (preceeding screen)

Use this command to move to the preceeding screen in the file.

> (following screen)

Use this command to move to the next screen in the file.

- number* (move directly to a screen)

After hitting the ESCAPE key, enter a number, then a RETURN. The EDITOR will move directly to that screen.

Chapter 6#

ATARI Input/Outputy#

MesaForth provides a complete set of Forth extensions interfacing to the ATARI Operating System. The Input/Output (I/O) words fall into the following categories:

1. Central Input/Output (CIO)
2. Disk Handler
3. Disk Operating System (DOS)
4. Special I/O (sound, graphics, joysticks)

Refer to Appendix A for a complete summary of these functions. The remainder of this chapter will give a brief description of the features provided in MesaForth.

CIO functions#

The interface to CIO provides both input and output similar to what is available in ATARI BASIC. The ATARI Operating System allows up to 8 files/devices to be operated at a time. They are identified by individual I/O Control Blocks (IOCB). These IOCB's are identified by #0, #1, ... #7. MesaForth and the ATARI O.S. use some of these internally. The IOCB's available for your use are #0, #3, #4, #5, and #6. These words are defined for you convenience.

The functions provided are:

CLOSE	close file/device	
GET	get character(byte)	
GETBUF	get buffer from file/device	
GETREC	get record (terminated by End-Of-Line)	
JSRCIO	call CIO (assumes IOCB set-up)	
NOTE	note position in disk file	
OPEN	open file/device	
POINT	point to position in disk file	
PUT	write character(byte)	
PUTBUF	write buffer to file/device	
PUTREC	write record (terminated by End-Of-Line)	
STATUS	return status of file/device	
XIO	call CIO (like BASIC XIO)	

For a detailed description of OPEN, CLOSE, NOTE, POINT, and XIO refer to the ATARI BASIC manual. The calling sequence and meaning of arguments is similar, with the exception of the file name arguments. The ATARI O.S. requires that file name be terminated by an EOL. A special string function is provided ($FILE) which converts the top string on the string stack into a file name, and returns the address of the name on the top of the numeric stack. You can use this address in OPEN (or XIO). Then use a $DROP to drop the file name from the string stack. Example:

```
#3 4 0 " D:FILE" $FILE OPEN $DROP
#3 GET
#3 CLOSE
```

The above example opens D:FILE and gets the first byte from the file. The file is then closed. Another word (?DISKERROR) can be used to abort your program if a disk error occurs.

Disk Handler#

The disk handler words interface to the ATARI O.S. disk handler routines. They support reading and writing of individual sectors on a disk (without using the DOS). Two extra words are defined which will dump sectors from disk, and do a sector by sector copy of a disk.

DOS Functions#

Some of the DOS functions are supported in MesaForth. You can delete, lock, rename, and unlock files. You can also do a directory of your disks. These functions use the string stack for the name of the file(s). For example:

```
" D:*.*" DIR
" D:*.BAK" DELETE
```

An additional function (SCRCOPY) can be used to create a copy of the currently selected file (DR0/DR1).

Special I/O#

Words have been defined to access the sound, graphics, and joystick functions provided by the ATARI O.S. These words are similar to the ATARI BASIC commands providing the same features:

COLOR	selects color	
DR.	draws line (DRAWTO)	
GR.	opens screen for graphics (GRAPHICS)	
LOC.	locates color at point (LOCATE)	
PL.	plots point (PLOT)	
POS.	positions graphics cursor (POSITION)	
SE.	sets color register (SETCOLOR)	
SO.	sets voice for sound (SOUND)	
STICK	tests joystick position	
STRIG	tests joystick trigger	

An additional word (CVTSTK) has been provided to convert the joystick position values to something more meaningful.

Chapter 7#

Miscellaneous Functions#

MesaForth provides two additional features useful for creating turnkey applications in Forth. A text formatter is provided to support on-line documentation to the screen or to a printer. A TURNKEY word is provided to save a set of loaded Forth words. A NEW-ABORT word allows you to chain your own functions into the warm restart sequence (SYSTEM RESET).

The Text Formatter reads a file generated using the MesaForth Editor (i.e., a Forth file of screens with no carriage returns). It looks for a small set of commands starting in column 1 of a line:

.BREAK	pause on screen (so it can be read)	
.CENTER text	center text on line	
.END	end of file	
.FILL	begin filling of text (right-margin justify)	
.NOFILL	end filling of text	
.PAGE	force a start of a new page (clear screen)	
.	force a blank line	

Use the text formatter to generate instructions for programs you have written. You call the text formatter (after it is loaded) by using the FORMAT word. It will display the text on the screen or print it on a printer. It will stop formatting if the BREAK key is hit.

