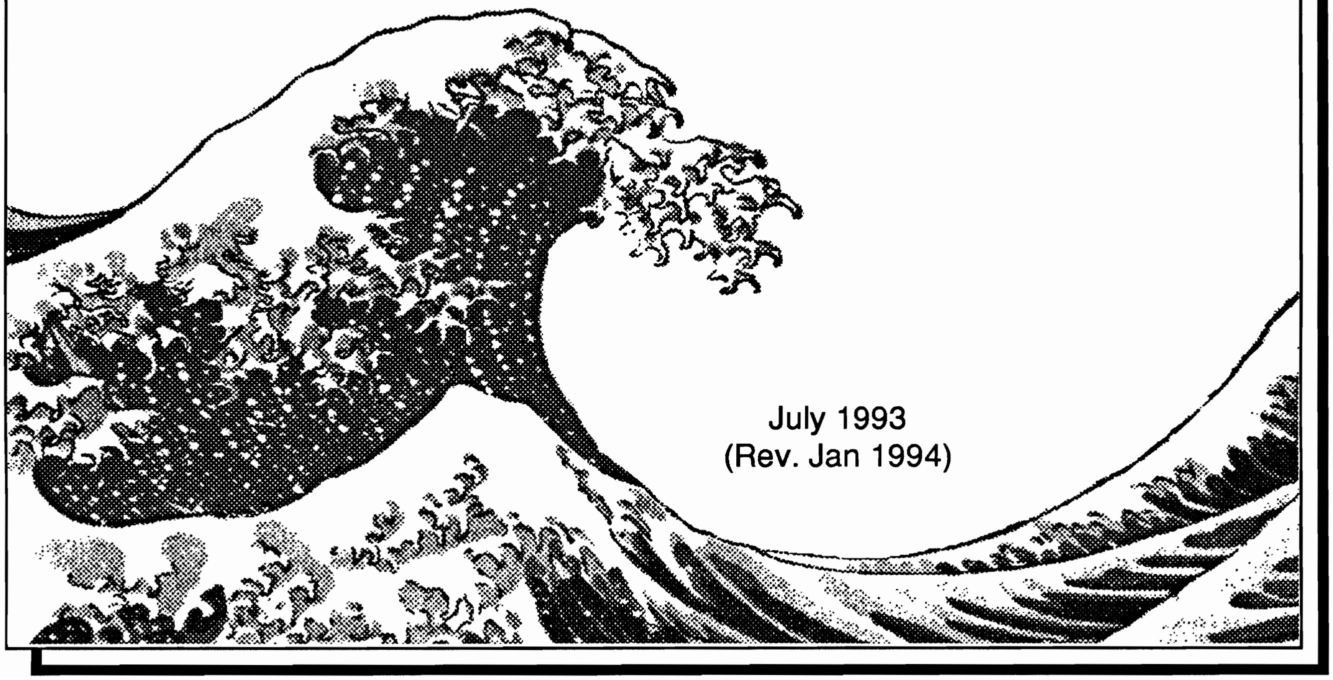
MOSS LANDING MARINE LABORATORIES
P.O. Box 450 Moss Landing, CA USA 95039-0450 (408) 633-3304

FORTH for NOAA/MLML Instruments

Richard E.Reaves and William W. Broenkow

Moss Landing Marine Laboratories Technical Publication 93-2



July 1993
(Rev. Jan 1994)

FORTH for NOAA/MLML Instruments

Richard E. Reaves and William W. Broenkow

Moss Landing Marine Laboratories

Moss Landing Marine Laboratories Technical Publication 93-2
Moss Landing, CA 95039

July 1993
(Rev. January 1994)

FORTH for NOAA/MLML Instruments

Table of Contents

page

List of Figures

FORTH for NOAA/MLML Instruments

Richard E. Reaves and William W. Broenkow
Moss Landing Marine Laboratories

Overview

This report describes FORTH software written for several instruments used in the NASA-sponsored project to design and build Marine Optical Buoy System (MOBS) and in the NOAA-sponsored project "EOS MODIS Execution: Oceanographic Profiling, Data Acquisition and Management for the Marine Optical Buoy System". In the NOAA project MLML and NOAA personnel will participate in quarterly cruises at the MOBS Hawaiian site to validate performance of SeaWiFS and will participate in several extended "process" cruises to provide wide geographic surface truthing investigations similar to those lead by Dennis Clark (NOAA) following the launch of CZCS in 1979. In the NASA project we are designing and building MOBS, a high resolution spectroradiometer that will operate autonomously in a buoy moored west of Lanai in the Hawaiian Islands. That instrument, the "Marine Optical System" (MOS), will transmit by cellular phone in near real time observations of upwelled radiance and downwelled irradiance from three depths.

During the EOS MODIS cruises several instruments have hardware and firmware designed and built at MLML: the MLML CTD-Rosette, MOS, and the NOAA/MLML "Surface Irradiance Spectrometer" (SIS). SIS was designed and built at MLML by Mark Yarbrough. He chose to use the Onset TattleTale Model 7 because of its low power, superior performance, built-in real-time clock, large RAM, the TPU which will control the fiber optics multiplexer, multiple serial interfaces, disk, and good factory support, among other factors. For that system the first author wrote a FORTH core and implemented data acquisition commands to transmit data under control of data acquisition computer. Mike Feinholz wrote a high level (VMS) data acquisition program (Feinholz and Broenkow, 1993) that archives data in the MLML_DBASE format (Broenkow and Reaves, 1993).

During the past three years, Dennis Clark and Mark Yarbrough have designed and built a prototype MOS instrument. Components for this instrument had been acquired over a several year period and each was supplied with its own individual controller. Two SC spectrographs were used, each having its own 80C85 controller. Research Support Instruments installed a third 8085 that provided A/D conversion on temperature, pressure and supply voltages, which transmitted the analog and spectral data. Those controllers are no longer supported by the manufacturer, and their firmware has caused problems in reading certain pixels in the diode arrays. A fourth and different controller was supplied with the fiber optics multiplexer. The multiplexer suffered serial communication problems, and glitches in its firmware prohibited smooth multiplexer control.

In building the second generation MOS instruments, which will be used both in the MOBS buoy and as a free-standing profiling instrument, we chose to replace the multitude of controllers by a single robust CPU, the TattleTale Model 7. Our experience with OEM software for limited production devices suggested that we cannot always implement certain commands we consider essential, and that OEM support for their firmware is not consistent. To allow this CPU to be used in all MLML instruments (including MOS, SIS, CTD/Rosette and the buoy), we needed to assemble a FORTH vocabulary that is common in all instruments. Because we anticipate that our future requirements may change, choice of the FORTH environment allows us to download new commands remotely. Thus we will be able to reprogram MOBS via cellular phone. FORTH provides a multi-tasking, interactive and flexible environment in contrast with the commercially available cross-compilers that requires an external computer such as the MacIntosh to create and download programs which is time consuming. Because we find the need to have online storage of large data sets we chose to implement the MS-DOS disk capabilities in FORTH.

Since the instrumentation and its use are rapidly evolving, this report will require periodic updating. This project has been time consuming, but will pay off by allowing us control over all functions.

FORTH Organization

The FORTH dictionary for the Onset TattleTale Model 7 (TT7), which uses the Motorola 68332 CPU, was developed from several sources. First the 8085 FORTH provided by Research Support Instruments, Inc. which was then modified for use on MLML's MOS instrument. This is a Fig-FORTH version. The second source was from FORTH, Inc. Target Compiler for the 68332. The drawback of the target compiler is that the 68332 must communicate with an IBM PC using the Background Debug Mode (BDM) interface built into the CPU. The goal was to have FORTH on the TT7 to run independently of any external machines. Consequently, the entire FORTH core was rewritten based on the source code from these two sources. Additional words were provided to make use of the 68332 and TT7 features such as the Flash EPROM, A/D converter, Time processor unit (TPU), and Serial EEPROM.

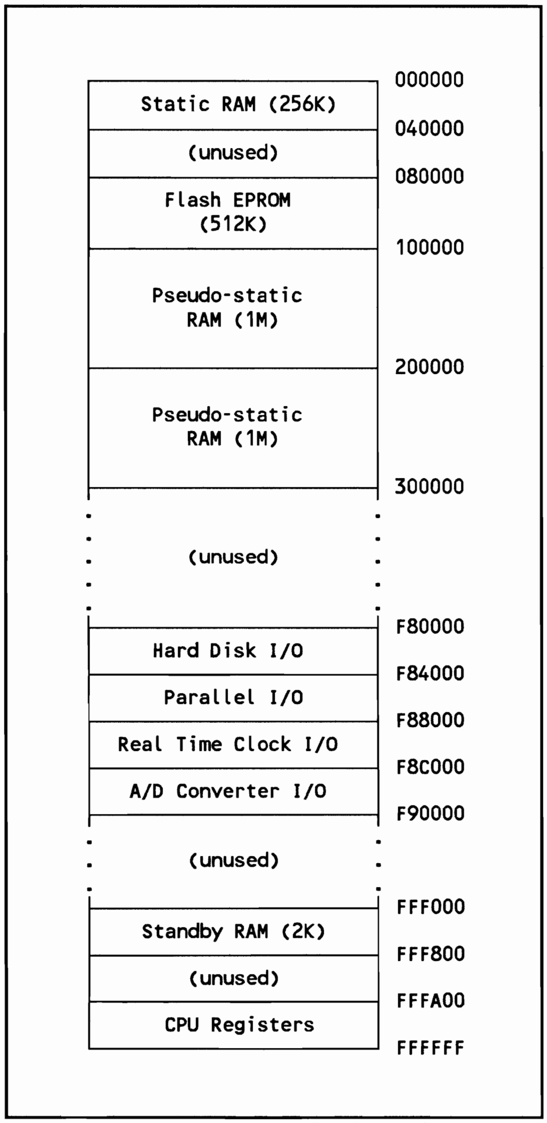


Figure 1. Memory organization of the TT7. Hexadecimal addresses are shown on the right.

The source code is available from the authors, and it contains 28 files totalling about 330K which can be put on a IBM PC 360K floppy disk. The assembler used in this project came from the Motorola Freeware Bulletin Board and was modified for use on the MLML's VAX. This assembler may also be obtained from the authors.

The organization of the TT7 contains three components: RAM, EPROM and I/O. The RAM is divided into three sections: Static, Pseudo-static and Standby (Figure 1). These sections and EPROM placement were governed by the TT7 schematic. FORTH uses the static RAM for stacks, user area, buffers, system variables and CPU operations (Figure 2).

The FORTH language is organized into two sections: core and RAM. The core section contains the basic FORTH core that is common to all instruments developed by MLML. This resides in the Flash EPROM. The second section, RAM, contains routines specific to each instrument. Although it also resides in the Flash EPROM, it is copied to Static RAM and can be modified there and burned into the EPROM. This allows us to add words to the TT7 FORTH core dictionary without need of an external interface. Updating the FORTH core requires use of an external machine such as a PC interfaced through the BDM. That requirement can be eliminated by sending the code via the console port and manually burning the EPROM, though this is time consuming.

Features on the TT7

The architecture and operation of the Motorola 68332 microprocessor are described in detail in the Motorola manuals. Motorola (1990a) describes the CPU registers in more detail as well as the CPU architecture. Motorola (1990b) describes the instruction set, exceptions, and the BDM interface on the CPU. This is mainly used for assembly programming. Motorola (1990c) describes in detail the Time Processing Unit that is built into the CPU chip. Includes description of built-in time functions. The Motorola manuals may be ordered from the Motorola Literature Distribution, P.O. Box 20912, Phoenix, AZ 85036.

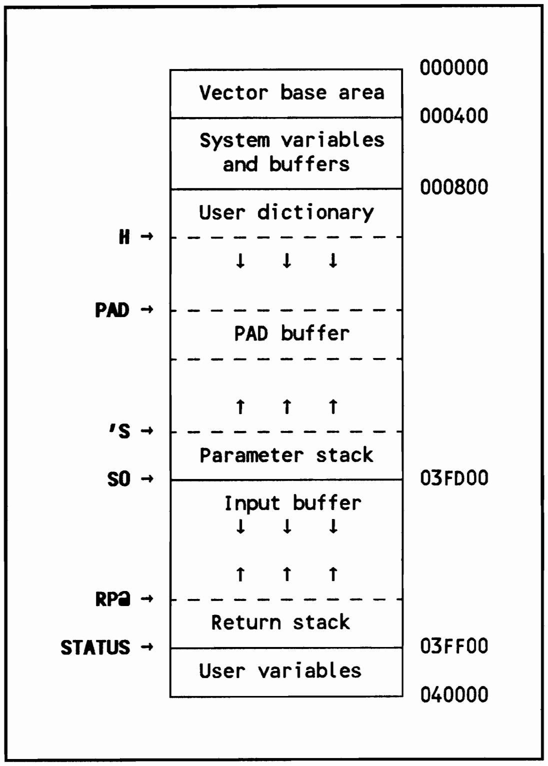


Figure 2. Memory organization of the FORTH RAM area. FORTH words that points to memory areas are shown on the left with hexadecimal addresses on the right.

In addition to the Motorola 68332 CPU, the TT7 contains additional components that enhance its usefulness. These include the SDA1812 A/D converter, ICM7170 real time clock, CAT35C104 serial EEPROM, and Conner CP-2084 disk drive.

A/D Converter

A Siemens SDA1812D 12-bit A/D converter is also included on the TattleTale Model 7. This chip can read four channels one at a time and return the 12-bit result. This is accomplished by using the FORTH word **SDA**. In case of timeouts, **SDA** returns the value given by **ADTIMEOUT**. All four channels of the A/D converter may be printed using **.SDA**.

To make use of the SDA chip from an assembly routine, the address returned by **sdasr** points to a subroutine also used by **SDA**.

Hard Disk Drive

An 80 megabyte Conner CP-2084 3.5" disk drive is attached to each of the TattleTale Model 7. A series of low level FORTH words were created to make use of this drive to read and write sectors and obtain information about the drive and registers:

DRIVE	Turn on/off drive
.DINFO	Print drive information
.DSTAT	Print drive registers
DREAD	Read sector(s)
DWRITE	Write sector(s)

Other low level words are used in the above but may be used for other purposes:

DRIVE.TABLE	Returns disk configuration
DINFO	Copies disk info to buffer
DREADY	Checks drive status
DSECTOR	Positions drive for read/write

Using these low level routines, additional FORTH words are available to setup and access the disk drive in DOS (IBM PC) format. The DOS disk format and directory structure was implemented here from material provided in Angermeyer, et. al. (1989, pp. 577-618). This allows the drive to be connected to a PC-compatible computer so files can be transferred easily between the PC and the drive. This connection is accomplished using the National Instrument PC-DIO-24 card and a appropriate driver for the PC. The following words operates on files:

FOPEN	Opens a file for read/write
FSEEK	Sets the file position
FREAD	Reads data from file
FWRITE	Writes data to file
FGETS	Reads a string from file
FPUTS	Writes a string to file
FEOF	Sets the End of File marker
FCLOSE	Closes the file

Other words, which emulate their DOS counterparts, may be used for disk and file handling:

FORMAT	Formats disk
CHKDSK	Checks contents of disk
FTYPE	Displays ASCII contents of file
FDUMP	Displays binary contents of file
COPY	Copy files
DEL	Remove files from disk
REN	Rename files
MKDIR or **MD**	Create a sub-directory
CHDIR or **CD**	Set current directory
RMDIR or **RD**	Remove a sub-directory
DIR	Display content of directory

The DOS disk format comprises the disk header sector, the File Allocation Table (FAT) and the root directory. The data following the root directory are grouped in clusters. Each cluster contains a number of sectors (in powers of two) depending on the disk size. In this case of 80-Mb drive, the cluster size is 4 sectors. These parameters, including the number of FAT sectors, are calculated in **FORMAT**.

Real Time Clock

The TattleTale Model 7 provides date and time from an onboard Real Time Clock using the Harris Semiconductor ICM7170 chip. FORTH words are available to set and retrieve date and time from the Real Time Clock:

!DATE	Sets the date
@DATE	Gets the date
!TIME	Sets the time of day
@TIME	Gets the time of day

Leap years are also accounted for. Since only the last two digits of the year are used, the years range from 1980 to 2079. The time of day is based on the 24-hour clock.

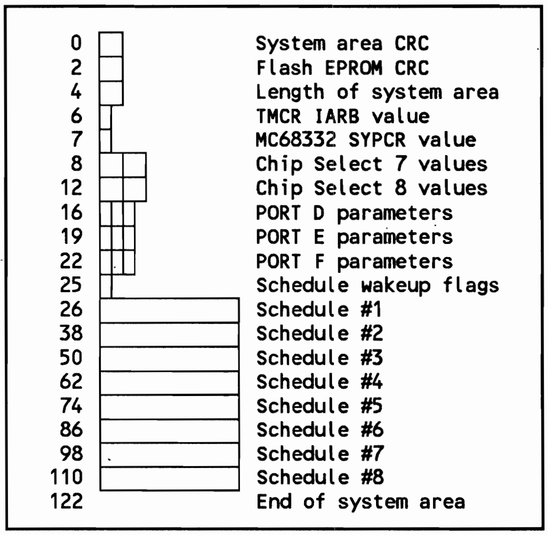
The Real Time Clock also provides scheduling by using its alarm registers. The TT7 was modified so it can turn itself off and wakes up at a later time set in the alarm. The following words provides this function:

SETALARM	Sets the alarm registers
?ALARM	Check alarm interrupt flag
.ALARM	Print content of alarm registers
?SLEEP	Check if can turn off TT7
SLEEP	Sets alarm and turn off TT7
SCHED	Sets the schedule for alarm interrupts
.SCH	Print schedules
SCANSCHED	Scan schedule upon alarm interrupt and execute routines associated with the interrupt

Note the scheduling information are stored in the system area of the Serial EEPROM described below. The entire contents of the Real Time Clock may be printed using **.RTC**.

Serial EEPROM

A 4096-bit serial EEPROM chip is included with the TattleTale Model 7 for storing permanent data even when the power is off. This uses the Catalyst Semiconductor CAT35C104H chip connected to the CPU's QSPI interface. The chip is configured for 512 byte addressing range and the data can be written and read using the FORTH words **!SEE**, **!SEE2**, **!SEE4**, **@SEE**, **@SEE2** and **@SEE4**. The entire contents of the serial EEPROM may be printed using **.SEE**.



0	System area CRC
2	Flash EPROM CRC
4	Length of system area
6	TMCR IARB value
7	MC68332 SYPCR value
8	Chip Select 7 values
12	Chip Select 8 values
16	PORT D parameters
19	PORT E parameters
22	PORT F parameters
25	Schedule wakeup flags
26	Schedule #1
38	Schedule #2
50	Schedule #3
62	Schedule #4
74	Schedule #5
86	Schedule #6
98	Schedule #7
110	Schedule #8
122	End of system area

Figure 3. Serial EEPROM System Area Map. The values on the left represent SEEPROM addresses.

Memory organization of the serial EEPROM is divided in two parts: system area and user area. These two parts are evenly divided into 256 bytes each.

The first half (address 000 to 0FF) of the EEPROM is the system area, which contains parameters for the system upon startup and is identical in all TattleTale units (Figure 3).

The second half (address 100 to 1FF) of the EEPROM is the user area. This is used to store parameters that are specific for each instrument. For example, the calibrated half-step counts for each multiplexer positions for the MOS can be stored.

TPU Serial Interface

Onset has provided the Time Processing Unit (TPU) microcode that allows the TPU to function as asynchronous serial interfaces. Up to 7 paired serial interfaces (one input and the other output) or up to 15 one-way serial interfaces may be used. Each TPU channel can act as either an input (RXD) or output

(TXD) channel. The FORTH words that provide this function are:

TSEROPEN	Opens a TPU channel for serial input or output
TSERBAUD	Sets the baud rate
TSERPAIR	Link input and output channels for handshaking
TSERXSHAKE	Enable/disable XON/XOFF handshaking
TSERFLUSH	Empties the serial buffer
TSERTIMEOUT	Sets the timeout value
TSERLEN	Gets the number of bytes residing in the buffer
TSERGET	Gets a received character
TSERPUT	Sends a character
TSERPUTS	Send a string of characters
TSERCLOSE	Closes the TPU channel

TSEROPEN must be executed first before any other words can be used. This allocates a portion of the user dictionary to be reserved for buffering input or output.

Glossary of terms

BDM
An acronym for Background Debug Mode, which is part of the 68332 CPU. This allows data transfers between an external machine (with appropriate driver) and the CPU. Used for downloading updated FORTH core.

Cell
A memory unit used for general storage. For the 68332 FORTH, this is 4 bytes long.

Colon definition
Creates a new FORTH word to execute a group of FORTH words. This begins with the FORTH word : (colon) and ends with ; (semicolon).

Console
An external device that acts as a terminal. All communications through this port use the SCI.

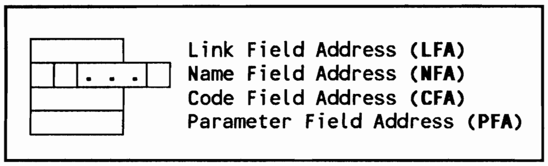


Figure 4. Structure of a dictionary entry.

CRC
An acronym for Cyclic Redundancy Check. This 16-bit checksum is used for checking data integrity in the Serial EEPROM, data transfers across serial interface, etc.

Dictionary
Contains all FORTH words in the system. Each dictionary **entry** (Figure 4) contains the following:

The **LFA** is a cell that contains the pointer to the LFA of the previous word. Used in searching for matching words until a zero encountered.

The **NFA** contains the length byte and the word (up to 31 bytes long) with the last character having the 7th bit set. A zero may be padded after the last character to make the entire length of the NFA an even number (as required by the 68332 bus addressing). The length byte actually contains a series of bits:

Bit	Description
0-4	Length of word (0 to 31)
5	Smudge bit, to hide the word from searches
6	Immediate flag bit
7	Precedence bit, indicate start of NFA

The **CFA** is a cell that contains the address of a routine that determines how to process the PFA.

The **PFA** is a cell that contains either an address of the word to be executed or a value.

Exception
A system interrupt caused by internal errors such as bus errors, invalid address, divide by zero, etc.

FORTH word
A word containing any ASCII characters (except control characters and space) identifying the operation to be performed.

Parameter stack (or simply "**stack**")
A group of cells that the FORTH word uses as "arguments" for input and output.

Pin
A one bit data I/O line that functions as either an input or an output. Used mainly for controlling and check status of external devices.

QSM

An acronym for Queued Serial Module, which is part of the 68332 CPU. This contains both the QSPI (synchronous) and SCI (asynchronous) interfaces.

QSPI

An acronym for Queued Serial Peripheral Interface, which is part of the 68332 CPU. This is a synchronous serial I/O useful for high speed communications.

Return stack

A group of cells used mainly as a return pointer after a FORTH word is executed. Also used in loops (see **DO**) and can be used as place holders from the parameter stack (see **>R** and **R>**).

SCI

An acronym for Serial Communication Interface, which is part of the 68332 CPU. This is an asynchronous serial I/O used as the console port for interactive FORTH.

SIM

An acronym for System Integration Module, which is part of the 68332 CPU. This controls the operation of the CPU such as system clock, timers, etc.

TPU

An acronym for Time Processing Unit, which is part of the 68332 CPU. Its 16 channels provide various time or serial interface functions.

User space

A portion of the memory set aside for user variables.

Vocabulary

Contains a set of selected FORTH words. Several vocabularies may exist and selecting a vocabulary may combine more than one vocabulary. Words are searched in the **context** vocabulary. New words are created in the **current** vocabulary.

References

Baker, L. and Derick, M. 1983.
Pocket Guide to FORTH. Addison-Wesley Publishing Company. 104 p.

Brodie, Leo 1987.
Starting FORTH, 2nd Ed., FORTH, Inc. Prentice Hall. 346 p.

Broenkow, W.W. and Reaves, R.E. 1993.
Introduction to MLML_DBASE programs. Moss Landing Marine Laboratories Tech. Pub 93-1. Moss Landing, CA 95039

Conner Peripherals 1991.
CP2084 Intelligent Disk Drive Product Manual. Conner Peripherals, Inc. 61 pp.

FORTH Inc. 1986.
polyFORTH ISD-4, Reference Manual and CPU Supplement. FORTH, Inc.

McCabe, C. Kevin 1983.
FORTH Fundamentals, Vols 1 and 2. dilithium Press. 365 p.

