Glossary

Kernel V4.08 QED-Forth Functions

This Glossary provides detailed descriptions of all of the library routines, called "Words" in FORTH, provided in firmware on the QVGA Controller, Panel-Touch Controller, and QED-4 Board (aka QED-Flash Board). This Glossary contains the following:

- ⇨*A Categorized List of All Functions (Words) in the V4.08 Kernel;*
- ⇨*A Description of Stack Symbols, Abbreviations and Naming Conventions;*
- ⇨ *A List of Words that Disable Interrupts;*
- ⇨*A Main Glossary of the QED-Forth Library Words;*
- ⇨*A Glossary of Assembler Words;*
- ⇨*A Glossary of C Debugger Words; and,*
- ⇨*A Glossary of Kernel Extension Words Available in Source Form.*

Entries are alphabetized in the following ASCII word order:

! " # $ % & ' ( ) * + , - . / 0 1 … 9 : ; < = > ? @ A B C … Z [\ ] ^ _ ` a b c … z { | } ~

Categorized Word List

Word Categories

ARRAY
ASSEMBLER
C DEBUGGING
COMPARISON
CONTROL STRUCTURES
DEBUG, TRACE, AND BENCHMARK
DEFINITION
DEVICE DRIVERS
Analog I/O Drivers
Digital I/O Drivers
High Current Drivers
Keypad and Display Drivers
Real-Time Clock
Serial I/O Drivers
Stepper Motor Drivers
DICTIONARY
FLASH MANAGEMENT
FLOATING POINT COMPARISON
FLOATING POINT CONSTANTS
FLOATING POINT MATH
FLOATING POINT STRING CONVERSION AND I/O
HEADERLESS FUNCTIONS
HEAP
INTERPRETER AND COMPILER
INTERRUPTS
LOGICAL
MASS MEMORY
MATH
MATRIX DIMENSIONING AND ACCESS
MATRIX EDITING
MATRIX I/O
MATRIX MATH
MATRIX ROW/COL OPERATIONS
MEMORY
MEMORY MAP
MULTITASKING AND TIME-KEEPING
NUMERIC CONVERSION
OPERATING SYSTEM
SERIAL I/O ROUTINES (Character, String, and Number I/O)
SOURCE FORM ROUTINES
STACK
STRING
STRUCTURES
VECTOR

ARRAY

2ARRAY.FETCH	COPY.ARRAY	MAX#DIMENSIONS
2ARRAY.STORE	DELETED	PF.STACK.FRAME
?ARRAY.SIZE	DIM.CONSTANT.ARRAY:	SWAP.ARRAYS
?DIMENSIONS	DIMENSIONED	ZERO.ARRAY
ARRAY.PF	FILL.ARRAY	[0]
ARRAY:	BLANK.ARRAY	[]

ASSEMBLER (SEE ASSEMBLER GLOSSARY)

>ASSM	BLT	DIR	LSLA	SBCA
>FORTH	BMI	ELSE,	LSLB	SBCB
ABA	BNE	END-CODE	LSLD	SEC
ABX	BPL	END.CODE	LSR	SEI
ABY	BRA	ENDIF,	LSRA	SEV
ADCA	BRCLR	EORA	LSRB	STAA
ADCB	BRN	EORB	LSRD	STAB
ADDA	BRSET	EQ	LT	STD
ADDB	BSET	EXT	MI	STOP
ADDD	BSR	FDIV	MUL	STS
AGAIN,	BVC	GE	NE	STX
ALWAYS	BVS	GT	NEG	STY
ANDA	CALL	HI	NEGA	SUBA
ANDB	CBA	HS	NEGB	SUBB
ANY.BITS.CLR	CC	IDIV	NEVER	SUBD
ANY.BITS.SET	CLC	IF,	NOP	SWI
ASL	CLI	IMM	ORAA	TAB
ASLA	CLR	INC	ORAB	TAP
ASLB	CLRA	INCA	PL	TBA
ASLD	CLRB	INCB	PSHA	TEST
ASR	CLV	IND,X	PSHB	THEN,
ASRA	CMPA	IND,Y	PSHX	TPA
ASRB	CMPB	INH	PSHY	TST
ASSEMBLER	CODE	INS	PULA	TSTA
BCC	COM	INX	PULB	TSTB
BCLR	COMA	INY	PULX	TSX
BCS	COMB	JMP	PULY	TSY
BEGIN,	CPD	JSR	REL	TXS
BEQ	CPX	LDAA	REPEAT,	TYS
BGE	CPY	LDAB	ROL	UNTIL,
BGT	CS	LDD	ROLA	VC
BHI	DAA	LDS	ROLB	VS
BHS	DEC	LDX	ROR	WAI
BITA	DECA	LDY	RORA	WHILE,
BITB	DECB	LE	RORB	XGDX
BLE	DES	LO	RTI	XGDY
BLO	DEX	LS	RTS	
BLS	DEY	LSL	SBA	

C DEBUGGING (SEE C DEBUGGER GLOSSARY)

=CHAR	CHAR	FP_QtoC	MAIN
=FLOAT	CHAR*	FPtoString	PrintFP
=INT	DO[]	INT	
=LONG	FLOAT	INT*	
C$>COUNTED$	FLOAT*	LONG	
CALL.CFN	FP_CtoQ	LONG*	

COMPARISON

<	0=	DMAX	U<	XU<
< >	0>	DMIN	U>	XU>
<=	D<	DRANGE	UMAX	
=	D< >	DU<	UMIN	
>	D=	DU>	URANGE	
>=	D>	MAX	X< >	
0<	D0< >	MIN	X=	
0< >	D0=	RANGE	XRANGE	

CONTROL STRUCTURES

+LOOP	ENDIFTRUE	I+	J\I	REPEAT
AGAIN	ENDOF	IF	K	THEN
BEGIN	EXIT	IFTRUE	LEAVE	UNLOOP
CASE	FOR	IXN-	LOOP	UNTIL
DO	I	IXN+	NEXT	URANGE.OF
ELSE	I'	IXU-	OF	WHILE
ENDCASE	I-	IXU+	OTHERWISE	
ENDIF	I\J	J	RANGE.OF	

DEBUG, TRACE, AND BENCHMARK

(BENCHMARK:)	DEFAULT.TRACE.ACTION	IS.TRACE.ACTION
BENCHMARK:	DUMP.REGISTERS	SINGLE.STEP
BREAK	F*/COUNTER	TRACE
DEBUG	F+COUNTER	

DEFINITION

:	CODE	INTEGER:	VARIABLE
;	CONSTANT	LOCALS{	XADDR:
<DBUILDS	DOES>	MATRIX:	XCONSTANT
<VBUILDS	DOUBLE:	NO.OP	XVARIABLE
2CONSTANT	END.CODE	REAL:	
2VARIABLE	END-CODE	REDEFINE	
ADDR:	FCONSTANT	REGISTER:	
ARRAY:	FVARIABLE	USER	

DEVICE DRIVERS : Analog I/O Drivers

(>DAC)	A/D12.MULTIPLE	A/D8.SAMPLE
(A/D12.MULTIPLE)	A/D12.SAMPLE	INIT.A/D12&DAC
(A/D12.SAMPLE)	A/D8.MULTIPLE	INIT.SPI
(A/D8.MULTIPLE)	A/D8.OFF	SPI.OFF
(A/D8.SAMPLE)	A/D8.ON	SPI.RESOURCE
>DAC	A/D8.RESOURCE	

DEVICE DRIVERS : Digital I/O Drivers

INIT.PIA	PIA.TOGGLE.BITS	PPA
PIA.C!	PORTA	PPB
PIA.C@	PORTA.DIRECTION	PPC
PIA.CHANGE.BITS	PORTD	
PIA.CLEAR.BITS	PORTD.DIRECTION	
PIA.SET.BITS	PORTE	

DEVICE DRIVERS : High Current Drivers

CLEAR.HIGH.CURRENT	SET.HIGH.CURRENT

DEVICE DRIVERS : Keypad and Display Drivers

$>DISPLAY	COMMAND>DISPLAY	KEYPAD
(UPDATE.DISPLAY)	DISPLAY.BUFFER	LINES/DISPLAY
?KEYPAD	DISPLAY.HEAP	PUT.CURSOR
?KEYPRESS	DISPLAY.OPTIONS	UPDATE.DISPLAY
BUFFER.POSITION	GARRAY.XPFA	UPDATE.DISPLAY.LINE
CHAR>DISPLAY	INIT.DISPLAY	
CHARS/DISPLAY.LINE	IS.DISPLAY	
CLEAR.DISPLAY	IS.DISPLAY.ADDRESS	

DEVICE DRIVERS : Real-Time Clock

READ.WATCH	SET.WATCH

DEVICE DRIVERS : Serial I/O Drivers (See also SERIAL I/O ROUTINES)

#INPUT.CHARS	INIT.RS485	SERIAL1.AT.STARTUP
#OUTPUT.CHARS	INIT.SERIAL2	SERIAL1.RESOURCE
?KEY1	KEY1	SERIAL2.AT.STARTUP
?KEY2	KEY2	SERIAL2.RESOURCE
BAUD1.AT.STARTUP	PARITY	TRANSMITTING
BAUD2	PARITY.IN	USE.SERIAL1
DISABLE.SERIAL2	PARITY.OUT	USE.SERIAL2
EMIT1	RS485.RECEIVE	
EMIT2	RS485.TRANSMIT	

DEVICE DRIVERS : Stepper Motor Drivers

CREATE.RAMP	SPEED.TO.DUTY	STEP.MANAGER

DICTIONARY

'	CFA.PTR	FORTH	NP	VOCABULARY
(HERE)	CFA>NAME	HERE	NPAGE	VP
,	CFA>NFA	ID.	ON.FORGET	WIDTH
?HAS.PFA	CFA>PFA	LATEST	PFA>NAME	WORDS
ALLOT	CONTEXT	LINK	PFA>NFA	
ANEW	CURRENT	NFA.FOR	V,	
ASSEMBLER	DEFINITIONS	NFA>CFA	VALLOT	
AXE	DP	NFA>LFA	VC,	
C,	DPAGE	NFA>PFA	VFORTH	
CFA.FOR	FORGET	NHERE	VHERE	

FLASH MANAGEMENT

DOWNLOAD.MAP	PAGE.TO.RAM	TO.FLASH
PAGE.TO.FLASH	STANDARD.MAP	WHICH.MAP

FLOATING POINT COMPARISON

F<	F=	F0< >	F0>	FMIN
F< >	F>	F0<=	F0>=	
F<=	F0<	F0=	FMAX	

FLOATING POINT CONSTANTS

1/INFINITY	1/PI	360/2PI	LOG10(2)	SQRT(2)
-1/INFINITY	1/SQRT(2)	INFINITY	ONE	TEN
1/LN(2)	1/TEN	-INFINITY	PI	ZERO
1/LOG10(2)	2PI/360	LN(2)	PI/2	

FLOATING POINT MATH

>DEGREES	F**	FALOG2	FLOT	FTAN
>RADIANS	F/	FASIN	FNEGATE	INT.FLOOR
1/F	F^N	FATAN	FP.ERROR	INT.PART
10^N	F+	FCOS	FP.POP	OVERFLOW
DFIXX	F2*	FINT	FP.PUSH	RANDOM#
DFLOT	F2/	FIXX	FRANDOM	RANDOM.GAUSSIAN
DINT	FABS	FLN	FRTI	UFIXX
DINT.FLOOR	FACOS	FLOG10	FSCALE	UFLOT
F-	FALN	FLOG2	FSIN	UNDERFLOW
F*	FALOG10	FLOOR	FSQRT	

FLOATING POINT STRING CONVERSION AND I/O

$>F	FILL.FIELD	FP&STRING.POP	NO.SPACES
ASK.FNUMBER	FIXED	FP&STRING.PUSH	RIGHT.PLACES
F.	FIXED.	FP.DEFAULTS	SCIENTIFIC
F>FIXED$	FLOATING	LEFT.PLACES	SCIENTIFIC.
F>FLOATING$	FLOATING.	MANTISSA.PLACES	TRAILING.ZEROS
F>SCIENTIFIC$	FNUMBER	NEXT.NUMBER	

HEADERLESS FUNCTIONS : (See their glossary entries for details)

BUFFER>SPI	CALC.CHECKSUM

HEAP

.HANDLES	CURRENT.HEAP	HANDLE.PTR	ROOM
?HANDLE.SIZE	DEALLOCATED	HEAP.PTR	START.HEAP
+CURRENT.HEAP	DUP.HEAP.ITEM	HEAP.STRUCTURE.PF	TO.HEAP
+HEAP.HANDLE	FREE.HANDLE	IS.HEAP	TRANSFER.HEAP.ITEM
+HEAP.PAGE	FROM.HEAP	RECOVER.HANDLE	
ALLOCATED	H.INSTANCE:	RESIZE.HANDLE	

INTERPRETER AND COMPILER

#FIND	;	2LITERAL	EXECUTE	QUERY
#TIB	?IMMEDIATE	ASCII	FIND	QUIT
(	[	BACKTRACK	HAS.PFA	RECURSE
(#FIND)	[COMPILE]	BLK	IMMEDIATE	SMUDGE
(COMPILE.CALL)	\	CALL	INTERPRET	TIB
(CREATE)	]	COMPILE	LITERAL	UNIQUE.MSG
(EXECUTE)	>ASSM	COMPILE.CALL	LOCALS{	WORD
(FIND)	>FORTH	CREATE	PARSE	
:	>IN	EVALUATE	POCKET	

INTERRUPTS

ATTACH	IC1.ID	OC4.ID
CLOCK.MONITOR.ID	IC2.ID	PULSE.EDGE.ID
COP.ID	IC3.ID	PULSE.OVERFLOW.ID
DISABLE.INTERRUPTS	IC4/OC5.ID	RTI.ID
ENABLE.INTERRUPTS	ILLEGAL.OPCODE.ID	SCI.ID
FP&STRING.POP	IRQ.ID	SPI.ID
FP&STRING.PUSH	OC1.ID	SWI.ID
FP.POP	OC2.ID	TIMER.OVERFLOW.ID
FP.PUSH	OC3.ID	XIRQ.ID

LOGICAL

AND	COMPLEMENT	NOT	TRUE
BOOLEAN	FALSE	OR	XOR

MASS MEMORY

BLOCK	LIMIT	PREV	UFIRST	UREAD/WRITE
BUFFER	LINK_FILE_IO	READ/WRITE	ULIMIT	USE
FIRST	OFFSET	SCR	UPDATE	

MATH

-	1XN+	4XN-	DABS	SIGNED.D>S	XALIGN
*	2-	4XN+	DNEGATE	U*/MOD	XD-
*/	2*	8*	DSCALE	U/	XD+
*/MOD	2/	8/	LOG2	U/MOD	XN-
/	2+	8XN+	M*	U>D	XN+
/MOD	2XN-	ABS	M/MOD	U2/	XU-
|X1-X2|>U	2XN+	D-	MOD	UD*S	XU+
+	3*	D+	NEGATE	UM*	
1-	4-	D>S	RANDOM	UM/MOD	
1+	4*	D>S?	RANDOM#	UMOD	
10*	4/	D2*	S>D	X1-X2>D	
1XN-	4+	D2/	SCALE	X1-X2>N	

MATRIX DIMENSIONING AND ACCESS

?DIM.MATRIX	M[]	MATRIX->V
?MATRIX.SIZE	M[]!	PF.STACK.FRAME
DELETED	M[]@	REDIMMED
DIM.CONSTANT.MATRIX:	MATRIX.PF	
DIMMED	MATRIX:	

MATRIX EDITING

COLUMN.CONCATENATE	ROW.CONCATENATE	ROW/COL.INSERTED
COLUMN.TRUNCATE	ROW/COL.DELETED	SELECT.COLUMNS

MATRIX I/O

LOAD.MATRIX	M..	MATRIX
M.	M.PARTIAL	

MATRIX MATH

?DETERMINANT	M*MT	S*MATRIX
ALL.COLUMNS.SCALED	MATRIX-	S/MATRIX
ALL.ROWS.SCALED	MATRIX*	S+MATRIX
COPY.MATRIX	MATRIX.ELEMENT*	S-MATRIX
FFT	MATRIX.ELEMENT/	SOLVE.EQUATIONS
IFFT	MATRIX.MAX	SWAP.MATRIX
INVERTED	MATRIX.MIN	TRANSFORM.MATRIX
IS.IDENTITY	MATRIX.SUM	TRANSPOSED
L.INVERTED	MATRIX.VARIANCE	U.INVERTED
LEAST.SQUARES	MATRIX+	ZERO.MATRIX
LU.BACKSUBSTITUTION	MT*M	
LU.DECOMPOSITION	RANDOMIZED	

MATRIX ROW/COL OPERATIONS

ROW/COL-	ROW/COL.FILL	ROW/COL/
ROW/COL*	ROW/COL.INSERTED	ROW/COL+
ROW/COL*+	ROW/COL.IS.UNITY.LENGTH	ROW/COL->V
ROW/COL.ALL=	ROW/COL.MAX	S.ROW/COL-
ROW/COL.ANY=	ROW/COL.MIN	S.ROW/COL*
ROW/COL.CENTERED	ROW/COL.SUM	S.ROW/COL/
ROW/COL.COPY	ROW/COL.SWAP	S.ROW/COL+
ROW/COL.DELETED	ROW/COL.TRANSFORMED	S.ROW/COL<
ROW/COL.DOT.PRODUCT	ROW/COL.VARIANCE	S.ROW/COL>

MEMORY

!	(EE2!)	|2!|	CHANGE.BITS	MOVE.MANY
(!)	(EEC!)	|2@|	CLEAR.BITS	OFF
(@)	(EEF!)	|F!|	CMOVE	ON
(+!)	(EEX!)	|F@|	CMOVE.IN.PAGE	SET.BITS
(+C!)	(F!)	|X!|	CMOVE.MANY	THIS.PAGE
(2!)	(F@)	|X@|	DEFAULT.PAGE	TO
(2@)	(MOVE)	+!	ERASE	TOGGLE.BITS
(C!)	(PAGE.LATCH)	+C!	F!	X!
(C@)	(SET.BITS)	2!	F@	X@
(CHANGE.BITS)	(TOGGLE.BITS)	2@	FILL	
(CLEAR.BITS)	(X!)	BLANK	FILL.MANY	
(CMOVE)	(X@)	C!	MOVE	
(EE!)	@	C@	MOVE.IN.PAGE	

MEMORY MAP

DP	SO	USE.PAGE
NP	UPAD	UTIB
RO	UPOCKET	VP

MULTITASKING AND TIME-KEEPING

(STATUS)	INSTALL.MULTITASKER	RESOURCE.VARIABLE:
*100US=TIMESLICE.PERIOD	KILL	SEND
?GET	MAILBOX:	SERIAL
?RECEIVE	MICROSEC.DELAY	SERIAL.ACCESS
?SEND	NEXT.TASK	SET.WATCH
ACTIVATE	PAUSE	START.TIMESLICER
ASLEEP	READ.ELAPSED.SECONDS	STATUS
AWAKE	READ.ELAPSED.TIME	STOP.TIMESLICER
BUILD.STANDARD.TASK	READ.WATCH	TASK:
BUILD.TASK	RECEIVE	TASK'S.USER.VAR
DISK.RESOURCE	RELEASE	TIMESLICE.COUNT
GET	RELEASE.AFTER.LINE	UP
HALT	RELEASE.ALWAYS	
INIT.ELAPSED.TIME	RELEASE.NEVER	

NUMERIC CONVERSION

#	BASE	FP&STRING.PUSH	PAD
#>	CONVERT	HEX	SIGN
#S	DECIMAL	HOLD	
<#	DIGIT	NEXT.NUMBER	
ASK.NUMBER	FP&STRING.POP	NUMBER	

OPERATING SYSTEM

#USER.BYTES	CR.BEFORE.MSG	RESTORE
((ERROR))	CUSTOM.ABORT	RP!
(ABORT)	CUSTOM.ERROR	SAVE
(ERROR)	DEFAULT.REGISTER.INITS	STANDARD.RESET
(RP)	INIT.VITAL.IRQS.ON.COLD	STATE
ABORT	INSTALL.REGISTER.INITS	UABORT
ABORT"	NO.AUTOSTART	UERROR
AUTOSTART	NO.VITAL.IRQ.INIT	WARM
COLD	PRIORITY.AUTOSTART	
COLD.ON.RESET	RO	

SERIAL I/O ROUTINES: Character, String, and Number I/O
(see also Serial I/O Drivers in the DEVICE DRIVERS Section)

.	D.R	QUIET
."	DIN	RECEIVE.HEX
.R	DUMP	SPACE
.S	DUMP.INTEL	SPACES
?	DUMP.S1	SPAN
?KEY	DUMP.S2	TAB.WIDTH
ASK.FNUMBER	EMIT	TYPE
ASK.NUMBER	EXPECT	U.
BEEP	ID.	U?KEY
CFA>NAME	INPUT.STRING	UD.R
CHARS/LINE	KEY	UEMIT
COUNT.TYPE	NEXT.WORD	UKEY
CR	PAUSE.ON.KEY	XMIT.DISABLE
D.	PFA>NAME	

SOURCE FORM ROUTINES (SEE SOURCE FORM GLOSSARY)
(These utilities are provided as source code on diskette)

.LINE	EMPTY.BUFFERS	LOAD
-->	FLUSH	SAVE.BUFFERS
>L	INIT.UREAD/WRITE	SUBSTRING
2xN.MATRIX*	IS.RAMDISK	THRU
3xN.MATRIX*	LINE>$	
BLOCK.BUFFERS	LIST	

STACK

(SP)	2SWAP	DUP>R	NDROP	SWAP
.S	3	F.OVER.N	NEEDED	TUCK
?DUP	3DROP	F>R	NIP	X.OVER.N
> <	3DUP	F2DROP	OVER	X>R
>R	4	F2DUP	PF.STACK.FRAME	X2DROP
0	4DROP	FDROP	PICK	X2DUP
0\0	4DUP	FDUP	R@	XDROP
1	D.OVER.N	FDUP>R	R>	XDUP
-1	D>R	FOVER	R>DROP	XDUP>R
2	DEPTH	FPICK	ROLL	XOVER
-2	DPICK	FR@	-ROLL	XPICK
2DROP	DR@	FR>	ROT	XR@
2DUP	DR>	FR>DROP	-ROT	XR>
2DUP>R	DR>DROP	FRAME.DROP	S0	XR>DROP
2OVER	DROP	FROT	SP!	XROT
2ROT	DUP	FSWAP	STACK.FRAME	XSWAP

STRING

"	,"	COUNT	SKIP>
$COMPARE	/STRING	SCAN	-TRAILING
$MOVE	BL	SKIP	UPPER.CASE

STRUCTURES

+CURRENT.HEAP	DEALLOCATED	MEMBER->	STRUCTS->
+HEAP.HANDLE	DOUBLE->	OR.TYPE.OF:	STRUCTURE.BEGIN:
+HEAP.PAGE	DOUBLES->	PAGE->	STRUCTURE.END
ADDR->	FIELD	REAL->	TYPE.END
ADDRS->	H.INSTANCE:	REALS->	TYPE.OF:
ALLOCATED	HEAP.STRUCTURE.PF	RESERVED	V.INSTANCE:
BYTE->	HNDL->	SIZE.OF	XADDR->
BYTES->	INT->	STRING->	XADDRS->
D.INSTANCE:	INTS->	STRUCT->	XHNDL->

VECTOR MATH

2V.TRANSFORM	S.V-	S.V<	V.ANY=	V.SWAP
COL->V	S.V*	S.V>	V.COPY	V.TRANSFORM
DOT.PRODUCT	S.V.ALL=	V-	V.FILL	V/
MATRIX->V	S.V.ANY=	V*	V.MAX	V+
ROW/COL->V	S.V/	V*+	V.MIN	V<
ROW->V	S.V+	V.ALL=	V.SUM	V>

Stack Symbols and Naming Conventions

Stack Conventions and Notation

Understanding stack notation is very important when programming in Forth. Each glossary entry contains a "stack comment" (also called a "stack picture") that describes the inputs and outputs of the function. In addition, we strongly recommend that you include in your own source code a stack comment for each function that you write. This chapter describes the standard stack notation used throughout this glossary and all of the QED-Forth Software and Hardware documentation.