To save your own turnkey programs, use the TURNKEY word. It will save the currently loaded Forth words to a binary file on disk. Usually you will have a blank disk (formatted, with DOS).

You will use TURNKEY to save your Forth program as AUTORUN.SYS. TURNKEY sets the fence so that your words cannot be deleted when you load the new file. Example:

```
" D:AUTORUN.SYS" TURNKEY
```

A useful feature is to create programs that can be run, but that can be set so that the user cannot enter Forth to see what you are doing. This can be done by using the NEW-ABORT word. To automatically call one of your words when the SYSTEM RESET key is hit (or when the program is first loaded), define the following word:

```
: START  NEW-ABORT  yourword ;
```

The START word will automatically be executed whenever Forth restarts (on SYSTEM RESET or initial load). This can be used to prevent someone from ever getting into Forth. It can also be used to reset some application specific feature (like the string stack).

Appendix A -- MesaForth Reference#

The following reference table describes all of the Forth words available on the Forth system disk. This version of the Forth Interest Group Forth (figFORTH) is based on Forth 78. The difference (from figFORTH) is the block and screen size (512 bytes). Some of the words are defined in the standard Forth object file, others are defined in Forth source files on the system disk and can be modified and loaded.

The first column contains the name of the word.

The second column describes the stack operation:

```
( input --> output )
( $input -$> $output )
```

The normal stack is described with -->, the string stack is described by -$>. The top of the stack is the rightmost item in a list. The input items reflect the stack before the word is executed. The output items indicate the stack state after the word is executed. The operands are defined as follows:

n, n1, ...	16-bit signed integer numbers	
d, d1, ...	32-bit signed integer numbers	
u	16-bit unsigned integer number	
addr	16-bit address	
b	8-bit byte	
c	8-bit ATASCII character	
f	boolean flag (0 is false)	
iocb	offset of I/O control block (i.e., hex 00, 10, 20, ...)	

The third column indicates the source of the word. The values for this column are:

fig	fig-Forth word in normal Forth object file	
ext	MesaForth extensions in Forth object file	
SYS	in SYSTEM.4TH, words usually needed	
DISK	in DISK.4TH, disk handler interface	
DOS	in DOS.4TH, ATARI DOS words	
EDIT	in EDIT.4TH, Forth screen editor	
FORM	in FORMAT.4TH, text-formatting words for use in program HELP files	
TURN	in TURNKEY.4TH, saves current Forth words for turnkey operation	

Stack Manipulation#

-DUP	(n --> n ?)	fig	Duplicate only if non-zero	
>R	(n -->)	fig	Move top item to "return stack" for temporary storage (use caution)	
DUP	(n --> n n)	fig	Duplicate top of stack	
DROP	(n -->)	fig	Throw away top of stack	
OVER	(n1 n2 --> n1 n2 n1)	fig	Make copy of second item on top	
PICK	(nm...n1 --> nm...n1 nm)	ext	Pick the mth item into the stack and copy it to the top of the stack	
R>	(--> n)	fig	Retrieve item from return stack	
R	(--> n)	fig	Copy top of return stack onto stack	
ROT	(n1 n2 n3 --> n2 n3 n1)	fig	Rotate third item to top	
SWAP	(n1 n2 --> n2 n1)	fig	Reverse top two stack items	

Number Bases#

BASE	(--> addr)	fig	System variable containing number base.	
DECIMAL	(-->)	fig	Set decimal base.	
HEX	(-->)	fig	Set hexadecimal base.	