Motorola, Inc. 1990a.
MC68332 User's Manual. Part number MC68332UM/AD

Motorola, Inc. 1990b.
CPU32 Central Processing Unit Reference Manual. Part number CPU32RM/AD REV 1

Motorola, Inc. 1990c.
TPU Time Processing Unit Reference Manual. Part number TPURM/AD

Onset Computer Corp. 1991.
Tattletale Model-7 Hardware Reference Manual. North Falmouth, MA 02556-1030.

The Waite Group 1989.
The Waite Group's MS-DOS Developer's Guide, 2nd Ed. Howard W. Sams and Company. 783 p.

FORTH Dictionary

Details on using the FORTH language is beyond the scope of this report. Brodie (1981) provides an excellent treatment of FORTH for beginning and advanced programmers, while McCabe (1983), Baker and Derick (1983), FORTH Inc. (1986) are more technical. The FORTH dictionary is organized in the following manner:

Word **Stack Diagram**
Description...

Word represents the FORTH word used to perform a specific operation. Note that the FORTH words are case sensitive -- they must be typed exactly as shown. Note the delimiters between words are spaces or tabs. Do not put spaces between the characters in the word.

Stack Diagram shows the input requirements and output results of the parameter stack which is separated by the hyphens ("--"). The contents of the stack diagram are ordered so the last value in the list is at the top of the stack. The following lists the symbols found in the stack diagram:

Symbol	Stack values	Length in bits
a	Memory address	32
b	Unsigned byte-precision number	8*
c	ASCII character	8*
d	Signed double-precision number	64
f	Boolean flag	32
ff	Boolean false flag (0)	32
h	Signed half-precision number	16*
n	Signed single-precision number	32
t	Signed triple-precision number	96
tf	Boolean true flag (-1)	32
u	Unsigned single-precision number	32
ud	Unsigned double-precision number	64
uh	Unsigned half-precision number	16*

Note: * indicates the numbers are padded with zeros or sign-extended to 32 bits.

For example: (a -- n)

The input required by the word is a memory address *a*. The output result is left on stack as a 32-bit number *n*.

A vertical bar | may be used to separate a group of stack values. The action(s) or the result(s) depends on the returned boolean flag.

Lastly, **Description** explains the action of the FORTH word. Some examples are included to clarify the use of the FORTH words. With these examples, the **boldface** indicates input from the keyboard, and the *italics* indicates output to the terminal. FORTH usually terminates the output with an *ok*. Related word(s) are FORTH words that perform similar or opposite actions.

! (n a --)

Store a single-precision number *n* at the address *a*.

Related word: @

!CSP (--)

Stores the current parameter stack pointer in the user variable **CSP**. This is used for error checking during compiling. The stack pointer should be the same before and after compilation.

Related words: **?CSP CSP**

" (quote) (-- a b)

Format: " ..."

An immediate word that returns the address *a* and length *b* of the string. The " must be followed by a space, then a string of characters, and finally terminate with an ending quote ("). The beginning and ending quotes as well as the space after the beginning quote are not saved in the string. The string may contain up to 255 characters.

In the colon definition, the string is compiled inside the definition, and the address and count will be put on the stack upon execution of the word.

If not in a colon definition, the string is stored at **HERE**+128 and the address and count is put on stack. Overwriting this string can be easily done, so this should be used immediately or it may be lost.

Example:

" This is a string." TYPE *This is a string. ok*
: HELLO " Hello" TYPE ; *ok*
HELLO *Hello ok*

Related word: **."**

(ud1 -- ud2)

Format: **<# ... # ... #>**

Takes the least significant digit from the double-precision number *ud1*, converts to an ASCII character and puts it in memory for output. The value of this digit depends on the value in **BASE**. The result on stack *ud2* is the input *ud1* divided by the value in **BASE** and is used for further processing.

Example:

DECIMAL 123. <# # # # #> TYPE *23 ok*

Related words: **<# #S SIGN #ASC HOLD #>**

#> (ud -- a n)

Format: **<# ... #>**

Terminates the conversion of an unsigned double-precision number to an ASCII string. The number *ud* is dropped from stack and the address *a* and length *n* are placed on the stack.

Related words: **<# # #S SIGN #ASC HOLD**

#S (ud -- 0 0)

Format: **<# ... #S ... #>**

This puts all the digits in the double-precision value *ud* into the output buffer defined by **<#**. This is similar to repeating **#** until *ud* becomes zero. At least one digit will be converted.

Related words: **<# # SIGN #ASC HOLD #>**

' (apostrophe) (-- a)

Format: **'** *name*

Searches the FORTH dictionary for the next word *name* using **CONTEXT** vocabulary. If *name* is found, then returns the PFA of the word. In the colon definition, this address is stored in the next dictionary location. Error results if *name* is not found.

Related words: **COMPILE [COMPILE]**

'ABORT (-- a)

A user variable containing the PFA of a FORTH word to be executed by **ABORT"**. By default, **.ABORT** is used.

'CLEAN (-- a)

A user variable containing the PFA of a routine to be used in **CLEAN**. Defaults to **(CLEAN)**.

'CR (-- a)

A user variable containing the PFA of a routine to be used in **CR**. Defaults to **(CR)**.

'EXPECT (-- a)

A user variable containing the PFA of a routine to be used in **EXPECT**. Defaults to **(EXPECT)**.

'IDLE (-- a)

A user variable containing the PFA of a routine to be used in **ABORT**. Defaults to an internal routine that sets the FORTH system to its default state and executes **QUIT**.

'KEY (-- a)

A user variable containing the last input character. Note unlike other user variables which are stored as longwords, this variable stores as word only.

'MARK (-- a)

A user variable containing the PFA of a routine to be used in **MARK**. Defaults to **(MARK)**.

'PAGE (-- a)

A user variable containing the PFA of a routine to be used in **PAGE**. Defaults to **(PAGE)**.

'S (-- a)

Places the current parameter stack pointer on top of stack.

Related word: **SP@**.

'TAB (-- a)

A user variable containing the PFA of a routine to be used in **TAB**. Defaults to **(TAB)**.

'TYPE (-- a)

A user variable containing the PFA of a routine to be used in **TYPE**. Defaults to **(TYPE)**.

((--)

Used to enclose a comment in the source code. It must terminate with a ending parenthesis ()).

Example:

(this is a comment) *ok*

(CLEAN) (--)

A default routine used by **CLEAN** for a "dumb" terminal. It overwrites remainder of line with spaces and return cursor to beginning of line.

(CR) (--)

A default routine used by **CR** for a "dumb" terminal. It just sends a carriage return (ASCII 13) and a line feed (ASCII 10).

(ECHO) (f c --)

Outputs character *c* to the terminal and if the flag *f* is true, outputs space and backspace characters to erase (i.e rubout) the last character outputted. This is used in **(EXPECT)** to echo input characters.

Note: **PTR** and **CTR** are preserved before outputting characters and then restored.

(EMIT) (c --)

Outputs character *c* to the terminal. This is used in **(EXPECT)** to send a character if the input buffer length is negative (see **EXPECT**).

Note: **PTR** and **CTR** are preserved before outputting characters and then restored.

(EXPECT) (c1 | -- c2 |)

A default routine used by **EXPECT**. The action depends on the user variables **CTR** and **SPAN**:

1. If **CTR** is greater than zero, then sends a character *c1* from the stack and then negates **CTR**. Note that **CTR** contains the negated value of the maximum buffer size. Consequently, this buffer size must be a negative value before executing **EXPECT**.

2. If **CTR** and **SPAN** are both not zero, then execution is performed for **STRAIGHT**. Note the XON/XOFF handshaking is disabled to allow the XON and XOFF characters to be saved in the buffer.

3. If **CTR** is zero, then checks to see if a character was sent from the console. If so, then returns the character as *c2*; otherwise returns zero.

(FIND) (a1 a2 -- a3 n tf | ff)

A word at *a1* is searched beginning at the LFA *a2* of the vocabulary. If found, then puts PFA *a3* and word length *n* on stack along with the true flag. Otherwise, only the false flag is returned.

(MARK) (a n --)

A default routine used by **MARK** for a "dumb" terminal. It sends a caret character ("^") and types the string.

(PAGE) (--)

A default routine used by **PAGE** for a "dumb" terminal. It clears the screen by executing 25 **CR**'s.

(SAVEKEY) (c1 -- f1 f2 c2)

Interprets input character *c1* in the following manner. The flags *f1* and *f2* are set to false unless otherwise noted and *c2* is used for echoing the input character. This is used in **(EXPECT)** to save a stream of characters into an input buffer.

1. If *c1* is a carriage return character (ASCII 13), set **CTR** to zero (to terminate **EXPECT**), set *c2* to a space character (ASCII 32) and set flag *f1* true.

2. If *c1* is a backspace (ASCII 8) or DEL (ASCII 127) character, set *f2* to true and *c2* to backspace character (ASCII 8).

3. All other characters are saved in buffer pointed by **PTR**. **PTR**, **CTR** and **SPAN** are each incremented by one. *c2* is a copy of *c1*.

(STRAIGHT) (c -- f)

Saves character *c* into a buffer pointed by **PTR**. **PTR** and **CTR** are each incremented by one. The flag *f* sets to true if **CTR** becomes zero (i.e. buffer becomes full). This is used in **(EXPECT)** to save characters to an input buffer. Unlike **(SAVEKEY)**, this routine does not interpret characters.

(TAB) (n1 n2 --)

A default routine used by **TAB** for a "dumb" terminal. It ignores the line number *n1* and column number *n2* and executes **CR**.

(TYPE) (--)

A default routine used by **TYPE**. It sends a string from **PTR** and **CTR**. If the XON/XOFF handshaking is enabled, the output stops when an XOFF character is received. The output is resumed upon receipt of the XON character.

***** (n1 n2 -- n3)

Multiplies two single-precision numbers, *n1* and *n2*, and leaves result *n3* on the stack. Overflow is not checked.

Related words: ***/ / M* T* U***

***/** (n1 n2 n3 -- n4)

Multiplies the single-precision numbers, *n1* and *n2*, and then divide by *n3*. The result, truncated from an integer division, is left on stack as *n4*. Overflow is not checked and divide by zero causes an exception.

Example:

DECIMAL 12 5 8 * . *7 ok*

Related words: *** */MOD /**

***/MOD** (n1 n2 n3 -- n4 n5)

Multiplies two single-precision numbers, *n1* and *n2*, and then divide by *n3*. The remainder *n4* and quotient *n5*, truncated from an integer division, are left on stack. Overflow is not checked and divide by zero causes an exception.

Example:

DECIMAL 12 5 8 */MOD . . *7 4 ok*

Related words: *** / /MOD MOD U/MOD**

+ (plus) (n1 n2 -- n3)

Adds the two single-precision numbers, *n1* and *n2*, and the result is left on stack as *n3*. Overflow is not checked.

Related words: **- D+ M+**

+! (n a --)

Adds a single-precision number to the content of address *a*. Result is left at *a*. Overflow is not checked.

Related words: **C+! H+!**

+LOOP (n --)

Format: : *name* ... **DO** ... **+LOOP** ... ;

Used only in a colon definition. Increments the loop index by a *signed* single-precision number *n* and then the limit is checked for continuation of loop. This is useful for loops counting backwards. **1 +LOOP** is equivalent to **LOOP.**

Example:

: DSPNUM 0 5 DO I . -1 +LOOP ; *ok*
DSPNUM *5 4 3 2 1 0 ok*

Related words: **DO LEAVE LOOP /LOOP**

, (comma) (n --)

Compiles a longword from the stack to the dictionary and increments its pointer by four.

Related words: **C, H,**

- (minus) (n1 n2 -- n3)

Subtracts single-precision number *n2* from *n1* (i.e. *n1* - *n2*). The result is left on stack as *n3.*

Related words: **+ D-**

-1 (-- n)

Puts the value -1 on the stack. Note this also has all 32 bits set and is identical to the true flag.

-CELL (-- n)

Puts the negative of the cell size (i.e. -4) on the stack.

-FIND (-- a n tf | ff)

A copy of the word from the input stream (delimited by whitespaces) is placed at the top of the dictionary with the first byte containing the word length. The word is then searched in both the context and current vocabularies. If found, then the PFA *a* and the word length *n* are placed on the stack along with the true flag. Otherwise just the false flag is placed on stack.

-MATCH (a1 n1 a2 n2 -- a3 n3 ff | a1 n1 tf)

Searches the string *a1* with length *n1* for the substring *a2* with length *a2.* If found, then the address of the first non-matching character *a3* and the remaining length of the string *n3* along with a false flag is placed on stack. Otherwise, *a1* and *n1* is copied to *a3* and *n3* and returns a true flag.

-TEXT (a1 u a2 -- n)

Two strings, *a1* and *a2*, with same length *u* are compared and the result is left on stack:

a1 = *a2* returns zero
a1 > *a2* returns a positive number (1)
a1 < *a2* returns a negative number (-1)

Note: The strings are compared on a cell by cell basis.

-TRAILING (a n1 -- a n2)

Ignores trailing whitespaces in the string *a* with length *n1* by adjusting the length to the last non-space character.

. (period) (n --)

Displays a single-precision number *n* to the console according to the value in **BASE,** unformatted and followed by a space.

Related words: **D. U.**

." (--)

Format: **."** ..."

An immediate word that prints the string to the console. The **."** must be followed by a space, then a string of characters, and finally terminate with an ending quote ("). The beginning and ending quotes as well as the space after the beginning quote are not saved in the string. The string may contain up to 255 characters.

Example:

: HELLO ." Hello, how are you?" ; *ok*
HELLO *Hello, how are you? ok*

Related word: **"**

.ABORT (a --)

Displays a string to the console. The first byte of the address must contain a length byte. This is the default execution for **ABORT"**.

Related words: **ABORT ABORT"**

/ (n1 n2 -- n3)

Divides a single-precision number *n1* by *n2*. The result is put on stack as *n3*. If *n2* is zero, an exception will occur.

Related words: *** */MOD /MOD M/ MOD T/ U/ U/MOD**

/LOOP (u --)

Format: **:** *name* ... **DO** ... **/LOOP** ... **;**

Used only in a colon definition. Increments the loop index by an *unsigned* single-precision number *n* and then the limit is checked for continuation of loop. **1 /LOOP** is equivalent to **LOOP**.

Example:

```
: DSPNUM 10 0 DO I . 2 /LOOP ; ok
DSPNUM 0 2 4 6 8 ok
```

Related words: **DO LEAVE LOOP +LOOP**

/MOD (n1 n2 -- n3 n4)

Returns the remainder *n3*, which has the same sign as *n1*, and quotient *n4* of *n1* divided by *n2*. If *n2* is zero, an exception will occur.

Example:

```
DECIMAL 47 11 /MOD . . 4 3 ok
```

Related words: ***/MOD / MOD U/MOD**

0 (-- n)

Puts a single-precision zero (0) on the stack.

0. (-- d)

Puts a double-precision zero (0) on the stack. This is equivalent to putting 2 single-precision zeros on the stack.

0< (n -- f)

Test a single-precision number. Returns true if it's negative, else returns false.

0= (n -- f)

Test a single-precision number. Returns true if it's equal to zero, else return false.

Related word: **D0=**

0> (n -- f)

Test a single-precision number. Returns true if it's positive, else returns false.

1 (-- n)

Puts a single-precision one (1) on the stack.

1+ (n1 -- n2)

Adds one to the single-precision number on stack.

1- (n1 -- n2)

Subtract by one the single-precision number on stack.

1. (-- d)

Puts a double-precision one (1) on the stack. This is equivalent to putting single-precision one and zero on the stack.

1COM (n1 -- n2)

Perform a 1's complement on the single-precision number on stack.

Example:

```
HEX 1FF 1COM . FFFFFE00 ok
```

2 (-- n)

Puts a single-precision two (2) on the stack.

2! (d a --)

Stores a double-precision number *d* to the memory address *a*. The higher longword goes to the lower memory address.

2* (n1 -- n2)

Multiplies the single-precision number on stack by two.

2+ (n1 -- n2)

Adds two to the single-precision number on stack.

2- (n1 -- n2)

Subtract by two the single-precision number on stack.

2/ (n1 -- n2)

Divides the single-precision number on stack by two.

2>R (d --)

Saves a double-precision number to the return stack.

Related word: **2R>**

2@ (a -- d)

Fetches a double-precision from memory address *a*. The higher longword comes from the lower memory address.

2CONSTANT (d --)

Format: **2CONSTANT** *name*

Creates a dictionary entry *name* which contains the double-precision constant specified by *d*. Executing *name* will place the number on stack.

Example:

45. 2CONSTANT 45DEG *ok*
45DEG D. *45 ok*

Related words: **CONSTANT HCONSTANT**

2DROP (d --)

Throws away the double-precision number from the top of stack.

Related word: **DROP**

2DUP (d -- d d)

Copies a double-precision number on top of the stack.

Related word: **DUP**

2H! (h1 h2 a --)

Stores two half-precision numbers to the memory address. The lower memory address contains h1.

Related words: **2H@ 2U@**

2H@ (a -- h1 h2)

Fetches two half-precision numbers from the memory address. The numbers are sign-extended from 16-bits to 32-bits. The lower memory address contained h2.

Related words: **2H! 2U@**

2OVER (d1 d2 -- d1 d2 d1)

Copies double-precision number *d1* below the top of the stack to the top of stack.

Related word: **OVER**

2R> (-- d)

Retrieves a double-precision number from the return stack.

Related word: **2>R**

2ROT (d1 d2 d3 -- d2 d3 d1)

Rotates the top three double-precision values on the stack by moving the third value *d1* to the top while shifting the upper two values down.

Related word: **ROT**

2SWAP (d1 d2 -- d2 d1)

Exchange two double-precision numbers on top of the stack.

Related word: **SWAP**

2U@ (a -- uh1 uh2)

Fetches two half-precision numbers from the memory address. The lower memory address contained uh2.

Related words: **2H! 2H@**

2VARIABLE (d --)

Format: **2VARIABLE** *name*

Creates a dictionary entry *name* which provides a memory space for the double-precision number. This space is initialized by *d*. Executing *name* will place the address on stack.

Example:

45 2VARIABLE DEG *ok*
DEG 2@ D. *45 ok*

Related words: **HVARIABLE VARIABLE**

3 (-- n)

Puts a single-precision three (3) on the stack.

4 (-- n)

Puts a single-precision four (4) on the stack.

4* (n1 -- n2)

Multiplies the single-precision number on stack by four.

4+ (n1 -- n2)

Adds four to the single-precision number on stack.

4- (n1 -- n2)

Subtract by four the single-precision number on stack.

4/ (n1 -- n2)

Divides the single-precision number on stack by four.

6 (-- n)

Puts a single-precision six (6) on the stack.

8 (-- n)

Puts a single-precision eight (8) on the stack.

: (--)

Format: **:** *name* ... **;**

This initiates the colon definition. *name* is compiled as a dictionary entry (with the smudge bit set) in the current vocabulary with the CFA containing the address to a routine to execute the words starting at PFA. The word(s) between *name* and ; are searched and if found, the CFA of each word is stored in the dictionary.

Example:

: AVG + 2/ . ; *ok*
Creates a word "AVG" to average two values and display the result.
8 15 AVG *11 ok*

Related word: **;**

; (--)

Format: **:** *name* ... **;**

This terminates the colon definition by clearing the smudge bit of *name* and store the address of the routine to exit the word to the dictionary.

Related word: **:**

;CODE (--)

Format: **:** *name* ... **;CODE** ... **C;**

Terminates the colon definition by clearing the smudge bit of *name* and store the address of the routine to invoke *name* to the dictionary. This creates a new defining word. **;CODE** sets the context to **ASSEMBLER** to compile the mnemonics of the run-time action of *name*.

Related words: **: C; ASSEMBLER**

< (n1 n2 -- f)

Compares two single-precision numbers and returns a true flag if *n1* is less than *n2*; otherwise, leaves a false flag.

Related words: **D< U<**

<# (--)

Format: **<# ... #>**

Initiates the conversion of a double-precision number to an ASCII string. This creates the string in reverse order, starting at **PAD**, and its final address is stored in the user variable **PTR**.

Related words: **# #S SIGN #ASC HOLD #>**

<BUILDS (--)

Format: **:** *name* ... **<BUILDS ... DOES>** ... **;**

Causes *name* to define another word and clears the smudge bit and stores zero (a dummy value) to the PFA. This is equivalent to **0 CONSTANT**.

Related word: **DOES>**

<CMOVE (a1 a2 u --)

Copies memory from *a1* to *a2* with length *u* one byte at a time. Memory is copied starting at the end of the string, working towards lower memory.

Related word: **CMOVE**

<MOVE (a1 a2 u --)

Copies memory from *a1* to *a2* with length *u* one cell at a time. Memory is copied starting at the end of the string, working towards lower memory. This is more faster than **<CMOVE**.

Related word: **MOVE**

= (n1 n2 -- f)

Compares two single-precision numbers and returns a true flag if *n1* is equal to *n2*; otherwise, leaves a false flag.

Related word: **D=**

> (n1 n2 -- f)

Compares two single-precision numbers and returns a true flag if *n1* is greater than *n2*; otherwise, leaves a false flag.

Related word: **D>**

>4< (n1 -- n2)

Reverses the order of bytes in the longword *n1*.

Example:

HEX 12345678 >4< . *78563412 ok*

Related words: **>< >H<**

>< (n1 -- n2)

Reverses the order of bytes in the lower half of *n1*.

Example:

HEX 12345678 >< . *12347856 ok*

Related words: **>4< >H<**

>H< (n1 -- n2)

Reverses the order of words in the longword *n1*.

Example:

HEX 12345678 >H< . *56781234 ok*

Related words: **>4< ><**

>IN (-- a)

A user variable containing the offset of the input stream.

>R (n --)

Saves a single-precision number to the return stack.

Related word: **R>**

? (a --)

Fetches a single-precision number from address *a* and prints it.

?COMP (--)

Aborts with an error message if not in compiling mode.

?CSP (--)

Aborts with an error message if the current stack pointer is not the same as in the user variable **CSP**. This is used mainly at the end of the colon definition to check if nothing remains on the stack.

Related words: **!CSP CSP**

?DIGIT (a1 -- a2 n tf | a2 ff)

Fetches a character at *a1*. If the characters is a digit according to **BASE**, returns *n* the value of the digit and a true flag on stack. Otherwise returns a false flag. The address *a1* is incremented by one in *a2* for the next character.

?DUP (n -- n n | 0)

If *n* is not zero, then copy *n* on the stack. Otherwise leave the zero on the stack. This is useful to avoid having to remove the top of the stack for zero values (i.e. **DUP IF** ... **ELSE DROP THEN** is equivalent to **?DUP IF** ... **THEN**).

Related word: **DUP**

?ERROR (f n --)

If *f* is true, then aborts with an error message specified by *n*.

Related words: **ERROR MESS**

?EXEC (--)

Aborts with an error message if not in executing mode.

?KEY (-- c)

Determines if a character has been received into the system buffer. If so, then gets the character and puts it on stack. Otherwise, leaves a false flag (zero) on stack.

Related words: **KEY EMIT**

?PAIRS (n1 n2 --)

Aborts with an error message if *n1* is not equal to *n2*.

?STACK (--)

Aborts with an error message if the parameter stack is out of bounds.

@ (a -- n)

Fetches a single_precision number from memory address *a*.

Related word: **!**

@EXECUTE (a --)

Fetches the PFA from memory address *a* and if it is not zero, executes compiled FORTH words starting at PFA.

Related word: **EXECUTE**

ABORT (--)

Resets the parameter and return stack pointers and executes the routine stored in **'IDLE**. By default, this routine sets **BASE** to 10 and resets **CONTEXT** and **CURRENT** to **FORTH** vocabulary while preserving the user dictionary. This is similar to a "warm" system reset.

Related words: **.ABORT ABORT"**

ABORT" (f --)

Format: **ABORT" ..."**

If *f* is true, then it executes the routine stored in **'ABORT**; otherwise, it does nothing. By default, this routine is **.ABORT**, which prints the compiled string (terminated by an ending quote) and executes **ABORT**. The string may contain up to 255 characters.

Related words: **ABORT .ABORT**

ABS (n -- u)

Returns the absolute value of *n*.