Each stack picture is enclosed in parentheses, so it is automatically treated as a comment by the QED-Forth compiler. A stack picture indicates the input(s) and output(s) of a routine. Inputs are listed to the left of the -- symbol, and outputs are listed to the right. Stack items are separated by the \ character which can be read as "under". For example, the stack picture of the addition routine + is:

```
( n1\n2 --  n3 )
```

which indicates that the function expects two integer inputs (n1 under n2), and returns the integer result n3 as an output. If we want to convey a bit more information within the stack picture, we can add an explanatory phrase within the parentheses. The standard way to do this is to insert a vertical bar symbol before the explanatory phrase. For example, a more instructive stack picture for the + routine would be:

```
( n1\n2 --  n3  |  n3 = n1 + n2 )
```

which tells us that the output n3 is the sum of the two inputs.

Even words that expect no inputs and return no outputs should be documented with a stack picture; in this case the stack picture would be as follows:

```
(  --  )
```

Alternate stack inputs or outcomes are indicated by placing the alternatives within brackets as in

```
( -- [n1\true] or [false] )
```

The word associated with this stack picture either places a signed integer under a true flag on the stack, or places a false flag on the stack.

The smallest item that can be stored on the stack is called a "cell". Each stack cell consists of two bytes. The most significant byte is stored on the top of the stack (in the lower memory location), and the data stack grows downward toward low memory. For stack items whose data size is less than two bytes, for example a character, page or byte, the datum is stored in the least significant byte of the stack cell.

The remainder of this chapter describes the standard stack symbols, data structure names, and naming conventions used in the QED-Forth Glossary and documentation.

Stack Symbols

The following table describes the standard symbols used to represent items placed on the data stack.

Stack Symbol	Size on Stack	Size in Memory	Valid Data Size	Meaning
addr	1 cell	2 bytes	2 bytes	16-bit address. Range: 0 to 65,535 (0-0xFFFF).
page	1 cell	2 bytes	1 byte	Page. Range: 0 to 255 (0-0xFF).
xaddr	2 cells	4 bytes	3 bytes	32-bit extended address comprising an address and page: addr\page. In paged memory, the address immediately after 0x7FFF on a specified page is 0000 on the following page.
$addr	1 cell	2 bytes	2 bytes	16-bit address of the count byte of an ASCII string. The count byte precedes the string in memory.
x$addr	2 cells	4 bytes	3 bytes	Extended string address, same as: $addr\page.
xtask.id	2 cells	4 bytes	3 bytes	Extended task identifier address, also called STATUS addr or base addr of user area.
xresource	2 cells	4 bytes	3 bytes	Extended address of a resource variable.
xmailbox	2 cells	4 bytes	3 bytes	Extended address of a mailbox variable.
cfa	1 cell	2 bytes	2 bytes	16-bit code field address.
nfa	1 cell	2 bytes	2 bytes	16-bit name field address.
pfa	1 cell	2 bytes	2 bytes	16-bit parameter field address.
xcfa	2 cells	4 bytes	3 bytes	Code field xaddr.
xnfa	2 cells	4 bytes	3 bytes	Name field xaddr.
xpfa	2 cells	4 bytes	3 bytes	Parameter field xaddr.
xhandle	2 cells	4 bytes	3 bytes	32-bit address of a memory location that contains a 32-bit xaddr. Typically used to hold the base xaddr of a heap item.
flag	1 cell	2 bytes	2 bytes	Boolean flag, 0 indicates false. Non-zero indicates true.
true	1 cell	2 bytes	2 bytes	Boolean flag, = -1 = 0XFFFF.
false	1 cell	2 bytes	2 bytes	Boolean flag, = 0
-1	1 cell	2 bytes	2 bytes	-1
0	1 cell	2 bytes	2 bytes	0
1	1 cell	2 bytes	2 bytes	1
char	1 cell	1 byte	1 byte	ASCII character.
byte	1 cell	1 byte	1 byte	Unspecified single byte datum.
cnt	1 cell	1 byte	1 byte	Unsigned byte-sized integer, the count of an ASCII string. Range: 0 to 255.
n	1 cell	2 bytes	2 bytes	Signed 16-bit (single) integer. Range: -32,768 to 32,767.
+n	1 cell	2 bytes	2 bytes	Signed positive 16-bit (single) integer. Range: 0 to 32,767.
u	1 cell	2 bytes	2 bytes	Unsigned 16-bit (single) integer range: 0 to 65,535.

Stack Symbol	Size on Stack	Size in Memory	Valid Data Size	Meaning
w	1 cell	2 bytes	2 bytes	Unspecified signed or unsigned 16-bit integer, either n or u.
d	2 cells	4 bytes	4 bytes	Signed 32-bit integer. Range: -2,147,483,648 to +2,147,483,647
+d	2 cells	4 bytes	4 bytes	Signed positive 32-bit (double) integer. Range: 0 to +2,147,483,647.
ud	2 cells	4 bytes	4 bytes	Unsigned 32-bit (double) integer. Range: 0 to 4,294,967,295.
wd	2 cells	4 bytes	4 bytes	Unspecified signed or unsigned double integer, either d or ud.
r	2 cells	4 bytes	4 bytes	Floating point (real) number, sign and exponent are in most significant cell (top of stack, low memory) mantissa is in least significant cell (bottom of stack, high memory).
mode	1 cell	2 bytes	2 bytes	Synonym for n, specifies the address mode for an assembly mnemonic.
condition	1 cell	2 bytes	2 bytes	Synonym for n, specifies the condition for assembly branches and control structures.
arg	1 cell	2 bytes	2 bytes	Synonym for w, an argument passed to an assembly mnemonic. The interpretation of arg depends on the specified mode.
array.xpfa	2 cells	4 bytes	3 bytes	The xpfa of an array. Used to refer to an array as a whole.
matrix.xpfa	2 cells	4 bytes	3 bytes	The xpfa of a matrix. Used to refer to a matrix as a whole.
#rows	1 cell	2 bytes	2 bytes	Specifies the number of rows in a matrix. Range: 0 to 16,383.
row#	1 cell	2 bytes	2 bytes	Used as row index for matrices, range: 0 to 16,383.
#cols	1 cell	2 bytes	2 bytes	Specifies the number of columns in a matrix. Range: 0 to 16,383.
col#	1 cell	2 bytes	2 bytes	Used as column index for matrices. Range: 0 to 16,383.
xvaddr	2 cells	4 bytes	3 bytes	32-bit extended address specifying the starting xaddr of a vector (a vector is a collection of floating point numbers evenly spaced in memory). For the vector operations to function properly, xvaddr must be 4-byte aligned (i.e., it must be an even multiple of 4 bytes). The heap manager and array and matrix dimensioning words perform this alignment automatically.
sep	1 cell	2 bytes	2 bytes	Specifies the separation used by vector words expressed in multiples of 4 bytes. Range: 0 to 16,383. sep=1 means a vector of contiguous floating point numbers, sep=2 means elements are separated by 8 bytes, etc.

Stack Symbol	Size on Stack	Size in Memory	Valid Data Size	Meaning
d.#el	2 cells	4 bytes	3 bytes	Synonym for +d. Number of elements in a vector; the allowed number of elements is limited only by available memory.
\...	? cells			Unspecified number of bytes.
<name>				Indicates that a single word is taken from the input stream.
<text>				Indicates that text is taken from the input stream or placed into the output stream.

Stack Representations of Data Structures

The following table describes the stack pictures used to represent data structures such as arrays, matrices, rows, columns, and vectors.

Data Structure	Stack Picture	Representation
array	**array.xpfa**	An array is represented by its extended parameter field address, which is obtained by using the word **'** (tick). For example, **' MY.ARRAY** returns the **xpfa**.
matrix	**matrix.xpfa**	A matrix is represented by its extended parameter field address, which is obtained by using the word **'** (tick). For example, **' MY.MATRIX** returns the **xpfa**.
row/col	**[row#\\-1] or [-1\\col#]**	A number of words operate on a specified row or column in a matrix. A matrix row is specified by a row number under a -1 under a **matrix.xpfa**: **row#\\-1\\matrix.xpfa --** A matrix column is specified by a -1 under a column number under a **matrix.xpfa**: **-1\\col#\\matrix.xpfa --**
vector	**xvaddr\\sep\\d.#el**	A vector is a collection of floating point numbers evenly spaced in memory. All matrices and their constituent rows and columns can be represented as vectors, and fast vector operators form the basis of all matrix math operations. A vector is represented by an extended base address **xvaddr**, a vector element separation value **sep**, and the 32-bit number of elements in the vector **d.#el**. **xvaddr** must be 4-byte aligned; that is, it must be an even multiple of 4 bytes; the heap manager and array and matrix dimensioning words perform this alignment automatically (see **XALIGN**). **sep** is expressed as a multiple of 4 bytes (e.g., **sep**=1 means a vector of contiguous floating point numbers, **sep**=2 means elements are separated by 8 bytes, and so on). The number of elements in the vector is limited only by available memory. A vector may cross one or more page boundaries.

Naming Conventions

Hexadecimal (base sixteen) numbers are denoted by the prefix 0x. The letters and symbols listed below are commonly used in the names of words to help convey their meaning.

Symbol	Meaning
(	Words delimited by parentheses are either page-less variations of words, as in **(!)** versus **!** , or subsidiary words, as in (CREATE) versus CREATE .
,	Words ending with **,** (comma) are assembler control words. The comma suggests that execution of the word causes code to be compiled into the dictionary. For example, **BEGIN,** ... **UNTIL,** are assembler loop control words analogous to **BEGIN** ... **UNTIL** in high level Forth.
->	These are the final two characters of each structure defining word. They suggest the defining nature of the word.
:	A word ending with a colon indicates a defining word.
2	A word beginning with **2** indicates that it operates on double width 32-bit values.
D	The D prefix indicates that a word operates on double numbers (32-bit values). This prefix is also used to distinguish words related to the definitions area of the dictionary (for example, **DP**, **DPAGE**).
F	The **F** prefix indicates that a word operates on floating point numbers.
M	An **M** indicates that a word operates on matrices or on mixed double/single number types (for example, **UM/**).
N	The **N** prefix distinguishes words related to the names area of the dictionary (for example, **NP**, **NPAGE**).
ROW/COL	Words that operate on a specified row or column in a matrix include **ROW/COL** in their name.
S	The **S** prefix indicates that a word performs an operation involving a scalar (for example, **S*MATRIX**).
U	The **U** prefix indicates that a word operates on unsigned numbers.
V	The **V** prefix indicates that a word operates on vectors. It also distinguishes words related to the variable area (for example, **VP**, **VPAGE**).
X	The X prefix indicates that a word operates on extended addresses.
|	Words delimited by the **|** (vertical bar) character are uninterruptable operators (for example, **|2@|**). These provide robust behavior even when multiple concurrent tasks or routines are accessing a shared memory location or 32 bit variable.

Attributes

Some glossary entries include an "Attributes" field. The following table presents the meaning of the single-character codes used in the Attributes field of the Main Glossary.

Symbol	Meaning
C	Compilation only. The word may be used only during compilation of a colon definition.
D	Defining. The word is a defining word.
I	Immediate. An immediate word is executed (rather than compiled) even if the interpreter is in compilation mode.
M	Multi-programming impact. The word may execute PAUSE and cause a task switch. See the Chapter in this Glossary titled "Words That Disable Interrupts" for a list of routines that call PAUSE.
S	Scratchpad impact. The word modifies one or more scratchpad user variables related to floating point math, floating point string/number conversion, or integer string/number conversion. If the word is to be used inside an interrupt service routine that shares the same user area as another program, then the affected scratchpad variables should be saved at the start of the interrupt service routine and restored before the service routine terminates. The words **FP.PUSH**, **FP.POP**, **FP&STRING.PUSH**, and **FP&STRING.POP** are helpful utilities for saving and restoring the affected scratchpad variables; consult their glossary entries for examples of use.
U	User. The word is a user value.

Words that Disable Interrupts

Certain QED-Forth routines temporarily disable interrupts by setting the I bit in the condition code register. These routines are summarized here to assist you in planning the time-critical aspects of your application.

The kernel provides a set of uninterruptable memory operators that disable interrupts for a few microseconds during the memory access. These are very useful in applications where several tasks or interrupt routines must access a shared memory location. The glossary entries for these words detail the length of time that interrupts are disabled.

```
(CHANGE.BITS)  (CLEAR.BITS)     (SET.BITS)       (TOGGLE.BITS)
CHANGE.BITS    CLEAR.BITS       PIA.CHANGE.BITS  PIA.CLEAR.BITS
PIA.SET.BITS   PIA.TOGGLE.BITS  SET.BITS         TOGGLE.BITS
|2!|           |F!|             |X!|
|2@|           |F@|             |X@|
```

Accessing the PIA (Peripheral Interface Adapter) and LCD display require the insertion of wait states, and the architecture of the QED Board requires that interrupts be disabled while a wait state memory access is in progress. In addition to the PIA routines listed above, the following routines disable interrupts to insert wait states:

```
$>DISPLAY         (UPDATE.DISPLAY)     ?KEYPAD
?KEYPRESS         CHAR>DISPLAY         CLEAR.DISPLAY
COMMAND>DISPLAY   DISPLAY.OPTIONS      INIT.DISPLAY
INIT.PIA          IS.DISPLAY.ADDRESS   KEYPAD
PIA.C@            PIA.C!               PUT.CURSOR
UPDATE.DISPLAY    UPDATE.DISPLAY.LINE
```

The multitasker mediates access to shared resources and ensures smooth transfer of information among tasks. The routines that manage resource variables and mailboxes must disable interrupts for short periods of time to ensure proper access to shared resources and messages. Consequently, the following routines temporarily disable interrupts:

```
?GET    ?RECEIVE   ?SEND
GET     RECEIVE    RELEASE   SEND
```

Consult their glossary entries for details.

The following routines temporarily disable interrupts to ensure that a new task is not corrupted while it is being built:

```
BUILD.STANDARD.TASK    BUILD.TASK
```

These routines disable interrupts to ensure that the elapsed time clock is not updated while it is being read:

```
READ.ELAPSED.SECONDS   READ.ELAPSED.TIME
```

The multitasker is charged with smoothly transferring control among tasks via timeslicing or cooperative task switching. The timeslicer is an interrupt service routine associated with output compare#2. It disables interrupts for the duration of a task switch which requires 25 microseconds plus 3.25 microseconds for each **ASLEEP** task encountered. The cooperative task switch routine

```
PAUSE
```

switches tasks in (27 + 3.25n) microseconds, where n is the number of **ASLEEP** tasks encountered in the round robin task list. Of this time, interrupts are disabled for (20 + 3.25n) microseconds.

The **PAUSE** routine (which temporarily disables interrupts) is called by the following built-in device drivers:

```
EMIT      EMIT1    EMIT2
KEY       KEY1     KEY2
?KEYPAD   KEYPAD
```

The following device driver routines GET and RELEASE resource variables, and so disable interrupts for short periods of time:

```
?KEY              ?KEY1           ?KEY2
?KEYPAD           ?KEYPRESS       >DAC
A/D12.MULTIPLE    A/D12.SAMPLE    A/D8.MULTIPLE
A/D8.SAMPLE       PAUSE.ON.KEY
```

The battery-backed real-time clock option shares the RAM socket on the QED Board. While the "watch" is being read or set by the routines

```
READ.WATCH     SET.WATCH
```

the RAM cannot be accessed, so interrupts cannot be properly serviced. Therefore these routines disable interrupts for approximately 0.5 msec while the watch is being accessed.

All of the routines that write to the EEPROM disable interrupts for 20 msec per programmed byte. This results from the 68HC11's design which prohibits any EEPROM locations from being read while other EEPROM locations are being modified. Since all interrupts are vectored through

EEPROM, interrupts cannot be serviced while an EEPROM storage operation is in progress. The following fundamental EEPROM storage routines

```
(EEC!)   (EE!) (EEX!)   (EEF!)   (EE2!)
```

disable interrupts for 20 msec per programmed byte. These routines are smart enough to avoid programming a byte that already has the correct contents. The following routines may modify EEPROM locations:

```
ATTACH                    AUTOSTART
COLD.ON.RESET             DEFAULT.REGISTER.INITS
DOWNLOAD.MAP              INIT.VITAL.IRQS.ON.COLD
INSTALL.MULTITASKER       INSTALL.REGISTER.INITS
IS.DISPLAY                NO.AUTOSTART
SAVE                      SERIAL1.AT.STARTUP
SERIAL2.AT.STARTUP        STANDARD.MAP
STANDARD.RESET            START.TIMESLICER
```

All of the routines that write to the Flash memory disable interrupts for 20 msec per programmed sector, where a standard sector is 256 bytes. This results from the flash architecture which prohibits any flash locations from being read while other flash locations are being modified. Since interrupts invoke flash-resident code, interrupts cannot be serviced while an flash storage operation is in progress. The following flash routines disable interrupts:

```
PAGE.TO.FLASH   PAGE.TO.RAM    PRIORITY.AUTOSTART    TO.FLASH
```

The following routines disable interrupts and do not re-enable them:

```
DISABLE.INTERRUPTS    SEI
COLD                  WARM
```

DISABLE.INTERRUPTS and its assembly language counterpart **SEI** explicitly set the I bit in the condition code register. The routines **ENABLE.INTERRUPTS** and **CLI** clear the I bit to globally enable interrupts. The restart routines **COLD** and **WARM** disable interrupts so that the initialization process is not interrupted.

Main Glossary

(Alphabetized in ASCII Order)

!	(w\xaddr --) Stores a 16-bit number w at the extended address xaddr. The high order byte is stored at xaddr and the low order byte at xaddr+1. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page. Pronunciation: "store"
"	(-- x$addr) Compile Time: (<text> --) Parses the <text> string in the input stream delimited by a terminating " character. If compiling, emplaces the text in the dictionary as a counted string along with a call to a routine that pushes x$addr to the stack at runtime. If executing, emplaces the string in the dictionary at HERE and leave x$addr (the address of the string) on the stack. Pronunciation: "quote" Attributes: I
#	(ud1 -- ud2) Divides unsigned double number ud1 by the value in BASE to compute the unsigned double quotient ud2 and the integer remainder n. Converts n to an appropriate single ASCII digit character in the current number base and inserts it into the pictured numeric output string below PAD. # is used between <# and #> commands to create a pictured numeric string. Pronunciation: "number-sign" Attributes: S
#>	(d -- xaddr\cnt) Drops d and leaves the xaddr under the count of the pictured numeric output string resulting from the number conversion process initiated by <#. The character count is also stored in location xaddr - 1, so xaddr - 1 can be used as an x$addr. Used to terminate a pictured numeric output sequence which was opened by <# . The pictured numeric output string is located below PAD. Pronunciation: "number-sign-greater"
#FIND	(<name> -- [xcfa\xnfa\flag] or [0]) Executes BL WORD to parse the next space-delimited word from the input stream, and then searches the dictionary for a match of the parsed word. #FIND first searches the CONTEXT vocabulary. Then, if the word is not found and if the CONTEXT and CURRENT vocabularies are different, it searches the CURRENT vocabulary. If the word is not found in the dictionary, it leaves a 0 on the stack. If the word is found, it leaves the word's extended code field address under its extended name field address under a flag on the stack. The flag is +1 if the word is immediate and -1 if the word is not immediate. An error occurs if the input stream is exhausted while WORD executes. A COLD restart will occur if more than 255 page changes are made during the search through either vocabulary. This prevents the interpreter from going on an infinite search through a corrupted dictionary. A COLD restart will also occur if POCKET is not in common memory. Pronunciation: "hash-find"

#INPUT.CHARS (-- n)

Places on the stack the number of characters in the input queue of the secondary serial port (serial2). In other words, returns the number of characters that have been received by the serial2 input interrupt service routine that have not yet been removed from the circular input buffer by KEY2. The default serial2 input buffer holds 80 characters and is located in the system RAM (consult the memory map appendix in the Software Manual for its location). The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output).
Pronunciation: "number-input-chars"

#OUTPUT.CHARS (-- n)

Places on the stack the number of characters in the output queue of the secondary serial port (serial2). In other words, returns the number of characters that have been placed in the output buffer by EMIT2 that have not yet been removed from the circular output buffer by the serial2 output interrupt service routine. The default serial2 output buffer holds 80 characters and is located in the system RAM (consult the memory map appendix in the Software Manual for its location). The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output).
Pronunciation: "number-output-chars"

#S (ud1 -- ud2 | ud2 = 0\0)

Converts all digits of the unsigned double number ud1 by iteratively dividing quotients by BASE and inserting the ASCII symbol for the remainder into the pictured numeric output starting at the left of the string and working towards the right. ud2 is a double number zero. If ud1 equals zero, a single 0 is added to the pictured output buffer below PAD. #S is used between <# and #> commands to create pictured numeric output.
Pronunciation: "number-sign-s" Attributes: S

#TIB (-- xaddr)

User variable that contains the number of characters in the terminal input buffer (TIB). See QUERY .
Pronunciation: "number-t-i-b" Attributes: U

#USER.BYTES (-- n)

n is the number of USER bytes already allocated by QED-Forth. This quantity is useful if the programmer wants to define more user variables. The first additional user variable would be defined as #USER.BYTES USER <name> .