Arithmetic and Logical#

*	(n1 n2 --> prod)	fig	Multiply.	
*/	(n1 n2 n3 --> quot)	fig	Multiply, then divide (n1*n2/n3), using double-precision intermediate.	
*/MOD	(n1 n2 n3 --> rem quot)	fig	Multiply, then divide (n1*n2/n3), using double-precision intermediate.	
+	(n1 n2 --> sum)	fig	Add.	
-	(n1 n2 --> diff)	fig	Subtract (n1-n2).	
/	(n1 n2 --> quot)	fig	Divide (n1/n2).	
/MOD	(n1 n2 --> rem quot)	fig	Divide (n1/n2), giving both remainder and quotient.	
1+	(n --> n+1)	fig	Increment number by 1.	
2+	(n --> n+2)	fig	Increment number by 2.	
ABS	(n --> absolute)	fig	Absolute value of n.	
AND	(n1 n2 --> and)	fig	Logical AND (bitwise).	
D+	(d1 d2 --> sum)	fig	Add double-precision numbers.	
D+-	(d1 n --> d2)	fig	Apply the sign of n to d1, leaving it as d2.	
DABS	(d --> absolute)	fig	Absolute value of double precision number.	
DMINUS	(d --> -d)	fig	Change sign of double-precision number.	
M*	(n1 n2 --> d)	fig	Multiplies two numbers, producing a double-precision number.	
M/	(d n1 --> rem quot)	fig	Divide double precision number by single precision number, producing single-precision numbers.	
M/MOD	(ud1 u2 --> u3 ud4)	fig	Unsigned divide of double-precision number, producing single precision remainder(u3) and double-precision quotient(ud4).	

MAX	(n1 n2 --> max)	fig	Maximum of n1 and n2.	
MIN	(n1 n2 --> min)	fig	Minimum of n1 and n2.	
MINUS	(n --> -n)	fig	Change sign of number.	
MOD	(n1 n2 --> rem)	fig	Modulo (i.e. remainder of n1/ n2).	
OR	(n1 n2 --> or)	fig	Logical OR (bitwise).	
U*	(u1 u2 --> ud)	fig	Unsigned multiplication of two numbers, producing unsigned double-precision number.	
U/	(ud u1 --> u2 u3)	fig	Unsigned divide of double-precision number by single-precision number, producing unsigned remainder (u2) and quotient (u3)	
XOR	(n1 n2 --> xor)	fig	Logical exclusive OR (bitwise).	

Comparison#

0<	(n --> f)	fig	True if number is negative.	
0=	(n --> f)	fig	True if top number zero (i.e., reverses truth value).	
<	(n1 n2 --> f)	fig	True if n1 less than n2.	
=	(n1 n2 --> f)	fig	True if n1 equals n2.	
>	(n1 n2 --> f)	fig	True if n1 greater than n2.	

Memory#

!	(n addr -->)	fig	Store word value at address in memory.	
+!	(n addr -->)	fig	Add number to value of word at address in memory.	
<CMOVE	(from to n -->)	SYS	Like CMOVE, except bytes moved starting at high address first.	
?	(addr -->)	fig	Print numeric value of word at address in memory.	
@	(addr --> n)	fig	Fetch one word from memory at indicated address.	
BLANKS	(addr u -->)	fig	Fill u bytes in memory with blanks.	
C!	(b addr -->)	fig	Store byte value at address in memory.	
C@	(addr --> b)	fig	Fetch one byte at address from memory.	
CMOVE	(from to u -->)	fig	Move u bytes in memory.	
ERASE	(addr u -->)	fig	Fill u bytes in memory with zeroes.	
FILL	(addr u b -->)	fig	Fill u bytes in memory with a byte value.	

Control Structures#

BEGIN ...UNTIL	until: (f -->)	fig	Loop back to BEGIN until true at UNTIL.	
BEGIN ...REPEAT	while: (f -->)	fig	Loop while true at WHILE. ...WHILE.	
DO...+LOOP	do: (end+1 start -->) +loop: (n-->)	fig	Like DO...LOOP, except adds stack value to index at end of loop.	
DO...LOOP	do: (end+1 start -->)	fig	Set up loop, given index range.	
I	(--> index)	fig	Place current index value on stack.	
IF...(true) ...ENDIF	if: (f -->)	fig	If top of stack is true (non-zero), execute.	
IF...(true) ELSE...(false) ...ENDIF	if: (f -->)	fig	Like IF...ENDIF, except if false, the ELSE clause is executed.	
LEAVE	(-->)	fig	Terminate loop at next LOOP or +LOOP.	