Related word: **NEGATE**

ACTIVATE (a --)

Used only in a colon definition. Starts the task at PFA *a* executing the remainder of the definition. This definition must have **ABORT**, **STOP**, **QUIT**, or an infinite loop such as **BEGIN ... AGAIN** before terminating with ; or the system will crash.

Example:

: START TASK1 ACTIVATE BEGIN PAUSE AGAIN ; *ok*

Related words: **BACKGROUND TERMINAL**

AGAIN (--)

Format: **:** *name* **... BEGIN ... AGAIN ... ;**

Used only in a colon definition. Completes compilation of an infinite loop (i.e. no testing done to terminate the loop). This may be terminated with **ABORT**, **EXIT** or **QUIT** or other outside intervention.

ALLOT (n --)

Increments the dictionary pointer **H** by *n* bytes. Prints an error message if the new dictionary pointer gets too close to the parameter stack.

Example:

40 ALLOT *ok*
allocates 40 bytes of the dictionary space and **H** is incremented by 40.

AND (n1 n2 -- n3)

Performs a logical (i.e. bit-wise) AND of two single-precision numbers.

Example:

HEX 1234 1FF AND . *34 ok*

Related words: **1COM OR XOR**

ASSEMBLER (--)

Selects the assembler vocabulary as the context vocabulary. See **MC68332 FORTH Assembler** for more information on using the assembler. This is useful if the user needs to use the assembler rather than FORTH to create a routine for speed and/or more efficient coding.

Related words: **;CODE CODE LABEL C;**

BACK (a --)

Computes the offset from the current dictionary pointer to the address *a* and compiles the offset as a signed word to the dictionary. This is used in the last loop words such as **AGAIN**, **REPEAT**, **LOOP** and **UNTIL** to branch back to the word after **BEGIN** or **DO**.

BACKGROUND (n1 n2 n3 --)

Format: **BACKGROUND** *name*

Sets up a background (non-terminal) task definition table containing *n1* bytes of user area, *n2* bytes of parameter stack area and *n3* bytes of return stack area. Executing *name* puts the address of the task table on the stack. This table contains an array of 2 addresses: First the pointer to the user variable area and the second is the initial parameter stack pointer.

Example:

48 64 32 BACKGROUND TASK1 *ok*

Related words: **TERMINAL BUILD**

BASE (-- a)

A user variable containing the numeric base for input and output ASCII conversions. By default, the system starts with base 10.

BEGIN (--)

Formats: **:** *name* ... **BEGIN** ... **AGAIN** ... **;**
: *name* ... **BEGIN** ... **UNTIL** ... **;**
: *name* ... **BEGIN** ... **WHILE** ... **REPEAT** ... **;**

Used only in a colon definition. This starts compiling the indefinite loop. During compilation, the address at that point is left on stack.

Related words: **AGAIN REPEAT UNTIL WHILE**

BL (-- n)

The ASCII value of a space (32) is put on stack.

BLANKS (a n --)

Fills the memory with spaces (ASCII 32) starting at *a* with length *n*.

Related word: **FILL**

BUILD (a --)

Sets up the background task RAM area from the task definition table *a*. This also connects the background task to the multitasking loop. By default, the task remains dormant until **ACTIVATE** is used. Also, inputs are disabled and the outputs are directed to the console.

Example:

TASK1 BUILD *ok*

Related word: **BACKGROUND**

C! (b a --)

Stores a byte *b* to the memory address *a*.

Related word: **C@**

C# (-- a)

A user variable containing the current column number of the cursor.

Related word: **L#**

C+! (b a --)

Increments a byte in the memory address *a* by *b*. Result is left at *a*. Overflow is ignored.

Related words: **+! H+!**

C, (b --)

Compiles a byte to the dictionary and increments its pointer by one.

Related words: **, H,**

C/L (-- n)

Puts the maximum number of columns per line on stack. Currently, this value is 80.

C; (--)

Format: **CODE** *name* ... **C;**

Terminates the assembler and returns the current vocabulary back to the context vocabulary. Also the smudge bit is cleared and the stack pointer is checked.

Related words: **CODE ;CODE LABEL**

C@ (a -- b)

Fetches a byte from the memory address *a*.

Related word: **C!**

CELL+ (n1 -- n2)

Increments the number on top of stack by cell length, which is 4.

CELL- (n1 -- n2)

Decrements the number on top of stack by cell length, which is 4.

CELLS (n1 -- n2)

Multiplies the number on top of stack by cell length, which is 4.

CFA (a1 -- a2)

Converts the PFA on top of stack to CFA.

Related words: **LFA NFA PFA**

CLEAN (--)

Clears the line on screen from cursor position to the end of line. This executes a routine stored in user variable **'CLEAN.**

CMOVE (a1 a2 u --)

Copies memory from *a1* to *a2* with length *u* one byte at a time. Memory is copied starting at the beginning of the string, working towards higher memory.

Related word: **<CMOVE**

CMOVE> (a1 a2 u --)

This is identical to **CMOVE** for compatibility with some standards.

CODE (--)

Format: **CODE** *name* ... **C;**

Creates a dictionary entry of *name* and sets the current vocabulary to **ASSEMBLER.**

FORTH words are not compiled but rather executed. See **MC68332 FORTH Assembler** for more details on using the assembler.

Related words: **;CODE C; LABEL**

COMPILE (--)

Format: **:** *name* ... **COMPILE** *fname* ... **;**

Used only in a colon definition. Upon executing *name*, the CFA of the FORTH word *fname* is compiled in the dictionary and increment its pointer by cell size.

Related words: **' [COMPILE]**

CONSTANT (n --)

Format: **CONSTANT** *name*

Creates a dictionary entry *name* which contains the single-precision constant specified by *n*. Executing *name* will place the number on stack.

Example:

100 CONSTANT GAIN *ok*
GAIN . *100 ok*

Related words: **2CONSTANT HCONSTANT**

CONSTRUCT (a --)

Sets up the terminal task RAM area from the task definition table *a*. This also connects the terminal task to the multitasking loop. By default, the task is activated with **ABORT** and waits for input from keyboard.

Example:

TERMTASK CONSTRUCT *ok*

Related word: **TERMINAL**

CONTEXT (-- a)

A user variable containing the address of the dictionary pointer to the LFA of the last vocabulary word. This is used for vocabulary word searches. Up to 8 vocabularies may be stored in the user space following *a*.

Related words: **CURRENT VOCABULARY**

CONVERT (d1 a1 -- d2 a2)

Convert an ASCII string *a1* into a double-precision number according to the value in **BASE** until the first non-digit character is encountered in *a2*. This number is accumulated in *d1* and returned as *d2*.

Related words: **?DIGIT NUMBER**

COUNT (a1 -- a2 b)

Fetches the length *b* of the string at address *a1* and increments the address by one. FORTH typically stores strings with the first byte is the string length and the string text follows.

COUNTER (-- n)

Returns the time in milliseconds since power-up or last system reset. This uses the CPU's periodic interrupt timer (PITR) which interrupts internally approximately every one millisecond and updates a system variable.

Related words: **TIMER counter**

CR (--)

Positions the cursor to the line below the current line and to the left side of the screen. This executes a routine stored in user variable **'CR**.

CREATE (--)

Format: **CREATE** *name*

Creates a dictionary entry *name*, linking it to the current vocabulary. If *name* already exist, a warning message will be printed, but it will compile anyway. The smudge bit will be set and the CFA contains the routine **VARIABLE**. Note that the PFA is not yet established.

CSP (-- a)

A user variable containing the current parameter stack pointer.

Related words: **!CSP ?CSP**

CTR (-- a)

A user variable containing the length remaining to output a string or to input characters. In the case of inputs, this value is negative until a carriage return detected or buffer pointed by **PTR** is full; both results in **CTR** being zero.

Related word: **PTR**

CURRENT (-- a)

A user variable containing the address of the dictionary pointer to the LFA of the last vocabulary word. This is used for defining new vocabulary words as well as word searches.

Related words: **CONTEXT DEFINITIONS**

D+ (d1 d2 -- d3)

Adds the two double-precision numbers, *d1* and *d2*, and the result is left on stack as *d3*. Overflow is not checked.

Related words: **+ D- M+**

D- (d1 d2 -- d3)

Subtracts double-precision number *d2* from *d1* (i.e. *d1* - *d2*). The result is left on stack as *d3*.

Related words: **- D+**

D. (d --)

Displays a double-precision number *d* to the console, unformatted and followed by a space.

Related words: **. U.**

D.R (d n --)

Displays a double-precision number *d* to the console, right-justified by width *n*.

Example:

```
345. 5 ." T=" D.R T=  345  ok
```

Related word: **U.R**

D0= (d -- f)

Tests a double-precision number. Returns true if it's equal to zero, else return false.

Related word: **0=**

D< (d1 d2 -- f)

Compares two double-precision numbers and returns a true flag if *d1* is less than *d2*; otherwise, leaves a false flag.

Related words: **< U<**

D= (d1 d2 -- f)

Compares two double-precision numbers and returns a true flag if *d1* is equal to *d2*; otherwise, leaves a false flag.

Related word: **=**

D> (d1 d2 -- f)

Compares two double-precision numbers and returns a true flag if *d1* is greater than *d2*; otherwise, leaves a false flag.

Related word: **>**

DABS (d -- ud)

Returns the absolute value of *d*.

Related word: **DNEGATE**

DECIMAL (--)

Stores 10 to the user variable **BASE** for decimal input/outputs.

Related words: **BINARY HEX**

DEFINITIONS (--)

Sets the current vocabulary to the context vocabulary to allow new words to be defined under the context vocabulary.

Example:

```
FORTH DEFINITIONS ok
```

Related words: **CONTEXT CURRENT**

DEVICE (-- a)

A user variable containing a pointer to an array of 10 PFA specific for terminal I/O. This information used by a terminal task:

Array Index	Used in routine
1	**EXPECT**
2	**TYPE**
3	**CR**
4	**PAGE**
5	**MARK**
6	**TAB**
7	**CLEAN**
8	**!BAUD**
9	**@BAUD**
10	**XSHAKE**

Related word: **TERMINAL**

DLITERAL (d --)

Format: : *name* ... **DLITERAL** ... ;

Used only in a colon definition. Compiles the routine address and the double-precision number to the dictionary. Upon execution the compiled number is fetched and placed on the stack.

Related word: **LITERAL**

DMAX (d1 d2 -- d3)

Returns *d3* as the maximum of the two double-precision numbers *d1* and *d2*.

Related word: **MAX**

DMIN (d1 d2 -- d3)

Returns *d3* as the minimum of the two double-precision numbers *d1* and *d2*.

Related word: **MIN**

DNEGATE (d1 -- d2)

Change the sign of the double-precision number on stack.

Related word: **DABS**

DO (n1 n2 --)

Formats: : *name* ... **DO** ... **LOOP** ... ;
: *name* ... **DO** ... **+LOOP** ... ;
: *name* ... **DO** ... **/LOOP** ... ;

Used only in a colon definition. Starts the indexed loop by saving the loop limit *n1* and the initial loop index *n2* to the return stack. The loops continues until the index becomes greater than or equal to the limit. Since the testing occurs at the end of the loop, the loop is executed at least once.

Example:

: DSPNUM 4 0 DO I . LOOP ; *ok*
DSPNUM *0 1 2 3 ok*

Related words: **LEAVE LOOP +LOOP /LOOP I I'**

DOES> (--)

Formats: : *name* ... **<BUILDS** ... **DOES>** .. ;
: *name* ... **CREATE** ... **DOES>** ... ;

Used only in a colon definition. Specifies the run-time behavior of the defining word *name* when used to define other words.

Related words: **<BUILDS CREATE**

DROP (n --)

Throws away the single-precision number from the top of stack.

Related word: **2DROP**

DUP (n -- n n)

Copies a single-precision number on top of the stack.

Related words: **?DUP 2DUP**

ELSE (--)

Format: : *name* ... **IF** ... **ELSE** ... **THEN** ... ;

Used only in a colon definition. Branches to the words after **THEN** in a conditional group. **IF** tests a flag on stack and if true, then executes the words between **IF** and **ELSE**. Otherwise, executes the words between **ELSE** and **THEN**.

Related words: **IF THEN**

EMIT (c --)

Outputs a character to the console.

Example:

42 EMIT * *ok*

Related words: **?KEY KEY TYPE**

EMPTY (--)

Clears the user dictionary beginning at **FENCE** to the end at **H**.

ENCLOSE (a c -- a n1 n2 n3)

Scans the string beginning at *a* until either the delimiting character *c* or a null (zero) is encountered. Returns with the offset to the first non-delimiter character *n1*, the offset to the first delimiter after a valid character *n2* and the offset to the first character not scanned *n3*.

Related word: **WORD**

ERASE (a n --)

Fills the memory with zeros starting at *a* with length *n*.

Related word: **FILL**

ERROR (n --)

Checks **WARNING** and aborts if its value is less than zero. Otherwise, outputs the word causing the error along with a question mark to the console. Additional message may be printed from *n*.

Related words: **?ERROR MESS**

EXECUTE (a --)

Executes the word starting at the PFA *a*.

Related word: **@EXECUTE**

EXIT (--)

Format: : *name* ... **EXIT** ... ;

Used only in a colon definition. Terminates the execution of *name*. Before terminating, the return stack must be the same as when *name* starts executing; otherwise, a system crash can result.

Related word: ;

EXPECT (a n --)

Receives a string of characters at *a* until either a carriage return is detected or *n* characters are received. A null character is placed at the end of string. Also the characters received are echoed back to the console.

Related words: **?KEY KEY STRAIGHT**

FENCE (-- a)

A user variable containing the top of the protected memory to prevent **FORGET** and **EMPTY** from erasing any of that memory.

FILL (a n b --)

Fills the memory with byte *b* starting at *a* with length *n*.

Related words: **BLANKS ERASE**

FORGET (--)

Format: **FORGET** *name*

Used only as an executable word. Deletes *name* and all the words created after *name* from the dictionary. The context vocabulary must be the same as the current vocabulary. Any error returns an error message.

Related word: **EMPTY**

FORTH (--)

Selects the FORTH core words as the context vocabulary.

GET (a --)

Assigns a resource, such as a hardware device, disk drive, etc., to the task by storing the task's user area address to the resource variable *a*. If the resource variable contains zero or equal to *a*, then this resource is available. Otherwise, waits until the resource becomes available.

Related word: **RELEASE**

H (-- a)

A user variable containing the current dictionary pointer.

Related word: **HERE**

H! (h a --)

Stores a half-precision number to the memory address *a*.

Related words: **H@ U@**

H+! (h a --)

Increments a half-precision number in the memory address *a* by *h*. Result is left at *a*. Overflow is not checked.

Related words: **+! C+!**

H, (h --)

Compiles a word to the dictionary and increments its pointer by two.

Related words: **, C,**

H@ (a -- h)

Fetches a sign-extended half-precision number from the memory address *a*.

Related words: **H! U@**

HCONSTANT (h --)

Format: **HCONSTANT** *name*

Creates a dictionary entry *name* which contains the half-precision constant specified by *h*. Executing *name* will place the sign-extended number on stack.

Example:

80 HCONSTANT DEFTEMP *ok*
DEFTEMP . *80 ok*

Related words: **2CONSTANT CONSTANT**

HERE (-- a)

Puts the current dictionary pointer on stack.

HEX (--)

Stores 16 to the user variable **BASE** for hexadecimal input/outputs.

Related words: **BINARY DECIMAL**

HIS (a1 a2 -- a3)

Obtain the address of another task's user variable *a3* from the task address *a1* and the current task's user variable *a2*.

HOLD (c --)

Format: **<# ... HOLD ... #>**

Places the ASCII character *c* in the output buffer.

Example:

DECIMAL 5. <# 35 HOLD #S #> *#5 ok*

Related words: **<# # #S SIGN #ASC #>**

HVARIABLE (h --)

Format: **HVARIABLE** *name*

Creates a dictionary entry *name* which provides a memory space for the half-precision number. This space is initialized by *h*. Executing *name* will place the address on stack.

Example:

45 HVARIABLE DEG *ok*
DEG H@ . *45 ok*

Related words: **2VARIABLE VARIABLE**

I (-- n)

Copies a number from the top of the return stack to the parameter stack. This also can get the index value in an indexed loop.

Related words: **I' J J' K K**

I' (-- n)

Copies a number from the cell below the top of the return stack to the parameter stack. This also can get the limit value in an indexed loop.

Related words: **I J J' K K'**

ID. (a --)

Outputs the name of the FORTH word at the NFA *a*. Letters that are omitted during compilation are filled with underscores (_).

IF (f --)

Formats: **:** *name* ... **IF** ... **THEN** ... **;**
: *name* ... **IF** ... **ELSE** ... **THEN** ... **;**

Used only in a colon definition. Tests a flag from the stack and if true, then begin executing the word immediately after **IF**. Otherwise, skips to the word after **ELSE** or **THEN**.

Related words: **ELSE THEN**

IMMEDIATE (--)

Sets the immediate bit of the most recent definition. Commonly used after ;. Words that have this bit set will execute during compilation.

Example:

: DSP ." Compiling" ; IMMEDIATE *ok*
: TMP DSP ." this routine" ; *Compiling ok*
TMP *this routine ok*

Related word: **SMUDGE**

INTERPRET (--)

Begins an indefinite loop to parse and execute or compile from keyboard inputs until the end of a null terminated string. Any errors will abort the loop.

J (-- n)

Copies a number from the second cell below the top of the return stack to the parameter stack. This also can get the index value in an outer indexed loop.

Related words: **I I' J' K K'**

KEY (-- c)

Gets a character *c* from the console. If a character is not available, then it waits until a character is received.

Related words: **?KEY EMIT EXPECT TRAIGHT**

L# (-- a)

A user variable containing the current line number of the cursor.

Related word: **C#**

LABEL (--)

Format: **LABEL** *name* ... **C;**

Creates a dictionary entry of *name* and sets the current vocabulary to **ASSEMBLER**. This does not form an executable word, but instead it returns the address of the routine. Hence, it is useful to create subroutines at the machine level.

FORTH words are not compiled but rather executed. See **MC68332 FORTH Assembler** for more details on using the assembler.

Related words: **;CODE CODE C;**

LAST (-- a)

A user variable containing the LFA of the last dictionary entry created.

LATEST (-- a)

Returns the LFA of the recent dictionary entry in the current vocabulary.

LEAVE (--)

Causes the indexed loop to terminate after executing the words after this to the end of loop. This is done by copying the loop limit to the index.

Example:

: DSPNUM 5 0 DO I 3 = IF LEAVE THEN I . LOOP ; *ok*
DSPNUM *0 1 2 3 ok*

Related words: **DO LOOP +LOOP /LOOP**

LFA (a1 -- a2)

Converts the PFA on top of stack to LFA.

Related words: **CFA NFA PFA**

LITERAL (n --)

Format: **:** *name* ... **LITERAL** ... **;**

Used only in a colon definition. Compiles the routine address and the single-precision number to the dictionary. Upon execution the compiled number is fetched and placed on the stack.

Related word: **DLITERAL**

LOOP (--)

Format: **:** *name* ... **DO** ... **LOOP** ... **;**

Used only in a colon definition. Increments the loop index by one and then the limit is checked for continuation of loop.

Example:

: DSPNUM 5 0 DO I . LOOP ; *ok*
DSPNUM *0 1 2 3 4 ok*

Related words: **DO LEAVE +LOOP /LOOP**

M* (n1 n2 -- d)

Multiplies two single-precision numbers, *n1* and *n2*, and leave the double-precision result on stack.

Related word: *** M*/ M/ T* U***

M*/ (d1 n1 n2 -- d2)

Multiplies the double-precision number *d1* and the single-precision number *n1*, and then divide by a single-precision number *n2*. The double-precision result *d2*, truncated from an integer division, is left on stack. This uses the 96-bit intermediate arithmetic. Overflow is not checked and divide by zero causes an exception.

Example:

124. 34 7 M*/ D. *602 ok*

Related words: ***/ M* M/**

M+ (d1 n -- d2)

Adds the double-precision number *d1* to the single-precision number *n*, and leave the double-precision result on stack.

Related words: **+ D+**

M/ (d n1 -- n2)

Divides the double-precision number *d* by the single-precision number *n1*. A single-precision result *n2*, truncated from an integer division, is left on stack. Divide by zero causes an exception.

Related words: **/ M*/ T/**

MARK (a n --)

Marks the current cursor on the screen and types the string *a* with length *n*. This executes a routine stored in user variable **'MARK.**

MAX (n1 n2 -- n3)

Returns *n3* as the maximum of the two single-precision numbers *n1* and *n2*.

Related word: **DMAX**

MESS (n --)

Outputs a message based on *n*. This is used to print error messages.

Related words: **?ERROR ERROR**

MIN (n1 n2 -- n3)

Returns *n3* as the minimum of the two single-precision numbers *n1* and *n2*.

Related word: **DMIN**

MOD (n1 n2 -- n3)

Returns the modulus or the remainder of *n1* divided by *n2*. *n3* has the same sign as *n1*.

Example:

124 13 MOD . *9 ok*

Related words: **/ /MOD */MOD U/MOD**

MOVE (a1 a2 u --)

Copies memory from *a1* to *a2* with length *u* one cell at a time. Memory is copied starting at the beginning of the string, working towards higher memory. This is more faster than **CMOVE**.

Related word: **<MOVE**

MOVE> (a1 a2 u --)

This is identical to **MOVE** for compatibility with some standards.

MS (n --)

Delays the execution for *n* milliseconds. This includes **PAUSE** so other tasks may execute during the delay.

NEGATE (n1 -- n2)

Change the sign of the single-precision number on stack.

Related word: **ABS**

NFA (a1 -- a2)

Converts the PFA on top of stack to NFA.

Related words: **CFA LFA PFA**

NOT (f1 -- f2)

Complements the flag on top of stack; i.e. returns true if false and vice versa. This is equivalent to **0=**.

NUMBER (a -- n | d)

Converts a number according to the value in **BASE** in the input string *a* into either a single-precision *n* (content of **PTR** is negative) or a double-precision number *d* (content of **PTR** is zero). If the number cannot be converted, aborts with an error message.

Related words: **?DIGIT CONVERT**

OPERATOR (-- a)

Returns the address of the operator task. This is the task that controls the operation of the FORTH system.

OR (n1 n2 -- n3)

Perform a logical (i.e. bit-wise) OR of two single-precision numbers.

Example:

HEX 123 7890 OR . *79B3 ok*

Related words: **1COM AND XOR**

OVER (n1 n2 -- n1 n2 n1)

Copies single-precision number *n1* below the top of the stack to the top of stack.

Related word: **2OVER**

PAD (-- a)

Returns an address for a scratch pad. In this implementation, this address is the dictionary pointer plus 256 bytes (i.e. **HERE** + 256).

PAGE (--)

Clears the entire screen. This executes a routine stored in user variable **'PAGE**.

PAUSE (--)

Waits for the completion of one multitasking loop before continuing in the task. This is useful to prevent the task from taking up CPU time.

Related word: **STOP**

PFA (a1 -- a2)

Converts the NFA on top of stack to PFA.

Related words: **CFA LFA NFA**

PICK (n1 -- n2)

Copies a single-precision number from *n1* cells below the top of the stack, not including *n1*, to the top of the stack. **0 PICK** is equivalent to **DUP** and **1 PICK** is equivalent to **OVER**.

Example:

```
11 12 13 14 3 PICK .S
11 12 13 14 11 <-Top  ok
```

Related words: **DUP OVER**

PTR (-- a)

A user variable containing the pointer to the input or output strings.

Related word: **CTR**

QUERY (--)

Initiates and reads a string from keyboard into the input buffer using **EXPECT**. This terminates when either a carriage return is received or the number of characters received equal to 80.

Related word: **EXPECT**

QUIT (--)

Terminates the current routine and enters the interpreter loop. The return stack is set to its initial value.

Related word: **ABORT**

R0 (-- a)

A user variable containing the initial return stack pointer.

Related words: **RP! RP@**

R> (-- n)

Retrieves a single-precision number from the return stack.

Related word: **>R**

R@ (-- n)

Copies a number from the top of the return stack to the parameter stack. This identical to **I**.

RELEASE (a --)

Releases a resource, such as a hardware device, disk drive, etc., by storing zero at the resource variable *a* only if the content is equal to the task's user area address.

Related word: **GET**

REPEAT (--)

Format: **:** *name* ... **BEGIN** ... **WHILE**
... **REPEAT** ... **;**

Used only in a colon definition. Branches unconditionally to the words after **BEGIN**.