$>DISPLAY(x$addr\n1\n2 -- | n1 = line#, n2 = character#)

Moves the contents of the counted string specified by x$addr to the location in DISPLAY.BUFFER starting at the specified character number n2 on the specified line number n1. Does not move the count of x$addr. Confines the string to the specified line in DISPLAY.BUFFER by clamping the number of characters moved to a maximum equal to the number of character positions remaining after the specified position on the specified line. The line number n1 should be less than LINES/DISPLAY, and the character number n2 should be less than CHARS/DISPLAY.LINE. If the most significant byte of the page in x$addr equals 0xFF, $>DISPLAY accepts a null-terminated (C-style) string. $>DISPLAY does not modify the contents of the LCD display; this will occur upon the next execution of UPDATE.DISPLAY.LINE or UPDATE.DISPLAY or (UPDATE.DISPLAY).

Pronunciation: "string-to-display"

$>F (x$addr -- [r\-1] or [0])
Removes the next space-delimited text string from the input stream and attempts to convert it into a valid floating point number r. Returns r under a true flag if the conversion is successful; otherwise returns a false flag.
Pronunciation: "string-to-f" Attributes: S

$COMPARE (xaddr1\xaddr2\+n1 -- +n2)
Finds the number of common characters +n2 in the strings whose first characters are stored at xaddr1 and xaddr2, respectively. +n1 specifies the maximum number of characters to be compared. Comparison starts at the specified addresses and terminates as soon as an unmatched pair of characters is encountered. +n2 = 0 if there are no common characters found (i.e., if the character at xaddr1 is different from that at xaddr2) or if +n1 is negative. The strings may cross page boundaries.
Pronunciation: "string-compare"

$MOVE (x$addr\xaddr\n --)
Moves the contents of the counted string specified by x$addr to the destination starting at xaddr. Does not move the count byte. The number of characters moved is clamped to a maximum of n bytes.
Pronunciation: "string-move"

' (-- [xpfa] or [0\0])
Compile Time: (<name> --)
Removes <name> from the input stream and returns <name>'s extended parameter field address. Returns 0\0 if <name> has no parameter field (see ?HAS.PFA). If in execution mode, leaves the xpfa on the stack. If in compilation mode, compiles the xpfa as a 2-cell literal in the current definition; the xpfa is pushed to the stack when the definition later executes. An error occurs if no <name> is given or if <name> cannot be found in the dictionary.
Pronunciation: "tick" Attributes: I

((--)
Compile Time: (<text)> --)
Ignores all further input until) or the end of the input stream is encountered. Used to enclose comments. No error occurs if the end of the input stream is encountered. When used in a block, the terminating) can be on a different line than (. However, when used from the terminal, only single line comments are allowed. See also \.
Pronunciation: "paren" Attributes: I

(!) (w\addr --)
Stores a 16-bit number at addr on the current page or in common memory. The high order byte is stored at addr and the low order byte at addr+1.
Pronunciation: "paren-store"

(#FIND) ($addr -- [xcfa\xnfa\flag] or [0])
Searches the dictionary for a match of the counted string at $addr in the common memory. (#FIND) first searches the CONTEXT vocabulary. Then, if the word is not found and if the CONTEXT and CURRENT vocabularies are different, it searches the

CURRENT vocabulary. If the word is not found in the dictionary, (#FIND) leaves a 0 on the stack. If the word is found, it leaves the word's extended code field address under its extended name field address under a flag on the stack. The flag is +1 if the word is immediate and -1 if the word is not immediate. A COLD restart will occur if more than 255 page changes are made during the search through either vocabulary. This prevents the interpreter from going on an infinite search through a corrupted dictionary. A COLD restart will also occur if POCKET is not in common memory.
Pronunciation: "paren-hash-find"

((ERROR)) ([...]\error.id --)
The default routine called by (ERROR) if a system error is detected and the CUSTOM.ERROR flag is false. Prints a descriptive error message, if possible printing the name of the routine that detected the error and any arrays or matrices involved in producing the error condition. Unlike (ERROR), ((ERROR)) does not execute ABORT. In multitasking applications, the availability of ((ERROR)) allows a task with access to the serial line to print intelligible system error messages without executing ABORT. Since ABORT invokes the user-installed autostart routine in a turnkeyed system, the ability to handle errors without invoking ABORT increases the programmer's options. For example, the following user-defined error handler can be installed in UERROR to allow a task to print standard error messages without calling ABORT or invoking an installed autostart routine:

```
: MY.ERROR.HANDLER          ( -- )
        ((ERROR))           \ prints proper error messages
        SP!  RP!            \ initialize data & return stacks
        FORTH DEFINITIONS   \ initialize vocabulary
        QUIT   ;            \ enter interpreter
CFA.FOR MY.ERROR.HANDLER UERROR X!
CUSTOM.ERROR ON             \ install task's error handler
```

See (ERROR), CUSTOM.ERROR, UERROR, and ABORT.
Pronunciation: "paren-paren-error"

(+!) (w\addr --)
Adds w to the 16-bit value stored at addr on the current page or in common memory and stores the result at addr.
Pronunciation: "paren-plus-store"

(+C!) (byte\addr --)
Adds byte to the 8-bit value at addr on the current page or in common memory and stores the result at addr.
Pronunciation: "paren-plus-c-store"

(2!) (w1\w2\addr -- | [addr] gets w2, [addr+2] gets w1)
Stores two 16-bit integers at addr on the current page or in common memory. w2 is stored at addr and w1 is stored at addr+2. Can also be used to store a double number at addr.
Pronunciation: "paren-two-store"

(2@) (addr -- w1\w2)

Fetches two 16-bit integers from addr and addr+2 on the current page or in common memory. w2 is taken from addr and the w1 is from addr+2. Can also be used to fetch a double number from addr.
Pronunciation: "paren-two-fetch"

(>DAC) (byte\n -- | byte = data, n = channel#)
Writes the specified data byte to the digital to analog converter (DAC) channel specified by n. Note that the eight valid DAC channel numbers are 1 <= n <= 8. The data transfer uses the SPI (serial peripheral interface); use INIT.A/D12&DAC to initialize the SPI interface. To maximize speed, this routine does not GET or RELEASE the SPI.RESOURCE. Consequently, this routine should not be used in a multitasking environment where another task might require access to the SPI. Executes in about 40 microseconds with a 16 MHz processor crystal. See >DAC and INIT.A/D12&DAC.
Pronunciation: "paren-to-dac"

(@) (addr -- w)
Fetches a 16-bit number from addr on the current page or in common memory. The high order byte is taken from addr and the low order byte from addr+1.
Pronunciation: "paren-fetch"

(A/D12.MULTIPLE) (xaddr\u1\u2\flag\n -- | u1=timing,u2=#samples,flag=mode,n= channel#)
Acquires u2 samples from the 12 bit analog to digital (A/D) converter and stores the samples as sequential 16 bit values starting at the specified xaddr. u1 is a timing parameter, flag specifies the conversion mode, and n specifies the channel number of the A/D (0 < = n < = 7). The meaning of the flag is as follows:

Flag value	Type of conversion
-1	single ended, unipolar
0	differential, unipolar
1	single ended, bipolar
2	differential, bipolar

Single-ended sampling means that the input voltage of the specified channel is referenced to VRL which is typically analog ground. Differential sampling means that the voltage input of the specified channel's "partner" is subtracted from the voltage of the specified channel and the resulting voltage is digitized by the A/D. The pairing of "partner" channels is as follows: [0,1], [2,3], [4,5], and [6,7]. Unipolar sampling means that the input is a positive voltage that swings from VRL to VRH (typically 0 to +5 V), while bipolar sampling means that the input is interpreted as a signed 12 bit representation of an input that swings from -VRH to +VRH (typically -5V to +5V). Note that conversion of inputs more negative than -4.0 V may require an external -5V supply connected to pin 39 of the Analog I/O connector. The data transfer uses the SPI (serial peripheral interface); use INIT.A/D12&DAC to initialize the SPI interface. To maximize speed, this routine does not GET or RELEASE the SPI.RESOURCE. Consequently, this routine should not be used in a multitasking environment where another task might require access to the SPI; see A/D12.MULTIPLE. If the specified xaddr is in common memory, the first sample is taken after 58 microseconds and subsequent samples are taken every (27.5+2.5*u1) microseconds, where u1 is the specified timing parameter passed to this routine. If the specified xaddr is in paged memory, the first sample is taken after 68 microseconds and subsequent samples are taken every (50+2.5*u1) microseconds. Of course, the operation of interrupts (including timesliced multitasking)

will affect these sampling times. See (A/D12.SAMPLE), A/D12.SAMPLE, A/D12.MULTIPLE, and INIT.A/D12&DAC.
Pronunciation: "paren-A-to-D-twelve-multiple"

(A/D12.SAMPLE) (flag\n -- u | flag=conversion mode, n=channel#, u=result)

Acquires and places on the stack a single sample u from the 12 bit analog to digital (A/D) converter. n specifies the channel number of the A/D (0 < = n < = 7). The meaning of the flag is as follows:

Flag value	Type of conversion
-1	single ended, unipolar
0	differential, unipolar
1	single ended, bipolar
2	differential, bipolar

Single-ended sampling means that the input voltage of the specified channel is referenced to VRL which is typically analog ground. Differential sampling means that the voltage input of the specified channel's "partner" is subtracted from the voltage of the specified channel and the resulting voltage is digitized by the A/D. The pairing of "partner" channels is as follows: [0,1], [2,3], [4,5], and [6,7]. Unipolar sampling means that the input is a positive voltage that swings from VRL to VRH (typically 0 to +5 V), while bipolar sampling means that the input is interpreted as a signed 12 bit representation of an input that swings from -VRH to +VRH (typically -5V to +5V). Note that conversion of inputs more negative than -4.0 V may require an external -5V supply connected to pin 39 of the Analog I/O connector. The data transfer uses the SPI (serial peripheral interface); use INIT.A/D12&DAC to initialize the SPI interface. To maximize speed, this routine does not GET or RELEASE the SPI.RESOURCE. Consequently, this routine should not be used in a multitasking environment where another task might require access to the SPI; see A/D12.SAMPLE. Executes in 88 microseconds. See A/D12.SAMPLE, (A/D12.MULTIPLE), A/D12.MULTIPLE, and INIT.A/D12&DAC.
Pronunciation: "paren-A-to-D-twelve-sample"

(A/D8.MULTIPLE) (xaddr\u1\u2\n -- | u1=timing param; u2 = #samples, n = channel#)

Acquires u2 samples from the 8 bit analog to digital (A/D) converter in the 68HC11 and stores the samples as sequential unsigned 8 bit values starting at the specified xaddr. n specifies the channel number of the A/D (0 < = n < = 7). To maximize speed, this routine does not GET or RELEASE the A/D8.RESOURCE. Consequently, this routine should not be used in a multitasking environment where another task might require access to the 8 bit A/D; see A/D8.MULTIPLE. If the specified xaddr is in common memory, the first sample is taken after 16 microseconds and subsequent samples are taken every (10+2.5*u1) microseconds, where u1 is the specified timing parameter passed to this routine. If the specified xaddr is in paged memory, the first sample is taken after 11 microseconds and subsequent samples are taken every (32.5+2.5*u1) microseconds. Of course, the operation of interrupts (including timesliced multitasking) will affect these sampling times. See (A/D8.SAMPLE), A/D8.SAMPLE, A/D8.MULTIPLE, and A/D8.ON.
Pronunciation: "paren-A-to-D-eight-multiple"

(A/D8.SAMPLE) (n -- byte | n = channel#)

Acquires and places on the stack a single sample byte from the 8 bit analog to digital converter in the 68HC11. n specifies the A/D channel number (0 < = n < = 7). To maximize speed, this routine does not GET or RELEASE the A/D8.RESOURCE.

Consequently, this routine should not be used in a multitasking environment where another task might require access to the 8 bit A/D; see A/D8.SAMPLE. This routine executes in 23 microseconds. See A/D8.SAMPLE, (A/D8.MULTIPLE), A/D8.MULTIPLE, and A/D8.ON.
Pronunciation: "paren-A-to-D-eight-sample"

(ABORT) ([...] --)
Return Stack: (R: [...] --)
The default abort routine called by ABORT if the CUSTOM.ABORT flag is false. Clears the data and return stacks, sets the page to the default page (0), and executes FORTH DEFINITIONS to set CONTEXT and CURRENT equal to FORTH. If an autostart vector has been installed (see AUTOSTART), (ABORT) executes the specified routine; otherwise it executes QUIT which sets the compilation mode and enters the interpreter. If R0 and S0 aren't in common RAM, a COLD restart is initiated.
Pronunciation: "paren-abort"

(BENCHMARK:) (<name> -- u1\ud\u2\u3 | u1=#msec, ud=#sec, u2=#F*, u3=#F+)
Removes the next <name> from the input stream and measures and places on the stack the execution time and operations count of <name>. Use as:

```
    (BENCHMARK:) <name>
```

The multitasker's timeslicer clock must be running for the execution time to be measured; use START.TIMESLICER to start it before calling (BENCHMARK:). Any stack arguments needed by <name> should be placed on the stack before (BENCHMARK:) is invoked. If <name> leaves any items on the stack, they will be below the stack items left by (BENCHMARK:). Net execution time of <name> is represented by ud seconds + u1 msec. The resolution of the measurement equals the timeslice period which can be set using the command *100US=TIMESLICE.PERIOD; the default is 5 msec. u2 is the number of F* and F/ operations performed by <name> and u3 is the number of F+ and F- operations performed by <name>. Operations counts up to 65,535 can be reported; after that the 16-bit operations counter rolls over to 0 and continues counting. This word is a subsidiary to BENCHMARK: which prints the results instead of leaving them on the stack.
Pronunciation: "paren-benchmark" Attributes: S

(C!) (byte\addr --)
Stores the byte at addr on the current page or in common memory.
Pronunciation: "paren-c-store"

(C@) (addr -- byte)
Fetches the byte stored at addr on the current page or in common memory.
Pronunciation: "paren-c-fetch"

(CHANGE.BITS) (byte1\byte2\addr -- | byte1 = data; byte2 = mask)
At the byte specified by addr on the current page or in common memory, modifies the bits specified by 1's in byte2 to have the values indicated by the corresponding bits in byte1. In other words, byte2 serves as a mask that specifies the bits at addr that are to be modified, and byte1 provides the data that is written to the modified bits. Disables interrupts for 16 cycles (4 microseconds) to ensure an uninterrupted read/modify/write operation. Executes more rapidly than CHANGE.BITS.
Pronunciation: "paren-change-bits"

(CLEAR.BITS) (byte1\addr --)

For each bit of byte1 that is set, clears the corresponding bit of the 8 bit value at addr on the current page or in common memory. Disables interrupts for ten cycles (2.5 microseconds) to ensure an uninterrupted read/modify/write operation. Executes more rapidly than CLEAR.BITS.
Pronunciation: "paren-clear-bits"

(CMOVE) (addr1\addr2\u -- | addr1=src, addr2=dest, u = byte count)

If u is greater than 0, u consecutive bytes starting at addr1 are copied to the destination addresses starting at addr2 on the current page or in common memory. Does not change the page. Speed is approximately 7.5 microseconds per byte. If the source and destination regions overlap and addr1 < addr2, (CMOVE) starts at high memory and moves toward low memory to avoid propagation of the moved contents. (CMOVE) always moves the contents in such a way as to avoid memory propagation.
Pronunciation: "paren-c-move"

(COMPILE.CALL) (cfa --)

Compiles a call to the specified cfa. Compiles a JSR (jump to subroutine) opcode followed by cfa into the definitions area at HERE and increments DP by 3. No page change is compiled.
Pronunciation: "paren-compile-call"

(CREATE) ($addr --)

See CREATE. Similar to CREATE which creates a header for <name>. The difference is that CREATE removes <name> from the input stream by executing BL WORD, while (CREATE) accepts <name> as a counted string at $addr in common memory. Converts the counted string at $addr to upper case letters and searches the dictionary via (FIND) to check for uniqueness. Issues a warning if <name> is not unique. Creates a new header for <name> starting at the address pointed to by NP, links the header to the CURRENT vocabulary, and initializes the code field address in the header to the current value of DP. Updates the CURRENT vocabulary xhandle to point to the xnfa of <name> and updates NP to point to the byte after <name>'s header. The number of characters saved in the header is the lesser of the value in WIDTH or the actual number of characters in <name>. If locals are compiling, all characters are saved in the header to avoid non-uniqueness of local variables. An abort error occurs if the header cannot be stored (e.g. if NP does not point to RAM). If WIDTH is less than or equal to 1, (CREATE) resets WIDTH to 2.
Pronunciation: "paren-create" Attributes: D

(EE!) (w\addr -- | addr is an EEPROM address)

Stores w at the specified addr in EEPROM. Any byte that already contains the specified contents is not re-programmed; this helps lengthen the lifetime of the EEPROM. Requires 20 msec per programmed byte. Disables interrupts during the programming of each byte. Caution: the prolonged disabling of interrupts by (EE!) can adversely affect real-time servicing of interrupts including those associated with the secondary serial line.
Pronunciation: "paren-e-e-store"

(EE2!) (w1\w2\addr -- | [addr] gets w2, [addr+2] gets w1)

Stores w1 and w2 at the specified addr in EEPROM. w2 is stored at addr and w1 is stored at addr+2. Can also be used to store a double number at addr. Any byte that already contains the specified contents is not re-programmed; this helps lengthen the lifetime of the EEPROM. Requires 20 msec per programmed byte. Disables interrupts during the programming of each byte. Caution: the prolonged disabling of interrupts by (EE2!) can adversely affect real-time servicing of interrupts including those associated with the secondary serial line.
Pronunciation: "paren-e-e-two-store"

(EEC!) (byte\addr -- | addr is an EEPROM address)
Stores byte at the specified addr in EEPROM. If addr already contains the specified contents it is not re-programmed; this helps lengthen the lifetime of the EEPROM. Requires 20 msec per programmed byte. Disables interrupts during the programming of each byte. Caution: the prolonged disabling of interrupts by (EEC!) can adversely affect real-time servicing of interrupts including those associated with the secondary serial line.
Pronunciation: "paren-e-e-c-store"

(EEF!) (r\addr -- | addr is an EEPROM address)
Stores r at the specified addr in EEPROM. Any byte that already contains the specified contents is not re-programmed; this helps lengthen the lifetime of the EEPROM. Requires 20 msec per programmed byte. Disables interrupts during the programming of each byte. Caution: the prolonged disabling of interrupts by (EEF!) can adversely affect real-time servicing of interrupts including those associated with the secondary serial line.
Pronunciation: "paren-e-e-f-store"

(EEX!) (xaddr\addr -- | addr is an EEPROM address)
Stores xaddr at the specified addr in EEPROM. Any byte that already contains the specified contents is not re-programmed; this helps lengthen the lifetime of the EEPROM. Requires 20 msec per programmed byte. Disables interrupts during the programming of each byte. Caution: the prolonged disabling of interrupts by (EEX!) can adversely affect real-time servicing of interrupts including those associated with the secondary serial line.
Pronunciation: "paren-e-e-x-store"

(ERROR) ([...] --)
Return Stack: (R: [...] --)
The default routine called if a system error is detected and the CUSTOM.ERROR flag is false. Prints a descriptive error message, if possible printing the name of the routine that detected the error and any arrays or matrices involved in producing the error condition. After printing the message, executes ABORT. See ((ERROR)), CUSTOM.ERROR, UERROR, and ABORT.
Pronunciation: "paren-error"

(EXECUTE) (cfa --)
Executes (calls) the routine whose executable machine instructions begin at the specified code field address cfa on the current page or in common memory.
Pronunciation: "paren-execute"

(F!) (r\addr --)
Stores a floating point number at addr on the current page or in common memory.
Pronunciation: "paren-f-store"

(F@) (addr -- r)
Fetches a floating point number from addr on the current page or in common memory.
Pronunciation: "paren-f-fetch"

(FIND) ($addr -- [xcfa\flag] or [0])
Searches the dictionary for a match of the counted string at $addr in the common memory. (FIND) first searches the CONTEXT vocabulary. Then, if the word is not found and if the CONTEXT and CURRENT vocabularies are different, it searches the CURRENT vocabulary. If the word is not found in the dictionary, (FIND) leaves a 0 on the stack. If the word is found, it leaves the word's extended code field address under a flag on the stack. The flag is +1 if the word is immediate and -1 if the word is not immediate. A COLD restart will occur if more than 255 page changes are made during the search through either vocabulary. This prevents the interpreter from going on an infinite search through a corrupted dictionary. A COLD restart will also occur if POCKET is not in common memory.
Pronunciation: "paren-find" Attributes: D

(HERE) (-- addr)
Places on the stack the addr of the next available location in the definitions area. Equivalent to DP 2XN+ @
Pronunciation: "paren-here" Attributes: U

(MOVE) (addr1\addr2\u -- | addr1=src, addr2=dest, u = count)
If u is greater than 0, u consecutive 16-bit numbers (i.e., 2*u consecutive bytes) starting at addr1 are copied to the destination addresses starting at addr2 on the current page or in common memory. Does not change the page. Speed is approximately 15 microseconds per 2-byte cell. If the source and destination regions overlap and addr1 < addr2, (MOVE) starts at high memory and moves toward low memory to avoid propagation of the moved contents. (MOVE) always moves the contents in such a way as to avoid memory propagation.
Pronunciation: "paren-move"

(PAGE.LATCH) (-- addr)
Returns the address of the page latch whose contents indicate the current page. See THIS.PAGE. Be careful when explicitly changing the contents of the page latch. Note that the word C! cannot be used to alter the contents of the page latch because C! saves and restores the page. Rather, the page-less store operator (C!) must be used.