Terminal Input-Output#

.	(n -->)	fig	Print number.	
."	(-->)	fig	Print message (terminated by ").	
.R	(n fieldwidth -->)	fig	Print number, right-justified in field.	
?TERMINAL	(--> f)	fig	True if terminal break request present.	
BELL	(-->)	SYS	Ring console bell.	
BL	(--> n)	fig	Leaves the .ATASCII. value of blank on the stack.	
COUNT	(addr --> addr+1 u)	fig	Change length-byte string to TYPE form.	
CR	(-->)	fig	Do a carriage return.	
D.	(d -->)	fig	Print double-precision number.	
D.R	(d fieldwidth -->)	fig	Print double-precision number, right-justified in field.	
DUMP	(addr u -->)	SYS	Dump u .bytes. starting at address.	
EMIT	(c -->)	fig	Type character c.	
EXPECT	(addr n -->)	fig	Read n characters (or until a carriage return) from input to address.	
KEY	(--> c)	fig	Read key, put .ATASCII. value on stack.	
PR-OFF	(-->)	SYS	Turn the printer off (for terminal I/O).	
PR-ON	(-->)	SYS	Turn the printer on (for terminal I/O). All output from TYPE, EMIT, etc. will appear on the screen and the printer. All keyboard input will also be echoed on the printer.	
SPACE	(-->)	fig	Type a space.	
SPACES	(n -->)	fig	Type n spaces.	
SPEMIT	(c -->)	SYS	Allows special characters to be sent to the screen.	

TYPE	(addr u -->)	fig	Type string of u characters starting at address.	
U.	(u -->)	SYS	Type the unsigned number.	
WORD	(c -->)	fig	Read one word from input stream, stopping at character c (usually blank).	

Input-Output Formatting#

#	(d --> d)	fig	Convert next digit of double-precision number and add character to output string.	
#>	(d --> addr u)	fig	Terminate output string (ready for TYPE).	
#S	(d --> 0 0)	fig	Convert all significant digits of double-precision number to output string.	
<#	(-->)	fig	Start output string.	
HOLD	(c -->)	fig	Insert .ATASCII. character into output string.	
NUMBER	(addr - d)	fig	Convert string at address to double-precision number.	
SIGN	(n d --> d)	fig	Insert sign of n into output string.	

Disk Handling#

B/BUF	(--> n)	FIG	System constant giving disk block size in bytes	
B/SCR	(--> n)	FIG	System constant giving blocks per editing screen	
BLK	(--> addr)	FIG	System variable containing current block number	
BLOCK	(block --> addr)	FIG	Read disk block to memory address	
DR0	(-->)	FIG	Selects use of .file. 0 for LIST, LOAD, and .EDIT.	
DR1	(-->)	FIG	Selects use of .file. 1 for LIST, LOAD, and .EDIT.	
EMPTY-BUFFERS	(-->)	FIG	Erase all buffers	
FLUSH	(-->)	FIG	Write all updated buffers to disk	
INDEX	(from to -->)	FIG	Lists the first line of the screens indicated	
LIST	(screen -->)	FIG	List a disk screen (.512 bytes.)	
LOAD	(screen -->)	FIG	Load disk screen (compile or execute)	
PLIST	(strt end -->)	SYS	List the screens from strt to end to the printer (and screen)	
SCR	(--> addr)	FIG	System variable containing current screen number	
UPDATE	(-->)	FIG	Mark last buffer accessed as updated	

Defining Words#

: xxx f pointer to context vocabulary (searched first)		
CURRENT	(--> addr)	FIG
DEFINITIONS	(-->)	FIG
FORTH	(-->)	FIG
VLIST	(-->)	FIG
VOCABULARY	(-->)	FIG

Miscellaneous and System#

' xxx	(--> addr)	FIG	Find address of xxx in dictionary; if used in definition, compile address	
(	(-->)	FIG	Begin comment, terminated by right paren on same line; space after (	
,	(n -->)	FIG	Compile a number into the dictionary	
ABORT	(-->)	FIG	Error termination of operation	
ADDBLKS	(0/1 n -->)	EDIT	Adds n blocks (screens) to the end of DR0(0) or DR1(1)	
ALLOT	(-->)	FIG	Leave a gap of n bytes in the dictionary	
COLD	(-->)	FIG	Performs a system cold start (erases application program and restarts)	
EDIT	(n -->)	EDIT	Enters screen editor for screen n of the current file (DR0/DR1)	
FORGET xxx	(-->)	FIG	Forget all definitions back to and including xxx	
FORMAT	(filespec -$>)	FORM	Read the file, and format the text on the screen or printer	
FREE	(--> n)	SYS	Returns the number of free bytes left in memory	
HERE	(--> addr)	FIG	Returns address of next unused byte in the dictionary	
IN	(--> addr)	FIG	System variable containing offset into input buffer (used by WORD)	
NEW-ABORT	(-->)	SYS	This word is used when compiling another word that is to be executed on a warm-reset (like the SYSTEM RESET key). This word	

			should be the very first word in a colon definition. The remainder of the new word definition will be executed each warm-restart	
PAD	(--> addr)	FIG	Returns address of scratch area (usually 68 bytes beyond HERE)	
SP@	(n -->)	FIG	Returns address of top stack item	