Related words: **BEGIN WHILE**

ROLL (n --)

Places the *n*th stack value, not including *n*, on top of the stack while shifting the remaining stack values down. **1 ROLL** is equivalent to **SWAP** and **2 ROLL** is equivalent to **ROT**. **0 ROLL** does nothing.

Example:

```
11 12 13 14 3 ROLL .S
12 13 14 11 <-Top  ok
```

Related words: **SWAP ROT**

ROT (n1 n2 n3 -- n2 n3 n1)

Rotates the top three single-precision values on the stack by moving the third value *n1* to the top while shifting the upper two values down.

Related word: **2ROT**

RP! (--)

Sets the return stack pointer to the initial value.

Related word: **R0**

RP@ (-- a)

Places the current return stack pointer on top of the parameter stack.

S->D (n -- d)

Converts a singed single-precision number to a signed double-precision number.

S0 (-- a)

A user variable containing the initial parameter stack pointer.

Related words: **'S SP@ SP!**

SIGN (n ud -- ud)

Format: **<# ... SIGN ... #>**

If the value *n* is negative, puts a minus sign ("-") in the output buffer.

Related words: **<# # #S #ASC HOLD #>**

SMUDGE (--)

Toggles the smudge bit (bit 5) in the first byte of the most recent word's NFA. If the bit is set, the word is hidden from dictionary searches (i.e. make the word unavailable). Otherwise, the word becomes available to the user.

Related word: **IMMEDIATE**

SP! (--)

Sets the parameter stack pointer to its initial value.

Related word: **S0**

SP@ (-- a)

Places the current parameter stack pointer on top of stack. This is identical to **'S.**

Related word: **'S**

SPACE (--)

Outputs a space to the console.

Related word: **SPACES**

SPACES (n --)

Output *n* spaces to the console. **1 SPACES** is equivalent to **SPACE.**

Related word: **SPACE**

SPAN (-- a)

A user variable containing the actual number of characters input by **EXPECT.**

STATE (-- a)

A user variable containing the state of the interpreter. A zero indicates an execute mode. Otherwise indicates compiling mode.

Related words: **[ ]**

STATUS (-- a)

A user variable containing the status of the task. This is actually either the machine level jump to the next task or a wake up call to resume executing the task where it left off.

Related word: **WAKE**

STOP (--)

Suspends a task indefinitely until resumed from another task or an interrupt.

Related words: **NOD PAUSE**

STRAIGHT (a n --)

Receives a string of characters at *a* until *n* characters are received. All characters, including XON/XOFF and carriage return, are stored in the string. The characters are not echoed.

Related words: **?KEY KEY EXPECT**

SWAP (n1 n2 -- n2 n1)

Exchange two single-precision numbers on top of the stack.

Related word: **2SWAP**

T* (d n -- t)

Multiplies a double-precision number and a single-precision number, and leave the triple-precision number on stack.

Related words: *** M* T/ U***

T/ (t n -- d)

Divides a triple-precision number by a single-precision number, and leave the double-precision result, truncated from an integer division, on stack.

Related word: **/ M/ T* U/**

TAB (n1 n2 --)

Sets the cursor to the position specified by line *n1* and column *n2*. This executes a routine stored in user variable **'TAB**.

TERMINAL (a n --)

Format: **TERMINAL** *name*

Sets up a terminal task definition table based on the **DEVICE** address *a* and the **PAD** plus parameter stack size *n*. If the device is not required, use zero. Executing *name* puts the address of the task table on the stack. This table contains an array of 3 addresses: First the pointer to the user variable area, next the initial parameter stack pointer and lastly, the address of the terminal specific I/O.

Example:

EXDEV 256 TERMINAL TERMTASK *ok*

Related words: **BACKGROUND CONSTRUCT DEVICE**

THEN (--)

Format: **:** *name* ... **IF** ... **THEN** ... **;**

Used only in a colon definition. Terminates a conditional branch.

Related words: **IF ELSE**

TIB (-- a)

A user variable containing the address of the input buffer. This is used in **QUERY** and **WORD**.

TIMER (n --)

Subtracts *n* from the time in milliseconds returned by **COUNTER**. The result is output to console. This is useful for determining the time delay for an operation.

Example:

COUNTER TIMETEST TIMER *154 ok*

Related word: **COUNTER**

TOGGLE (a b --)

Performs a bit-wise exclusive-OR of the byte in address *a* with the byte mask *b*.

Related word: **XOR**

TRAVERSE (a1 n -- a2)

Scans the NFA to find the beginning or the end of the name depending on *n*. If *n* equals to 1, then returns the CFA from NFA. If *n* equals to -1, then returns the NFA from the CFA. Any other value may cause erroneous result.

Related words: **CFA NFA**

TYPE (a n --)

Outputs a string beginning at address *a* and length *n*. The routine that does the output is stored in the user variable **'TYPE**. By default, this routine is **(TYPE)** which outputs to the console.

Related words: **'TYPE (TYPE)**

U* (u1 u2 -- ud)

Multiplies two *unsigned* single-precision numbers, *u1* and *u2*, and leave the *unsigned* double-precision result on stack.

Related words: *** M* T* U/**

U. (u --)

Displays an *unsigned* single-precision number *u* to the console according to the value in **BASE**, unformatted and followed by a space.

Example:

DECIMAL -123 U. *4294967173 ok*

Related words: **. D.**

U.R (u n --)

Displays an *unsigned* single-precision number *u* to the console, right-justified by width *n*.

Example:

DECIMAL 56 5 ." X=" U.R *X= 56 ok*

Related word: **D.R**

U/ (ud u1 -- u2)

Divides the *unsigned* double-precision number *ud* by the *unsigned* single-precision number *u1*. The *unsigned* single-precision result *u2*, truncated from an integer division, is left on stack. Divide by zero causes an exception.

Example:

DECIMAL 1456 43 U/ . *33 ok*

Related words: **/ M/ T/ U***

U/MOD (ud u1 -- u2 u3)

Divides the *unsigned* double-precision number *ud* by *u1*. The *unsigned* remainder *u2* and the *unsigned* quotient *u3*, truncated from an integer division, is left on stack. Divide by zero causes an exception.

Example:

DECIMAL 1435. 22 U/MOD . . *65 5 ok*

Related words: ***/MOD / /MOD MOD**

U< (n1 n2 -- f)

Compares two *unsigned* single-precision numbers and returns a true flag if *u1* is less than *u2*; otherwise, leaves a false flag. This is useful for comparing addresses which can have signed values.

Related words: **< D<**

U@ (a -- word)

Fetches an unsigned half-precision number from the memory address *a*.

Related words: **H! H@**

UNTIL (f --)

Format: **:** *name* **... BEGIN ... UNTIL ... ;**

Used only in a colon definition. Tests the flag on stack and if false, then branches to the word following **BEGIN**. Otherwise continue to the word after **UNTIL**.

USER (n --)

Format: **USER** *name*

Creates a dictionary entry *name* with the offset *n* in the user area. This allows *name* to become a user variable. Upon execution of *name*, the address of the variable is left on stack.

Example:

128 USER TEMP *ok*
TEMP @ . *2254 ok*

Related word: **VARIABLE**

VARIABLE (n --)

Format: **VARIABLE** *name*

Creates a dictionary entry *name* which provides a memory space for the single-precision number. This space is initialized by *n*. Executing *name* will place the address on stack.

Example:

45 VARIABLE DEG *ok*
DEG @ . *45 ok*

Related words: **2VARIABLE HVARIABLE**

VOC-LINK (-- a)

A user variable containing a pointer to the LFA of the most recent vocabulary entry.

VOCABULARY (--)

Format: **VOCABULARY** *name* **IMMEDIATE**

Creates a new vocabulary *name* which links to the current vocabulary. If *name* becomes the current vocabulary, any new words will be defined under this vocabulary. Up to 8 vocabularies may be found in the system (in this implementation, 3 are used, i.e. **FORTH**, **ASSEMBLER** and **TT7**, leaving 5 additional entries).

Related word: **DEFINITIONS**

WAKE (-- n)

A constant used to store in **STATUS** of a task to resume its execution caused by **STOP**. In this implementation, this is a two byte code equivalent to JSR (A4) in assembly code.

WARNING (-- a)

A user variable containing the warning flag.

Related word: **ERROR**

WHILE (f --)

Format: **:** *name* ... **BEGIN** ... **WHILE**
... **REPEAT** ... **;**

Used only in a colon definition. Tests the flag on stack and if true, executes the words following **WHILE** and branches back to **BEGIN** at **REPEAT**. A false flag branches to the word following **REPEAT**.

Related words: **BEGIN REPEAT**

WIDTH (-- a)

A user variable containing the maximum length of the word names. The default value in this implementation is 31.

WITHIN (n1 n2 n3 -- f)

Returns a true flag if *n2* < = *n1* < *n3* (for *n2* > *n3*, either *n2* < = *n1* or *n1* < *n3*). Otherwise returns a false flag.

WORD (c --)

Parse the next word from the input buffer with *c* as the delimiter (or a null character, whichever comes first). A copy of the word, with delimiters removed, is placed at the dictionary pointer with a length of the word stored in the first byte. This is used by defining words such as **CREATE** to create a dictionary entry with the word name.

XOR (n1 n2 -- n3)

Perform a logical (i.e. bit-wise) exclusive-OR of two single-precision numbers.

Example:

HEX 123 4567 XOR . *4444 ok*

Related words: **1COM AND OR**

[(--)

Suspends compiling to execute the following words in a colon definition. This sets the user variable **STATE** to zero.

Related word: **]**

[COMPILE] (--)

Format: **:** *name* ... **[COMPILE]** *fname* ... ;

Force compilation of an immediate word *fname* to the colon definition. Upon executing *name*, *fname* will be executed as if it was executed during compilation.

Related words: **' COMPILE**

] (--)

Resume compiling in a colon definition. This sets the user variable **STATE** to a non-zero value.

Related word: **[**

counter (-- a)

Returns the system address containing the time elapsed in milliseconds since the last power-up or system reset. This is useful in writing assembly code that utilize this for timeouts, delays, etc.

Example:

CODE CNTR counter S -) MOV
NEXT *ok*

Related word: **COUNTER**

MC68332 CPU Secific FORTH Words

This section describes the FORTH words that are specific to the Motorola 68332 CPU. These words utilizes the registers and memory of the CPU. Refer to Motorola (1990a, 1990b and 1990c) manuals for more details regarding the registers and the operation of the CPU.

!CFSR (n1 n2 --)

Sends a channel function value to the TPU Channel Function Select Register (CFSR). The function value *n1* range from 0 to 15 (modulo 16). See **TPUF.xxx** for function values. The channel number *n2* ranges from 0 to 15 (modulo 16).

Related word: **@CFSR**

!CIER (f n --)

Sets or clears a bit in the TPU Channel Interrupt Enable Register (CIER) depending on the flag *f*. A true flag enables and a false flag disables interrupts from the channel. The channel number *n* ranges from 0 to 15 (modulo 16).

Related word: **@CIER**

!CISR (f n --)

Sets or clears a bit in the TPU Channel Interrupt Status Register (CISR) depending on the flag *f*. The channel number *n* ranges from 0 to 15 (modulo 16).

Related words: **@CISR ^CISR**

!CPR (n1 n2 --)

Sets the priority in the TPU Channel Priority Register (CPR). The priority value *n1* ranges from 0 to 3 (modulo 4). This value governs the priority the TPU acts upon the channel:

0 = No priority (channel disabled)
1 = Low priority
2 = Middle priority
3 = High priority

The channel number *n2* ranges from 0 to 15 (modulo 16).

Related word: **@CPR**

!HSQR (n1 n2 --)

Sends a sequence to the TPU Host Sequence Register (HSQR). The sequence values *n1* ranges from 0 to 3 (modulo 4). This selects the mode of operation of the TPU channel and is dependent on the function. The channel number *n2* ranges from 0 to 15 (modulo 16).

Related word: **@HSQR**

!HSRR (n1 n2 --)

Sends a service request to the TPU Host Service Request Register (HSRR). The request *n1* ranges from 0 to 3 (modulo 4) with 0 being no action to the function. This is used to select the microcode entry address in the TPU and the action is dependent on the function. A zero value returned in the register indicates the action is completed. The channel number *n2* ranges from 0 to 15 (modulo 16).

Related words: **@HSRR HSRRWAIT**

!PRAM (uh n1 n2 --)

Stores *uh* into the TPU Parameter RAM in which the location is specified by the channel number *n2*, which ranges from 0 to 15 (modulo 16), and the index *n2*. The index species which word to store *uh* for the given channel, and it ranges from 0 to 7 (modulo 8).

Example:

20 1 4 !PRAM *ok*
stores word value 20 to the TPU Parameter RAM at word 1 in channel 4, which makes the actual location at hex FFFFFE42.

Related word: **@PRAM**

!QBAUD (n --)

Change the baud rate (bits/second) of the QSPI (synchronous) port. The range is from 0 to the system clock divided by 4. The maximum possible rate is 4M baud. An out of range value will not cause an error; however, it may cause garbled data. The value on stack is also saved in variable **QBAUD**.

Related word: **@QBAUD**

!SYSCLOCK (u --)

Changes the system clock frequency. The maximum frequency is the maximum allowed for the CPU, which is 16,777,216 MHz. The minimum frequency is 131 KHz; however, in actual practice, the minimum should be 320 KHz. Anything lower than this would result in difficulty in communicating through the SCI (console) port.

Notes: The SCI and QSPI baud rate registers are automatically adjusted for the new system clock.

Changing the frequency to a lower value lowers power consumption.

Example:

DECIMAL 8000000 !SYSCLOCK *ok*
sets the system clock to 8 MHz.

Related word: **@SYSCLOCK**

!TPUBUF (a1 a2 n --)

Stores the buffer address *a1* and the interrupt routine address *a2* to a memory location specified by the TPU channel number *n*.

Each TPU channel can have its own interrupt, and when an interrupt occurs, the internal routine extracts the interrupt routine address. If this address is not zero, then the routine loads in the buffer address to register **A0** and then jumps to the interrupt routine.

CPU registers **A0** and **A1** are also saved in the return stack before getting the buffer and interrupt addresses. This preserves their contents upon exiting the interrupt routine. The user's interrupt routine need not save the contents of these registers.

Related words: **tpuvend** **@TPUBUF** **?TPUBUF**

'SCI (-- a)

Contains the address of the task's user space which uses the SCI as the console port. On power-up, SCI is assigned to **OPERATOR**.

.CS (--)

Prints information about the SIM chip select registers with their starting address, address length, wait states, etc.

.PIN (--)

Print contents of the CPU port D, E and F registers as binary values.

.QSM (--)

Prints contents of the QSM registers and the QSPI RAM in tabular format.

.REG (--)

Prints contents of the CPU data, status and control registers.

.SIM (--)

Prints contents of the SIM registers in tabular format.

.TPU (--)

Prints contents of the TPU registers and the parameter RAM in tabular format.

@CFSR (n1 -- n2)

Returns a channel function value from the TPU Channel Function Select Register (CFSR). The channel number *n1* ranges from 0 to 15 (modulo 16).

Related word: **!CFSR**

@CIER (n -- f)

Returns the bit value from the TPU Channel Interrupt Enable Register (CIER) as a flag *f*. The channel number *n* ranges from 0 to 15 (modulo 16).

Related word: **!CIER**

@CISR (n -- f)

Returns the bit value from the TPU Channel Interrupt Status Register (CISR) as a flag *f*. The channel number *n* ranges from 0 to 15 (modulo 16).

Related word: **!CISR ^CISR**

@CPR (n1 -- n2)

Returns the priority value from the TPU Channel Priority Register (CPR). The channel number *n1* ranges from 0 to 15 (modulo 16). See **!CPR** for information about the priority values.

Related word: **!CPR**

@HSQR (n1 -- n2)

Returns a sequence value from the TPU Host Sequence Register (HSQR). The channel number *n1* ranges from 0 to 15 (modulo 16).

Related word: **@HSQR**

@HSRR (n1 -- n2)

Returns a service request value from the TPU Host Service Request Register (HSRR). The channel number *n1* ranges from 0 to 15 (modulo 16).

Related words: **!HSRR HSRRWAIT**

@PRAM (n1 n2 -- uh)

Fetches *uh* from the TPU Parameter RAM in which the location is specified by the channel number *n2*, which ranges from 0 to 15 (modulo 16), and the index *n1*. The index specifies which word to retrieve for the given channel, and it ranges from 0 to 7 (modulo 8).

Example:

> **1 4 @PRAM .** *20 ok*
> fetches word from the TPU Parameter RAM at word 1 in channel 4, which makes the actual location at hex FFFFFE42.

Related word: **!PRAM**

@QBAUD (-- u)

Returns the actual baud rate (bits/second) of the QSPI (synchronous) port bases on the actual system clock and QSPI setting.

Related word: **!QBAUD**

@SYSCLOCK (-- u)

Returns the actual system clock frequency based on the value found in the SIM's SYNCR register.

Related words: **!SYSCLOCK sysclk**

@TPUBUF (n -- a)

Fetches the buffer address *a* from a memory location specified by the TPU channel number *n*.

Related words: **!TPUBUF ?TPUBUF**

?TPUBUF (n -- f)

Returns a true flag if the buffer specified by the TPU channel *n* is not being used.

Related words: **!TPUBUF @TPUBUF**

CPUPIN (n --)

Designate a pin to function as a CPU line. Pin number *n* ranges from 0 to 23 (modulo 24) which corresponds to the following ports:

Port D	= pins 0 to 7
Port E	= pins 8 to 15
Port F	= pins 16 to 23

This sets a bit in the CPU Pin Assignment Register (PAR). For more information see Motorola (1990a).

Example:

DECIMAL 18 CPUPIN *ok* sets bit 2 in the Port F PAR to enable interrupt request input at interrupt level 2.

Related words: **PCLR PIN PSET**

CRYSTAL (-- n)

Returns the external crystal frequency of the system. It usually ranges from 20 to 50 KHz. In this implementation, the value is 40 KHz.

CSWAIT (n1 n2 --)

Sets the wait state *n1* in the chip select register *n2*. The wait state indicates how many clock cycles to wait before storing or fetching data from memory or I/O. The value ranges from 0 to 13. The chip select register value ranges from 0 to 10. Out of range values prints an error message.

Example:

4 9 CSWAIT *ok*
sets 4 wait states to the chip select register 9.

Related word: **.CS**

EXCEPTION (a n --)

Stores the exception address *a* (i.e. the start address of an interrupt routine) to the vector *n* in the vector base area. The interrupt routine must be in assembly code and terminated with an **RTE.** Each vector is 4 bytes long. See Motorola (1990b) for details on exceptions.

Example:

LABEL DUMMYINT RTE C; *ok*
DUMMYINT 54 EXCEPTION *ok*

Related word: **TPUEXCEPTION**

HSRRWAIT (n1 n2 -- f)

Examines repeatedly a service request value from the TPU Host Service Request Register (HSRR) until either the value is zero or a timeout occurs. The timeout value *n1* is expressed in milliseconds. The channel number *n2* ranges from 0 to 15 (modulo 16). A flag is returned indicating if a timeout occured.

Example:

2500 6 HSRRWAIT . *0 ok*
waits at TPU channel 6 for up to 2.5 seconds and returns zero indicating the TPU channel successfully performed its operation.

Related words: **!HSRR @HSRR**

PCLR (n --)

Sets pin *n* to output a low signal. *n* ranges from 0 to 23 (modulo 24) corresponding to the following ports:

Port D	= pins 0 to 7
Port E	= pins 8 to 15
Port F	= pins 16 to 23

This clears a bit in the CPU Pin Assignment Register (PAR) and sets the bit CPU Data Direction Register (DDR) for output. For more information see Motorola (1990a).

Example:

DECIMAL 3 PCLR *ok* causes the output of bit 3 in Port D to become low.

Related words: **CPUPIN PIN PSET**

PIN (n -- f)

Returns the port input value *f* from the pin *n* which ranges from 0 to 23 (modulo 24) corresponding to the following ports:

Port D = pins 0 to 7
Port E = pins 8 to 15
Port F = pins 16 to 23

This clears a bit in the CPU Pin Assignment Register (PAR) and clears the bit CPU Data Direction Register (DDR) for input. For more information see Motorola (1990a).

Example:

DECIMAL 8 PIN . *1 ok* examines bit 0 of Port E, which returns a one.

Related words: **CPUPIN PCLR PSET**

PSET (n --)

Sets pin *n* to output a high signal. *n* ranges from 0 to 23 (modulo 24) corresponding to the following ports:

Port D = pins 0 to 7
Port E = pins 8 to 15
Port F = pins 16 to 23

This clears a bit in the CPU Pin Assignment Register (PAR) and sets the bit CPU Data Direction Register (DDR) for output. For more information see Motorola (1990a).

Example:

DECIMAL 23 PSET *ok* causes the output of bit 7 in Port F to become high.

Related words: **CPUPIN PCLR PIN**

QBAUD (-- a)

A variable containing the user specified QSPI baud rate. This is used for changing the system clock to maintain the QSPI baud rate.

Related words: **!QBAUD @QBAUD**

QSM.CMDRAM (-- a)

Returns the address of the QSPI Command RAM array. This is used in conjunction with **QSM.RECRAM** and **QSM.XMTRAM** to control the serial data transfer. Each element is a byte long. See Motorola (1990a) for more details.

QSM.QDDR (-- a)

Returns the address of the QSM Data Direction Register. This specifies the direction of the port "D" pins. This register is one byte long. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

QSM.QILVR (-- a)

Returns the address of the QSM Interrupt Level and Vector Register. This is used to set up the interrupt vector and levels. See Motorola (1990a) for more details.

QSM.QMCR (-- a)

Returns the address of the QSM Configuration Register. This controls the operation of the QSM. See Motorola (1990a) for more details.

QSM.QPAR (-- a)

Returns the address of the QSM Pin Assignment Register. This sets the function of the port "D" pins. This register is one byte long. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

QSM.QPDR (-- a)

Returns the address of the QSM Port Data Register. This is used read or write pins to port "D". This register is one byte long. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

QSM.RECRAM (-- a)

Returns the address of the QSPI Receive Data RAM array. This is used in conjunction with **QSM.CMDRAM** to receive serial data from external devices. Each element is a word (2 bytes) long. See Motorola (1990a) for more details.

QSM.SCCR0 (-- a)

Returns the address of the SCI Control Register 0. This controls the operation of the SCI, including setting the baud rate. See Motorola (1990a) for more details.

Related words: **!BAUD @BAUD**

QSM.SCCR1 (-- a)

Returns the address of the SCI Control Register 1. This controls the operation of the SCI. See Motorola (1990a) for more details.

QSM.SCDR (-- a)

Returns the address of the SCI Data Register. Data is read and written to this register. See Motorola (1990a) for more details.

QSM.SCSR (-- a)

Returns the address of the SCI Status Register. This contains the status of the SCI. See Motorola (1990a) for more details.

QSM.SPCR0 (-- a)

Returns the address of the QSPI Control Register 0. This controls the operation of the QSPI including setting the baud rate. See Motorola (1990a) for more details.

Related words: **!QBAUD @QBAUD**

QSM.SPCR1 (-- a)

Returns the address of the QSPI Control Register 1. This controls the operation of the QSPI. See Motorola (1990a) for more details.

QSM.SPCR2 (-- a)

Returns the address of the QSPI Control Register 2. This controls the operation of the QSPI. See Motorola (1990a) for more details.

QSM.SPCR3 (-- a)

Returns the address of the QSPI Control Register 3. This controls the operation of the QSPI. This register is one byte long. See Motorola (1990a) for more details.

QSM.SPSR (-- a)

Returns the address of the QSPI Status Register. This contains the status of the QSPI. This register is one byte long. See Motorola (1990a) for more details.

QSM.XMTRAM (-- a)

Returns the address of the QSPI Transmit Data RAM array. This is used in conjunction with **QSM.CMDRAM** to transmit serial data to external devices. Each element is a word (2 bytes) long. See Motorola (1990a) for more details.

SCIBUF (-- a)

Returns the address of the SCI input buffer which is 260 bytes long. The first 4 bytes contains the header and the remaining 256 bytes contains the data received from SCI. The header contains:

Pos	Len	Description
0	2	Control/Status
2	1	Pointer to the last character received
3	1	Pointer to the first character received

For the Control/Status field, the bit positions have the following meanings:

0 enable/disableXON/XOFFhandshaking
1 enable/disable hardware handshaking
2 suspend transmission of data (i.e. XOFF received or hardware line low)

SIM.CSPAR (-- a)

Returns the starting address of the SIM chip select registers. See Motorola (1990a) for more details.