(RP) (-- addr)
Places on the stack the address of the most significant byte of the top item on the return stack.
Pronunciation: "paren-r-p" Attributes: U

(SET.BITS) (byte1\addr --)
For each bit of byte1 that is set, sets the corresponding bit of the 8 bit value at addr on the current page or in common memory. Disables interrupts for ten cycles (2.5

microseconds) to ensure an uninterrupted read/modify/write operation. Executes more rapidly than SET.BITS.
Pronunciation: "paren-set-bits"

(SP) (-- addr)
Places on the stack the address of the most significant byte of the top cell of the data stack just before (SP) is executed.
Pronunciation: "paren-s-p" Attributes: U

(STATUS) (-- addr | addr is also the base addr of the user area)
Returns the address but not the page of the STATUS user variable which is also the task identification address at the base of the task's user area. Because STATUS must be in common memory, a 16-bit address is sufficient to specify its location. Using (STATUS) instead of STATUS leads to faster code because page-less memory operators execute more rapidly than operators that take full extended addresses. See the glossary entry for STATUS.
Pronunciation: "paren-status" Attributes: U

(TOGGLE.BITS) (byte1\addr --)
For each bit of byte1 that is set, reverses the state of the corresponding bit of the 8 bit value at addr on the current page or in common memory. Disables interrupts for ten cycles (2.5 microseconds) to ensure an uninterrupted read/modify/write operation. Executes more rapidly than TOGGLE.BITS.
Pronunciation: "paren-toggle-bits"

(UPDATE.DISPLAY) (--)
Writes the contents of the DISPLAY.BUFFER to the LCD display, but does not re-home the cursor/display RAM pointer to the default upper left position. This facilitates "scrolling" the contents of a graphics display when used in conjunction with the IS.DISPLAY.ADDRESS routine. For character displays, the cursor is turned off during the write to the display and is restored to its prior state after the update is complete, thus avoiding "flickering" of the cursor. Intermittently disables interrupts for 28 cycles (7 microseconds) per byte to implement clock stretching. See also UPDATE.DISPLAY and UPDATE.DISPLAY.LINE.

(X!) (xaddr\addr --)
Stores an extended address xaddr at addr on the current page or in common memory.
Pronunciation: "paren-x-store"

(X@) (addr -- xaddr)
Fetches an extended address from addr on the current page or from common memory.
Pronunciation: "paren-x-fetch"

* (n1\n2 -- n3)
Multiplies n1 by n2 giving n3 which is the least significant cell of the product.
Pronunciation: "star"

*/ (n1\n2\n3 -- n4 | do n1*n2/n3 ; n4 = quotient)

Multiplies n1 and n2 producing an intermediate double number result which is divided by n3 to yield the integer quotient n4. Uses signed math. An unchecked error occurs on overflow. Division by zero (n2=0) yields n4 = -1.
Pronunciation: "star-slash"

*/MOD (n1\n2\n3 -- n4\n5 | do n1*n2/n3; n4 = remainder; n5 = quotient)
Multiplies n1 and n2 producing an intermediate double number result which is divided by n3 to yield remainder n4 and quotient n5. Uses signed math. An unchecked error occurs on overflow. Division by zero (n2=0) yields n4 = -1 and n5 = -1. See U*/MOD.

Pronunciation: "star-slash-mod"

*100US=TIMESLICE.PERIOD (u --)
Sets u as the period of the timeslice clock (the OC2 interrupt) in units of 100 microseconds. For example, to set the timeslice period to 0.8 msec, execute

```
8  *100US=TIMESLICE.PERIOD
```

Note that the default timeslice increment set after a COLD restart is 5 msec. Implementation detail: Based on the prescaler bits PR1 and PR0 in the TMSK2 register, this routine calculates the period of the clock driving the OC2 timer. It then calculates the number of these periods in the requested timeslice period u, and stores the resulting OC2 timer increment in a headerless system variable called TIMESLICE.INCREMENT. This stored increment sets the period of the OC2 timer. The period of the OC2 timer determines the timeslice period and also the resolution of the elapsed time clock (see READ.ELAPSED.TIME). Aborts if the calculated increment is 0 or is greater than 65,535.
Pronunciation: "times-100-microseconds-equals-timeslice-period"

+ (n1\n2 -- n3)
Adds n1 to n2 and puts the sum n3 on the data stack.
Pronunciation: "plus"

+! (w\xaddr --)
Adds w to the 16-bit value stored at xaddr and stores the result at xaddr.
Pronunciation: "plus-store"

+C! (byte\xaddr --)
Adds byte to the 8-bit value stored at xaddr and stores the result at xaddr.
Pronunciation: "plus-c-store"

+CURRENT.HEAP (xpfa-- xpfa+u)
Adds the offset u to the extended parameter field address xpfa. +CURRENT.HEAP is defined as a member of the structure HEAP.STRUCTURE.PF. Use as:

```
' <name.of.heap.item> +CURRENT.HEAP
```

to find the address in the heap item's parameter field where the 16-bit current.heap address is stored. CURRENT.HEAP specifies the heap in which the item is allocated. See HEAP.STRUCTURE.PF, CURRENT.HEAP, +HEAP.PAGE.
Pronunciation: "plus-current-heap"

+HEAP.HANDLE (xpfa-- xpfa+u)

Adds the offset u to the extended parameter field address xpfa. +HEAP.HANDLE is defined as a member of the structure HEAP.STRUCTURE.PF. Use as:

    ' <name.of.heap.item> +HEAP.HANDLE

to find the address in the parameter field that contains of the handle which contains the base xaddr of the heap item. See HEAP.STRUCTURE.PF and +HEAP.PAGE.
Pronunciation: "plus-heap-handle"

+HEAP.PAGE (xpfa-- xpfa+u)

Adds the offset u to the extended parameter field address xpfa. +HEAP.PAGE is defined as a member of the structure HEAP.STRUCTURE.PF. Use as:

    ' <name.of.heap.item> +HEAP.PAGE

to find the address of the page of the heap as saved in the item's parameter field. This is the page of the handle as well as the page of CURRENT.HEAP in which the item resides. See HEAP.STRUCTURE.PF.
Pronunciation: "plus-heap-page"

+LOOP (n --)

Return Stack: (R: w1\w2 -- [w1\w2] or [] | drops w1,w2 when loop terminates)
Adds the signed integer n to the loop index. If the loop index was incremented across the boundary between limit and limit+1, terminates the loop by allowing execution to continue with the word following +LOOP. Otherwise, continues looping by transferring control to the word following DO. Use as:

    w1 w2 DO   words to be executed   n +LOOP

where w1 is the loop limit and w2 is the starting index. An error is issued if +LOOP is not properly paired with DO inside a colon definition. See DO I J K I' LEAVE.
Pronunciation: "plus-loop" Attributes: C, I

, (w --)

Stores w at the next available location in the definitions area and increments the definitions pointer DP by 2. An error occurs if w is not correctly stored (for example, if DP does not point to RAM). An error occurs if the operation causes DP to be incremented across the boundary between 0x7FFF (the last valid address in a given page) and 0x8000 (the start of the register area).
Pronunciation: "comma"

," (--)

Compile Time: (<text> --)
Parses the <text> string delimited by a " character from the input stream and emplaces the string in the dictionary starting at HERE. An error occurs if the compiled string crosses a page boundary.
Pronunciation: "comma-quote"

- (n1\n2 -- n3 | n3 = n1 - n2)

Subtracts n2 from n1 and puts the result n3 on the data stack.
Pronunciation: "minus"

-1 (-- -1)

Puts the value negative one on the data stack.
Pronunciation: "minus-one"

-1/INFINITY (-- r)
Pushes the smallest representable negative floating point number onto the data stack.
Pronunciation: "minus-one-over-infinity"

-2 (-- -2)
Puts the value negative two on the data stack.
Pronunciation: "minus-two"

-INFINITY (-- r)
Pushes the negative of the largest representable floating point number onto the data stack.
Pronunciation: "minus-infinity"

-ROLL (wn\...\w0\+n -- w0\wn\...\w1 | 0 <= +n <= 255)
Transfers the top item (not including +n) on the data stack to the nth position from the top of the data stack, where the top stack item is item#0, the next is item#1, etc. For example, 0 ROLL does nothing, 1 -ROLL is equivalent to SWAP, and 2 -ROLL is equivalent to -ROT.
Pronunciation: "minus-roll"

-ROT (w1\w2\w3 -- w3\w1\w2)
Rotates the top three stack entries by moving the top cell below the next two cells on the data stack.
Pronunciation: "minus-rot"

-TRAILING (xaddr\u1 -- xaddr\u2)
Strips trailing space characters from the string located at xaddr by returning the new character count, u2, of the text string with spaces removed. Equivalent to BL SKIP>
Pronunciation: "dash-trailing"

. (n --)
Prints n with no leading spaces and 1 trailing space. If the number base is decimal, w is printed as a signed number in the range -32,768 to +32,767. In other bases w is printed as an unsigned positive number. Use U. to print w as a positive unsigned number in decimal base.
Pronunciation: "dot" Attributes: M, S

." (--)
Compile Time: (<text> --)
Parses the <text> string in the input stream delimited by a terminating " character. If executing, types the <text>. If compiling, emplaces the text in the dictionary as a counted string along with a call to a routine that types the <text> at runtime. An error occurs if the compiled string crosses a page boundary.
Pronunciation: "dot-quote" Attributes: I, M

.HANDLES (--)
Prints the allocated handles of the current heap in tabular format, listing the size, base xaddress, and handle xaddress of each heap item. All quantities are displayed in hexadecimal base. An * displayed in the size field indicates that the handle is not in use (i.e., it has been returned to the heap) or its contents are invalid.

Pronunciation: "dot-handles" Attributes: M

.R (w\+byte -- | +byte is field width)
Prints w right-justified in a field of +byte characters. If +byte is less than or equal to the number of characters to be printed, the number is printed with no extra spaces. If the number base is decimal, w is printed as a signed number in the range -32,768 to +32,767. In other bases w is printed as an unsigned positive number. To print w as a positive unsigned number in decimal base, place a 0 on the stack above w to convert it into a positive double number and call UD.R
Pronunciation: "dot-r" Attributes: M, S

.S (--)
Displays the contents of the data stack without modifying the contents of the stack. Prints the number of 1-cell stack items in brackets and displays the stack items separated by \ (read as "under"). A maximum of 5 items are displayed. For example, if there are 7 stack items having values 1...7 with 1 on top of the stack and 7 farthest down, executing .S yields

```
( 7 ) \ 5 \ 4 \ 3 \ 2 \ 1   ok
```

In execution mode, the stack contents are automatically displayed after each line is interpreted if the DEBUG flag is true. The stack is also displayed during a TRACE of a compiled routine.
Pronunciation: "dot-s" Attributes: M, S

/ (n1\n2 -- n3 | n3 = n1/n2)
Divide n1 by n2, giving the quotient n3. If the division does not produce an integer quotient, the quotient is rounded towards 0. Division by 0 (n2=0) produces a quotient of -1. See U/.
Pronunciation: "slash"

/MOD (n1\n2 -- n3\n4 | n3 = remainder, n4 = quotient)
Divide n1 by n2, giving the remainder n3 and the quotient n4. The quotient is rounded towards 0, and the remainder carries the sign of n1. Division by 0 (n2=0) yields n3 = n4 = -1. See U/MOD.
Pronunciation: "slash-mod"

/STRING (xaddr1\u1\n -- xaddr2\u2)
Shortens the text string whose first character is at xaddr1 by computing xaddr2 = xaddr1 + n and u2 = u1 - n. u1 and u2 are 16 bit text string counts. n may be negative, and the string may cross a page boundary.
Pronunciation: "slash-string"

0 (-- 0)
Puts the value zero on the data stack.
Pronunciation: "zero"

0< (n -- flag)
Flag is TRUE if n is less than zero and FALSE otherwise.
Pronunciation: "zero-less-than"

0< > (w -- flag)

Flag is TRUE if w is not equal to zero and FALSE otherwise.
Pronunciation: "zero-not-equal"

0= (w -- flag)
Flag is TRUE if w is equal to zero and FALSE otherwise.
Pronunciation: "zero-equals"

0> (n -- flag)
Flag is true if n is greater than zero and FALSE otherwise.
Pronunciation: "zero-greater-than"

0\0 (-- 0\0)
Places two zeros on the top of the stack.
Pronunciation: "0-under-0"

1 (-- 1)
Puts the value one on the data stack.
Pronunciation: "one"

1+ (w1 -- w2 | w2 = w1 + 1)
Adds 1 to w1 giving the sum w2.
Pronunciation: "one-plus"

1- (w1 -- w2 | w2 = w1 - 1)
Subtracts 1 from w1 giving w2.
Pronunciation: "one-minus"

1/F (r1 -- r2)
r2 is the multiplicative inverse of r1; r2 = 1.0/r1.
Pronunciation: "one-over-f" Attributes: S

1/INFINITY (-- r)
Places the smallest representable positive floating point number on the data stack.
Pronunciation: "one-over-infinity"

1/LN(2) (-- r)
Places the floating point representation of the inverse of the natural logarithm of 2 (1.4427) on the stack.
Pronunciation: "one-over-l-n-of-two"

1/LOG10(2) (-- r)
Places the floating point representation of the inverse of the base 10 logarithm of 2 (3.3219) on the stack.
Pronunciation: "one-over-log-ten-of-two"

1/PI (-- r)
Places the floating point representation of 1/pi (0.3183) on the stack.
Pronunciation: "one-over-pi"

1/SQRT(2) (-- r)

	Places the floating point representation of the inverse of the square root of 2 (0.7071) on the stack. Pronunciation: "one-over-square-root-of-two"
1/TEN	(-- r) Places r = 0.1 on the data stack. Pronunciation: "one-over-ten"
10*	(n1 -- n2 | n2 = n1 * 10) Multiplies n1 by 10 (decimal) giving n2. Pronunciation: "ten-star"
10^N	(n -- r) r equals 10 to the nth power; r = 10^n. Pronunciation: "ten-to-the-n"Attributes: S
1XN+	(xaddr1 -- xaddr2) Adds 1 to xaddr1 yielding xaddr2. Equivalent to 1 XN+ . Pronunciation: "one-x-n-plus"
1XN-	(xaddr1 -- xaddr2) Subtracts 1 from xaddr1 yielding xaddr2. Equivalent to 1 XN- . Pronunciation: "one-x-n-minus"
2	(-- 2) Puts the value two on the data stack. Pronunciation: "two"
2!	(w1\w2\xaddr -- | [xaddr] gets w2, [xaddr+2] gets w1) Stores two 16-bit integers at xaddr. w2 is stored at xaddr and w1 is stored at xaddr+2. Can also be used to store a double number at xaddr. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page. Pronunciation: "two-store"
2*	(n1 -- n2 | n2 = n1 * 2) Multiplies n1 by 2 giving n2. Pronunciation: "two-star"
2+	(w1 -- w2 | w2 = w1 + 2) Adds 2 to w1 giving the sum w2. Pronunciation: "two-plus"
2-	(w1 -- w2 | w2 = w1 - 2) Subtracts 2 from w1 giving w2. Pronunciation: "two-minus"
2/	(n1 -- n2 | n2 = n1 / 2) Divides n1 by 2 giving n2. See U2/. Pronunciation: "two-slash"

2@ (xaddr -- w1\w2)

Fetches two 16-bit integers from xaddr. w2 is taken from xaddr and w1 is from xaddr+2. Can also be used to fetch a double number from xaddr. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "two-fetch"

2ARRAY.FETCH (row#\col#\xpfa -- d)

Fetches and places on the data stack the contents of the element at row#, column# in the specified 2-dimensional array or matrix. The size of the element that is fetched depends upon the number of bytes per element of the array or matrix as specified by DIMENSIONED or DIMMED, and the result is always padded out to 4 bytes on the stack. There is an unchecked error if the specified array or matrix does not have 2 dimensions or if the number of bytes per element does not equal 1, 2, or 4. See also M[]@.
Pronunciation: "two-array-fetch"

2ARRAY.STORE (d\ row#\col#\xpfa --)

Stores the specified byte-, 2 byte-, or 4 byte-data at the element specified by row#, column# in the specified 2-dimensional array or matrix. The size of the element that is stored can be 1 byte, 2 bytes, or 4 bytes depending upon the number of bytes per element of the array or matrix as set by DIMENSIONED or DIMMED. There is an unchecked error if the specified array or matrix does not have 2 dimensions or if the number of bytes per element does not equal 1, 2, or 4. See also M[]!.
Pronunciation: "two-array-store"

2CONSTANT (wd <name> --)

Removes the next <name> from the input stream and defines a child word called <name> which when executed leaves the 32-bit value wd on the data stack. wd is stored in the definitions area of the dictionary. <name> is referred to as a "2constant".
Use as:

```
    wd  2CONSTANT  <name>
```

Pronunciation: "two-constant" Attributes: D

2DROP (w1\w2 --)

Drops the top two cells from the data stack.
Pronunciation: "two-drop"

2DUP (w1\w2 -- w1\w2\w1\w2)

Duplicates the top two cells on the data stack.
Pronunciation: "two-dupe"

2DUP>R (w1\w2 -- w1\w2)

Return Stack: (R: -- w1\w2)
Copies the top cell pair on the data stack to the return stack.
Pronunciation: "2-dup-to-r" Attributes: C

2LITERAL (-- wd)

Compile Time: (wd --)

If QED-Forth is in execution mode when 2LITERAL is invoked, 2LITERAL does nothing. If QED-Forth is in compilation mode, 2LITERAL removes wd from the stack and compiles it into the dictionary along with code that, when later executed, pushes wd to the stack. 2LITERAL can be used within a colon definition to compile a numeric value into the definition. For example,

```
: <name>
        ... [ 1234 1000 * ] 2LITERAL ...
;
```

This compiles the value calculated between [and] as a double literal into the definition of <name>. When <name> is executed, this value will be placed on the stack.
Pronunciation: "two-literal" Attributes: C, I

2OVER (w1\w2\w3\w4 -- w1\w2\w3\w4\w1\w2)
Places a copy of cell pair w1\w2 on the top of the stack.
Pronunciation: "two-over"

2PI/360 (-- r)
Places the floating point representation of 2pi/360 (0.017453) on the stack.
Pronunciation: "two-pi-over-three-sixty"

2ROT (w1\w2\w3\w4\w5\w6 -- w3\w4\w5\w6\w1\w2)
Rotates the top three cell pairs on the data stack.
Pronunciation: "two-rote"

2SWAP (w1\w2\w3\w4 -- w3\w4\w1\w2)
Exchanges the top two cell pairs on the data stack.
Pronunciation: "two-swap"

2V.TRANSFORM (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el\xcfa --)
xcfa specifies a floating point transformation that operates on two input numbers to produce a single floating point result. 2V.TRANSFORM applies this transformation to each pair of corresponding elements in two source vectors specified by xvaddr1\sep1\d.#el and xvaddr2\sep2\d.#el and places the result in the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be either of the sources.
Pronunciation: "two-v-transform" Attributes: S

2VARIABLE (<name> --)
Removes the next <name> from the input stream, defines a child word called <name>, and VALLOTs 2 cells in the variable area. When <name> is executed it leaves the extended address xaddr of the two cells reserved in the variable area that hold <name>'s contents. <name> is referred to as a "2variable". Use as:

```
2VARIABLE <name>
```

Pronunciation: "2-variable" Attributes: D

2XN+ (xaddr1 -- xaddr2)
Adds 2 to xaddr1 yielding xaddr2. Equivalent to 2 XN+ .
Pronunciation: "two-x-n-plus"

2XN- (xaddr1 -- xaddr2)

	Subtracts 2 from xaddr1 yielding xaddr2. Equivalent to 2 XN- . Pronunciation: "two-x-n-minus"
3	(-- 3) Puts the value three on the data stack. Pronunciation: "three"
3*	(n1 -- n2 | n2 = n1 * 3) Multiplies n1 by 3 giving n2. Pronunciation: "three-star"
360/2PI	(-- r) Places the floating point representation of 360/2pi (57.296) on the stack. Pronunciation: "three-sixty-over-two-pi"
3DROP	(w1\w2\w3 --) Drops the top three cells from the data stack. Pronunciation: "three-drop"
3DUP	(w1\w2\w3 -- w1\w2\w3\w1\w2\w3) Duplicates the top three cells on the data stack. Pronunciation: "three-dupe"
4	(-- 4) Puts the value four on the data stack. Pronunciation: "four"
4*	(n1 -- n2 | n2 = n1 * 4) Multiplies n1 by 4 giving n2. Pronunciation: "four-star"
4+	(w1 -- w2 | w2 = w1 + 4) Adds 4 to w1 giving the sum w2. Pronunciation: "four-plus"
4-	(w1 -- w2 | w2 = w1 - 4) Subtracts 4 from w1 giving w2. Pronunciation: "four-minus"
4/	(n1 -- n2 | n2 = n1 / 4) Divides n1 by 4 giving n2. Pronunciation: "four-slash"
4DROP	(w1\w2\w3\w4 --) Drops the top 4 cells from the data stack. Pronunciation: "four-drop"
4DUP	(w1\w2\w3\w4 -- w1\w2\w3\w4\w1\w2\w3\w4) Duplicates the top four cells on the data stack. Pronunciation: "four-dupe"