String Functions#

" text"	(-$> $)	SYS	Pushes a string constant on top of the str ng stack	
""	(-$> $)	SYS	Push the empty string on top of the string stack	
!	(vaddr -->) ($ -$>)	SYS	Store the string at the top of the string stack into the string variable	
+	($1 $2 -$> $1+$2)	SYS	Concatenate the top two strings on the string stack	
+!	(vaddr -->) ($ -$>)	SYS	Concatenate the top string of the string stack onto the end of the string variable	
.	($ -$>)	SYS	Type the string at the top of the string stack	
<	(--> f) ($1 $2 -$>)	SYS	Compare the top two strings on the string stack and return true if $1 < $2	
=	(--> f) ($1 $2 -$>)	SYS	Compare the top two strings on the string stack and return true if $1 = $2	
@	(vaddr -->) (-$> $)	SYS	Fetch the string from the string variable, and place it on top of the string stack	
COMPARE	(--> n) ($1 $2 -$>)	SYS	Compare the top two strings on top of the string stack. Return 1, 0, -1	
DROP	($2 $1 -$> $2)	SYS	Drop top value from string stack	
DUP	($1 -$> $1 $1)	SYS	Duplicate the string at the top of the string stack	
EXTRACT	(vaddr offset char --> offset true) (-$> wd) --> false) (-$>)	SYS	Extracts substrings from a string, starting at the offset within the string looking for the next occurrence of the character. If there	

			is another of the character (or a substring between the offset and the end of the string variable), then true is returned along with the new offset. Otherwise false is returned. This function is useful to extract words from a sentence.	
FETCH	(addr len -->) (-$> $)	SYS	Fetch the string starting at the address, with its length as indicated, and place it on the top of the string stack	
FILL	(n c -->) (-$> $)	SYS	Create a string at the top of the string stack which has n characters (c)	
LEN	(--> n)	SYS	Return length of the string on top of the string stack	
P!	(-->)	SYS	Reset string stack pointer	
P@	(--> n)	SYS	Returns value of string stack pointer	
P2	(--> n)	SYS	Return the address of the 2nd string on the string stack (the string below the top string)	
STORE	(addr max --> actlen) ($ -$>)	SYS	Store the string on top of the string stack at the indicated address. The string will be stored up to the indicated maximum, the actual length of the string stored will be returned on top of the stack	
SWAP	($1 $2 -$> $2 $1)	SYS	Swap the top two strings on the string stack	
VARFILL	(vaddr c -->)	SYS	Fill the string variable with the character (c)	
VARIABLE xx	(len -->)	SYS		

			Creates a string variable with maximum length as indicated	
VARLEN	(vaddr --> len)	SYS	Return the length of the string currently in the string variable	
VARMAX	(vaddr --> max)	SYS	Return the maximum string length of the string variable	
$	(-->)	SYS	Variable containing the size of the string stack (512 byte default)	

ATARI Input/Output(CIO) Functions#

#0	(-->)	SYS	Iocb offset for #0	
#3	(-->)	SYS	Iocb offset for #3	
#4	(-->)	SYS	Iocb offset for #4	
#5	(-->)	SYS	Iocb offset for #5	
#6	(-->)	SYS	Iocb offset for #6	
FILE	(--> addr) ($ -$> $ +EOL)	SYS	Makes the string at the top of the string stack into a file name (terminated by an EOL) and returns the starting address on top of the stack. This file name can now be used by CIO functions (OPEN, XIO)	
(STAT)	(--> n)	ext	Status of previous CIO call	
?DISKERROR	(-->)	ext	Aborts and prints an error message if the previous CIO operation had an error	
CLOSE	(iocb -->)	ext	Closes file using iocb	
GET	(iocb --> b)	ext	Gets a single byte using the given iocb. Check (STAT) for End-Of-File	
GETBUF	(iocb addr len --> actlen)	ext	Performs a get buffer operation using the indicated iocb. The buffer starts at the indicated address and has the indicated length. The transfer will end when the buffer is full, or the end-of-file is reached. The actual length of the data transferred will be returned on the stack	
GETREC	(iocb addr len --> actlen)	ext	Same as GETBUF, except the transfer will be terminated at an EOL (End Of Line). This is a get record operation	