Related words: **.CS CSWAIT**

SIM.DDRE (-- a)

Returns the address of the SIM Port E Data Direction Register. This register is one byte long and controls the direction of the pins to either input or output. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

SIM.DDRF (-- a)

Returns the address of the SIM Port F Data Direction Register. This register is one byte long and controls the direction of the pins to either input or output. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

SIM.MCR (-- a)

Returns the address of the SIM Module Configuration Register. This controls the operation of the CPU. See Motorola (1990a) for more details.

SIM.PEPAR (-- a)

Returns the address of the SIM Port E Pin Assignment Register. This register is one byte long and controls the function of the pins. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

SIM.PFPAR (-- a)

Returns the address of the SIM Port F Pin Assignment Register. This register is one byte long and controls the function of the pins. Setting the pins to "one" causes the pin to become an interrupt request pin. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

SIM.PICR (-- a)

Returns the address of the SIM Periodic Interrupt Control Register. Used in conjunction with **SIM.PITR**, this sets the interrupt level and vector of the timer. See Motorola (1990a) for more details.

SIM.PITR (-- a)

Returns the address of the SIM Periodic Interrupt Timer Register. This specifies the time between timer interrupts. This is used by **COUNTER**. See Motorola (1990a) for more details.

SIM.PORTE (-- a)

Returns the address of the SIM Port E Data Register. This register is one byte long and is used for bit-wise input and output. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

SIM.PORTF (-- a)

Returns the address of the SIM Port F Data Register. This register is one byte long and is used for bit-wise input and output. See Motorola (1990a) for more details.

Related words: **CPUPIN PCLR PIN PSET**

SIM.RSR (-- a)

Returns the address of the SIM Reset Status Register. This contains the status of the cause of system reset. This register is one byte long. See Motorola (1990a) for more details.

SIM.SWSR (-- a)

Returns the address of the SIM Software Watchdog Service Register. This allows resetting the software watchdog by first writing hex value 55 and then write hex value AA to the register. The register is one byte long. See Motorola (1990a) for more details.

SIM.SYNCR (-- a)

Returns the address of the SIM Clock Synthesizer Control Register. This sets the system clock. See Motorola (1990a) for more details.

Related words: **!SYSCLOCK @SYSCLOCK**

SIM.SYPCR (-- a)

Returns the address of the SIM System Protection Control Register. This controls the monitoring of the system. See Motorola (1990a) for more details.

TPU.BUFPTR (-- a)

Returns the address of the TPU interrupt buffer array. Each of the 16 array elements that correspond to the TPU channels contains two longwords: the first is the interrupt service routine address, and the other is the buffer address.

Related words: **!TPUBUF @TPUBUF ?TPUBUF**

TPU.CFSR (-- a)

Returns the address of the TPU Channel Function Select Register (CFSR). This register contains four words at four bits per channel. See Motorola (1990c) for more details.

Related words: **!CFSR @CFSR**

TPU.CIER (-- a)

Returns the address of the TPU Channel Interrupt Enable Register (CIER). This register contains one word at one bit per channel. See Motorola (1990c) for more details.

Related words: **!CIER @CIER**

TPU.CISR (-- a)

Returns the address of the TPU Channel Interrupt Status Register (CISR). This register contains one word at one bit per channel. See Motorola (1990c) for more details.

Related words: **!CISR @CISR ^CISR**

TPU.CPR (-- a)

Returns the address of the TPU Channel Priority Register (CPR). This register contains two words at two bits per channel. See Motorola (1990c) for more details.

Related words: **!CPR @CPR**

TPU.HSQR (-- a)

Returns the address of the TPU Host Sequence Register (HSQR). This register contains two words at two bits per channel. See Motorola (1990c) for more details.

Related words: **!HSQR @HSQR**

TPU.HSRR (-- a)

Returns the address of the TPU Host Service Request Register (HSRR). This register contains two words at two bits per channel. See Motorola (1990c) for more details.

Related words: **!HSRR @HSRR HSRRWAIT**

TPU.PRAM (-- a)

Returns the start address of the TPU Parameter RAM. Each of the 16 TPU channels has 8 words or 16 bytes of RAM space for various purposes. See Motorola (1990c) for more details.

Related words: **!PRAM @PRAM**

TPU.TICR (-- a)

Returns the address of the TPU Interrupt Configuration Register (TICR). This sets the interrupt level and vector. See Motorola (1990c) for more details.

TPU.TMCR (-- a)

Returns the address of the TPU Module Configuration Register (TMCR). This controls the operation of the TPU. This register may be written only once until system reset. See Motorola (1990c) for more details.

TPUF.DIO (-- n)

Returns the function value of the TPU Discrete I/O operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.ITC (-- n)

Returns the function value of the TPU Input Capture/Input Transition Counter operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.OC (-- n)

Returns the function value of the TPU Output Compare operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.PM (-- n)

Returns the function value of the TPU Period Measurement with Additional/Missing Transition Detect operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.PPWA (-- n)

Returns the function value of the TPU Period/Pulse Width Accumulator operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.PSP (-- n)

Returns the function value of the TPU Position-Synchronized Pulse Generator operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.PWM (-- n)

Returns the function value of the TPU Pulse-Width Modulation operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.SM (-- n)

Returns the function value of the TPU Stepper Motor operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.SPWM (-- n)

Returns the function value of the TPU Synchronized Pulse-Width Modulation operation for the TPU CFSR register. See Motorola (1990c) for details on using this function.

TPUF.UART (-- n)

Returns the function value of the TPU Asynchronous Serial I/O operation for the TPU CFSR register. This function was provided by Onset in microcode form and is not part of the standard TPU functions.

Related word: **TSEROPEN**

TPUCLOCK (-- u)

Returns the TPU clock frequency in Hz based on the system clock and the TCR1 value and PSCK bit in the TMCR register. This is useful in computing various parameters for the TPU channel functions such as number of counts per cycle, delay counts, baud rate, etc. See Motorola (1990c) for more details.

TPUEXCEPTION (a n --)

Stores the exception address *a* (i.e. the start address of an interrupt routine) at the vector corresponding to the TPU channel *n* (modulo 16) in the vector base area. The interrupt routine must be in assembly code and terminated with an **RTE**. See Motorola (1990b) for details on exceptions.

Example:

LABEL CH5INT RTE C; *ok*
CH5INT 5 TPUEXCEPTION *ok*

Related word: **EXCEPTION**

TSERBAUD (n1 n2 --)

For use with TPU function **TPUF.UART** only. Sets the baud rate *n1* in channel *n2* for serial input or output. The maximum baud rate is the same as returned by **TPUCLOCK**.

TSERCLOSE (n --)

For use with TPU function **TPUF.UART** only. Disables the function at channel *n* and clears the interrupt routine address from the TPU buffer area.

Related word: **TSEROPEN**

TSERFLUSH (n --)

For use with TPU function **TPUF.UART** only. Sets the buffer pointers to zero at channel *n*. This clears any received data in the buffer as well as any output data.

TSERGET (n -- c)

For use with TPU function **TPUF.UART** only. Begins an indefinite loop until a character *c* has been received in the buffer at channel *n*. If a timeout occurred, returns -1 on the stack.

Related words: **TSERPUT TSERTIMEOUT**

TSERLEN (n1 -- n2)

For use with TPU function **TPUF.UART** only. Returns the number of bytes *n2* remaining in the buffer of channel *n1*.

TSEROPEN (f a n1 n2 --)

Opens a TPU channel *n2* for serial input or output, using the TPU function **TPUF.UART**. This routine performs the following actions:

This channel can be designated as with serial input or serial output depending on flag *f*. If the flag is true, the channel is set for output; otherwise, the channel is set for input. This flag also specifies the proper interrupt routine address to store in the TPU buffer area.

The buffer with address *a* and length *n1* is used for storing information and data. This buffer may be allocated using **BUFFER**. 26 bytes of the buffer is reserved for channel information, so the buffer length must be 27 bytes or more.

The baud rate is set to 9600 and the timeout value is initially zero.

Related words: **TSERCLOSE TSERBAUD**

TSERPAIR (n1 n2 --)

For use with TPU function **TPUF.UART** only. This links two TPU channels, *n1* and *n2*, for handshaking purposes. One of the two channels must be an input channel and the other the output channel. Aborts with a message if one or both channels are invalid (i.e. both inputs, both outputs, identical channel numbers, not opened with **TSEROPEN**, etc.)

Related word: **TSERXSHAKE**

TSERPUT (c n --)

For use with TPU function **TPUF.UART** only. Transmits a character *c* to channel *n*.

Related words: **TSERGET TSERPUTS**

TSERPUTS (a n1 n2 --)

For use with TPU function **TPUF.UART** only. Transmits a string of characters starting at address *a* and length *n1* to output channel *n2*.

Related words: **TSERGET TSERPUT**

TSERTIMEOUT (n1 n2 --)

For use with TPU function **TPUF.UART** only. Sets the timeout *n1* in milliseconds for the channel *n2* for use with **TSERGET**. If *n1* is zero, then the timeout check is disabled and **TSERGET** would hang infinitely until a character is received.

TSERXSHAKE (f n --)

For use with TPU function **TPUF.UART** only. Sets the XON/XOFF handshaking mode on the TPU channel *n*. Enables handshake if *f* is non-zero, else disables handshake.

Note: Handshaking, if enabled, will not take effect until **TSERPAIR** is executed.

Related word: **TSERPAIR XSHAKE**

^CISR (n --)

Complements a bit in the TPU Channel Interrupt Status Register (CISR) at the channel number *n*, which ranges from 0 to 15 (modulo 16).

Related words: **!CISR @CISR**

sysclk (-- a)

Returns the subroutine address to get the system clock frequency (Hz). This is useful for writing assembly code that utilizes the system clock frequency.

This subroutine uses the following CPU registers:

D0 -	output: system clock frequency
D1 -	temporary

Example:

CODE SCLK sysclk JSR D0 S -) MOV NEXT *ok*

Related word: **@SYSCLOCK**

tpuvend (-- a)

Returns the routine address to terminate the TPU interrupt. All this does is to restore the CPU registers **A0** and **A1** from the return stack and return from exception (i.e. return back to the system where it left off when the TPU interrupt occurred).

Example:

CODE ENDTPU tpuvend JMP C; *ok*

Related word: **!TPUBUF.**

TattleTale Model 7 Specific FORTH Words

The following FORTH words utilizes the features of the TattleTale Model 7 in addition to the MC68332 CPU specific words described above. These features include the disk drive, serial EEPROM, A/D converter and the Real Time Clock. See Onset (1991) and Conner (1991) for more details on these features.

!DATE (d --)

Sets the date in the Real Time Clock. The input *d* must have a *dd/mm/yyyy* format (*dd* = day, *mm* = month, *yyyy* = year). Only the last two digits of the year is put into the Real Time Clock, so the years must be from 1980 to 2079. The day of week is not needed since it is computed from the date.

Example:

DECIMAL 15/04/1993 !DATE *ok*
sets the date in the Real Time Clock to April 15, 1993.

Related word: **@DATE**

!SEE (b a --)

Stores a byte *b* to the address in the TattleTale's Serial EEPROM. The address ranges from 0 to 511 (modulo 512). This is a non-volatile RAM useful to save data for the next power up or reset.

Related words: **@SEE**

!SEE2 (h a --)

Stores a word *h* to the address in the TattleTale's Serial EEPROM. The address ranges from 0 to 510 (modulo 512). This is a non-volatile RAM useful to save data for the next power up or reset.

Related words: **@SEE2**

!SEE4 (n a --)

Stores a longword *n* to the address in the TattleTale's Serial EEPROM. The address ranges from 0 to 508 (modulo 512). This is a non-volatile RAM useful to save data for the next power up or reset.

Related words: **@SEE4**

!TIME (d --)

Sets the time of day in the Real Time Clock. The input must have a *hh:mm:ss* format (*hh* = hour, *mm* = minutes, *ss* = seconds). Note that the hour is a 24-hour clock (i.e. hours from 0 to 23, 0 = midnight).

Example:

DECIMAL 14:30:45 !TIME *ok*
sets the time of day in the Real Time Clock to 2:30:45.00 PM.

Related word: **@TIME**

(DATE) (n1 -- a n2)

Converts date *n1* in *ddmmyyyy* format to an ASCII string, returned with address *a* and length *n2*. The string has the format of *Www dd Mon yyyy* (*Www* = alpha weekday, *dd* = day, *Mon* = alpha month, *yyyy* = year).

Example:

15041993 (DATE) TYPE *Thu 15 Apr 1993 ok*

(TIME) (n1 -- a n2)

Converts time *n1* in centiseconds into an ASCII string returned by its address *a* and length *n2*. The string has the format of *hh:mm:ss.ss* (*hh* = hour in 24-hour clock, *mm* = minutes, *ss.ss* = seconds and fraction of second).

Example:

DECIMAL 1234567 (TIME) TYPE *03:25:45.67 ok*

.ALARM (--)

Prints the set date and time of the Real Time Clock's alarm registers. The output format appears as *Www dd Mon yyyy hh:mm:ss.ss* (*Www* = alpha weekday, *dd* = day, *Mon* = alpha month, *yyyy* = year, *hh* = hour in 24-hour clock, *mm* = minutes, *ss.ss* = seconds and fraction of second). A string of 'x' in one or more of the fields indicates that time or date field is disabled from alarm interrupt.

Example:

.ALARM
Alarm is set to: Tue 21 Jan 1994 18:30:xx.xx ok

Related words: **.RTC SETALARM**

.DINFO (--)

Prints information about the disk drive in a tabular format. This assumes the drive has been turned on. For details about the contents, see Conner (1991).

Related words: **DINFO DRIVE**

.DSTAT (--)

Prints contents of the disk drive registers in tabular format. For details about the contents, see Conner (1991).

.RTC (--)

Prints contents of the Real Time Clock registers in decimal and hexadecimal values.

Note: Executing this word clears the interrupt status bit in the Real Time Clock register. The value should be saved to a variable or kept on stack to use it in a program.

Related words: **.ALARM DATE TIME**

.SCH (--)

Prints the eight schedules set by **SCHED** in tabular format. For disabled schedules, '(unused)' will be printed. Otherwise, the date (in Julian minutes and in date/time format), period in minutes, number of cycles and the name of the routine to execute upon alarm interrupt.

.SDA (--)

Prints four channels of the SDA analog converter values in unsigned decimal, signed decimal and hexadecimal.

.SEE (--)

Prints the entire 512 byte contents of the Serial EEPROM in tabular format. The printout is identical to **DUMP** with two 256 byte sections separated as system and user areas.

Related words: **!SEE @SEE**

@DATE (-- u)

Gets the date from the Real Time Clock and returns as a single-precision number with a *ddmmyyyy* format (*dd* = day, *mm* = month, *yyyy* = year). Since the Real Time Clock uses only the last two digits of the year, the year ranges from 1980 to 2079.

Related word: **!DATE**

@SEE (a -- b)

Fetches a byte *b* from the address in the TattleTale's Serial EEPROM. The address ranges from 0 to 511 (modulo 512). This is a non-volatile RAM useful to save data for the next power up or reset.

Related word: **!SEE**

@SEE2 (a -- h)

Fetches a word *h* from the address in the TattleTale's Serial EEPROM. The address ranges from 0 to 510 (modulo 512). This is a non-volatile RAM useful to save data for the next power up or reset.

Related word: **!SEE2**

@SEE4 (a -- n)

Fetches a longword *n* from the address in the TattleTale's Serial EEPROM. The address ranges from 0 to 508 (modulo 512). This is a non-volatile RAM useful to save data for the next power up or reset.

Related word: **!SEE4**

@TIME (-- u)

Gets the time of day from the Real Time Clock in centiseconds since midnight.

Related word: **!TIME**

?ALARM (-- f)

Returns the interrupt status of the Real Time Clock. If *f* is non-zero, then the interrupt has occurred, otherwise no interrupt had occurred.

Note: Executing this word clears the interrupt status bit in the Real Time Clock register. The value should be saved to a variable or kept on stack to use it in a program.

Related words: **.ALARM .RTC SETALARM**

?SLEEP (-- f)

Returns a non-zero flag *f* indicates the TT7 will be turned off upon executing **SLEEP**, otherwise it will not be turned off.

This is useful, for example, to execute routines only when the TT7 is ready to be turned off (i.e. the last executing schedule in the list set up by **SCHED**).

ADTIMEOUT (-- n)

Returns the timeout value for A/D converters which would be returned by a routine that does the A/D conversion on the event of a timeout. In this implementation, the hex value is 8000 (i.e. the 15th bit is set).

Related word: **SDA**

CD (a n --)

Short form of **CHDIR**. See **CHDIR**.

CHDIR (a n --)

Changes to the default directory specified by the string *a* and its length *n*. The sub-directory names may be separated by a backslash (\). This sets the directory for default disk operations and for directory listing. Prints an error message if the disk drive is not on, on invalid sub-directory names or non-DOS disk format.

Related words: **MKDIR RMDIR DIR**

CHKDSK (--)

This checks the disk for clusters that are not linked to any directory entry and valid directory and sub-directory entries. If any clusters are not linked, directory entries will be created for these chained clusters in the root directory as *FILExxxx.CHK* where *xxxx* is the sequential number of the file. At the end, this routine will print the results. Prints an error message if the drive is not on or if the disk is not in DOS format.

Related word: **FORMAT**

CLRSCHED (n --)

Disables alarm interrupt for schedule number *n*. The number must be from 1 to 8.

Related word: **SCHED**

COPY (a1 n1 a2 n2 --)

Copies the contents of the source file to the destination file on DOS disk. The source file name starts at *a1* with length *n1*, and the destination file name starts at *a2* with length *n2*. The file names may contain sub-directory names separated with the backslash character (\).

DATE (--)

Gets date from the Real Time Clock and prints it. See **(DATE)** for information on the output format.

Example:

DATE *Fri 21 Jan 1994 ok*

Related words: **!DATE @DATE (DATE)**

DAY/WEEK (n1 -- a n2)

Returns an ASCII string *a* with length *n2* from day of the week *n1* (0 = "Sun", 1 = "Mon", etc.). In this implementation, *n2* will always be 3.

Example:

4 DAY/WEEK TYPE *Thu ok*

Related word: **WEEKDAY**

DEL (a n --)

Deletes a file from the DOS disk. The file name, *a* with length *n*, may include sub-directory separated by a backslash (\).

DINFO (a --)

Receives information about the disk drive into a buffer *a*. This assumes the disk drive is turned on. See Conner (1991) for details on content.

Related words: **.DINFO DRIVE**

DIR (--)

Outputs a DOS-like directory listing to the console. The directory is specified with **CHDIR**.

DREAD (a n1 n2 --)

Reads data from disk to the buffer *a* starting at the logical sector number *n1* and the number of sectors to read *n2*. Each sector is 512 bytes long so the memory at *a* must be large enough to hold 512**n2* bytes. Any error aborts with a message.

Related word: **DWRITE**

DREADY (--)

Checks the drive power-on status and the drive status registers. Waits until the drive is ready to read or write. Aborts with a message on timeouts or errors.

DRIVE (f --)

Turns the disk drive on or off depending on the flag *f*. If true, then turn drive on; otherwise, turn it off. Turning it on may take about 8 seconds.

DRIVE.TABLE (-- n1 n2 n3)

Extracts the disk identification and matchs it with the table. Returns the number of heads *n1*, the number of cylinders per head *n2* and the number of sectors per cylinder *n3*. The product of the three values gives the total number of sectors on disk. If found, returns the default values from the table (which may be different than found in the drive information). Otherwise extracts the values from the drive information.

Related word: **DINFO**

DRV.AVLSECT (-- a)

A variable containing the number of available sectors on disk.

DRV.CLUSTSZ (-- a)

A variable containing the number of sectors per cluster.

DRV.NDISK (-- a)

A variable containing the number of disk drives on-line. Currently, this is set to one.

DRV.TOTSECT (-- a)

A variable containing the total number of sectors on disk.

DSECTOR (n1 n2 --)

Sets the drive registers to point to the logical sector number *n1* and the number of sectors *n2*. The number of sectors must be between 1 and 255. Any error will abort with a message.

Related words: **DREAD DWRITE**

DWRITE (a n1 n2 --)

Writes data to disk from the buffer *a* starting at the logical sector number *n1* and the number of sectors to be read *n2*. Each sector is 512 bytes long. Any error aborts with a message.

Related word: **DREAD**

FCLOSE (n --)

Closes the file at file pointer *n* and frees up the memory in the file descriptor area.

Related word: **FOPEN**

FDUMP (a n --)

Outputs the content of a file to the console in a tabular format containing the offset from the beginning of the file, the data bytes (up to 16 per line) in hexadecimal values and the ASCII representation of the bytes. The output is similar to **DUMP**. The file name begins at address *a* and length *n*, and may contain the subdirectory separated by backslashes (\).

Example:

```
" DUMMY.DAT" FDUMP
00000000  EA AA 2A AA AE A2 AA AA  j***."**  ok
```

Related word: **FTYPE**

FEOF (n --)

At file pointer *n*, causes the current file offset to become the end of file. The remaining contents of the file is deleted.

Related word: **FSEEK**

FGETS (a n1 n2 -- n3)

Reads text data from a file specified by the file pointer *n2* to the string *a*. Returns *n3* as the actual number of characters read from the following conditions:

1. Up to the maximum length *n1*.
2. Encounters a carriage return character, which is not included in buffer *a*.
3. Reached the end of file.

Related word: **FPUTS FREAD FSEEK FWRITE**

FOPEN (a n1 -- n2)

Opens a DOS disk file, with its name beginning at address *a* and length *n1*, for reading and/or writing. The file name may contain the sub-directory separated by backslashes (\). Returns *n2* as the file pointer which is actually the index to the file pointer area. This area contains information about the file such as a copy of the directory entry, location of the directory entry, file offset, etc. The file offset (used in **FSEEK**) is set to zero.

One of three things may happen:

1. If the file does not exist, then a new file is created.
2. If the file exists, it is opened for read and write.
3. If the file has already been opened, the file pointer is returned and the file offset is reset to zero.

Example:

```
" \DATA\MOS023.DAT" FOPEN . 1  ok
```

Related words: **FCLOSE FGETS FSEEK FPUTS FREAD FWRITE FEOF**

FORMAT (--)

Initializes and formats the disk to DOS format. All sectors will be scanned for bad blocks and be filled with zeros. A root directory is created and result of the formatting are printed.

CARE MUST BE TAKEN SINCE THIS WILL ERASE ALL DATA FROM DISK

Before attempt to format the disk, a warning and a prompt will appear asking the user if he wish to continue. The user must type **yes** in lowercase letters to start the formatting process.

For an 80 megabyte disk, this takes about 40 minutes.

Related word: **CHKDSK**

FPUTS (a n1 n2 --)

Writes text data to a file specified by the file pointer *n2* from the string *a* and length *n1*. Appends a carriage return character to the string in the file. If needed, new clusters will be added to the file if the data goes past the end of file.

Related word: **FGETS FREAD FWRITE**

FREAD (a n1 n2 -- n3)

Reads binary data from a file specified by the file pointer *n2* to the buffer *a* up to length *n1*. Returns *n3* as the actual number of bytes read from the file since *n1* may go past the end of file.

Related word: **FSEEK FWRITE**

FSEEK (n1 n2 --)

Sets the file offset *n1* bytes from the beginning of the file specified by the file pointer *n2*. If the offset goes beyond the end of file (i.e. greater than the number of bytes in the file) or if *n1* is negative, then the offset will be set to the end of file.

Related words: **FEOF FREAD FWRITE**

FTYPE (a n --)

Outputs the content of a file to the console in ASCII strings (each terminated with a carriage return character). The file name begins at address *a* and length *n*, and may contain the subdirectory separated by backslashes (\).

Example:

" README.TXT" FTYPE
This is a file containing text used for informational purposes. Each line is separated in the file by a carriage return.
.
.
.
End of file.
ok

Related word: **FDUMP FGETS FPUTS**

FWRITE (a n1 n2 --)

Writes binary data to a file specified by the file pointer *n2* from the buffer *a* and length *n1*. If needed, new clusters will be added to the file if the data goes past the end of file.