4XN+	(xaddr1 -- xaddr2) Adds 4 to xaddr1 yielding xaddr2. Equivalent to 4 XN+ . Pronunciation: "four-x-n-plus"
4XN-	(xaddr1 -- xaddr2) Subtracts 4 from xaddr1 yielding xaddr2. Equivalent to 4 XN- . Pronunciation: "four-x-n-minus"
8*	(n1 -- n2 | n2 = n1 * 8) Multiplies n1 by 8 giving n2. Pronunciation: "eight-star"
8/	(n1 -- n2 | n2 = n1 / 8) Divides n1 by 8 giving n2. Pronunciation: "eight-slash"
8XN+	(xaddr1 -- xaddr2) Adds 8 to xaddr1 yielding xaddr2. Equivalent to 8 XN+ . Pronunciation: "eight-x-n-plus
:	(< name> --) Starts the compilation of a new definition. Removes <name> from the input stream and creates a header for <name> in the dictionary. The header is SMUDGEd so that it cannot be found until ; executes to successfully terminate the definition. Enters the compile mode so that words following : are compiled into the code field of <name> (but IMMEDIATE words are executed immediately instead of being compiled). A "<name> isn't unique" warning is issued if <name> already exists in the dictionary. The contents of CONTEXT and CURRENT are not modified. Use as : <name> ...body of new definition... ; Pronunciation: "colon" Attributes: D
;	(--) Marks the end of a colon definition and enters the execution mode. Checks the stack to make sure that no extra items were placed on or removed from the data stack during compilation of the definition. SMUDGEs the header created by : so that the new word can be found in the dictionary. Compiles code to cause control to be passed to the calling word when the definition is later executed. If locals were used in the definition, the code compiled by ; also removes the local variables from the return stack. Pronunciation: "semicolon" Attributes: C, I
<	(n1\n2 -- flag) Flag is TRUE if n1 is less than n2 and FALSE otherwise. Pronunciation: "less-than"
< #	(--)

Prepares for pictured numeric output by initializing the headerless user variable #PTR to be equal to PAD. #PTR points to the current character position in the pictured numeric output, which starts 1 byte below PAD and builds towards low memory.
Pronunciation: "less-number-sign" Attributes: S

< = (n1\n2 -- flag)
Flag is TRUE if n1 is less than or equal to n2 and FALSE otherwise.
Pronunciation: "less-than-or-equal"

< > (w1\w2 -- flag)
Flag is TRUE if w1 is not equal to w2 and FALSE otherwise.
Pronunciation: "not-equal"

<DBUILDS (<name> --)
Used in a high level defining word to mark the beginning of the specification of the action taken when a child word is defined. Removes <name> from the input stream and creates a header for <name> in the dictionary. Sets the HAS.PFA bit in the header to indicate that <name> has a parameter field. Use as:

```
: <namex> <DBUILDS    code to set up child's parameter field
                DOES>    run time action
;
```

where <namex> is referred to as a "defining word". Executing the statement

```
<namex> <child's.name>
```

defines the child word. The code after <DBUILDS specifies the action to be taken while defining the child word. This usually involves allotting and/or initializing the parameter field of the child. The "D" in <DBUILDS stands for "definitions area"; the parameter field is located at the next available location in the definitions area pointed to by DP. Thus the words ALLOT and , (as opposed to VALLOT and V,) should be used after <DBUILDS to reserve and initialize locations in the child's parameter field. Use <DBUILDS to define child words whose parameter fields are to be in non-modifiable write-protected memory once the application program is finished. Restrictions: LOCALS{ } cannot be used between <DBUILDS and DOES>. If you need to use local variables to perform the building action, define and call a subsidiary word that performs the action.
Example of use: a version of CONSTANT could be defined using <DBUILDS:

```
: MYCONSTANT        ( n <name> -- )
        <DBUILDS      ,
        DOES>         @
;
```

MYCONSTANT is a defining word. To define a child word named THIS.CON initialized to the value 1234 execute

```
1234 MYCONSTANT  THIS.CON
```

When MYCONSTANT executes, it initializes the first 2 bytes in the child's parameter field to 1234 and increments DP by 2. Executing THIS.CON places on the stack the value stored at the extended pfa. See DOES>.
Pronunciation: "d-builds" Attributes: D

<VBUILDS (--)
Used in a high level defining word to mark the beginning of the specification of the action taken when a child word is defined. Removes <name> from the input stream and

creates a header for <name> in the dictionary. Sets the HAS.PFA bit in the header to indicate that <name> has a parameter field. Use as:

```
    : <namex>
            <VBUILDS      code to set up child's parameter field
            DOES>         run time action
    ;
```

where <namex> is referred to as a "defining word". Executing the statement

```
    <namex>  <child's.name>
```

defines the child word. The code after <VBUILDS specifies the action to be taken while defining the child word. This usually involves allotting and/or initializing the parameter field of the child. The "V" in <VBUILDS stands for "variable area"; the parameter field is located at the next available location in the variable area. Thus the words VALLOT and V, (as opposed to ALLOT and ,) should be used after <VBUILDS to reserve and initialize locations in the child's parameter field. Use <VBUILDS to define child words whose parameter fields must always be in modifiable non-write-protected memory. See the definition of <DBUILDS. Restrictions: LOCALS{ } cannot be used between <VBUILDS and DOES>. If you need to use local variables to perform the building action, define and call a subsidiary word that performs the action.
Example of use: a version of VARIABLE that initializes the variable's contents could be defined using <VBUILDS as,

```
    : INITIALIZED.VAR      ( n <name> -- )
            <VBUILDS      V,
            DOES>
    ;
```

INITIALIZED.VAR is a defining word. To define a child word named MYVAR initialized to the value 1234 execute

```
    1234 INITIALIZED.VAR  MYVAR
```

When INITIALIZED.VAR executes, it initializes the first 2 bytes in the child's parameter field to 1234 and increments VP by 2. Executing MYVAR leaves the extended pfa on the stack (see DOES>) and a fetch from this address will return the value 1234.
Pronunciation: "v-builds" Attributes: D

= (w1\w2 -- flag)
Flag is TRUE if w1 is equal to w2 and FALSE otherwise.
Pronunciation: "equals"

> (n1\n2 -- flag)
Flag is TRUE if n1 is greater than n2 and FALSE otherwise.
Pronunciation: "greater-than"

> < (w1 -- w2 | w2 has upper and lower bytes of w1 swapped)
Swaps the two bytes of the top data stack cell.
Pronunciation: "swap-bytes"

>= (n1\n2 -- flag)
Flag is TRUE if n1 is greater than or equal to n2 and FALSE otherwise.
Pronunciation: "greater-than-or-equal"

>ASSM (--)

Executes ASSEMBLER so that assembler mnemonics can be found in the dictionary, and enters execution mode. Normally used to compile in-line assembly code into a high level FORTH definition, or to return to assembly after using >FORTH in a CODE definition. See >FORTH .
Pronunciation: "to-assembly" Attributes: I

>DAC (byte\n -- | byte = data, n = channel#)
Writes the specified data byte to the digital to analog converter (DAC) channel specified by n. The eight valid DAC channel numbers are 1 <= n <= 8. The data transfer uses the SPI (serial peripheral interface); use INIT.A/D12&DAC to initialize the SPI interface. To ensure proper operation in a multitasking environment, executes SPI.RESOURCE GET before writing to the DAC and SPI.RESOURCE RELEASE before terminating. Executes in approximately 125 microseconds. See (>DAC) for a faster version suitable for non-multitasking applications. See also INIT.A/D12&DAC.
Pronunciation: "to-dac" Attributes: M

>DEGREES (r1 -- r2)
Converts r1 in radians into r2 in degrees.
Pronunciation: "to-degrees" Attributes: S

>FORTH (--)
Sets vocabulary equal to FORTH and enters compilation mode. Normally used to compile high level FORTH words into an assembly CODE definition, or to return to high level after using >ASSM to assemble in-line code. See >ASSM .
Pronunciation: "to-forth"

>IN (-- xaddr)
User variable that contains the offset from the start of the current input stream to the next character to be parsed. The contents of >IN may range from 0 to the number of characters in the input stream. See QUERY, WORD.
Pronunciation: "to-in" Attributes: U

>R (w --)
Return Stack: (R: -- w)
Transfers the top cell on the data stack to the return stack.
Pronunciation: "to-r" Attributes: C

>RADIANS (r1 -- r2)
Converts r1 in degrees into r2 in radians.
Pronunciation: "to-radians" Attributes: S

? (xaddr --)
Prints the integer contents of xaddr.
Pronunciation: "question" Attributes: S

?ARRAY.SIZE (array.xpfa -- d | d = number of elements in array)
Returns the number of elements d (not the number of bytes!) in the array or matrix designated by array.xpfa. 0\0 is returned for undimensioned arrays and matrices.
Pronunciation: "question-array-size"

?DETERMINANT (matrix.xpfa\n -- r | n = sign, r = determinant)

Calculates the determinant r of an LU decomposed matrix specified by matrix.xpfa given the sign of the determinant n. May be used after LU.DECOMPOSITION executes. LU.DECOMPOSITION puts the source matrix in the proper form and calculates n. ?DETERMINANT then returns the value of the determinant of the matrix.
Attributes: S

?DIM.MATRIX (matrix.xpfa -- #rows\#cols)

Returns the number of rows and columns in the specified matrix. If DEBUG is ON, ABORTs if matrix.xpfa is undimensioned or is not a matrix.
Pronunciation: "question-dim-matrix"

?DIMENSIONS (array.xpfa -- [u1\u2\...uN\N\n] or [0\0] | N=#dim, n=bytes/element)

Returns the number of elements u1, u2, ...uN in each dimension under the number of dimensions N under the number of bytes per element n for the array or matrix designated by array.xpfa. Returns 0\0 if the array is undimensioned.
Pronunciation: "question-dimensions"

?DUP (w -- [w\w] or [0])

Duplicates the top cell of the data stack if it is non-zero.
Pronunciation: "question-dupe"

?GET (xresource -- flag | flag is true if resource is available)

Checks the resource variable xresource. If the resource is available (i.e., if it contains 0\0 or the current task's xtask.id), ?GET claims the resource by storing the current task's xtask.id in xresource, and returns a true flag. Otherwise, ?GET returns a false flag. Does not execute PAUSE. To ensure that the state of the resource is correctly determined, ?GET disables interrupts for 27 to 57 cycles (6.75 to 14.25 microseconds). See GET, RELEASE, and RESOURCE.VARIABLE:.
Pronunciation: "question-get"

?HANDLE.SIZE (xhandle -- +d)

+d is the number of heap bytes allocated to the heap item associated with xhandle. An unchecked error occurs if xhandle is not a valid heap handle.
Pronunciation: "question-handle-size"

?HAS.PFA (xnfa -- flag)

Returns a TRUE flag if the word referenced by xnfa has a parameter field address.
Pronunciation: "question-has-p-f-a"

?IMMEDIATE (xnfa -- flag)

Returns a TRUE flag if the word referenced by xnfa is an immediate word.
Pronunciation: "question-immediate"

?KEY (-- flag)

Returns a flag indicating receipt of a character. If flag is TRUE, a character has been received; otherwise, no character has been received. Depending on the value in SERIAL.ACCESS, may execute SERIAL RELEASE and SERIAL GET. See GET, RELEASE and SERIAL.ACCESS .

?KEY is a vectored routine that executes the routine whose xcfa is stored in the user variable U?KEY. Thus the programmer may install a different routine to tailor the behavior of ?KEY to the application's needs. For example, ?KEY could access a serial port other than that on the 68HC11 chip, or different tasks could use different ?KEY routines. See ?KEY1 and ?KEY2.
Pronunciation: "question-key" Attributes: M, U

?KEY1 (-- flag)
Returns a flag indicating whether a character has been received on the primary serial port (serial1) associated with the 68HC11's on-chip hardware UART. If a character has been received a TRUE flag is returned; otherwise a FALSE flag is returned. ?KEY1 is the default ?KEY routine installed in the U?KEY user variable after the special cleanup mode is invoked, or if SERIAL1.AT.STARTUP has been executed. If the value in SERIAL.ACCESS is RELEASE.AFTER.LINE, ?KEY1 does not GET or RELEASE the SERIAL1.RESOURCE. If SERIAL.ACCESS contains RELEASE.ALWAYS, ?KEY1 GETs and RELEASEs the SERIAL1.RESOURCE. If SERIAL.ACCESS contains RELEASE.NEVER, ?KEY1 GETs but does not RELEASE the SERIAL1.RESOURCE. See SERIAL.ACCESS, ?KEY, U?KEY, ?KEY2.
Pronunciation: "question-key-one" Attributes: M

?KEY2 (-- flag)
Returns a flag indicating whether a character has been received on the secondary serial port (serial2). The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). If one or more characters are present in the serial2 input buffer a TRUE flag is returned; otherwise a FALSE flag is returned. ?KEY2 can be made the default ?KEY routine installed in the U?KEY user variable after each reset or restart by executing SERIAL2.AT.STARTUP. If the value in SERIAL.ACCESS is RELEASE.AFTER.LINE, ?KEY2 does not GET or RELEASE the SERIAL2.RESOURCE. If SERIAL.ACCESS contains RELEASE.ALWAYS, ?KEY2 GETs and RELEASEs the SERIAL2.RESOURCE. If SERIAL.ACCESS contains RELEASE.NEVER, ?KEY2 GETs but does not RELEASE the SERIAL2.RESOURCE. See SERIAL.ACCESS, ?KEY, U?KEY, ?KEY1.
Pronunciation: "question-key-two" Attributes: M

?KEYPAD (-- [n\TRUE] or [FALSE] | 0 < = n < = 19)
Scans the 4 X 5 keypad or touchscreen. If a key is being depressed, PAUSEs and waits until the key is released, then returns the key number under a true flag. If no key is depressed, returns a false flag. Consult the hardware manual for a detailed description of keypad orientation. Briefly, in the standard QED Product Design Kit and Industrial Control System orientation with 4 rows and 5 columns and the connector at the bottom of the keypad, key #0 is in the lower right corner, key #1 is just above it, and key #19 is in the upper left corner. Disables interrupts for 48 cycles (12 μseconds) each time a row is scanned. See ?KEYPRESS and KEYPAD.
Pronunciation: "question-keypad"

?KEYPRESS (-- [n\TRUE] or [FALSE] | 0 < = n < = 19)
Scans the 4 X 5 keypad or touchscreen. If a key is being depressed, returns the key number under a true flag; unlike ?KEYPAD, ?KEYPRESS does not wait for the key to be released. If no key is depressed, returns a false flag. Consult the hardware manual for a detailed description of keypad orientation. Briefly, in the standard QED Product

Design Kit or Industrial Control System orientation with 4 rows and 5 columns and the connector at the bottom of the keypad, key #0 is in the lower right corner, key #1 is just above it, and key #19 is in the upper left corner. Disables interrupts for 48 cycles (12 microseconds) each time a row is scanned. See ?KEYPAD and KEYPAD.
Pronunciation: "question-keypress"

?MATRIX.SIZE (matrix.xpfa -- d | d = number of elements in matrix)
Returns the number of elements d (not the number of bytes!) in the specified matrix. If DEBUG is ON, ABORTs if matrix.xpfa is undimensioned or is not a matrix. If DEBUG is OFF, no error checking is performed, and d is indeterminant if matrix is not dimensioned.
Pronunciation: "question-matrix-size"

?RECEIVE (xmailbox -- [wd\true] or [false] | wd = received message)
If xmailbox is empty (i.e., if it contains 0\0), returns a false flag. If xmailbox contains a message (i.e., if it does not contain 0\0), fetches the contents of xmailbox wd and stores a 0\0 into xmailbox to indicate that the message has been received and that the mailbox is now empty. Leaves the message wd on the stack under a true flag. Does not execute PAUSE. To ensure that the state of the mailbox is correctly determined, ?RECEIVE disables interrupts for 26 to 61 cycles (6.5 to 15.25 microseconds). See SEND, RECEIVE, and MAILBOX:.
Pronunciation: "question-receive"

?SEND (wd\xmailbox -- flag | flag is true if message was sent)
If the mailbox with extended address xmailbox is empty (i.e., contains 0\0), stores the 32-bit message wd in xmailbox and returns a true flag. If xmailbox is not empty, drops wd and returns a false flag. Does not execute PAUSE. The message wd can be any 32-bit quantity except 0\0. For example, the message can be an xaddress that points to a block of data. To ensure that the state of the mailbox is correctly determined, ?SEND disables interrupts for 16 to 50 cycles (4 to 12.5 microseconds). See SEND, RECEIVE, and MAILBOX:.
Pronunciation: "question-send"

@ (xaddr -- w)
Fetches a 16-bit number from the memory location specified by xaddr. The high order byte is taken from xaddr and the low order byte from xaddr+1. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "fetch"

A/D12.MULTIPLE (xaddr\u1\u2\flag\n -- | u1=timing,u2=#samples,flag=mode,n=channel#)
Acquires u2 samples from the 12 bit analog to digital (A/D) converter and stores the samples as sequential 16 bit values starting at the specified xaddr. u1 is a timing parameter, flag specifies the conversion mode, and n specifies the channel number of the A/D (0 <= n <= 7). The meaning of the flag is as follows:

Flag value	Type of conversion
-1	single ended, unipolar
0	differential, unipolar
1	single ended, bipolar
2	differential, bipolar

Single-ended sampling means that the input voltage of the specified channel is referenced to VRL which is typically analog ground. Differential sampling means that the voltage input of the specified channel's "partner" is subtracted from the voltage of the specified channel and the resulting voltage is digitized by the A/D. The pairing of "partner" channels is as follows: [0,1], [2,3], [4,5], and [6,7]. Unipolar sampling means that the input is a positive voltage that swings from VRL to VRH (typically 0 to +5 V), while bipolar sampling means that the input is interpreted as a signed 12 bit representation of an input that swings from -VRH to +VRH (typically -5V to +5V). Note that conversion of inputs more negative than -4.0 V may require an external -5V supply connected to pin 39 of the Analog I/O connector. The data transfer uses the SPI (serial peripheral interface); use INIT.A/D12&DAC to initialize the SPI interface. To ensure proper operation in a multitasking environment, this routine executes SPI.RESOURCE GET before reading the A/D and SPI.RESOURCE RELEASE before terminating. If the specified xaddr is in common memory, the first sample is taken after 128 microseconds and subsequent samples are taken every (27.5+2.5*u1) microseconds, where u1 is the specified timing parameter passed to this routine. If the specified xaddr is in paged memory, the first sample is taken after 138 microseconds and subsequent samples are taken every (50+2.5*u1) microseconds. Of course, the operation of interrupts (including timesliced multitasking) will affect these sampling times. For a faster version suitable for non-multitasking applications, see (A/D12.MULTIPLE); see also (A/D12.SAMPLE), A/D12.SAMPLE, and INIT.A/D12&DAC.
Pronunciation: "A-to-D-twelve-multiple" Attributes: M

A/D12.SAMPLE (flag\n -- u | flag=conversion mode, n=channel#, u=result)

Acquires and places on the stack a single sample u from the 12 bit analog to digital (A/D) converter. n specifies the channel number of the A/D (0 < = n < = 7). The meaning of the flag is as follows:

Flag value	Type of conversion
-1	single ended, unipolar
0	differential, unipolar
1	single ended, bipolar
2	differential, bipolar

Single-ended sampling means that the input voltage of the specified channel is referenced to VRL which is typically analog ground. Differential sampling means that the voltage input of the specified channel's "partner" is subtracted from the voltage of the specified channel and the resulting voltage is digitized by the A/D. The pairing of "partner" channels is as follows: [0,1], [2,3], [4,5], and [6,7]. Unipolar sampling means that the input is a positive voltage that swings from VRL to VRH (typically 0 to +5 V), while bipolar sampling means that the input is interpreted as a signed 12 bit representation of an input that swings from -VRH to +VRH (typically -5V to +5V). Note that conversion of inputs more negative than -4.0 V may require an external -5V supply connected to pin 39 of the Analog I/O connector. The data transfer uses the SPI (serial peripheral interface); use INIT.A/D12&DAC to initialize the SPI interface. To ensure proper operation in a multitasking environment, this routine executes SPI.RESOURCE GET before reading the A/D and SPI.RESOURCE RELEASE before terminating. Executes in 158 microseconds. For a faster version suitable for non-multitasking applications, see (A/D12.SAMPLE). See also (A/D12.MULTIPLE), A/D12.MULTIPLE, and INIT.A/D12&DAC.
Pronunciation: "A-to-D-twelve-sample" Attributes: M

A/D8.MULTIPLE (xaddr\u1\u2\n -- | u1=timing parameter, u2 = #samples, n = channel#)

Acquires u2 samples from the 8 bit analog to digital (A/D) converter in the 68HC11 and stores the samples as sequential unsigned 8 bit values starting at the specified xaddr. n specifies the channel number of the A/D (0 < = n < = 7). To ensure proper operation in a multitasking environment, this routine executes A/D8.RESOURCE GET before reading the A/D and A/D8.RESOURCE RELEASE before terminating. If the specified xaddr is in common memory, the first sample is taken after 86 microseconds and subsequent samples are taken every (10+2.5*u1) microseconds, where u1 is the specified timing parameter passed to this routine. If the specified xaddr is in paged memory, the first sample is taken after 81 microseconds and subsequent samples are taken every (32.5+2.5*u1) microseconds. Of course, the operation of interrupts (including timesliced multitasking) will affect these sampling times. For a faster version suitable for non-multitasking applications, see (A/D8.MULTIPLE). See also A/D8.SAMPLE, (A/D8.SAMPLE), and A/D8.ON.