JSRCIO	(iocb cmd -->)	ext	Performs a call to the Operating System CIO routine. Iocb is the I/O Control block offset and cmd is the CIO command code. The status is stored in (STAT)	
NOTE	(iocb --> sector disp)	ext	Notes position in disk file	
OPEN	(iocb aux1 aux2 nameaddr -->)	ext	Opens file using iocb, 2 auxilliary bytes (see CIO), and name (terminated by EOL)	
POINT	(iocb sector disp -->)	ext	Points to position in disk file	
PUT	(iocb b -->)	ext	Output a single byte using the iocb	
PUTBUF	(iocb addr len -->)	ext	Outputs the buffer using the iocb. The buffer starts at the indicated address and has the given length.	
PUTREC	(iocb addr len -->)	ext	Outputs the record starting at the address, with given length. The record will be terminated by an EOL	
STATUS	(iocb --> status)	ext	Performs a status I/O operation using given iocb	
XIO	(cmd iocb aux1 aux2 addr -->)	SYS	Sames as BASIC XIO function. Calls CIO	

ATARI Forth File Functions#

LOAD	(filespec -$>)	SYS	Open the file as DR1, select DR1, and load the file (starting at screen 1)	
SETDR0	(filespec -$>)	SYS	Open the file as DR0	
SETDR1	(filespec -$>)	SYS	Open the file as DR1	
LOAD-ED	(-->)	SYS	Load the screen editor from EDITOR.4TH	
LOAD-DOS	(-->)	SYS	Load the DOS utilities from DOS.4TH	
LOAD-TURN	(-->)	SYS	Load the turnkey software generator from TURNKEY.4TH	
TURNKEY	(filespec -$>)	SYS	Save the current loaded Forth words (including the entire Forth program, but not including the DOS from DOS.SYS)	

General ATARI Input/Output Functions#

COLOR	(color -->)	SYS	Selects color to use	
CVTSTK	(n1 --> n2)	SYS	Converts joystick value to a more usable number (0=nothing, 1=up, 2=up-right, 3=right, 4=down-right, 5=down, 6=down-left, 7=left, 8=up-left)	
DR.	(x y -->)	SYS	Draws a line from the current position to the X-Y coord.	
GR.	(mode -->)	SYS	Opens screen for graphics I/O using iocb #6	
LOC.	(x y --> value)	SYS	Sets the cursor to the X-Y coord. and determines the color of that point	
PL.	(x y -->)	SYS	Sets the cursor to the X-Y coord. and plots the point	
POS.	(x y -->)	SYS	Sets cursor to the X-Y coord.	
SE.	(reg hue lum -->)	SYS	Sets a color register to the indicated hue and luminosity	
SO.	(voice pitch dist vol -->)	SYS	Sets the voice to the desired pitch, distortion and volume	
STICK	(port -->)	SYS	Reads the indicated joystick port (0,1,2,3)	
STRIG	(port --> f)	SYS	Reads joystick trigger port, returns true if trigger not pushed	

ATARI Disk Handler Functions#

COPY	(-->)	DISK	Makes a bit copy of a disk from drive 1 to drive 2. It reports the status of each read and write on the screen. It will not abort if it should encounter read errors (say from a missing sector on the disk)	
DMP	(sector -->)	DISK	Reads a disk sector into a buffer at hex 8000. It reads the sector from drive 2. The sector is then dumped to the screen	
GETSECTOR	(drive addr sector --> status)	DISK	Performs a get of the sector on the indicated disk drive (1,2,3,4). The sector (128 bytes) will be read in starting at the address. The sector number (1 to 720) indicates the disk sector. The status of the read will be returned on the stack	
PUTSECTOR	(drive addr sector --> status)	DISK	Writes a sector to disk. Similar to GETSECTOR	

ATARI DOS Functions#

DELETE	(filespec -$>)	DOS	Deletes the file(s). .Warning. there will be no confirmation like normal DOS	
DIR	(filespec -$>)	DOS	Lists a directory of the given file(s)	
FORGET DOS	(-->)	DOS	Used to forget DOS (after it has been loaded)	
LOCK	(filespec -$>)	DOS	Locks the file(s)	
RENAME	(" Dn:file,file" -$>)	DOS	Renames file(s)	
SCRCOPY	(strt end -->) (filespec -$>)	DOS	Creates a new file, copying screens from the current file (DR0/DR1) to the new file	
UNLOCK	(filespec -$>)	DOS	Unlocks the file(s)	