Related word: **FPUTS FREAD FSEEK**

LOG (a n --)

Saves a string of characters starting at address *a* and length *n* to a log file on disk. This file is a text file named **FORTH.LOG** and is useful to save diagnostic information. This will also automatically turn on the disk drive. Use **FTYPE** to list contents.

Example:

" 21/01/94 09:00 MOBY STARTED" LOG *ok*
" FORTH.LOG" FTYPE
21/01/94 09:00 MOBY STARTED ok

MD (a n --)

A short form of **MKDIR**. See **MKDIR**.

MKDIR (a n --)

Creates a sub-directory on the DOS disk. A short form of this word can be used as **MD**. The sub-directory name is specified by *a* and length *n*, and the names can be separated by backslashes (\). Any error will abort with a message.

Related words: **CHDIR RMDIR**

MONTH (n1 -- a n2)

Returns an ASCII string *a* with length *n2* from a numeric month *n1* which ranges from 1 to 12 (1 = "Jan", 2 = "Feb", etc.). In this implementation, *n2* will always be 3.

Example:

6 MONTH TYPE *Jun ok*

Related word: **(DATE)**

PSRAMAUTO (--)

Sets the Pseudo-static RAM to Auto-Refresh mode. Although this consumes slighty more power, it is used to "turn-on" the RAM for read and write.

Related words: **PSRAMSELF**

PSRAMSELF (--)

Sets the Pseudo-static RAM to Self-Refresh mode. This is used to "turn-off" the RAM to conserve power. Anything written to the RAM may not be reliable.

Related words: **PSRAMAUTO**

RD (a n --)

A short form of **RMDIR**. See **RMDIR**.

REN (a1 n1 a2 n2 --)

Renames the source file name, *a1* and length *n1*, to the destination file name, *a2* and length *n2*, on the DOS disk. The file names may contain sub-directory names separated with backslashes (\). If the destination file name points to a different sub-directory than the source file name, the directory entry of the file will be moved to the new sub-directory.

Example:

" ML034.DAT" " ML0034A.DAT" REN *ok*

Related word: **COPY**

RMDIR (a n --)

Removes a sub-directory from the DOS disk. A short form of this word can be used as **RD**. The sub-directory name is specified by *a* and length *n*, and the names can be separated by backslashes (\). The contents of the sub-directory must be empty else returns an error. Any error will abort with a message.

Related words: **CHDIR MKDIR**

SCANSCHED (--)

Scans the schedule list and executes routines enabld in the scheule list. Although this may be executed at any time, this should be executed when an alarm interrupt occurs using **?ALARM**.

Example:

BEGIN ?ALARM IF SCANSCHED THEN AGAIN *ok*

Related words: **SCHED SETALARM**

SCHED (a d1 d2 n1 n2 n3 --)

Adds or modifies a schedule to the schedule list. The information required for the list is:

a the PFA of the routine to execute upon alarm interrupt. This may be obtained from executing ' with routine name following.

d1 the date to begin the alarm interrupt. This is entered as *dd/mm/yyyy* (*dd* = day of month, *mm* = month, *yyyy* = year). The current date may be used by setting this value to zero (i.e. **0.**).

d2 the time of day to begin the alarm interrupt. This is entered as *hh:mm:ss* (*hh* = hour in 24-hour time, *mm* = minutes, *ss* = seconds). Although seconds are not used, it should be set to zero.

n1 the period interval in minutes between alarm interrupts. A zero value indicates the schedule will execute only once.

n2 the number of schedule executions. A zero value indicates infinite number of alarm interrupts.

n3 the schedule number in the list. This value must be from 1 to 8. Upon alarm interrupt, the first schedule in the list will be executed first, the second next, and so forth. Hence, the lower the schedule number, the higher the priority.

Examples:

' MOBY 0. 02:00:00 120 0 1 SCHED *ok*
' VAX 1/01/1994 12:00:00 60 5 2 SCHED *ok*

Related words: **.SCH CLRSCHED SCANSCHED**

SDA (n -- uh)

Invokes the on-board A/D converter. The channel number *n* ranges from 0 to 3. The converter returns a 12-bit unsigned value *uh*. A 10 millisecond timeout is used, and if it occurs, returns the timeout value instead of the value from the A/D converter.

Related words: **ADTIMEOUT sdasr**

SEECPUPIN (n --)

Designate a pin to function as a CPU line at power up. Pin number *n* ranges from 0 to 23 (modulo 24) which corresponds to the following ports:

Port D	= pins 0 to 7
Port E	= pins 8 to 15
Port F	= pins 16 to 23

This sets a bit for the CPU Pin Assignment Register (PAR) in the Serial EEPROM. For more information see Motorola (1990a).

Example:

DECIMAL 18 SEECPUPIN *ok* sets bit 2 in the Port F PAR to enable interrupt request input at interrupt level 2.

Related words: **SEEPCLR SEEPIN SEEPSET**

SEECRC (--)

Calculates the CRC of the system area in the Serial EEPROM and store it at address 0. This checksum is compared at power-up to check integrity of the EEPROM. If the checksums do not match, a set of default values will be stored in the EEPROM.

Related words: **CALCCRC UPDATECRC**

SEEPCLR (n --)

Sets pin *n* to output a low signal on power up. *n* ranges from 0 to 23 (modulo 24) corresponding to the following ports:

Port D	= pins 0 to 7
Port E	= pins 8 to 15
Port F	= pins 16 to 23

This clears a bit for the CPU Pin Assignment Register (PAR) and sets the bit CPU Data Direction Register (DDR) for output in the Serial EEPROM. For more information see Motorola (1990a).

Example:

DECIMAL 3 SEEPCLR *ok* sets the output of bit 3 in Port D to low.

Related words: **SEECPUPIN SEEPIN SEEPSET**

SEEPIN (n --)

Sets the port pin *n* to input at power up which ranges from 0 to 23 (modulo 24) corresponding to the following ports:

Port D	=	pins 0 to 7
Port E	=	pins 8 to 15
Port F	=	pins 16 to 23

This clears a bit for the CPU Pin Assignment Register (PAR) and clears the bit CPU Data Direction Register (DDR) for input in the Serial EEPROM. For more information see Motorola (1990a).

Example:

DECIMAL 8 SEEPIN *ok* sets bit 0 of Port E to input.

Related words: **SEECPUPIN SEEPCLR SEEPSET**

SEEPSET (n --)

Sets pin *n* to output a high signal at power up. *n* ranges from 0 to 23 (modulo 24) corresponding to the following ports:

Port D	=	pins 0 to 7
Port E	=	pins 8 to 15
Port F	=	pins 16 to 23

This clears a bit for the CPU Pin Assignment Register (PAR) and sets the bit for the CPU Data Direction Register (DDR) for output in the Serial EEPROM. For more information see Motorola (1990a).

Example:

DECIMAL 23 SEEPSET *ok* sets the output of bit 7 in Port F to high.

Related words: **CPUPIN PCLR PIN**

SETALARM (d1 d2 --)

Sets the Real Time Clock alarm registers to allow alarm interrupt in the future. The inputs are date *d1* and time *d2*. The date format is *dd/mm/yyyy* (*dd* = day of month, *mm* = month, *yyyy* = year) and the time format is *hh:mm:ss* (*hh* = hour in 24-hour time, *mm* = minutes, *ss* = seconds).

Example:

01/06/1994 12:00:00 SETALARM *ok*

Related words: **.ALARM ?ALARM SCHED SLEEP**

SLEEP (--)

Examines and update the schedule list (if any), sets the alarm registers for the next wake-up time and turns off the TT7. Turning off the TT7 is accomplished by setting the port E pin 7 to zero. This would work only if the TT7 has been modified for external interrupt. When the alarm interrupt occurs, the TT7 powers up and executes **SCANSCHED.**

Related words: **SCHED SCANSCHED SETALARM**

TIME (--)

Gets time of day from the Real Time Clock and prints it. See **(TIME)** for information on the output format.

Example:

TIME *20:52:23.47 ok*

Related word: **(TIME)**

TT7 (--)

Invokes the TT7 vocabulary for burning a new program to the Flash EPROM. See **TattleTale EPROM Burner** for list of words.

WEEKDAY (n1 -- n2)

Returns the day of the week *n2* (0 = Sunday, 1 = Monday; etc.) from the date *n1* in *ddmmyyyy* format.

Example:

DECIMAL 15041993 WEEKDAY . *4 ok*

Related word: **DAY/WEEK**

sdasr (-- a)

Returns the subroutine address to get a 12-bit value from the on board A/D converter. This is useful for writing assembly code that would use the A/D converter.

This subroutine uses the following CPU registers:

D0 - input: channel number;
output: analog value
D7 - temporary

Example:

CODE AD 1 #Q D0 MOV sdasr JSR
D0 S -) MOV NEXT *ok*

Related word: **SDA**

Additional FORTH Words for MLML Use

This section describes the FORTH words that may not be part of the standard FORTH dictionary but were added for MLML use. These words are not necessarily specific to the TattleTale Model 7 or the MC68332 processor but can be transported to other machines if needed.

!BAUD (n --)

Change baud rate *n* of the serial port. The range for the port depends on the content of **DEVICE**. If **DEVICE** is zero then the SCI port of the CPU is used. In which case it ranges from 0 to the system clock divided by 32 and values out of range will return an error. The highest possible baud rate is about 500K bits/second. The value on stack is also saved in variable **BAUD**.

NOTE: Care should be taken in using this word. In changing the baud rate, the baud rate on your terminal or computer that is connected to the serial port must also be changed. Otherwise, transmitted characters become garbled.

Related word: **@BAUD**

#ASC (a n --)

Format: **<# ... #ASC ... #>**

Copies a string starting at address *a* and length *n* to a buffer in memory for output. This is useful for adding ASCII strings to the output buffer defined by **<#**.

Example:

DECIMAL 18. <# #S " sec" #ASC #>
18 sec ok

Related words: **<# # #S SIGN HOLD #>**

'AUTO (-- a)

A user variable containing the PFA of an autostart routine. When the system powers up or a cold reset is performed, the startup routine examines this variable. If the content is zero then no autostart is performed; otherwise, executes the autostart routine. By default, this variable is set to zero.

This is useful to define instrument specific startups such as initializing external A/D converters, specific TPU routines, etc.

To enable the autostart, create a FORTH word containing the startup routine(s) and obtain its PFA from '. Store this PFA in this variable and execute **TT7SAVE** so this will be saved along with the new FORTH word(s) to the Flash EPROM.

.S (--)

Displays contents of the parameter stack starting from the bottom to top of stack, terminated with "<-Top". An empty stack prints the message "Empty Stack".

Example:

1 5 88 .S
1 5 88 <-Top ok

:00 (ud1 -- ud2)

Format: **<# ... :00 ... #>**

Extracts minutes or seconds (i.e. 0 to 59) from the double-precision number *ud1* and converts to ASCII characters with a preceding colon. The result after dividing by 60 is left on stack. This is used by **(TIME)** to format the time of day.

Example:

DECIMAL 1234. <# :00 58 HOLD :00 #>
TYPE *20:36 ok*

Related word: **(TIME)**

<< (n1 n2 -- n3)

Shifts the single-precision number *n1* to the left by *n2* bits and leaves the result on stack. The least significant bits are filled with zeros. This is similar to the C language operator.

Example:

HEX 123 2 << . *48C ok*

Related word: **>>**

>> (n1 bits -- n2)

Shifts the single-precision number *n1* to the right by *n2* bits and leaves the result on stack. The most significant bits are filled with zeros. This is similar to the C language operator.

Example:

HEX 123 2 >> . *48 ok*

Related word: **<<**

?CELL (n -- n f)

Determines if *n* can be compiled as a longword or word. Returns a true flag if *n* must be compiled as a longword. Otherwise, returns a false flag indicating that *n* can be a sign-extended word.

@BAUD (-- u)

Returns the actual baud rate of the serial port. If the content of **DEVICE** is zero, this baud rate is based on the actual system clock and SCI setting.

Related word: **!BAUD**

BAUD (-- a)

A variable containing the user specified baud rate. This is used for changing the system clock to maintain the SCI baud rate.

Related words: **!BAUD @BAUD**

BEEP (--)

Outputs a bell character (ASCII 7) to the console.

BINARY (--)

Stores 2 to the user variable **BASE** for binary input/outputs.

Related words: **DECIMAL HEX**

BUFFER (n -- a)

Allocates memory space as buffer with length *n* bytes. Returns address *a* of the newly allocated buffer. This is useful to reserve memory space for buffering, storage, etc. such as used in **TSEROPEN**.

Note: The memory begins at the top of the parameter stack area and allocates towards the current dictionary pointer (**HERE**).

Example:

HEX 1000 BUFFER . *3EB00 ok*

Related word: **ALLOT**

BYE (--)

Resets the internal and external devices and performs a "cold" restart of the system. This is similar to power-on except the memory is preserved with the exception of the user variables and most system variables for debugging purposes.

Note: This word may be executed only by the **OPERATOR** task, otherwise it is ignored.

CALCCRC (uh1 a n -- uh2)

Updates the CRC checksum based on the previously calculated CRC *uh1* and the array of bytes starting at *a* and length *n*. The new CRC *uh2* is put on stack as an unsigned 16-bit word.

Note: *uh1* must be initially set to zero to properly compute CRC.

Example:

0 " CRC TEST" CALCCRC . *4255 ok*

Related word: **UPDATECRC**

CALDAY (n1 -- n2)

Converts Julian Day *n1* since 0 Jan 1900 to calendar date *n2* in *ddmmyyyy* format (*dd* = day, *mm* = month, *yyyy* = year).

Example:

```
DECIMAL 34073 CALDAY . 15041993  ok
```

Related word: **JULDAY**

DEPTH (-- n)

Returns the number of cells on the parameter stack, not including the value placed on stack.

DUMP (a n --)

Outputs the memory contents beginning at address *a* and length *n* in a tabular format containing the address, the data bytes (up to 16 per line) in hexadecimal values and the ASCII representation of the bytes.

Example:

```
HEX 1000 8 DUMP
00001000  EA AA 2A AA AE A2 AA AA  j***."**  ok
```

Related words: **LDUMP WDUMP**

HMS (d -- n1 n2 n3)

Breaks down the input time *d* in *hh:mm:ss* format into seconds *n1*, minutes *n2* and hour *n3*.

Example:

```
DECIMAL 12:30:00 HMS . . . 12 30 0  ok
```

J' (-- n)

Copies a number from the third cell below the top of the return stack to the parameter stack.

Related words: **I I' J K K'**

JULDAY (n1 -- n2)

Converts date *n1* in *ddmmyyyy* format (*dd* = day, *mm* = month, *yyyy* = year) to Julian Day *n2*. This is the number of days since 0 Jan 1900.

Example:

```
DECIMAL 15041993 JULDAY . 34073  ok
```

Related word: **CALDAY**

K (-- n)

Copies a number from the fourth cell below the top of the return stack to the parameter stack.

Related words: **I I' J J' K'**

K' (-- n)

Copies a number from the fifth cell below the top of the return stack to the parameter stack.

Related words: **I I' J J' K**

LDUMP (a n --)

Identical to **DUMP** except the contents are displayed as 4-byte numbers.

Example:

```
HEX 1000 8 LDUMP
00001000  EAAA2AAA AEA2AAAA  j***."**  ok
```

Related words: **DUMP WDUMP**

RECOVER (--)

Restores the dictionary pointer and some user variables to where it was before compiling whenever an error occurred during compilation.

Related word: **TRY**

TRY (--)

Saves the dictionary pointer and some user variables to a system buffer before compiling. Used to recover these values in case of errors during compilation.

Related word: **RECOVER**

UPDATECRC (uh1 b -- uh2)

Updates the CRC checksum based on the previously calculated CRC *uh1* and the current byte *b*. The new CRC *uh2* is put on stack as an unsigned 16-bit word.

Note: *uh1* must be initially set to zero to properly compute CRC.

Example:

```
0 255 UPDATECRC . 7920  ok
```

Related word: **CALCCRC**

VLIST (--)

Outputs the names found in the vocabulary to the console in columnar format beginning with the newest entry.

WDUMP (a n --)

Identical to **DUMP** except the contents are displayed as 2-byte numbers.

Example:

```
HEX 1000 8 WDUMP
00001000  EAAA 2AAA AEA2 AAAA  j***."**  ok
```

Related words: **DUMP LDUMP**

X. (u n --)

Outputs an unsigned number *u* in hexadecimal format, formatted with leading zeros at width *n*.

Example:

```
DECIMAL 12345 8 X. 00003039  ok
```

Related words: **D.R U.R**

XSHAKE (f --)

Sets the XON/XOFF handshaking mode on the serial port. Enables handshake if *f* is non-zero, else disables handshake.

YMD (n1 -- n2 n3 n4)

Breaks down the date *n1* in *ddmmyyyy* format into day *n2*, month *n3* and year *n4*.

Example:

```
DECIMAL 15041993 YMD . . . 1993 4 15  ok
```

Related words: **DAY/WEEK MONTH**

MC68332 FORTH Assembler

FORTH provides an assembler specific for the Motorola's 68332 CPU. The format order is similar to the non-FORTH assembler convention except that the mnemonic is the last word. For example, in assembler convention:

```
MOVE.L   (A6)+,D2
ADDI.W   #10,D0
CLR.B    DATA
```

would look like this in FORTH assembler:

```
A6 )+ D2       MOV
10 #H D0 H.    ADD
DATA AB B.     CLR
```

For more detailed description on the registers, addressing modes and the mnemonics, refer to Motorola (1990b).

Registers:

Data	Address	Control
D0	**A0**	**SFC**
D1	**A1**	**DFC**
D2	**A2** *or* **N**	**USP**
D3	**A3** *or* **U**	**VBR**
D4	**A4** *or* **W**	
D5	**A5** *or* **I**	
D6	**A6** *or* **S**	
D7	**A7** *or* **R**	

Note: The letters that corresponds to the address registers **A2** to **A7** represent pointers to FORTH code:

N	= Next word
U	= User space
W	= Pointer to PFA of Word
I	= Interpreter pointer
S	= Parameter stack pointer
R	= Return stack pointer

Addressing Modes:

By default, the length of the addressing modes are in longwords (4 bytes). The user can select shorter lengths with the following:

H. - Length is word (2 bytes)
B. - Length is byte (1 byte)

Example:

D2 D0 B. MOV *ok*
moves a byte from D2 to D0

Indirect addressing modes:

)	simple indirect
)+	with postincrement
-)	with predecrement
0)	with 16-bit displacement
1)	" " "
2) *or* **N)**	" " "
3) *or* **U)**	" " "
4) *or* **W)**	" " "
5) *or* **I)**	" " "
6) *or* **S)**	" " "
7) *or* **R)**	" " "
PC)	program counter with displacement
+X	index and 8-bit displacement (word)
+XL	index and 8-bit displacement (longword)

The index words (**+X** and **+XL**) must be between a data register (containing the offset) and the address register in indirect mode.

Note: The letters **N**, **U**, **W**, **I**, **S** and **R** are identical to the address register letters explained above (see **Registers**).

Examples:

8 D1 +X 0) D0 B. MOV *ok*
moves a byte from indexed A0 to D0, is also equivalent to:

MOVE.B (8,A0,D1.W),D0

A6)+ D7 ADD *ok*
adds contents in (A6) to D7 and increments A6 by 4.

D0 8 1) MOV *ok*
moves D0 to address in A1 with offset of 8.

AB (a --)

Sets the address into absolute addressing mode. The length of the address depends on the value of the address, i.e. if *a* is between -32768 and 32767, inclusive, then the length would be word (2 bytes), otherwise, the length would be longword (4 bytes).

Example:

1000 AB D0 MOV *ok*
moves a longword from address 1000 to D0.

Immediate addressing modes:

#Q	quick immediate
#B	byte-sized immediate
#H	word-sized immediate
#	longword-sized immediate

Examples:

200 # A0) MOV *ok*
moves a value (200) to the address in A0

1 #Q D0 ADD *ok*
increments D0 by one

Branching Words:

The following words are only part of the **ASSEMBLER** vocabulary which are not to be confused with the words in the **FORTH** vocabulary. The difference is that the following words assembles a branch instruction based on the conditional value on stack. The branching offset may range only from -128 to 127 (i.e. a signed byte) from the program counter. Anything else aborts with an error message.

NOT (n1 -- n2)

Complements the condition code on stack.

Example:

CS NOT *ok*
changes the condition of carry bit set to carry bit clear.

IF (n --)

Formats: **CODE** *name* ... **IF** ... **ENDIF** ... **C;**
CODE *name* ... **IF** ... **ELSE**
... **ENDIF** ... **C;**

Used only in an assembler coding. Assembles a conditional branch at the dictionary pointer. This branches to the code after **IF** if the condition is true; otherwise branches to the code after **ELSE** or **ENDIF**. See **Branch Condition Codes** for values of *n*.

ELSE (--)

Format: **CODE** *name* ... **IF** ... **ELSE**
... **ENDIF** ... **C;**

Used only in an assembler coding. Assembles an unconditional branch at the dictionary pointer. This branches to the code after **ENDIF**.

ENDIF (--)

Format: **CODE** *name* ... **IF** ... **ENDIF** ... **C;**

Used only in an assembler coding. Terminates a conditional branch from **IF** and **ELSE**.

BEGIN (--)

Format: **CODE** *name* ... **BEGIN** ... **UNTIL** ... **C;**

Used only in an assembler coding. Starts an indefinite loop with **UNTIL.**

UNTIL (n --)

Format: **CODE** *name* ... **BEGIN** ... **UNTIL** ... **C;**

Used only in an assembler coding. Assembles a conditional branch at the dictionary pointer. This branches to the code after **BEGIN** if the condition is true; otherwise resumes execution after **UNTIL.** See **Branch Condition Codes** for values of *n*.

NEXT (--)

Format: **CODE** *name* ... **NEXT**

Assembles the FORTH Next word interpreter instruction to the dictionary pointer and terminate the assembler coding. In this implementation, the instruction is equivalent to **N) JMP C;**.

BRAWAIT (--)

Format: **CODE** *name* ... **BRAWAIT**

Assembles a branch instruction to return to the multitasking loop and terminate the assembler coding. This causes the current task to become idle and pass control to the next task.

Branch Condition Codes:

The following condition codes are used with **IF** and **UNTIL** as described above as well as the mnemonics **DBCC, SCC** and **TRAPCC**. These leave a branching code on the stack which is OR'ed with the branch instruction. At execution, the CPU status register bit(s) are tested for condition. See Motorola (1990b), especially under **Bcc**, for more details.

0=	Tests the zero status bit.
0<	Tests the negative status bit.
0>	Tests if the value is greater than zero.
CS	Tests the carry status bit.
LS	Tests if the values is less than zero or same.
VS	Tests the overflow status bit.

Note: **CS, LS** and **VS** are part of the 68332 status tests, which may not be found as a standard FORTH assembler word.

Mnemonics:

ABCD	Add decimal (BCD values) with extend
ADD	Add two values
ADDX	Add two values with extend
AND	Logical AND two values
ANDSR	Logical AND immediate to Status Register
ASL	Arithmetic shift left (msb → C → X, 0 → lsb)
ASR	Arithmetic shift right (lsb → C → X, copy msb)
BCHG	Test a bit and change it
BCLR	Test a bit and clear it
BGND	Enter background mode
BKPT	Breakpoint
BRA	Branch to new PC
BSET	Test a bit and set it
BSR	Branch to subroutine
BTST	Test a bit
CHK	Check register against 0 and upper bounds
CHK2	Check register against lower and upper bounds
CLR	Clear an operand
CMP	Compare two values
CMP2	Compare register against lower and upper bounds
CMPM	Compare memory contents
COM	Logical one's complement (equivalent to MC68332's **NOT**)
DBCC	Test condition, decrement and branch (See **Branch Condition Codes**)
DBRA	Decrement and branch
DIVS	Signed divide
DIVU	Unsigned divide
EOR	Exclusive OR two values
EORSR	Exclusive OR immediate to Status Register
EXG	Exchange registers
EXT	Sign extend a register
ILLEGAL	Illegal instruction trap
JMP	Jump to new PC
JSR	Jump to subroutine
LEA	Load effective address
LINK	Link and allocate
LPSTOP	Low-power stop

LSL	Logical shift left (msb → C → X, 0 → lsb)
LSR	Logical shift right (lsb → C → X, 0 → msb)
MOV	Move data from source to destination
MOVC	Move data to/from control register
MOVM	Move data between memory and multiple registers. This mnemonic also uses the following words:

RL	Placeholder for register list, also tells which direction to place values. This must either precede a predecrement or follow an indirect or postincrement addressing.
\	This separates a sublist of registers.
\\	This ends the register list.