Pronunciation: "A-to-D-eight-multiple" Attributes: M

A/D8.OFF (--)

Turns off the 68HC11's on-chip 8 bit analog to digital (A/D) converter by clearing the ADPU bit in the processor's OPTION register. The 8 bit A/D is initialized to the off state upon every reset or restart. See A/D8.ON.

Pronunciation: "A-to-D-eight-off"

A/D8.ON (--)

Turns on the 68HC11's on-chip 8 bit analog to digital (A/D) converter by setting the ADPU bit in the processor's OPTION register, and waits 100 microseconds for the A/D to stabilize. Also initializes A/D8.RESOURCE to 0\0. This routine must be executed after a reset or restart before using the 8 bit A/D. See A/D8.OFF.

Pronunciation: "A-to-D-eight-on"

A/D8.RESOURCE (-- xaddr)

A resource variable associated with the 8 bit analog to digital (A/D8) converter. Should be accessed only by the words GET ?GET and RELEASE. Initialized to 0\0 by A/D8.ON and at each reset or restart. A/D8.RESOURCE is automatically invoked by many of the A/D8 device drivers. See RESOURCE.VARIABLE:.

Pronunciation: "A-to-D-eight-resource"

A/D8.SAMPLE (n -- byte | n = channel#)

Acquires and places on the stack a single sample byte from the 8 bit analog to digital (A/D) converter in the 68HC11. n specifies the channel number of the A/D (0 <= n <= 7). To ensure proper operation in a multitasking environment, this routine executes A/D8.RESOURCE GET before reading the A/D and A/D8.RESOURCE RELEASE before terminating. This routine executes in 93 microseconds. For a faster version suitable for non-multitasking applications, see (A/D8.SAMPLE). See also (A/D8.MULTIPLE), A/D8.MULTIPLE, and A/D8.ON.

Pronunciation: "A-to-D-eight-sample"

ABORT ([...] --)

Return Stack: (R: [...] --)

If the CUSTOM.ABORT flag is true, executes the abort routine whose xcfa is stored in the user variable UABORT, and then returns to the routine that called ABORT. If

CUSTOM.ABORT is false, executes the default routine (ABORT) which clears the data and return stacks, sets the page to the default page (0), and executes FORTH DEFINITIONS to set CONTEXT and CURRENT equal to FORTH. If an autostart vector has been installed (see AUTOSTART), (ABORT) executes the specified routine; otherwise it executes QUIT which sets the compilation mode and enters the interpreter. If R0 and S0 aren't in common RAM, a COLD restart is initiated.

ABORT" (flag --)
Compile Time: (<text> --)
If flag is true, prints the <text> string between ABORT" and the terminating " and then executes ABORT. If flag is false, drops flag and continue execution. Useful for error detection and reporting.
Pronunciation: "abort-quote" Attributes: C, I, M

ABS (n1 -- +n2 | +n2 = absolute value of n1)
Replace n1 with its absolute value +n2. If n1 is positive, +n2 = n1. If n1 is negative, +n2 is the negative of n1.
Pronunciation: "abs"

ACTIVATE (xcfa\xtask.id --)
Sets up the routine specified by xcfa as the action word of the task whose task identifier (STATUS xaddress) is xtask.id, and leaves the specified task AWAKE so that it will be entered on the next pass through the round robin task list. ACTIVATE assumes that the specified task has already been added to the task list by BUILD.TASK or BUILD.STANDARD.TASK. The task's action word is typically either an infinite loop or a finite routine that ends with a HALT instruction (which is itself an infinite loop). ACTIVATE buries a call to HALT in the return stack frame to ensure graceful termination of a finite activation routine. If cooperative multitasking is used exclusively (i.e., if the timeslicer is not used), then the loop of the action word must contain at least one PAUSE statement (or invoke a word that in turn executes PAUSE). Otherwise, no task switching occurs. If timeslicing is used, incorporation of PAUSE statements in the loop of the action word is optional. The typical form of an action word is:

```
: <action.name>
        words to be executed once
        BEGIN
                words to be executed infinitely
                PAUSE
                words to be executed infinitely
        AGAIN
;
```

or:

```
: <action.name>
        words to be executed
        PAUSE
        words to be executed
        HALT
;
```

For example, if a task has been defined with

```
TASK: <task.name>
```

and built using BUILD.TASK or BUILD.STANDARD.TASK, it can be activated by executing

    CFA.FOR <action.name> <task.name> ACTIVATE

ADDR-> (u1 <name> -- u2)

Adds a named member to the structure being defined and reserves room for one 16 bit address field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.

Pronunciation: "address" Attributes: D

ADDR: (<name> --)

ADDR: is a synonym for INTEGER: . It defines a 16-bit self-fetching variable. ADDR: is meant to hold a 16-bit address. See the glossary entry for INTEGER: .

Pronunciation: "address-colon" Attributes: D

ADDRS-> (u1\u2 <name> -- u3)

Adds a named member to the structure being defined and reserves room for u2 16 bit addresses in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.

Pronunciation: "addresses" Attributes: D

AGAIN (--)

AGAIN is used within a colon definition to mark the end of an infinite loop structure as:

```
BEGIN
        <words to be iterated>
AGAIN
```

The words between BEGIN and AGAIN are executed indefinitely. AGAIN is equivalent to FALSE UNTIL. An error is issued if BEGIN and AGAIN are not properly paired inside a definition.

Attributes: C, I

ALL.COLUMNS.SCALED (matrix.xpfa1\[row#1\-1] or [-1\col#1]\ matrix.xpfa2 --)

Scales each column of the matrix specified by matrix.xpfa1 by dividing each element in the column by the amount designated in the scaler row/col specified by [row#1\-1] or [-1\col#1]\ matrix.xpfa2 which must have the same number of elements as the number of columns of the matrix. The scaler row/col should not be part of the matrix being scaled. That is, matrix.xpfa1 may not equal matrix.xpfa2.

Attributes: S

ALL.ROWS.SCALED (matrix.xpfa1\[row#\-1] or [-1\col#]\ matrix.xpfa2 --)

Scales each row of the matrix specified by matrix.xpfa1 by dividing each element in the row by the amount designated in the scaler row/col specified by [row#\-1] or [-1\col#]\ matrix.xpfa2 which must have the same number of elements as the number of rows of the matrix. The scaler row/col should not be part of the matrix being scaled. That is, matrix.xpfa1 may not equal matrix.xpfa2.
Attributes: S

ALLOCATED (u\xpfa -- | u is heap item size in bytes)
Allocates u bytes of heap memory and associates it with the item having the specified parameter field address xpfa. The xpfa is typically associated with a word defined by H.INSTANCE:. Typical use:

```
size.of.heap.item  H.INSTANCE:  <name>
SIZE.OF <name>  ' <name> ALLOCATED
```

or:

```
size.of.heap.item  ' <name> ALLOCATED
```

See H.INSTANCE: and SIZE.OF.

ALLOT (n --)
Reserves n bytes in the dictionary by incrementing the definitions pointer DP by n. An error occurs if the ALLOT operation causes DP to be incremented across the boundary between 0x7FFF (the last valid address in a given page) and 0x8000 (the start of the register area).

AND (w1\w2 -- w3)
Performs a logical bit-wise 'and' of two 16 bit numbers w1 and w2 to produce the result w3.

ANEW (<name> --)
Tries to find <name> in the CURRENT vocabulary. If <name> is not found or was not created by ANEW, then creates <name>. If <name> is found, executes <name> which resets the variable pointer VP to the value it had when ANEW <name> was first executed, then FORGETs all words defined after <name> was created; this resets DP and NP to the values they had when ANEW <name> was first executed. ANEW should be used to avoid redundancy when reloading code during debugging. Note that heap items associated with forgotten words are not released by ANEW and should be handled by the programmer using ON.FORGET. See FORGET and ON.FORGET.

ARRAY.PF (-- u | u = size of an array parameter field)
Places on the stack the number of bytes in an array parameter field based on the current value of MAX#DIMENSIONS. Typically used to define a stack-based temporary array within a definition; temporary arrays defined in this manner preserve re-entrancy (see the chapter on Designing Re-entrant Code in the Software Manual). For example:

```
: ARRAY.FUNCTION
        LOCALS{ .... | x&temp.array.pfa }
        ARRAY.PF PF.STACK.FRAME   TO  x&temp.array.pfa
        10 1 6 x&temp.array.pfa DIMENSIONED           \ dimension
        ....                  \ use the temp array
        x&temp.array.pfa DELETED       \ delete from heap
        ARRAY.PF FRAME.DROP        \ drop temp pf off stack
;
```

See MATRIX.PF, PF.STACK.FRAME and FRAME.DROP.
Pronunciation: "array-p-f"

ARRAY: (<name> --)
Removes <name> from input stream and defines <name> as an array. Allots and clears a parameter field for <name> in the variable area. When executed, <name> returns the extended element address given the indices; its stack picture is:
    (indices -- xaddr)
The element xaddress is also returned by the command
    indices ' <name> []
ARRAY: does not allocate heap space or dimension the array; see DIMENSIONED.
Pronunciation: "array-colon" Attributes: D

ASCII (<name> -- char)
Removes <name> from the input stream and converts its first character to its ASCII value char. In execution mode the ascii value is left on the stack. In compilation mode the ascii code is compiled as a literal into the current definition.
Attributes: I

ASK.FNUMBER (<text> -- [r\-1] or [0])
Inputs a character string <text> to the PAD buffer, terminating when CHARS/LINE characters are received or a carriage return is received, whichever comes first. Leaves <text> as a counted string at PAD and, ignoring leading blanks, attempts to convert the <text> string to a valid floating point number. If <text> is an ascii representation of a valid integer or double number or floating point number, the equivalent floating point representation r is left on the stack under a true flag; otherwise, a false flag is left on the stack. See ASK.NUMBER and NEXT.NUMBER.
Pronunciation: "ask-f-number" Attributes: M, S

ASK.NUMBER (<text> -- [n\1] or [d\2] or [0])
Inputs a character string <text> to the PAD buffer, terminating when CHARS/LINE characters are received or a carriage return is received, whichever comes first. Leaves <text> as a counted string at PAD and, ignoring leading blanks, attempts to convert the <text> string to a single or double number. If the string is converted to a 16-bit integer n, leaves n under a 1 flag. If the string cannot be represented as a 16-bit integer but is a valid 32-bit double number d, leaves d on the stack under a 2 flag. Leaves a 0 flag on the stack if the <text> string cannot be converted to a valid integer. See ASK.FNUMBER and NEXT.NUMBER.
Attributes: M, S

ASLEEP (-- 1)
A constant that places the value 1 on the stack. When stored into a task's STATUS user variable, indicates to the multitasking executive that the task is asleep and cannot be entered.

ASSEMBLER (--)
Sets CONTEXT equal to the assembler vocabulary's xhandle so that the assembler vocabulary is the first vocabulary searched during dictionary searches.

ATTACH (xcfa\n -- | n = interrupt identity number)

Posts an interrupt handler routine specified by xcfa for the interrupt with identity number n (e.g., OC1.ID, OC2.ID, etc.) Compiles an 8-byte code sequence at the EEPROM location associated with the specified interrupt. When the interrupt is serviced, the code at xcfa will be executed. The xcfa can be on any page. If coded in high level, the interrupt handler routine should end with a ; and if coded in assembly should end with an RTS (as opposed to an RTI).

AUTOSTART (xcfa --)
Compiles a 6-byte sequence into the EEPROM in the 68HC11. On subsequent restarts and ABORTs, the routine having the specified xcfa will be executed. This allows a finished application to be automatically entered upon power up and resets. CAUTION: If your application is to be put into production and replicated, it is recommended that you use the PRIORITY.AUTOSTART function which stores the 6-byte autostart sequence in flash memory.
Implementation detail: At location 0xAE00 in EEPROM, AUTOSTART writes the pattern 1357 followed by the four byte xcfa. To undo the effects of this command and return to the default startup action, use NO.AUTOSTART. To recover from the installation of a buggy autostart routine, use the special cleanup mode. See PRIORITY.AUTOSTART, and consult the "Interrupts, Register Initializations, and Autostarting" chapter in the Software Manual.

AWAKE (-- 0)
A constant that places the value 0 on the stack. When stored into a task's STATUS user variable, indicates to the multitasking executive that the task is awake and may be entered.

AXE (<name> --)
If <name> is found in the CURRENT vocabulary, removes its header and compacts the vocabulary, but leaves the definition (i.e., the code field) of <name> intact. Once a word's header has been AXEd it can no longer be found in the dictionary. AXE is useful for conserving memory space in the names area of the dictionary. AXE works properly only if the name area after <name> is a contiguous single-vocabulary linked list in modifiable RAM . An error is issued if <name> is not found or if <name> is on a different page than that returned by LATEST. An unchecked error occurs if words have been defined into vocabularies other than the CURRENT vocabulary since <name> was defined, or if NP has been explicitly moved with an

```
    <xaddr> NP X!
```

command since <name> was defined.

BACKTRACK (--)
Resets >IN to point to the first character of the word in the input stream that was most recently parsed by WORD.

BASE (-- xaddr)
User variable that contains the current number base (number conversion radix) used for number I/O and numeric conversion. Unchecked error if the contents of BASE are less than 2 or greater than 72.
Attributes: U

BAUD1.AT.STARTUP (n --)

Configures the QED Board so that the baud rate of the primary serial port (serial1) supported by the 68HC11's hardware UART will equal the specified standard baud rate upon all subsequent resets and restarts. Standard baud rates for boards clocked at 16 MHz are 150, 300, 600, 1200, 2400, 4800, 9600, and 19200 baud.
Implementation detail: This routine calls INSTALL.REGISTER.INITS which writes into EEPROM the required contents of INIT (=0xB8), the contents of BAUD that corresponds to the specified baud rate, and the contents of OPTION, TMSK2, BPROT that are present when this routine is executed. These values are installed in their respective registers upon each subsequent reset and restart. To undo the effects of this command, execute DEFAULT.REGISTER.INITS or invoke the special cleanup mode.
Pronunciation: "baud-one-at-startup"

BAUD2 (n --)
Sets the baud rate of the secondary serial port (serial2) supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). Smooth file transfers can be achieved at up to 4800 baud, but note that the operation of other interrupt service routines may lower the attainable error-free baud rate. The baud rate of serial2 is initialized to 1200 baud by the COLD restart routine. See USE.SERIAL2.
Pronunciation: "baud-two"

BEEP (--)
Emits the bell character, ascii 07, if QUIET is OFF .

BEGIN (--)
BEGIN is used within a colon definition to mark the start of a loop structure as:

```
BEGIN ... UNTIL
BEGIN ... WHILE ... REPEAT
BEGIN ... AGAIN
```

The words after UNTIL or REPEAT are executed after the loop structure terminates. BEGIN ... AGAIN is an infinite loop. An error is issued if BEGIN is not properly paired in a loop structure.
Attributes: C, I

BENCHMARK: (<name> --)
Measures and displays the execution time and operations count for the word <name>. The timeslice clock must be running to benchmark a word. This can be accomplished by executing START.TIMESLICER before invoking BENCHMARK:. <name> may be any executable word. Data stack parameters required by <name> should be placed on the stack before calling BENCHMARK:. Typical use:

```
START.TIMESLICER    \ if timeslicer wasn't already running ...
            \ push necessary stack parameters for <name>
BENCHMARK: <name>
```

See also (BENCHMARK:).
Pronunciation: "benchmark" Attributes: M, S

BL (-- char)
Puts the ascii value for a blank (a space, ascii value 32) on the data stack.
Pronunciation: "b-l"

BLANK (xaddr\u -- | u = byte count)

The ascii character value for space (32) is stored in each of u consecutive bytes beginning at xaddr. The specified region may cross page boundaries. Does nothing if u = 0.

BLANK.ARRAY (array.xpfa --)
Stores an ascii blank (32) into each byte of the specified array.

BLK (-- xaddr)
A user variable whose contents equal the number of the file block currently being interpreted. Contents of 0 indicate that the input stream is to be taken from the terminal input buffer (TIB). Modified by -->, LOAD and THRU, cleared by QUERY.
Pronunciation: "b-l-k" Attributes: U

BLOCK (n -- xaddr | n = block#, xaddr = buffer)
Performs the function of BUFFER which assigns the block specified by n to the buffer starting at xaddr, and then reads the contents of the specified block n from mass memory into its assigned buffer at xaddr. See BUFFER. Note that the starting location of the specified block in mass memory is given by 1024*(n + offset) where offset equals the 32-bit contents of the user variable OFFSET.
Attributes: M

BOOLEAN (w -- flag)
Converts a 16-bit integer into a boolean flag. Flag is FALSE (0) if w is 0, otherwise flag is TRUE (-1).

BREAK (--)
Sets a software breakpoint when compiled into any function (including assembly language functions). At execution time, BREAK suspends the program flow, saves the machine state and invokes a FORTH-style text interpreter that can be distinguished from the standard interpreter by the BREAK> prompt displayed at the start of each line. Any valid commands may be executed from within the BREAK interpreter. From within the BREAK interpreter, typing a carriage return alone on a line exits the BREAK mode, restores the machine registers to the values they held just before BREAK was entered, and resumes execution of the program that was running when BREAK was entered. The BREAK routine's preservation of the register state and its ability to execute any valid command make it a very powerful debugging tool. BREAK may be compiled into any definition to stop program flow in order to debug or analyze a word at the point where BREAK was called. Once inside BREAK, the stack contents may be displayed (using .S) or altered. Variables and memory locations may be displayed or altered. New words can even be defined and executed. BREAK is called by the trace routine if the variable SINGLE.STEP is set. To single step through some code, compile the code with TRACE ON, then execute the code with DEBUG and SINGLE.STEP ON. After each call, the name of the traced word and the stack picture will be printed, and then the BREAK word will execute, letting you execute FORTH commands. To go to the next "step" in the word being debugged, enter a CR alone on a line. To display the register state after each traced line, execute DUMP.REGISTERS ON. To stop single-stepping but continue tracing, execute the forth command SINGLE.STEP OFF. The trace will continue, but BREAK will not be called again, unless you hit a key. The trace routine enters the BREAK mode when a keystroke is detected at the serial I/O port: if you hit a key, BREAK is called by TRACE, and again, a CR resumes execution. To exit the

traced definition completely, execute ABORT (or any illegal command) from within the BREAK interpreter. Any error encountered while in the BREAK routine executes ABORT which places the programmer back into the standard QED-Forth interpreter (unless ABORT has been revectored to perform some other action; see CUSTOM.ABORT). See DEBUG, TRACE, SINGLE.STEP, DUMP.REGISTERS, and IS.TRACE.ACTION.
Attributes: M, S

BUFFER (n -- xaddr | n = block#, xaddr = buffer)
Returns the extended address xaddr of the first byte of the block buffer assigned to the block specified by n. If the specified block is already in a buffer, writes the buffer's contents to mass memory if the block has been UPDATED, and returns the extended address of the buffer. If the specified block is not already in a buffer, assigns it to a buffer, writing the previous contents of the buffer to mass memory if the previous contents had been UPDATED. BUFFER does not load the buffer with the contents of the specified block; see BLOCK. Note that the starting location of the specified block in mass memory is given by 1024*(n + offset) where offset equals the 32 bit contents of the user variable OFFSET.
Attributes: M

BUFFER.POSITION (n1\n2 -- n3 | n1 = line#, n2 = char#, n3 = buffer.offset)
Given the specified LCD display line number n1 (0 <= n1 < LINES/DISPLAY) and the specified character position in the display line (0 <= n2 < CHARS/DISPLAY.LINE), calculates the offset n3 of the specified position relative to the extended base address returned by DISPLAY.BUFFER. Clamps n3 to ensure that the buffer.offset is not greater than the size of the buffer. Note that for a graphics-style display the line# n1 is interpreted differently depending on whether the display is being used in "text mode" or "graphics mode". In text mode, n1 corresponds to the character line#; in graphics mode, n1 corresponds to the pixel line#. See LINES/DISPLAY for further information.

BUFFER>SPI (xaddr\+n --)
This routine is headerless; its xcfa is stored at address 0x7FF8 on page 0x0C. Writes to the SPI the contents of the buffer specified by xaddr and +n, where xaddr is the starting address, and +n is the number of bytes (0 <= +n <= 32,768). The buffer must not cross a page boundary. This routine does not GET or RELEASE the SPI.RESOURCE, nor does it modify the configuration of the SPI or activate any chip selects. If required, these additional functions must be performed by the calling program. This routine is optimized for speed, and executes at 9 microseconds per byte. To access this routine, add the following definition to your Forth source code:

```
HEX  : BUFFER>SPI    7FF8  C X@  EXECUTE ;
```

BUILD.STANDARD.TASK (xaddr1\xaddr2\xaddr3\xtask.id -- |
xaddr1=xheap.start, xaddr2=xheap.end, xaddr3=VP)
Builds a task with a specified heap and variable area and no compilation privileges. The task's stacks, user area, PAD, POCKET, and TIB are assigned to a 1Kbyte block of common RAM starting at xtask.id (the base of the task's user area). The task is appended to the round-robin task list and left ASLEEP running the default action word HALT. xaddr1 is the extended heap starting address, and xaddr2 is the extended heap end address. BUILD.STANDARD.TASK passes these to IS.HEAP which initializes the heap accordingly. xaddr3 specifies the start of the variable area for the task, and

xtask.id is the task identifier xaddress (also called the task's STATUS address or the base of its user area.) DP and NP are set to xaddress 0\0 in ROM so that the task cannot compile new words (it can, however, interpret and execute previously defined words). The 256-byte user area of the parent task (i.e., the task that is active when this command executes) is copied to create the new task's user area, so the parent's configuration is initially "inherited" by the new task. This implies that the new task has access to all the words in the parent's dictionary. The variables that control the memory map of the new task are set so that R0 = xtask.id + 0x400, S0 = xtask.id + 0x300, (both stacks have 1/4K space and grow downward in memory), TIB extends upward for 94 bytes starting at xtask.id + 0x180, POCKET extends upward for 32 bytes starting at xtask.id + 0x1E0, and PAD extends upward for 82 bytes and downward for 36 bytes starting at xtask.id + 124H. (Implementation detail: The multitasker uses the 2 bytes below TIB to hold a C stack pointer.) To initialize CURRENT.HEAP without modifying the heap control variables, pass BUILD.STANDARD.TASK a heap start xaddress that is equal to the heap end xaddress (see IS.HEAP).