Note: The register list must follow the mnemonic

Example:

**A6)+ RL MOVM D0 D3 \ A1 ** *ok*

is equivalent to

MOVEM.L (A6)+,D0-D3/A1

MOVP	Move data to/from peripheral
MOVS	Move date to/from address space
MFSR	Move data from Status Register
MFUSP	Move data from User Stack Pointer
MTSR	Move data to Status Register
MTUSP	Move data to User Stack Pointer
MULS	Signed multiply
MULU	Unsigned multiply
NBCD	Negate decimal (BCD values) with extend
NEG	Negate a value (2's complement)
NEGX	Negate a value with extend
NOP	No operation
OR	OR two values
ORSR	OR immediate to Status Register
PEA	Push effective address
RESET	Reset system (CPU remains intact)
ROL	Rotate register left (msb → C → lsb)
ROR	Rotate register right (lsb → C → msb)
ROXL	Rotate register left with extend (X → lsb, msb → C → X)
ROXR	Rotate register right with extend (X → msb, lsb → C → X)
RTD	Return and deallocate (use with LINK)
RTE	Return from interrupt
RTR	Return and restore condition codes
RTS	Return from subroutine
SBCD	Subtract decimal (BCD values) with extend
SCC	Set according to condition (See **Branch Condition Codes**)
STOP	Load status register and stop
SUB	Subtract two values
SUBX	Subtract two values with extend
SWP	Swap words in a data register
TAS	Test and set an operand
TBLS	Table lookup and interpolate (signed)
TBLSN	Table lookup and interpolate (signed, result not rounded)
TBLU	Table lookup and interpolate (unsigned)
TBLUN	Table lookup and interpolate (unsigned, result not rounded)
TRAP	Cause a TRAP exception
TRAPCC	Trap on condition (See **Branch Condition Codes**)
TRAPV	Trap on overflow
TST	Test an operand
UNLK	Unlink

TattleTale Flash EPROM Burner

The TattleTale Model 7 (TT7) uses the Intel 28F020 Flash EPROM chips that allows up to 512K of read-only space. In this implementation, the FORTH core with the user dictionary appended is stored in this space.

The user must set up the hardware properly before attempting to burn a new program in the EPROM:

1. The power supply to the TT7 must be at 12.0 volts; no less than 11.4 volts and no more than 12.6 volts as per EPROM's specification.

2. Pins F13 (VPROG) and F17 (+12V) must be jumpered together on the TT7 to enable the EPROM for burning.

Then burning the EPROM may be done in one of two ways:

1. Use **TT7SAVE** to add new FORTH words created in the user dictionary to the end of the FORTH core in the EPROM.

2. Use **TT7LOAD** or **TT7SLOAD** to download the entire FORTH core plus any additions to the user's dictionary. This allows updates to the FORTH core through the console (SCI) interface rather than use the 68332's BDM interface which may not be accessible.

The following FORTH words are used to load the buffer in memory and/or burn the new program into the EPROM.

SRECORD (a1 n1 -- a2 n2 tf | 0 tf | ff)

Converts the Motorola S-record format at address *a1* and length *n1* into binary data which is copied to *a1*. Returns values to the stack on one of the three conditions:

1. If the record begins with **S0**, then a zero and a true flag is put on stack.

2. If the record begins with **S1**, **S2** or **S3**, then the memory address *a2* of the data and the data length *n2* is left on stack.

3. If the record begins with **S7**, **S8** or **S9**, which signifies the end of the S-records, then just the false flag is returned on stack.

All other record types and any errors such as invalid record, length and checksum will abort with a message.

The Motorola S-record format comprises the following string (note all digits are hexadecimals -- two digits for each byte):

Stllaaaadddd...ddcc

t numerical digit indicating the record type:

Type	Function
0	Start of the S-record file
1 to 3	Contains data
7 to 9	End of S-record file

ll length of the entire binary string not including the checksum field.

aaaa memory address at the start of the data, the length depends on the record type:

Type	Address length in bytes
0	0
1 or 9	2
2 or 8	3
3 or 7	4

dddd...dd data

cc checksum, which is the one's complement of the sum of the byte values from the length byte up to the checksum byte.

Example:

" S107010012042AFFB8" SRECORD .S DROP HERE 128 + SWAP DUMP
256 4 -1 <-Top
0000009A 12 04 2A FF ... ok*
the address for the **DUMP** is arbitrary.

TT7BURN (--)

Copies the Burner Program into the Standby RAM and executes. This should be used after **TT7LOAD** or **TT7SLOAD** or the user runs the risk of corrupting the data in the Pseudo-static RAM (the disk routines also use this RAM). During execution the program will print the status of its operation and the EPROM to the console. At the end it does a system reset and starts the new FORTH operating system.

Example:

```
TT7BURN
Begin EPROM Burning
EPROM ID Check  -- Completed, Retries = 0000
Zero EPROM      -- Completed, Retries = 0000
Erase EPROM     -- Completed, Retries = 0095
Burn EPROM      -- Completed, Retries = 0000

(system startup banner...)
```

The "retries" field in the status report indicates the quality of the Flash EPROM. Higher numbers indicate the EPROM will be unreliable during future EPROM burns. If, instead of the *Completed, Retries* status, a *Failed, Error = x* appears, this indicates something is wrong with the EPROM:

Error #	Cause
3	Could not erase EPROM -- more than 3000 retries
5	Could not zero or burn EPROM -- more than 25 retries
9	Invalid EPROM ID -- check the programming voltage and pins on the TT7.

If all else fails, the EPROM must be bad which means a new TT7 must be used or its EPROMs must be replaced. The EPROM chips are good for 10,000 zero, erase and burn operations, so care must be taken not to save the FORTH system too many times.

TT7CKSM (--)

Computes and prints the checksum in hexadecimal format. The data in the Pseudo-static RAM is used to derive the checksum after using **TT7LOAD** or **TT7SLOAD**. The checksum is defined to be the sum of all unsigned bytes, then truncated to 16 bits. This is to make sure the data were received correctly.

Example:

```
TT7CKSM Checksum = 1234 ok
```

TT7LOAD (n f --)

Reads in *n* 1024 byte blocks of data from the console with the type depending on the flag *f*. If this flag is true, receives data in hexadecimal format, otherwise in binary format. This routine uses **KEY**, so the data are not echoed back to the console. With the hexadecimal format, any non-digit character will be ignored.

Since this only places the data in the Pseudo-static RAM, the user must execute **TT7BURN** before doing anything else.

The maximum number of blocks depends on the size of the Flash EPROM. In this case, the number of blocks must be between 1 and 512.

Related words: **TT7SLOAD TT7BURN**

TT7SAVE (--)

Appends the contents of the user dictionary to the end of the FORTH core in the Flash EPROM. This is done by copying the FORTH core and then the user dictionary to the Pseudo-static RAM and executes **TT7BURN**. The user dictionary is permanently saved in the EPROM (at least until a new FORTH core update or **TT7SAVE** is executed later). When a system reset or power-up occurs, this user dictionary is copied to the Static RAM.

Related words: **EMPTY FORGET**

TT7SLOAD (--)

Reads in Motorola S-records from the console and places the data in the Pseudo-static RAM. When this routine displays "Receiving S Records", send the S-records from a file in an external computer. Records beginning with "S7", "S8" or "S9" indicates the end of the file. "S0" records are ignored.

Since this only places the data in the Pseudo-static RAM, the user must execute **TT7BURN** before doing anything else.

The address in the S-records must be in the range from the start of the Flash EPROM to the upper limit of the EPROM. In this implementation, the address must be from hexadecimal 080000 to 0FFFFF. An address outside this range or an invalid S-record format would abort with an error.

Related words: **SRECORD** **TT7LOAD** **TT7BURN.**

FORTH Commands Specific for the SIS

These FORTH words are specific for the Surface Irradiance Spectrometer (SIS). This instrument is designed to scan incident irradiance in 38 channels (i.e. wavelengths) using the Data Translation's DT5742-PGL A/D converter. The converter resides at the address FFF90000.

1X (-- n)

Returns the gain constant for use with the A/D converter. This returns a zero (0) for a gain of 1.

10X (-- n)

Returns the gain constant for use with the A/D converter. This returns a one (1) for a gain of 10.

100X (-- n)

Returns the gain constant for use with the A/D converter. This returns a two (2) for a gain of 100.

500X (-- n)

Returns the gain constant for use with the A/D converter. This returns a three (3) for a gain of 500.

ADC (n1 n2 -- u)

Sets gain *n1* and channel number *n2* for A/D conversion and returns an unsigned 16-bit value in *u*. The gain can be used from these constants: **1X 10X 100X 500X**. A timeout of 10 milliseconds is applied and if timed out, returns the value from **ADTIMEOUT**.

Note: Channels 0 to 15 uses TT7 on-board A/D chip and 16 to 31 uses the DT A/D converter.

Example:

```
10X 18 ADC . 234 ok
```

Related words: **SDA ADCAVG ADCDLY ADTIMEOUT ADCWAIT**

ADC.DEFAULT (-- a)

A variable array containing the default gains and analog channel numbers for use with **GD**. The first word contains the number of channels in the list. The remainder of the array contains alternating byte values of analog channel number and gain. Currently, 13 channels are used in the following list in order:

Channel	Gain
0	1X
1	1X
2	1X
3	1X
23	1X
24	1X
25	1X
26	1X
27	1X
28	1X
29	1X
30	1X
31	1X

Related words: **ADC.LIST ADC.NUMCHAN GD GDDEF**

ADC.LIST (-- a)

A variable array containing analog channel numbers and gains of up to 32 channels for use with **GD**. Each channel number and gain is stored as byte values together as one element. The user may modify these to suit his requirements.

Example:

```
DECIMAL 15 256 * 10X + ADC.LIST 3 2*
+ H! ok
```

stores channel 15 with gain of 10X to the 4th element in the array.

Related words: **ADC.DEFAULT ADC.NUMCHAN GD GDDEF**

ADC.NUMCHAN (-- a)

A variable containing the number of analog channels stored in **ADC.LIST** for use with **GD**.

Related words: **ADC.DEFAULT ADC.LIST GD GDDEF**

ADCAVG (n --)

Averages all analog conversions by *n* replicates. At startup, this value defaults to 1. Internally, the analog values are saved in 64-bit sums.

Related word: **ADC**

ADCDLY (n1 n2 --)

Sets the wait period *n1* in microseconds for one of the four gains *n2*, which range from 0 to 3. This is the time from STROBE to A/D TRIGGER of the DT converter.

Example:

50 10X ADCDLY *ok*
sets 50 microseconds delay for 10X gain.

Related word: **ADC**

ADCWAIT (--)

Waits until interrupt is complete or timeout.

Example:

1X 20 ADC ADCWAIT . *1234 ok*

Related word: **ADC**

DA38 (n1 n2 -- n3)

Sets gain *n1* and channel *n2*, which range from 0 to 47, for A/D conversion and returns value in *n3* from the 38-channel radiometer.

Example:

1X 30 DA38 ADCWAIT . *567 ok*

.DA38 (n --)

Formatted display of all 48 channels of the 38-channel radiometer for given gain *n*.

Related word: **SCAN38**

GD (--)

Gets data from analog channels specified in **ADC.LIST** and **ADC.NUMCHAN** and the spectrometer data and outputs them according to **BASE.**

If **BASE** contains the value 2 (see **BINARY**), the output will be binary data; otherwise, the output will be converted to the value in **BASE.**

The organization of the output (in binary format) is as follows, assuming the default setup, (Note: For fields 2 or more bytes long, the bytes are reversed to make it compatible with the VAX byte order):

Pos	Len	Description
0	2	Length of the data in bytes
2	1	Shutter value (0=open, 1=close)
3	2	Number of analog averages
5	1	Number of analog channels
6	4	analog channel values (def = 13)
.	.	
.	.	
.	.	
58	1	Number of spectrometer channels (def = 48)
59	4	Spectrometer channel values
.	.	
.	.	
.	.	
251	2	Checksum = byte summation of all data in the output, including the length field.

Note: For each of the analog and Spectrometer channel values, the four bytes are encoded as:

Pos	Len	Description
0	1	Gain value. If the 7th bit of this field is set, a timeout has occurred and the analog value field contains invalid data.
1	1	Channel number
2	2	analog value

Related words: **ADC ADC.NUMCHAN ADC.LIST DA38 GDGAIN**

GDDEF (--)

Sets **ADC.LIST** and **ADC.NUMCHAN** to the default specified by **ADC.DEFAULT** for use with **GD**.

Related words: **ADC.DEFAULT** **ADC.LIST** **ADC.NUMCHAN** **GD**

GDGAIN (-- a)

Variable containing the gain to be used with **GD**.

Example:

10X GDGAIN ! *ok*
sets gain for all analog channels to 10X.

Related word: **GD**

SCAN38 (n --)

Outputs a string of 48 channels based upon gain *n* and **BASE**. This is useful for transfer data to spreadsheets and other programs in different computers.

Related word: **.DA38**

SHUTTER (f --)

Sets the action of the shutter. If the flag is zero, then opens the shutter, otherwise closes the shutter. This is useful to obtain dark counts for correcting the radiometric data.

FORTH Commands Specific for the MOS

These FORTH words are specific for the Marine Optic Spectrometer (MOS). This instrument is designed to scan irradiance and radiance in 512 channels (i.e. wavelengths) of each the blue (340 to 640 nm) and the red (600 to 900 nm). In addition, analog channels are used to obtain oceanographic and internal data such as pressure, temperatures, etc.

1X (-- n)

Returns the gain constant for use with the A/D converter. This returns a zero (0) for a gain of 1.

10X (-- n)

Returns the gain constant for use with the A/D converter. This returns a one (1) for a gain of 10.

100X (-- n)

Returns the gain constant for use with the A/D converter. This returns a two (2) for a gain of 100.

ADC (n1 n2 -- u)

Sets gain *n1* and channel number *n2* of the A/D converter and returns an unsigned 12-bit value *u*. The gain can be used from these constants: **1X 10X 100X.** A timeout of 10 milliseconds is applied and if timed out, returns the value from **ADTIMEOUT.**

Note: Channels 0 to 15 uses TT7 on-board A/D chip and 16 to 31 uses the external A/D converter.

Example:

```
10X 18 ADC . 234  ok
```

The following are words that uses **ADC** with known channel numbers. The stack diagram for each is (n1 -- u), as described above.

TILTX	X axis tilt
TILTY	Y axis tilt
WTEMP	Water temperature
BATT	Battery voltage
DTEMP	LED diode block temperature
ITEMP	Internal temperature
PRESS	Pressure
BTEMP	Blue diode array temperature
RTEMP	Red diode array temperature

Related words: **SDA ADCAVG ADCDLY ADTIMEOUT ADCWAIT**

.ADC (--)

Retrieves and displays all the defined channels listed in **ADC** to the terminal in tabular format.

.SEEMUX (--)

Prints the calibrated MUX positions for the 10 MUX channels in tabular format.

ADC.DEFAULT (-- a)

A variable array containing the default gains and analog channel numbers for use with **GD.** The first word contains the number of channels in the list. The remainder of the array contains alternating byte values of analog channel number and gain. Currently, 19 channels are used in the following list in order:

Channel	Gain
0	1X
1	1X
2	1X
3	1X
16	1X
17	1X
18	1X
19	1X
20	1X
21	1X
22	1X
23	1X
24	1X
25	1X
26	1X
27	1X
28	1X
29	1X
30	1X

Related words: **ADC.LIST ADC.NUMCHAN GD GDDEF**

ADC.LIST (-- a)

A variable array containing analog channel numbers and gains of up to 32 channels for use with **GD**. Each channel number and gain is stored as byte values together as one element. The user may modify these to suit his requirements.

Example:

DECIMAL 21 256 * 10X + ADC.LIST 8 2* + H! *ok*
stores channel 21 with gain of 10X to the 9th element in the array.

Related words: **ADC.DEFAULT ADC.NUMCHAN GD GDDEF**

ADC.NUMCHAN (-- a)

A variable containing the number of analog channels stored in **ADC.LIST** for use with **GD**.

Related words: **ADC.DEFAULT ADC.LIST GD GDDEF**

ADCAVG (n --)

Averages all A/D conversions by *n* replicates. At startup, this value defaults to 1. Internally, the analog values are saved in 64-bit sums.

Related word: **ADC**

BLUBUF (-- a)
REDBUF (-- a)

Buffer of 1024 bytes long to hold each of the blue and red spectroradiometer data. Each element in the array is 2 bytes long.

Related words: **BRECV RRECV GD**

BCOOL (b --)
RCOOL (b --)

Sets the blue and red CCD coolers with an 8-bit value (0 to 255). This controls the temperature of the CCD arrays with the higher the value, the lower the temperature. A zero value turns off the cooler circuitry. These values are also saved in variables **BCOOL.VAL** and **RCOOL.VAL.**

Example:

30 BCOOL 37 RCOOL *ok*
sets blue CCD cooler to 30 and red CCD cooler to 37.

BCOOL.VAL (-- a)
RCOOL.VAL (-- a)

CCD array cooler values last used with **BCOOL** and **RCOOL.**

BDELAY (--)
RDELAY (--)

Wait for period of time (determined from **BLUTIM** and **REDTIM**) before acquiring data from the spectroradiometers.

Related words: **BINTEG RINTEG**

BDISP (--)
RDISP (--)

These obtains and displays in hexadecimal format the spectroradiometer scan for either the blue or red CCD.

BINTEG (n --)
RINTEG (n --)

Sets the integration time for the spectroradiometers. The input value *n* is the index to the times in the constant INTEGTIME. The following commands are available to set the integration times in seconds:

BINT.25	**RINT.25**	= 0.25 sec
BINT.5	**RINT.5**	= 0.5 sec
BINT1	**RINT1**	= 1.0 sec
BINT2	**RINT2**	= 2.0 sec
BINT4	**RINT4**	= 4.0 sec
BINT8	**RINT8**	= 8.0 sec
BINT16	**RINT16**	= 16.0 sec
BINT32	**RINT32**	= 32.0 sec
BINT64	**RINT64**	= 64.0 sec
BINT128	**RINT128**	= 128.0 sec
BINT256	**RINT256**	= 256.0 sec
BINT512	**RINT512**	= 512.0 sec
BINT1024	**RINT1024**	= 1024.0 sec

BINIT (--)
RINIT (--)

Initializes the blue and red spectroradiometers and resets the FORTH variables.

BLULEN (-- a)
REDLEN (-- a)

The actual length of data received in **BLUBUF** and **REDBUF**.

BRESET (--)
RRESET (--)

Resets the blue and red spectroradiometers to the default values and sets the baud rate of the ports.

BSTART (--)
RSTART (--)

Starts acquiring the data from the blue and red spectroradiometers based on the integration times defined. The data are saved into **BLUBUF** and **REDBUF** to be later retrieved. **BLULEN** and **REDLEN** contains the number of bytes received by the spectroradiometers.

Related words: **BINTEG** **RINTEG**

BLUTIM (-- a)
REDTIM (-- a)

Variables containing the blue and red integration time index used in **BINTEG** and **RINTEG.**

COMPASS (-- n)

Returns the compass direction in degrees magnetic. The value is in integer degrees.

This uses the KVH compass through a TPU asynchronous serial channel and the compass value is decoded from the string.

FID (-- n)

This returns the fiducial position value which is defined as:

0	= between positions
1	= dark position
3	= up position
5	= down position
7	= calibration position

.FID (--)

This reads the fiducial position and output position in English.

Related word: **FID**

FIDON (--)

Turns on the fiducial motor.

Related words: **FIDOFF** **FIDPOS**

FIDOFF (--)

Turns off the fiducial motor.

Related words: **FIDON** **FIDPOS**

FIDPOS (n --)

Sets the fiducial to a position (see **FID** for position values). The follow words are available to set the fiducial to known positions:

DARK	Sets the fiducial to dark position.
UP	Sets the fiducial to up position.
DOWN	Sets the fiducial to down position.
CALIB	Sets the fiducial to calib position.

GD (--)

Gets data from analog channels specified in **ADC.LIST** and **ADC.NUMCHAN**, compass direction, fiducial and fiber optic positions and blue and red spectrometers and outputs them according to **BASE**.

If **BASE** contains the value 2 (see **BINARY**), the output will be binary data; otherwise, the output will be converted to the value in **BASE**.

The organization of the output (in binary format) is as follows, assuming the default setup, (Note: For fields 2 or more bytes long, the bytes are reversed to make it compatible with the VAX byte order):

Pos	Len	Description
0	2	Length of the data in bytes
2	2	Number of analog averages
4	1	Number of analog channels
5	4	analog channel values
.	.	(def = 19)
.	.	
.	.	
81	2	Compass direction in deg M
83	1	Fiducial position
84	1	Fiber optic position
85	4	Blue integration time (msec)
89	1	Blue CCD cooler value
90	1024	Blue spectrometer data
1114	4	Red integration time (msec)
1118	1	Red CCD cooler value
1119	1024	Red spectrometer data
2143	2	CRC Checksum of all data in the output, including the length field.

Note: For each of the analog and Spectrometer channel values, the four bytes are encoded as:

Pos	Len	Description
0	1	Gain value. If the 7th bit of this field is set, a timeout has occurred and the analog value field contains invalid data.
1	1	Channel number
2	2	analog value

Related words: **ADC** **ADC.NUMCHAN** **ADC.LIST** **GDAD** **GDGAIN**

GDAD (--)

Gets data from analog channels specified in **ADC.LIST** and **ADC.NUMCHAN** and compass direction and outputs them according to **BASE.**

This is useful to get only the analog channels and the compass readings for a quick look at the instrument status since the spectrometers and the fiber optic multiplexer takes time to setup, acquire and transmit data.

If **BASE** contains the value 2 (see **BINARY**), the output will be binary data; otherwise, the output will be converted to the value in **BASE.**

The organization of the output (in binary format) is as follows, assuming the default setup, (Note: For fields 2 or more bytes long, the bytes are reversed to make it compatible with the VAX byte order):

Pos	Len	Description
0	2	Length of the data in bytes
2	2	Number of analog averages
4	1	Number of analog channels
5	4	analog channel values
.	.	(def = 19)
.	.	
.	.	
81	2	Compass direction in deg M
83	2	CRC Checksum of all data in the output, including the length field.

Note: For each of the analog and Spectrometer channel values, the four bytes are encoded as:

Pos	Len	Description
0	1	Gain value. If the 7th bit of this field is set, a timeout has occurred and the analog value field contains invalid data.
1	1	Channel number
2	2	analog value

Related words: **ADC** **ADC.NUMCHAN** **ADC.LIST** **GD** **GDGAIN**

GDDEF (--)

Sets **ADC.LIST** and **ADC.NUMCHAN** to the default specified by **ADC.DEFAULT** for use with **GD.**

Related words: **ADC.DEFAULT** **ADC.LIST** **ADC.NUMCHAN** **GD**

GDGAIN (-- a)

Variable containing the gain to be used with **GD.**

Example:

10X GDGAIN ! *ok*
sets gain for all analog channels to 10X.

Related word: **GD**

MUX.HLDTQ (f --)

Switches the holding torque. If the flag is true, turns it on; otherwise turns it off. This freezes the multiplexer so the armature will not move, especially in rough seas. This must be set after the multiplexer is positioned.

Note: This would draw battery current, so use this judicially.

Note: The multiplexer must be turned on with **MUXON**, otherwise an error will be displayed.

MUX.HOME (--)

Sets the multiplexer to its home position. This positions the armature to "zero" position to start positioning the armature to the correct position with **MUX.IDXDN** or **MUX.IDXUP.**

Note: The multiplexer must be turned on with **MUXON**, otherwise an error will be displayed.

MUX.IDXDN (n --)

Move the multiplexer armature *backwards* by *n* half-steps. This is a relative movement from the last position. Use **MUX.HOME** to define an absolute position.

Note: The multiplexer must be turned on with **MUXON**, otherwise an error will be displayed.

Example:

MUX.HOME 200 MUX.IDXDN *ok*
sets the armature 200 half-steps back from the home position.