BUILD.TASK (xheap.start\xheap.end\xvp\xdp\xnp\xtib\xpad\xpocket\xr0\xs0\xtask\n --)

Builds a task with a specified memory map. Appends the task to the round-robin task list and leaves it ASLEEP running the default action word HALT. The stack picture above uses non-standard symbols that are more descriptive than a long list of xaddr items. All but the last item on the stack are extended addresses (xaddr). The last item n is the integer size of the user area. The first n bytes of the user area of the parent task (i.e., the task that is active when this command executes) are copied to create the new task's user area, so the parent's configuration is initially "inherited" by the new task. This implies that the new task has access to all the words in the parent's dictionary because the values in the new task's CONTEXT and CURRENT user variables have been copied from the parent. The variables that control the memory map of the new task are set according to the parameters passed to BUILD.TASK. xheap.start\xheap.end are passed to IS.HEAP which initializes the heap accordingly. (To initialize CURRENT.HEAP without modifying the heap control variables, pass BUILD.STANDARD.TASK a heap start xaddress that is equal to the heap end xaddress; see IS.HEAP). xvp specifies the contents of VP in the new task's user area, xdp specifies DP, xnp specifies NP, xtib specifies the contents of UTIB, xpad specifies the contents of UPAD, xpocket specifies the contents of UPOCKET, xr0 specifies R0 which positions the return stack, and xs0 specifies S0 which positions the data stack. xtask.id is the base address of the user area; it is the xaddress placed on the stack when the task's name is invoked. xr0, xs0, and xtask.id must be in common ram. The user area grows upward in memory, and stacks grow downward. If the new task ever calls WORD or interprets input, a POCKET buffer must be allocated and must be in the common RAM. The heap, VP, DP, NP, TIB, and PAD can be anywhere in memory. BUILD.TASK gives the programmer complete flexibility in allocating memory resources to a task. Some tasks might not need all of these memory areas; in this case, default xaddresses in ROM such as 0\0 can be used to initialize the unneeded memory pointers. The minimum required resources for a task are the first 6 bytes of the user area (STATUS, NEXT.TASK, and RP.SAVE) and a return stack. At the other end of the complexity scale, tasks that can compile new definitions and perform math, I/O and heap operations need to allocate all of the memory areas. Passing values for xvp and xdp in RAM allows compilation of new definitions and the task could subsequently have a private dictionary segment that is not accessible to other tasks. (However, note that concurrent compilation by multiple tasks is discouraged, as some compilation variables such as local variable save locations are not in task-private memory). An error is issued if xtask.id, xr0 or xs0 is not in the common RAM. See BUILD.STANDARD.TASK.

BYTE-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a single byte field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "byte" Attributes: D

BYTES-> (u1\u2 <name> -- u3)
Adds a named member to the structure being defined and reserves room for u2 bytes in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "bytes" Attributes: D

C! (byte\xaddr --)
Stores byte at xaddr.
Pronunciation: "c-store"

C, (byte --)
Stores byte at the next available location in the definitions area and increments the definitions pointer DP by 1. An error occurs if byte is not correctly stored; e.g. if DP does not point to RAM. An error occurs if the C, operation causes DP to be incremented across the boundary between 0x7FFF (the last valid address in a given page) and 0x8000 (the start of the register area).
Pronunciation: "c-comma"

C@ (xaddr -- byte)
Fetches the byte stored at xaddr.
Pronunciation: "c-fetch"

CALC.CHECKSUM (xaddr\+n -- checksum | n MUST be even)
This routine is headerless; its xcfa is stored at address 7FFC on page C. Calculates a 16-bit checksum for the buffer specified by xaddr and +n, where xaddr is the starting address, and +n is the number of bytes (0 <= +n <= 32,768). The buffer must not cross a page boundary, and n must be an even number of bytes. The checksum is calculated by initializing a 16-bit accumulator to zero, then adding in turn each 2-byte number in the buffer to the accumulator; the checksum is the final value of the accumulator. Using this routine provides a method of checking whether the contents of an area of memory have changed since a prior checksum was calculated. This routine is optimized for

speed, and executes at less than 3 microseconds per byte. To access this routine, add the following definition to your Forth source code:

```
    HEX  : CALC.CHECKSUM        7FFC  C X@  EXECUTE ;
```

CALL (<name> --)
Removes <name> from the input stream and compiles a call to <name> into the current definition, where <name> is the name of an executable FORTH or assembly coded routine. Typically used in an assembly coded definition. See the Assembler chapter of the Software Manual for examples of use.

CASE (n -- n)
Used inside a colon definition to mark the beginning of a CASE statement which implements a multi-decision control structure. Use as:

```
    n1 CASE
            n2 OF words to be executed if n1 = n2           ENDOF
            n3 OF words to be executed if n1 = n3           ENDOF
            n4 OF words to be executed if n1 = n4           ENDOF
            words to be executed if n1 does not equal n2 or n3 or n4
    ENDCASE
```

An error is issued if CASE and ENDCASE are not properly paired in a definition. See ENDCASE, OF, ENDOF, RANGE.OF, and URANGE.OF.
Attributes: C, I

CFA.FOR (-- xcfa)
Compile Time: (<name> --)
Removes <name> from the input stream and returns <name>'s extended code field address xcfa. xcfa is the first byte of executable code associated with <name>'s definition. If in execution mode, leaves the xcfa on the stack. If in compilation mode, compiles the xcfa as a 2-cell literal in the current definition; the xcfa is pushed to the stack when the definition later executes. An error occurs if no <name> is given or if <name> cannot be found in the dictionary.
Pronunciation: "c-f-a-for" Attributes: I

CFA.PTR (xnfa -- xaddr)
Given the extended name field address xnfa of a header in the dictionary, returns the xaddr in the header that contains the 3-byte code field address associated with the header. The page of the code field is a single byte stored at xaddr, and the address of the code field is a 16-bit address stored at xaddr+1. See NFA.FOR.
Pronunciation: "c-f-a-pointer"

CFA>NAME (xcfa --)
Prints the name of the word associated with the specified extended code field address xcfa. Useful for error diagnostics to print the name of the word in which an error occurs. The name is printed as ?NAME? if no name corresponding to xcfa is found in the dictionary.
Pronunciation: "c-f-a-to-name"

CFA>NFA (xcfa -- [xnfa] or [0\0])
Given the extended code field address xcfa of a word in the dictionary, searches the dictionary and returns the extended name field address xnfa of the word. If the name

associated with xcfa cannot be found in the dictionary, returns 0\0. xcfa is the first byte of executable machine code associated with the definition, and xnfa is the count byte of the word's header. See CFA.FOR, ID. and NFA.FOR.
Pronunciation: "c-f-a-to-n-f-a"

CFA>PFA (xcfa -- [xpfa] or [0\0])
Given the extended code field address xcfa of a word in the dictionary, searches the dictionary and returns the extended parameter field address xpfa of the word. If the name associated with xcfa cannot be found in the dictionary or if it does not have a parameter field, returns 0\0. See CFA.FOR and '
Pronunciation: "c-f-a-to-p-f-a"

CHANGE.BITS (byte1\byte2\xaddr -- | byte1 = data; byte2 = mask)
At the byte specified by xaddr, modifies the bits specified by 1's in byte2 to have the values indicated by the corresponding bits in byte1. In other words, byte2 serves as a mask which specifies the bits at xaddr that are to be modified, and byte1 provides the data which is written to the modified bits. Disables interrupts for 16 cycles (4 microseconds) to ensure an uninterrupted read/modify/write operation. See also (CHANGE.BITS).

CHAR>DISPLAY (char --)
Writes the specified data byte char to the LCD display. Does not write to the Display Buffer. If an alphanumeric (character) display is being used, this command writes the specified ascii character at the current cursor position and increments the cursor position. (Caution: the cursor does not always follow a contiguous path as it is incremented; there may be discontinuities at the ends of lines.) Intermittently disables interrupts for 28 cycles (7 microseconds) per byte written to the display. See COMMAND>DISPLAY and UPDATE.DISPLAY.
Pronunciation: "char-to-display"

CHARS/DISPLAY.LINE (-- n)
Returns the number of characters per line in the LCD display as specified by the last execution of IS.DISPLAY. The default value of n after executing the "special cleanup mode" is 20, corresponding to the default 4-line by 20-character display. The result returned by this routine is used by BUFFER.POSITION, PUT.CURSOR, UPDATE.DISPLAY, and UPDATE.DISPLAY.LINE.
Pronunciation: "chars-per-display-line"

CHARS/LINE (-- xaddr)
A user variable that contains the maximum number of characters that can be received by EXPECT. Also used by matrix print words M. M.. and M.PARTIAL to format their output. CHARS/LINE is initialized to a default value of 96 upon each COLD restart, and its value should not be increased above 96 unless the TIB is moved from its default location.
Pronunciation: "chars-per-line" Attributes: U

CLEAR.BITS (byte1\xaddr --)
For each bit of byte1 that is set, clears the corresponding bit of the 8 bit value at xaddr. Disables interrupts for ten cycles (2.5 microseconds) to ensure an uninterrupted read/modify/write operation. See also (CLEAR.BITS) and SET.BITS.

CLEAR.DISPLAY (--)

Clears (blanks) the LCD display, moves the cursor to home position (at the start of line 0). If a character display is in use (as specified by IS.DISPLAY), fills the 80 character DISPLAY.BUFFER with ascii blank characters. If a graphics display is being used in text mode, fills the buffer specified by GARRAY.XPFA with ascii blanks. If a graphics display is being used in graphics mode, erases (zeros) the buffer specified by GARRAY.XPFA. Intermittently disables interrupts for 28 cycles (7 microseconds) per byte written to the display. See INIT.DISPLAY.

CLEAR.HIGH.CURRENT (byte --)

For each bit of the input mask byte that is set, turns the corresponding high current driver OFF (so that it is not sinking current). Bits 0-3 in the input mask byte control the high current drivers named HC0-HC3, respectively. Disables interrupts for 31 cycles (less than 8 microseconds). See also SET.HIGH.CURRENT.

CLOCK.MONITOR.ID (-- n)

Returns the interrupt identity code for the clock monitor interrupt. Used as an argument for ATTACH.
Pronunciation: "clock-monitor-i-d"

CMOVE (xaddr1\xaddr2\u -- | xaddr1=src, xaddr2=dest, u = byte count)

If u is greater than 0, u consecutive bytes are copied from addresses starting at xaddr1 to addresses starting at xaddr2. The source and destination extended addresses may be located on different pages and the move may cross page boundaries. If the source and destination regions overlap and xaddr1 < xaddr2, CMOVE starts at high memory and moves toward low memory to avoid propagation of the moved contents. CMOVE always moves the contents in such a way as to avoid memory propagation. Speed is approximately 19 microseconds per byte. See CMOVE.MANY.
Pronunciation: "c-move"

CMOVE.IN.PAGE (addr1\addr2\u\page -- | addr1=src, addr2=dest, u = byte count)

If u is greater than 0, u consecutive bytes starting at addr1 are copied to the destination addresses starting at addr2 on the specified page. If the source and destination regions overlap and addr1 < addr2, CMOVE.IN.PAGE starts at high memory and moves toward low memory to avoid propagation of the moved contents. CMOVE.IN.PAGE always moves the contents in such a way as to avoid memory propagation. Speed is approximately 7.5 microseconds per byte.
Pronunciation: "c-move-in-page"

CMOVE.MANY (xaddr1\xaddr2\d -- | xaddr1=src, xaddr2=dest, d = byte count)

If the 32-bit byte count d is greater than 0, d consecutive bytes are copied from addresses starting at xaddr1 to addresses starting at xaddr2. The source and destination extended addresses may be located on different pages and the move may cross page boundaries. If the source and destination regions overlap and xaddr1 < xaddr2, CMOVE.MANY starts at high memory and moves toward low memory to avoid propagation of the moved contents. CMOVE.MANY always moves the contents in such a way as to avoid memory propagation. Speed is approximately 19 microseconds per byte.
Pronunciation: "c-move-many"

CODE (<name> --)

Begins an assembly coded definition. Removes <name> from the input stream and creates a header for <name> that cannot be found in the dictionary until END.CODE executes. Executes ASSEMBLER so that the assembler mnemonics can be found by the interpreter. The assembly mnemonics between CODE and END.CODE form the body of the definition. See END.CODE.
Attributes: D

COL->V (col#\matrix.xpfa -- xvaddr\sep\d.#el)

Returns the vector representation xvaddr\sep\d.#el of the specified column in the specified matrix. xvaddr is the base address of the vector, sep is the element separation expressed as a multiple of 4 bytes (e.g., sep=1 means a vector of contiguous floating point numbers, sep=2 means elements are separated by 8 bytes, etc.) and the double number d.#el is the number of elements in the vector. In the case of a column in a matrix, sep=1 (i.e., column elements are stored in contiguous memory locations) and d.#el is the 32-bit equivalent of the number of rows in the matrix. Note that xvaddr must be 4-byte aligned (i.e., must be an even multiple of 4). The heap manager and array and matrix dimensioning words automatically perform the required 4-byte alignment. See also ROW->V.
Pronunciation: "col-to-v"

COLD (--)

Disables interrupts and restarts the QED-Forth system and initializes all of the user variables to their default values. Initializes the following machine registers:
PORTG, DDRG, TMSK2, SPCR, BAUD, SCCR1, SCCR2, BPROT,
OPT2, OPTION, HPRIO, INIT, CSSTRH, CSCTL, CSGADR, CSGSIZ.
Initializes the vectors of the vital interrupts if INIT.VITAL.IRQS.ON.COLD has been executed. Calls ABORT which clears the stacks and calls either the QED-Forth interpreter or an autostart routine that has been installed using AUTOSTART. If COLD.ON.RESET has been executed, every reset or power-up will invoke a COLD (as opposed to a WARM) initialization sequence. Consult the Program Development Techniques chapter and the Interrupts, Initializations, and Autostarting chapter in the Software Manual for more information about cold restarts.

COLD.ON.RESET (--)

Initializes a flag in EEPROM that causes subsequent resets to execute a cold restart (as opposed to the standard warm-or-cold restart). This option is useful to help "bullet-proof" turnkeyed systems that have an autostart word installed; any error or reset causes a full COLD restart which initializes all user variables, after which the autostart routine completes the system initialization and enters the application routine. See STANDARD.RESET. Implementation detail: Initializes location 0xAE1C in EEPROM to contain the pattern 13.

COLUMN.CONCATENATE (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)

Concatenates the two source matrices specified by matrix.xpfa1 and matrix.xpfa2 to form a destination matrix matrix.xpfa3 with more columns. The number of rows in the two source matrices must be the same. The destination may be one of the sources.

COLUMN.TRUNCATE (matrix.xpfa1\matrix.xpfa2\n --)

Copies all but the final n columns of the source matrix specified by matrix.xpfa1 to the destination specified by matrix.xpfa2. The destination may be the source.

COMMAND>DISPLAY (byte --)

Writes the specified byte to the LCD display as a command (as opposed to a character to be displayed). Does not modify the contents of the DISPLAY.BUFFER. Intermittently disables interrupts for 28 cycles (7 microseconds) per command byte written to the display. See CHAR>DISPLAY.
Pronunciation: "command-to-display"

COMPILE (<name> --)

Removes the next <name> from the input stream. Use as:

```
: <namex>
        ... COMPILE <name> ...
;
```

where <namex> is typically immediate and <name> is typically not immediate. Compiles into the current definition code that will cause <name> to be compiled when <namex> is executed. That is, COMPILE defers the compilation of <name> until <namex> executes. Consult the Advanced Topics chapter of the Software Manual for further description and an example.
Attributes: C, I

COMPILE.CALL (xcfa --)

Compiles a call to the assembly language subroutine whose first byte of executable machine code is stored at xcfa. If no page change is needed at runtime, (COMPILE.CALL) is executed. If a page change is needed, an 8-byte sequence is compiled into the definitions area to accomplish the page change at run time.

COMPLEMENT (w1 -- w2)

Returns the ones complement of w1. That is, inverts each bit of w1 to produce w2.

CONSTANT (w <name> --)

Removes the next <name> from the input stream and defines a child word called <name> which when executed leaves the value w on the data stack. w is stored in the definitions area of the dictionary. <name> is referred to as a "constant". Use as:

```
w CONSTANT <name>
```

Attributes: D

CONTEXT (-- xaddr)

A user variable that contains a 32-bit xhandle which in turn contains the xnfa of the top word in the vocabulary to be searched first. Thus CONTEXT X@ returns the xhandle of the search vocabulary, and CONTEXT X@ X@ returns the xnfa of the top word in the search vocabulary. In short, the contents of CONTEXT determine the search vocabulary. See FIND and CURRENT.
Attributes: U

CONVERT (ud1\xaddr1 -- ud2\xaddr2)

Converts the numeric string starting at xaddr1+1 into the 32-bit number ud2. Conversion is accomplished by multiplying the double accumulator ud1 by the value in BASE and then adding the next digit from the string at xaddr1. Conversion ends when a non-convertible ASCII character is encountered in the string. Isolated embedded commas are ignored and are not treated as a non-convertible character. xaddr2 is the address of the first non-convertible character encountered in the string. For example, executing 0\0 " 123 " CONVERT leaves a 32-bit representation of the number 123 on the data stack under the xaddr of the terminating space in the string " 123 " .

COP.ID (-- n)
Returns the interrupt identity code for the computer operating properly (COP) interrupt. Used as an argument for ATTACH.
Pronunciation: "cop-i-d"

COPY.ARRAY (array.xpfa1\array.xpfa2 --)
Dimensions the destination array specified by array.xpfa2 and copies the contents of the source array specified by array.xpfa1 into the destination. The source and destination can be in different heaps.

COPY.MATRIX (matrix.xpfa1\matrix.xpfa2 --)
Dimensions the destination matrix specified by matrix.xpfa2 and copies the contents of the source matrix specified by matrix.xpfa1 into the destination. The source and destination can be in different heaps.

COUNT (x$addr -- xaddr\cnt | xaddr = x$addr+1)
Unpacks the counted string whose count is stored at x$addr and whose first character is stored at x$addr+1. Returns the extended address of the first character under the count. The string may cross a page boundary.

COUNT.TYPE (x$addr --)
Unpacks count from x$addr on page and types the string. COUNT.TYPE is equivalent to COUNT TYPE
Attributes: M

CR (--)
Causes subsequent output to appear at the beginning of the next line by emitting a carriage return (ascii 13) followed by a line feed (ascii 10).
Pronunciation: "c-r" Attributes: M

CR.BEFORE.MSG (-- xaddr)
A user variable that contains a flag. If the flag is false (the default condition), system warnings and error messages are printed without first emitting carriage return/linefeed characters. This ensures smooth downloads if the host terminal is using the suggested technique of waiting for a linefeed character (ascii 10) before sending each new line of source code to the QED Board (see the Program Development Techniques chapter). If the CR.BEFORE.MSG flag is true, the error and warning messages are printed on a separate line, but the leading carriage return/linefeed that is emitted may cause the host terminal to send the next line of source code before the QED Board is capable of responding to it. Thus it is recommended that CR.BEFORE.MSG be kept in its default OFF state while downloading to the QED Board.
Pronunciation: "carriage-return-before-message" Attributes: U

CREATE (<name> --)

Adds a new header for <name> to the names area. Executes BL WORD to parse the next space-delimited word <name> from the input stream. Converts the parsed string to upper case letters and searches the dictionary via (FIND) to check for uniqueness. A warning is issued if <name> is not unique.
Implementation detail: Creates a new header for <name> starting at the address pointed to by NP, links the header to the CURRENT vocabulary, and initializes the code field address in the header to the current value of DP. Updates the CURRENT vocabulary xhandle to point to the xnfa of <name> and updates NP to point to the byte after <name>'s header. The number of characters saved in the header is the lesser of the value in WIDTH or the actual number of characters in <name>, to a maximum of 31 characters. If locals are compiling, all characters are saved in the header to avoid non-uniqueness of local variables. An abort error occurs if the header cannot be stored (e.g., if NP does not point to RAM). If WIDTH is less than or equal to 1, CREATE resets WIDTH to 2.
Attributes: D

CREATE.RAMP (start_speed\end_speed\accel\ticks_per_sec\start_ramp_addr\speeds_per_ramp
-- steps_in_ramp | all parameters are integers)

Writes speed_per_ramp +1 entries into the RAMP.ARRAY starting at the specified start_ramp_addr to attain the specified starting and ending speeds and acceleration (or deceleration). Returns the number of steps in the created ramp. start_speed, end_speed and acceleration are all interpreted as positive numbers. Speeds are in units of steps per second if the motor is configured for full stepping, or halfsteps per second if the motor is configured for half stepping. The acceleration is in units of (half) steps per second per second. Speeds are clamped to the attainable range (between 0 and ticks_per_second), and the acceleration is clamped such that a maximum of 10 seconds is spent at any one transient speed in a ramp. Each ramp entry comprises a step_limit which specifies the number of steps to be taken at the speed, and a duty_cycle which specifies the speed (see the glossary entry for SPEED.TO.DUTY). If the specified speeds_per_ramp = 0, this function simply writes a "final" speed by setting the step_limit to 0. For non-zero speeds_per_ramp, this routine writes the specified number of ramp entries, plus an additional entry at the final speed with the step_limit set to 0 which tells the STEP.MANAGER that this is the final speed in the ramp. Note that higher level calling routines can write over the final speed, or concatenate two ramps to achieve a speed profile that ramps up to a steady speed for a specified number of steps, and then smoothly ramps down to a stopped state. See the high level source file steppers.4th in the Demos_and_Drivers directory of the distribution.