Related words: **MUX.HOME** **MUX.IDXUP**

MUX.IDXUP (n --)

Move the multiplexer armature *forwards* by *n* half-steps. This is a relative movement from the last position. Use **MUX.HOME** to define an absolute position.

Note: The multiplexer must be turned on with **MUXON**, otherwise an error will be displayed.

Example:

MUX.HOME 340 MUX.IDXUP *ok*
sets the armature 340 half-steps forward from the home position.

Related words: **MUX.HOME** **MUX.IDXDN**

MUXCHAN (n --)

Sets the multiplexer to the fiber optic channel number *n*, which ranges from 0 to 10 (0 = home position which has no fiber optic cable connected). This value is stored in **MUXCHAN.VAL.**

Note: The multiplexer must be turned on with **MUXON**, otherwise an error will be displayed. The calibrated MUX positions (in counts from home position) are stored in the Serial EEPROM. Use **SEEMUX** to enter the calibrated values.

Example:

2 MUXCHAN *ok*
sets the multiplexer to the second fiber optic cable.

Related words: **MUX.HLDTQ MUX.HOME MUX.IDXUP MUXCHAN.VAL MUXON**

MUXCHAN.VAL (-- a)

A variable containing the current fiber optic channel position. The value stored ranges from 0 to 10 (0 = home position).

Related word: **MUXCHAN**

MUXON (n --)

Sets up the TPU channel *n* as the stepper motor function (i.e. **TPUF.SM)** to control the fiber optic multiplexer and turns on the multiplexer. The channel number ranges from 0 to 15 (modulo 16) and is stored in **MUXTPU.CHAN.** This also checks the CRC checksum in the Serial EEPROM and supply default values if CRC does not match. Use **SEEMUX** to enter calibrated MUX values.

Example:

10 MUXON *ok*

Related words: **MUXCHAN MUXOFF MUXTPU.CHAN**

MUXOFF (n --)

Disables the TPU channel (the value is stored in **MUXTPU.CHAN)** from the stepper motor function and turns off the fiber optic multiplexer. This is used to conserve power from the batteries.

Related word: **MUXON MUXTPU.CHAN**

MUXTPU.CHAN (-- a)

A variable containing the TPU channel number used for fiber optic multiplexer operations.

Related word: **MUXON**

SEEMUX (n1 n2 --)

Calibrated MUX positions *n1* in stepper motor counts from the home position is stored in the Serial EEPROM for MUX channel *n2*. The channel number ranges from 1 to 10. The calibrated values are stored beginning at address 258 in the EEPROM. Address 256 is reserved for the CRC checksum. The values for all channels occupies 22 bytes of the EEPROM.

Example:

DECIMAL 141 2 SEEMUX *ok*
stores 141 counts for MUX channel 2.

Related word: **.SEEMUX**

FORTH Quick Reference Guide

Constants

0	(-- n)
1	(-- n)
2	(-- n)
3	(-- n)
4	(-- n)
6	(-- n)
8	(-- n)
-1	(-- n)
-CELL	(-- n)
0.	(-- d)
1.	(-- d)
BL	(-- c)
C/L	(-- n)

User Variables

'ABORT
'AUTO
BASE
C#
'CLEAN
CONTEXT
'CR
CSP
CTR
CURRENT
DEVICE
'EXPECT
FENCE
H
'IDLE
>IN
'KEY
L#
LAST
'MARK
'PAGE
PTR
R0
S0
SPAN
STATE
STATUS
'TAB
TIB
'TYPE
VOC-LINK
WARNING
WIDTH

Arithmetic

Single-precision

+	(n1 n2 -- n1+n2)
-	(n1 n2 -- n1-n2)
*	(n1 n2 -- n1*n2)
/	(n1 n2 -- n1/n2)
*/	(n1 n2 n3 -- (n1*n2)/n3)
MOD	(n1 n2 -- r)
/MOD	(n1 n2 -- r q)
*/MOD	(n1 n2 n3 -- r q)
1+	(n -- n+1)
1-	(n -- n-1)
2+	(n -- n+2)
2-	(n -- n-2)
4+	(n -- n+4)
4-	(n -- n-4)
2*	(n -- n*2)
2/	(n -- n/2)
4*	(n -- n*4)
4/	(n -- n/4)
ABS	(n -- n)
CELL-	(n -- n-4)
CELL+	(n -- n+4)
CELLS	(n -- n*4)
NEGATE	(n -- -n)

Double-precision

D+	(d1 d2 -- d1+d2)
D-	(d1 d2 -- d1-d2)
DABS	(d -- d)
DNEGATE	(d -- -d)

Triple-precision

T*	(d n -- t)
T/	(t n -- d)

Mixed-precision

M+	(n1 d1 -- d2)
M*	(n1 n2 -- d)
M/	(d n1 -- n2)
M*/	(d1 d2 n -- d3)
U*	(n1 n2 -- d)
U/	(d n1 -- n2)
U/MOD	(d n1 -- r q)

Comparisons

Single-precision

=	(n1 n2 -- f)
>	(n1 n2 -- f)
<	(n1 n2 -- f)
0=	(n -- f)
0<	(n -- f)
0>	(n -- f)
U<	(n1 n2 -- f)
MIN	(n1 n2 -- n3)
MAX	(n1 n2 -- n3)
WITHIN	(n lower upper -- f)

Double-precision

D0=	(d -- f)
D=	(d1 d2 -- f)
D<	(d1 d2 -- f)
D>	(d1 d2 -- f)
DMIN	(d1 d2 -- d3)
DMAX	(d1 d2 -- d3)

Strings

-TEXT	(a1 n a2 -- f)
-MATCH	(a1 n1 a2 n2 -- a1 n1 tf | a3 n3 ff)

Logic

NOT	(n1 -- n2)
AND	(n1 n2 -- n3)
OR	(n1 n2 -- n3)
XOR	(n1 n2 -- n3)
1COM	(n1 -- n2)
<<	(n1 bits -- n2)
>>	(n1 bits -- n2)
>H<	(n:1234 -- n:3412)
><	(n:1234 -- n:1243)
>4<	(n:1234 -- n:4321)

Memory

!	(n a --)
+!	(n a --)
@	(a -- n)
C!	(b a --)
C+!	(b a --)
C@	(a -- b)
H!	(h a --)
H+!	(h a --)
H@	(a -- h)
U@	(a -- uh)
2H!	(h1 h2 a --)
2H@	(a -- h1 h2)
2U@	(a -- uh1 uh2)
2!	(d a --)
2@	(a -- d)
PAD	(-- a)
"	(-- a n)
CMOVE *or* CMOVE>	(src des n --)
<CMOVE	(src des n --)
MOVE *or* MOVE>	(src des n --)
<MOVE	(src des n --)
FILL	(a n fill-byte --)
ERASE	(a n --)
BLANKS	(a n --)
DUMP	(a n --)
WDUMP	(a n --)
LDUMP	(a n --)
BUFFER	(len -- a)

Stack Manipulations

Single-precision

DROP	(n --)
DUP	(n -- n n)
?DUP	(n -- n n | n)
OVER	(n1 n2 -- n1 n2 n1)
PICK	(n1 -- n2)
ROLL	(n --)
ROT	(n1 n2 n3 -- n2 n3 n1)
SWAP	(n1 n2 -- n2 n1)

Double-precision

2DROP	(d --)
2DUP	(d1 -- d1 d1)
2OVER	(d1 d2 -- d1 d2 d1)
2ROT	(d1 d2 d3 -- d2 d3 d1)
2SWAP	(d1 d2 -- d2 d1)

Return Stack

>R	(n --)
R>	(-- n)
2>R	(d --)
2R>	(-- d)
R@	(-- n)
I	(-- n)
I'	(-- n)
J	(-- n)
J'	(-- n)
K	(-- n)
K'	(-- n)

Structure Control

BEGIN ... UNTIL	
UNTIL	(f --)
BEGIN ... WHILE ... REPEAT	
WHILE	(f --)
BEGIN ... AGAIN	
AGAIN	(--)

IF ... ELSE ... THEN	
IF (f --)	
DO ... LOOP	
DO	(n1 n2 --)
LOOP	(--)
DO ... +LOOP	
DO	(n1 n2 --)
+LOOP	(n --)
DO ... /LOOP	
DO	(n1 n2 --)
/LOOP	(u --)
LEAVE	(--)
BACK	(a --)

Character Input

?KEY	(-- c)
KEY	(-- c)
EXPECT	(a n --)
QUERY	(--)
STRAIGHT	(a n --)
ENCLOSE	(a1 c -- a1 n1 n2 n3)
(FIND)	(a nfa -- pfa n tf | ff)
-FIND	(-- pfa n tf | ff)
WORD	(c --)
COUNT	(a -- a+1 n)

Character Output

EMIT	(c --)
TYPE	(a n --)
-TRAILING	(a n1 -- a n2)
."	(str --)
SPACE	(--)
SPACES	(n --)
CR	(--)
PAGE	(--)
TAB	(line col --)
MARK	(a n --)
CLEAN	(--)
BEEP	(--)

Input Number Conversion

?DIGIT	(a -- a+1 n tf | a+1 ff)
CONVERT	(d1 a -- d2 a)
NUMBER	(a -- n | d)

Output Number Conversion

.	(n --)
?	(a --)
D.	(d --)
D.R	(d n --)
U.	(n --)
U.R	(n width --)
X.	(n width --)

Number Formatting

<#	(--)
SIGN	(n d -- d)
#	(d -- d)
#S	(d -- d)
HOLD	(c --)
#ASC	(a n --)
#>	(d -- a n)

Interpretation

(	(--)
' *name*	(-- a)
INTERPRET	(--)

Compilation

,	(n --)
H,	(h --)
C,	(b --)
<BUILDS ... DOES>	
;CODE	(--)
[	(--)
]	(--)
COMPILE *name*	
[COMPILE] *name*	
IMMEDIATE	(--)
SMUDGE	(--)
LITERAL	(n --)
DLITERAL	(d --)

Defining Words

: *name* ... ;	(--)
CREATE *name*	(--)
CONSTANT *name*	(n --)
VARIABLE *name*	(n --)

HCONSTANT *name*	(h --)
HVARIABLE *name*	(h --)
2CONSTANT *name*	(d --)
2VARIABLE *name*	(d --)
USER *name*	(n --)

Vocabularies

VOCABULARY *name*	(--)
DEFINITIONS	(--)
VLIST	(--)
FORTH	(--)
ASSEMBLER	(--)
TT7	(--)

Dictionary Management

FORGET *name*	(--)
EMPTY	(--)
ALLOT	(n --)
HERE	(-- a)
LATEST	(-- a)
CFA	(pfa -- cfa)
LFA	(pfa -- lfa)
NFA	(pfa -- nfa)
PFA	(nfa -- pfa)
TRAVERSE	(a1 n -- a2)
ID.	(nfa --)

Operating System

General

ABORT	(--)
.ABORT	(--)
ABORT"	(--)
EXECUTE	(pfa --)
@EXECUTE	(a --)
EXIT	(--)
QUIT	(--)
PAUSE	(--)
STOP	(--)
TRY	(--)
RECOVER	(--)
BYE	(--)

Stack control

!CSP	(--)
'S	(-- a)
SP@	(-- a)
SP!	(--)
RP@	(-- a)
RP!	(--)

Resource control

GET	(a --)
RELEASE	(a --)

Multitasking control

ACTIVATE	(a --)
BACKGROUND	(nu ns nr --)
BUILD	(a --)
CONSTRUCT	(a --)
HIS	(a1 a2 -- a3)
TERMINAL	(a n --)
WAKE	(-- n)
OPERATOR	(-- a)

Error Handling

?ERROR	(f n --)
ERROR	(n --)
MESS	(n --)
?COMP	(--)
?CSP	(--)
?EXEC	(--)
?PAIRS	(n1 n2 --)
?STACK	(--)
EXCEPTION	(a vector --)
TPUEXCEPTION	(a chan --)

Date and Time Words

Internal clock

counter	(-- a)
COUNTER	(-- n)
TIMER	(n --)
MS	(n --)

Time of day

!TIME	(hh:mm:ss --)
@TIME	(-- time)
:00	(d1 -- d2)
(TIME)	(time -- a n)
TIME	(--)
HMS	(d -- s m h)

Date

YMD	(date -- d m y)
JULDAY	(date -- julday)
CALDAY	(julday -- date)
WEEKDAY	(date -- weekday)
!DATE	(dd/mm/yyyy --)
@DATE	(-- date)
MONTH	(n1 -- a n2)
DAY/WEEK	(n1 -- a n2)
(DATE)	(date -- a n)
DATE	(--)
.RTC	(--)

Alarm

SETALARM	(dd/mm/yyyy hh:mm:ss --)
.ALARM	(--)
?ALARM	(-- f)

Scheduling

SCHED	(pfa date time per cyc sch# --)
CLRSCHED	(sch# --)
.SCH	(--)
SCANSCHED	(--)
SLEEP	(--)
?SLEEP	(-- f)

Tools

BINARY	(--)
DECIMAL	(--)
HEX	(--)
DEPTH	(-- n)
S->D	(n -- d)
.S	(--)
TOGGLE	(a mask --)
?CELL	(n -- n f)
CALCCRC	(crc1 a n -- crc2)
UPDATECRC	(crc1 b -- crc2)

MC68332 Words

CPU Registers

SIM.MCR
SIM.SYNCR
SIM.RSR
SIM.PORTE
SIM.DDRE
SIM.PEPAR
SIM.PORTF
SIM.DDRF
SIM.PFPAR
SIM.SYPCR
SIM.PICR
SIM.PITR
SIM.SWSR
SIM.CSPAR

CPU Words

CRYSTAL	(-- n)
@SYSCLOCK	(-- n)
!SYSCLOCK	(n --)
sysclk	(-- a)
CSWAIT	(wait chan# --)
.CS	(--)
.REG	(--)
.SIM	(--)
.QSM	(--)
CPUPIN	(n --)
PIN	(n -- f)
PCLR	(n --)
PSET	(n --)
.PIN	(--)

QSM Registers

QSM.QMCR
QSM.QILVR
QSM.SCCR0
QSM.SCCR1
QSM.SCSR
QSM.SCDR
QSM.QPDR
QSM.QPAR
QSM.QDDR
QSM.SPCR0
QSM.SPCR1
QSM.SPCR2
QSM.SPCR3
QSM.SPSR
QSM.RECRAM
QSM.XMTRAM
QSM.CMDRAM

Serial I/O

SCIBUF	(-- a)
'SCI	(-- a)
BAUD	(-- a)
!BAUD	(baud --)
@BAUD	(-- baud)
XSHAKE	(f --)
QBAUD	(-- a)
!QBAUD	(baud --)
@QBAUD	(-- baud)
(TYPE)	(--)
(EXPECT)	(--)
(CR)	(--)
(PAGE)	(--)
(MARK)	(--)
(TAB)	(l c --)
(CLEAN)	(--)
(ECHO)	(f c --)
(EMIT)	(c --)
(SAVEKEY)	(c1 f1 f2 -- c2)
(STRAIGHT)	(c -- f)

TPU Registers

TPU.TMCR
TPU.TICR
TPU.CIER
TPU.CFSR
TPU.HSQR
TPU.HSRR
TPU.CPR
TPU.CISR
TPU.PRAM

TPU Functions

TPUF.DIO
TPUF.ITC
TPUF.OC
TPUF.PWM
TPUF.SPWM
TPUF.PM
TPUF.PSP
TPUF.SM
TPUF.PPWA
TPUF.UART

TPU I/O

.TPU	(--)
TPUCLOCK	(-- freq)
!CIER	(f chan --)
@CIER	(chan -- f)
!CISR	(f chan --)
@CISR	(chan -- f)
^CISR	(chan --)
!HSQR	(seq chan --)
@HSQR	(chan -- seq)
!HSRR	(req chan --)
@HSRR	(chan -- req)
HSRRWAIT	(chan --)
!CPR	(prior chan --)
@CPR	(chan -- prior)
!CFSR	(fn chan --)
@CFSR	(chan -- fn)
!PRAM	(n i chan --)
@PRAM	(i chan -- n)
TPU.BUFPTR	(-- a)
!TPUBUF	(bufa inta chan --)
@TPUBUF	(chan -- bufa)
?TPUBUF	(-- f)
tpuvend	(-- a)

TPU Serial I/O

TSEROPEN	(outflg a len chan --)
TSERBAUD	(baud chan --)
TSERPAIR	(chan1 chan2 --)
TSERXSHAKE	(f chan --)
TSERFLUSH	(chan --)
TSERTIMEOUT	(n chan --)
TSERLEN	(chan -- n)
TSERGET	(chan -- n)
TSERPUT	(n chan --)
TSERPUTS	(a n chan --)
TSERCLOSE	(chan --)

Hard Disk Management

Disk I/O

DRV.AVLSECT	(-- a)
DRV.CLUSTSZ	(-- a)
DRV.TOTSECT	(-- a)
DRV.NDISK	(-- a)
DRIVE.TABLE	(-- #head #cyl #sect)
DRIVE	(f --)
DREADY	(--)
DINFO	(a --)
.DINFO	(--)
.DSTAT	(--)
DSECTOR	(lsn ns --)
DREAD	(addr lsn ns --)
DWRITE	(addr lsn ns --)

DOS Disk Words

FORMAT	(--)
CHKDSK	(--)
FOPEN	(a n -- fptr)
FSEEK	(n fptr --)
FREAD	(buf n fptr -- bytesread)
FGETS	(buf n fptr -- bytesread)
FWRITE	(buf n fptr --)
FPUTS	(buf n fptr --)
FEOF	(fptr --)
FCLOSE	(fptr --)
FTYPE	(a n --)
FDUMP	(a n --)
DEL	(a n --)
REN	(a1 n1 a2 n2 --)
COPY	(a1 n1 a2 n2 --)
DIR	(--)
CHDIR *or* CD	(a n --)
MKDIR *or* MD	(a n --)
RMDIR *or* RD	(a n --)
LOG	(a n --)

Flash EPROM Burner

TT7LOAD	(n f --)
SRECORD	(a1 n1 -- a2 n2 tf | 0 tf | ff)
TT7SLOAD	(--)
TT7CKSM	(--)
TT7BURN	(--)
TT7SAVE	(--)

Serial EEPROM

!SEE	(b a --)
!SEE2	(h a --)
!SEE4	(n a --)
@SEE	(a -- b)
@SEE2	(a -- h)
@SEE4	(a -- n)
SEECRC	(--)
.SEE	(--)
SEECPUPIN	(n --)
SEEPIN	(n --)
SEEPCLR	(n --)
SEEPSET	(n --)

A/D Conversion

ADTIMEOUT	(-- n)
SDA	(chan# -- n)
sdasr	(-- a)
.SDA	(--)

MC68332 FORTH Assembler

CODE *name* ... C;
LABEL *name* ... C;
: *name* (FORTH words) ... ;CODE ... C;

Data Registers:

D0 D1 D2 D3 D4 D5 D6 D7

Address Registers:

A0 A1 A2 A3 A4 A5 A6 A7

A2 *or* N	Next word
A3 *or* U	User space
A4 *or* W	Word pfa address
A5 *or* I	Interpreter pointer
A6 *or* S	Parameter stack pointer
A7 *or* R	Return stack pointer

Control Registers:

SFC	Source function code
DFC	Destination function code
USP	User stack pointer
VBR	Vector base register

Addressing Modes:

H.	Word (2 bytes) operation
B.	Byte (1 byte) operation
Default	Longword (4 bytes) operation

Indirect addressing modes:

)	simple indirect
)+	with postincrement
-)	with predecrement
0)	with 16-bit displacement
1)	
2) *or* N)	(See **Address Registers**)
3) *or* U)	
4) *or* W)	
5) *or* I)	
6) *or* S)	
7) *or* R)	
PC)	program counter with displacement
+X	index w/8-bit word displacement
+XL	index w/8-bit longword displacement

Absolute addressing:

AB	(a --)

Immediate addressing modes:

#Q	quick immediate
#B	byte-sized immediate
#H	word-sized immediate
#	longword-sized immediate

Branching Words:

IF ... ELSE ... ENDIF
BEGIN ... UNTIL

NEXT
BRAWAIT

Condition Codes:

0= 0< 0> CS LS VS NOT

Mnemonic List:

ABCD
ADD
ADDX
AND
ANDSR
ASL
ASR
BCHG
BCLR
BGND
BKPT
BRA
BSET
BSR
BTST
CHK
CHK2
CLR
CMP
CMP2
CMPM
COM
DBCC
DBRA
DIVS
DIVU
EOR
EORSR
EXG
EXT
ILLEGAL
JMP
JSR
LEA
LINK
LPSTOP
LSL
LSR
MOV
MOVC
MOVP
MOVS
MFSR
MFUSP
MTSR
MTUSP
MULS
MULU
NBCD
NEG
NEGX
NOP
OR
ORSR
PEA
RESET
ROL
ROR
ROXL
ROXR
RTD
RTE
RTR
RTS
SBCD
SCC
STOP
SUB
SUBX
SWP
TAS
TBLS
TBLSN
TBLU
TBLUN
TRAP
TRAPCC
TRAPV
TST
UNLK

MOVM (register(s)) \ (registers(s)) \\
RL Register list placeholder

SIS Specific Words

A/D Converter

1X	(-- n)
10X	(-- n)
100X	(-- n)
500X	(-- n)
ADC	(gain chan -- u)
ADCAVG	(n --)
ADCDLY	(n1 n2 --)
ADCWAIT	(--)

Data Output

ADC.DEFAULT	(-- a)
ADC.LIST	(-- a)
ADC.NUMCHAN	(-- a)
DA38	(n1 n2 -- n3)
.DA38	(n --)
GD	(--)
GDDEF	(--)
GDGAIN	(-- a)
SCAN38	(n --)
SHUTTER	(f --)

MOS Specific Words

A/D Converter

1X	(-- n)
10X	(-- n)
100X	(-- n)
ADC	(gain chan -- u)
ADCAVG	(n --)
.ADC	(--)

Known A/D Channels

TILTX	(gain -- n)
TILTY	(gain -- n)
WTEMP	(gain -- n)
BATT	(gain -- n)
DTEMP	(gain -- n)
ITEMP	(gain -- n)
PRESS	(gain -- n)
BTEMP	(gain -- n)
RTEMP	(gain -- n)
COMPASS	(-- n)

CCD Array Acquisition

BINIT	(--)
RINIT	(--)
BRESET	(--)
RRESET	(--)
BSTART	(--)
RSTART	(--)
BLUTIM	(-- a)
REDTIM	(-- a)
BLUBUF	(-- a)
REDBUF	(-- a)
BLULEN	(-- a)
REDLEN	(-- a)
BCOOL	(b --)
RCOOL	(b --)
BCOOL.VAL	(-- a)
RCOOL.VAL	(-- a)
BDELAY	(--)
RDELAY	(--)
BDISP	(--)
RDISP	(--)

CCD Integration Times

BINTEG	(n --)
RINTEG	(n --)
BINT.25	(--)
RINT.25	(--)
BINT.5	(--)
RINT.5	(--)
BINT1	(--)
RINT1	(--)
BINT2	(--)
RINT2	(--)
BINT4	(--)
RINT4	(--)
BINT8	(--)
RINT8	(--)
BINT16	(--)
RINT16	(--)
BINT32	(--)
RINT32	(--)
BINT64	(--)
RINT64	(--)
BINT128	(--)
RINT128	(--)
BINT256	(--)
RINT256	(--)
BINT512	(--)
RINT512	(--)
BINT1024	(--)
RINT1024	(--)

Fiducial

FID	(-- n)
.FID	(--)
FIDON	(--)
FIDOFF	(--)
FIDPOS	(n --)
DARK	(--)
UP	(--)
DOWN	(--)
CALIB	(--)

Fiber Optic MUX

MUXON	(chan --)
MUXOFF	(chan --)
MUXTPU.CHAN	(-- a)
MUX.HLDTQ	(f --)
MUX.HOME	(--)
MUX.IDXDN	(n --)
MUX.IDXUP	(n --)
MUXCHAN	(n --)
MUXCHAN.VAL	(-- a)
SEEMUX	(pos mux# --)
.SEEMUX	(--)

Data Output

ADC.DEFAULT	(-- a)
ADC.LIST	(-- a)
ADC.NUMCHAN	(-- a)
GD	(--)
GDAD	(--)
GDDEF	(--)
GDGAIN	(-- a)