CURRENT (-- xaddr)

A user variable that contains a 32-bit xhandle which in turn contains the xnfa of the top word in the vocabulary to which new definitions are added by CREATE. Thus CURRENT X@ returns the xhandle of the definitions vocabulary, and CURRENT X@ X@ is equivalent to LATEST, returning the xnfa of the latest word defined. In short, the contents of CURRENT determine the vocabulary to which new words are added. See CREATE and CONTEXT.
Attributes: U

CURRENT.HEAP (-- xaddr)

A user variable that holds the 32-bit extended address that specifies the end of the current heap. Executing CURRENT.HEAP X@ places the xaddress of the last+1 byte in the current heap on the data stack; the other heap control variables are stored just below this address in the heap. See IS.HEAP
Pronunciation: "current-heap" Attributes: U

CUSTOM.ABORT (-- xaddr)
A user variable that contains a flag. If the flag is TRUE, the abort routine whose xcfa is in UABORT is executed each time that ABORT is called. If the flag is FALSE, ABORT executes the default (ABORT) routine. See ABORT, (ABORT), and UABORT.
Attributes: U

CUSTOM.ERROR (-- xaddr)
A user variable that contains a flag. If the flag is TRUE, the error routine whose xcfa is in UERROR is executed in response to every system error. If CUSTOM.ERROR is FALSE, all system errors call the default (ERROR) routine. See (ERROR) and UERROR.

Attributes: U

D+ (d1\d2 -- d3)
Adds two signed double numbers d1 and d2 giving the signed double number result d3.
Pronunciation: "d-plus"

D- (d1\d2 -- d3 | d3 = d1 - d2)
Subtracts two signed double numbers d1 and d2 giving the signed double number result d3.
Pronunciation: "d-minus"

D. (wd --)
Prints wd with no leading spaces and 1 trailing space. If the number base is decimal, wd is printed as a signed number in the range -2,147,483,648 to +2,147,483,647. In other bases wd is printed as an unsigned positive number.
Pronunciation: "d-dot" Attributes: M, S

D.INSTANCE: (u <name> -- | u is the size of the structure)
Removes <name> from the input stream and creates a structure instance called <name>, and allocates u bytes in the definitions area starting at HERE for the structure instance (the "D" in "D.INSTANCE:" refers to the Definitions area where the instance is allocated). Compare with V.INSTANCE:. When <name> is executed, the extended base address of the allocated structure instance is placed on the data stack. Typical use:

```
    <structure.name> D.INSTANCE: <name>
```

where <structure.name> was defined using

```
    STRUCTURE.BEGIN: <structure.name>
            ...
    STRUCTURE.END
```

Executing <structure.name> leaves the structure size u on the stack, and D.INSTANCE: <name> allocates and names the instance. Executing

```
    SIZE.OF <name>
```

places the allocated size of the instance on the stack. Note that the instance may cross page boundaries, and may increment the dictionary pointer DP so that it points to a new page.
Pronunciation: "d-instance" Attributes: D

D.OVER.N (d\n -- d\n\d)
Copies the double number located under the top data stack cell to the top of the data stack.
Pronunciation: "d-over-n"

D.R (wd\+byte -- | +byte is field width)
Prints wd right-justified in a field of +byte characters. If +byte is less than or equal to the number of characters to be printed, the number is printed with no extra spaces. If the number base is decimal, wd is printed as a signed number in the range -2,147,483,648 to +2,147,483,647. In other bases w is printed as an unsigned positive number. To print wd as a positive unsigned number in decimal base, use UD.R
Pronunciation: "d-dot-r" Attributes: M, S

D0< > (wd -- flag)
Flag is TRUE if double number wd is not equal to zero, and FALSE otherwise.
Pronunciation: "d-zero-not-equal"

D0= (wd -- flag)
Flag is TRUE if double number wd is equal to zero and FALSE otherwise.
Pronunciation: "d-zero-equal"

D2* (d1 -- d2 | d2 = d1 * 2)
Multiplies signed double number d1 by 2 giving d2. Overflow errors are not checked.
Pronunciation: "d-two-star"

D2/ (d1 -- d2 | d2 = d1 / 2)
Divides signed double number d1 by 2 giving d2.
Pronunciation: "d-two-slash"

D< (d1\d2 -- flag)
Flag is TRUE if the signed double number d1 is less than the signed double number d2 and FALSE otherwise.
Pronunciation: "d-less-than"

D< > (wd1\wd2 -- flag)
Flag is TRUE if the two double numbers are not equal and FALSE otherwise.
Pronunciation: "d-not-equal"

D= (wd1\wd2 -- flag)
Flag is TRUE if the two double numbers are equal and FALSE otherwise.
Pronunciation: "d-equal"

D> (d1\d2 -- flag)
Flag is TRUE if the signed double number d1 is greater than the signed double number d2 and FALSE otherwise.

Pronunciation: "d-greater-than"

D>R (d --)
Return Stack: (R: -- d)
Transfers the top double number on the data stack to the return stack.
Pronunciation: "d-to-r" Attributes: C

D>S (d -- n)
Converts the double number d to the single number n by dropping the most significant cell of d. There is an unchecked error if d cannot be represented by a 16-bit signed integer.
Pronunciation: "d-to-s"

D>S? (d -- [d\2] or [n\1])
If possible, converts double number d to a single number n and leaves n on the stack under a 1 flag. Otherwise leaves double number d on the stack under a 2 flag.
Pronunciation: "d-to-s-question"

DABS (d1 -- +d2)
Replaces double precision signed number d1 with its absolute value +d2. If d1 is positive, +d2 = d1. If d1 is negative, +d2 is the negative of d1.
Pronunciation: "d-abs"

DEALLOCATED (xpfa --)
De-allocates the heap memory associated with the data structure having the specified parameter field address xpfa. The xpfa is typically associated with a word defined by H.INSTANCE:. Typical use:

```
     size.of.heap.item  H.INSTANCE:  <name>
     SIZE.OF <name>   ' <name>  ALLOCATED
                ...
      ' <name>  DEALLOCATED
```

DEBUG (-- xaddr)
A user variable that holds a flag. If true, this flag enables error checking by the word NEEDED and by some array and matrix routines. It also enables the trace printout of words that are compiled while TRACE is ON.
Attributes: U

DECIMAL (--)
Set the numeric conversion base to ten by storing decimal 10 into the user variable BASE.

DEFAULT.PAGE (-- page | page = 0)
Places a zero onto the data stack; this represents the default page assigned to the common memory (i.e., addresses above 0x8000).

DEFAULT.REGISTER.INITS (--)
Undoes the effect of the INSTALL.REGISTER.INITS command.

Implementation detail: sets the contents of location AE06H in EEPROM to 0xFF to ensure that default initializations will be used after subsequent resets. The default register initializations are:

Register Name	Register Address	Default Value
OPTION	0x8039	0x33
TMSK2	0x8024	0x02
BPROT	0x8035	0x10
BAUD	0x802B	0x31

DEFAULT.TRACE.ACTION (--)

Installs NO.OP, a do-nothing word, as the trace action. Equivalent to

```
CFA.FOR  NO.OP  IS.TRACE.ACTION
```

See IS.TRACE.ACTION.

DEFINITIONS (--)

Stores the contents of CONTEXT into CURRENT so that the search vocabulary is also the vocabulary to which new definitions are appended.
Attributes: I

DELETED (array.xpfa -- | used for arrays and matrices)

De-allocates the heap space assigned to the specified array or matrix, and clears the parameter field to indicate that the data structure is no longer dimensioned. Use as:

```
' <name> DELETED
```

See DIMENSIONED, DIMMED.

DEPTH (-- +n | +n = stack depth)

+n is the number of cells on the data stack before +n was placed on the stack.

DFIXX (r -- d)

d is the 32 bit integer closest to the floating point number r. See FIXX.
Pronunciation: "d-fix" Attributes: S

DFLOT (d -- r)

Converts the double number d to the nearest floating point number r. Note that there is a potential loss of resolution in this conversion, since d is represented by 32 significant bits, while r has a 16-bit mantissa. See FLOT.
Pronunciation: "d-f-lot" Attributes: S

DIGIT (char -- [n\-1] or [0])

Converts ascii char to binary digit n in the current number base and leaves n on the stack under a true flag. If char cannot be converted to a valid digit, returns a false flag.

DIM.CONSTANT.ARRAY: (u1\...\uN\N\n <name> -- | N=#dim, n=bytes/element)

Removes <name> from the input stream and creates and dimensions an array in the definitions area. This is useful for building lookup tables that will reside in flash memory after the application is finished and write-protected. u1, u2, ...uN specify the number of elements in each dimension, N specifies the number of dimensions, and n specifies the number of bytes per element. <name> behaves exactly as an array does; its stack picture is

(indices -- xaddr)

and an element xaddress is also returned upon execution of the command

indices ' <name> []

DIM.CONSTANT.ARRAY: creates a header for <name> in the names area of the dictionary. It creates and initializes a parameter field and a "handle" (to mimic a heap handle) in the definitions area, and allots the required number of bytes for the array in the definitions area. ABORTs if #dim is invalid (<1 or >MAX#DIMENSIONS). The array may cross page boundaries, and may increment the dictionary pointer DP so that it points to a new page.

Example of use:

To define and dimension a constant array to have 2 dimensions (3 rows and 4 columns) with 6 bytes per element, execute:

3 4 2 6 DIM.CONSTANT.ARRAY: <name>

Restrictions: In general, constant arrays should be dimensioned only once at the time of creation; redimensioning to a larger size could write over other routines in the dictionary and cause a crash.

Pronunciation: "dim-constant-array" Attributes: D

DIM.CONSTANT.MATRIX: (#rows\#cols <name> --)

Removes <name> from the input stream and creates and dimensions a matrix in the definitions area. This is useful for building lookup tables that will reside in flash memory after the application is finished and write-protected. <name> behaves exactly as a matrix does; its stack picture is:

(row#\col# -- xaddr)

and an element xaddress is also returned upon execution of the command

row# col# ' <name> M[]

DIM.CONSTANT.MATRIX: creates a header for <name> in the names area of the dictionary. It creates and initializes a parameter field and a "handle" (to mimic a heap handle) in the definitions area, and allots the required number of bytes for the matrix in the definitions area. The matrix is assigned #rows rows and #cols columns, 2 dimensions, and 4 bytes/element. The matrix may cross page boundaries, and may increment the dictionary pointer DP so that it points to a new page.

Example of use:

To define and dimension a constant matrix to have 3 rows and 4 columns, execute:

3 4 DIM.CONSTANT.MATRIX: <name>

Restrictions: Care must be used when using matrix operators that assume that the matrix resides in the current heap. In general, constant matrices should be dimensioned only once at the time of creation; redimensioning to a larger size could write over other routines in the dictionary and cause a crash.

Pronunciation: "dim-constant-matrix"Attributes: D

DIMENSIONED (u1\...\uN\N\n\array.xpfa --)

Dimensions the array specified by array.xpfa. u1...uN specify the number of elements in each dimension, N specifies the number of dimensions, and n specifies the number of bytes per element. DIMENSIONED executes DELETED to de-allocate any heap space previously allocated to the array, and then writes the dimensioning information into the array's parameter field and allocates the required number of bytes in the heap. ABORTs if there is not enough heap space or if N is invalid (N must be between 1 and MAX#DIMENSIONS, inclusive).

Example of use:

To define and dimension an array to have 2 dimensions (3 rows and 4 columns) with 6 bytes per element, execute:

```
ARRAY:   <name>
3 4 2 6   ' <name>  DIMENSIONED
```

DIMMED (#rows\#cols\matrix.xpfa --)

Dimensions the matrix specified by matrix.xpfa to have #rows rows and #cols columns. The number of dimensions is 2, and there are 4 bytes per element (i.e., the size of a floating point number). DIMMED executes DELETED to de-allocate any heap space previously allocated to the matrix, and then writes the dimensioning information into the parameter field and allocates the required number of bytes in the heap. ABORTs if there is not enough heap space, or (if DEBUG is ON) if the number of rows or columns is greater than 16,383. For example, to define and dimension a matrix to have 3 rows and 4 columns, execute:

```
MATRIX: <name>
3 4 ' <name>  DIMMED
```

DIN (-- wd)

Compile Time: (<name> --)

DIN removes the next word from the input stream, converts it to a 32-bit double number wd in the current number base, and executes 2LITERAL which leaves the number on the stack if QED-Forth is in execution mode, or compiles it as a literal in the current definition if QED-Forth is in compilation mode. If DIN is not used, 32-bit numbers in the input stream are truncated to 16 bits. An error is issued if <name> cannot be converted to a valid number in the current number base. Typical use:

```
HEX DIN 12345678       ( -- 5678 \ 1234 )
D.                     12345678 ok
```

Pronunciation: "d-in"

DINT (r -- d)

d is the double number representation of the integer part of floating point number r. See INT.PART.

Pronunciation: "d-int"

DINT.FLOOR (r -- d)

d is the greatest double number less than or equal to r. See INT.FLOOR.

Pronunciation: "d-int-floor"

DISABLE.INTERRUPTS (--)

Sets the interrupt mask bit (the "I bit") in the condition code register to globally disable interrupts.

DISABLE.SERIAL2 (--)

Disables the secondary serial port (serial2) which is supported by QED-Forth's software UART. Implementation detail: Locally disables the serial2 output interrupt OC4 and disconnects the pin control logic associated with the PA4 output. Locally disables the serial2 input interrupt IC4/OC5. Clears the resource variable SERIAL2.RESOURCE to 0\0.

Pronunciation: "disable-serial-two"

DISK.RESOURCE (-- xresource)

A resource variable associated with the mass memory block buffers. After initialization to 0\0, should be accessed only by the words GET ?GET and RELEASE. Initialized to 0\0 by the system startup software. See RESOURCE.VARIABLE:.

DISPLAY.BUFFER (-- xaddr)

Returns the extended base address of the buffer that holds the display data. UPDATE.DISPLAY.LINE and UPDATE.DISPLAY write the contents of this buffer to the LCD display. If a character display is in use, xaddr is the base address of an 80 character buffer in the system RAM. If a graphics display is in use, xaddr is the starting address of the array associated with GARRAY.XPFA. Each byte in this buffer represents a character position or graphical byte on the LCD display. To display characters on the LCD display, simply write the desired ascii characters or graphical data into this buffer and execute UPDATE.DISPLAY.LINE or UPDATE.DISPLAY. UPDATE.DISPLAY.LINE causes the contents of a specified line in DISPLAY.BUFFER to be written to the corresponding line of the display. UPDATE.DISPLAY causes the contents of all lines in DISPLAY.BUFFER to be written to the corresponding lines of the display. See $>DISPLAY and BUFFER.POSITION.

DISPLAY.HEAP (-- xaddr)

Returns the extended address that points to the top of the heap containing the graphics array. The default display heap is located at 3000 to 0x45FF on page 0x0F. Caution: adding items to the DISPLAY.HEAP is not recommended; if there is not enough room for the GARRAY, INIT.DISPLAY will not dimension it. However, if you must dimension additional heap items in this heap, execute the following commands:

```
CURRENT.HEAP X@          \ save the prior heap specifier on dstack
DISPLAY.HEAP CURRENT.HEAP X!          \ display.heap is the heap
< dimension or delete heap items here>
CURRENT.HEAP X!          \ restore prior heap specifier
```

DISPLAY.OPTIONS (flag1\flag2\flag3\flag4 --)

Sets the display and cursor options on the LCD display. The meanings of the input flags are as follows:

flag1 = display.enabled?
flag2 = cursor.on?
flag3 = cursor.blinking?
flag4 = text.mode?

If flag1 is true, the contents of the display are visible; if false, the display appears blank. If flag2 is true, the cursor is on (typically an underscore character); if false, the cursor is off. If flag3 is true, the cursor blinks (typically a flashing box the size of a single character); if false, the cursor blink is turned off. If flag4 is true, the display is operating in "text mode"; if false, it is operating in "graphics mode". (If a Toshiba graphics display is in use, flag4 can take on the additional value 1, meaning that both text and graphics modes are enabled. In this case, you must set different home locations for the text and graphics regions.) Note that graphics mode should only be specified if a graphics display is in use; see IS.DISPLAY. Note also that the cursor is never visible in graphics mode. INIT.DISPLAY (which is executed upon each reset or restart) leaves the display enabled with the cursor off and cursor blink off and the "home" location in the upper left corner at display ram address 0. Implementation detail: In addition to writing the appropriate command byte to the display, DISPLAY.OPTIONS stores the command byte in a headerless system variable called *PRIOR.CURSOR.STATE. This variable is referenced by UPDATE.DISPLAY.LINE and UPDATE.DISPLAY to blank the cursor during updates to character displays (to prevent annoying flickering) and restore it to its prior state after the update is complete. It is also used by LINES/DISPLAY to infer whether the display is being operated in text mode or graphics mode, which in turn determines whether LINES/DISPLAY reports the number of character lines or the number of pixel lines in the display. This routine intermittently disables interrupts for 28 cycles (7 microseconds) per command byte written to the display.

DMAX (d1\d2 -- [d1] or [d2] | retains the greater of d1 and d2)
Retains the greater of two signed double numbers and drops the other.
Pronunciation: "d-max"

DMIN (d1\d2 -- [d1] or [d2] | retains the lesser of d1 and d2)
Retains the lesser of the two signed double numbers and drops the other.
Pronunciation: "d-min"

DNEGATE (d1 -- d2 | d2 = two's complement of d1)
Negates signed double number d1 to yield d2. The negative (two's complement) is computed by inverting all of the bits in d1 and adding a 32-bit 1 to the result.
Pronunciation: "d-negate"

DO (w1\w2 -- | w1 = limit, w2 = starting index)
Return Stack: (R: -- w1\w2)
Used inside a colon definition to mark the beginning of a counted loop structure that is terminated by LOOP or +LOOP. Sets up loop parameters on the return stack with w1 as the limit and w2 as the starting index. Because the loop parameters are maintained on the return stack, caution must be exercised when using the operators >R and R> inside a loop. DO...LOOPs may be nested as long as each DO is matched with a corresponding LOOP or +LOOP in the same definition as DO. w1 and w2 may either be a pair of signed integers or a pair of unsigned integers. If w1 = w2, the loop does not execute, and control is immediately passed to the word following LOOP or +LOOP. DO may only be used within a definition. Use as:

```
    w1 w2 DO ... LOOP
```

or

```
    w1 w2 DO ...  n +LOOP
```

An error is issued if DO is not properly paired with LOOP or +LOOP inside a definition.
See also LOOP +LOOP I J K I' LEAVE
Attributes: C, I

DOES> (--)
Used in a high level defining word to mark the beginning of the specification of the run-time action of the child words. Use as:

```
    : <namex>
            <DBUILDS   compile time action
            DOES>  run time action
    ;
```

or as

```
    : <namex>
```

```
        <VBUILDS   compile time action
        DOES>  run time action
    ;
```

where <namex> is referred to as a "defining word". Executing the statement

```
    <namex> <child's.name>
```

defines the child word. The code after <DBUILDS or <VBUILDS specifies the action to be taken while defining the child word, and the code after DOES> specifies the action to be taken when the child word executes.
The default run-time action of DOES> is to leave the extended parameter field address of the child word on the data stack. Thus, the code between DOES> and ; should expect the xpfa on the stack when the child executes. See the definitions of <DBUILDS and <VBUILDS for examples of use.
Pronunciation: "does"

DOT.PRODUCT (xvaddr1\sep1\xvaddr2\sep2\d.#el -- r)
Returns the dot product r of the vectors specified by xvaddr1\sep1\d.#el and xvaddr2\sep2\d.#el. The dot product is calculated by multiplying the corresponding elements of the two vectors and summing the result into a floating point accumulator. The two vector specifications may refer to the same vector.
Attributes: S

DOUBLE-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a double number field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "double" Attributes: D

DOUBLE: (<name> --)
Defines a 32-bit self-fetching variable. Removes <name> from the input stream and creates a child word (a self-fetching variable) called <name> and allots 4 bytes in the variable area as the parameter field where the self-fetching variable's value is stored. When <name> is executed it leaves its value (a 32-bit number) on the stack. Thus <name> behaves like a 2constant when executed. Unlike a 2constant, its parameter field is in the variable area and so can always be modified. The TO command is used to store a value into the self-fetching variable. In general, code using self-fetching variables runs faster than does similar code that uses standard variables because the fetch and store operations are integrated into the action of the variable. Use as:

```
    DOUBLE: <name>
```

Pronunciation: "double-colon" Attributes: D

DOUBLES-> (u1\u2 <name> -- u3)
Adds a named member to the structure being defined and reserves room for u2 double numbers in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining

word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "doubles" Attributes: D

DOWNLOAD.MAP (--)
Sets a flag in EEPROM and changes the state of a latch in the onboard PALs to put the download memory map into effect on flash-equipped QED-Flash Boards. After execution of this routine, and upon each subsequent reset or restart, pages 4, 5, and 6 are addressed in the S2 RAM, and pages 1, 2, and 3 are addressed in the S1 flash memory. This allows code (and Forth names) to be compiled into RAM on pages 4, 5 and 6 and then transferred to flash using the PAGE.TO.FLASH function. To establish the standard memory map, see the glossary entry for STANDARD.MAP. Note that the standard map is active after a "factory cleanup" operation.

DP (-- xaddr)
User variable that contains the 32-bit definitions pointer. The contents of DP are placed on the stack by HERE and are modified by ALLOT. The command DP X@ is equivalent to HERE; it yields the xaddr of the next available dictionary location. The command DP @ is equivalent to DPAGE; it yields the page of the definitions area.
Pronunciation: "d-p" Attributes: U

DPAGE (-- page)
Returns the page of the definitions area of the dictionary. Equivalent to DP @
Pronunciation: "d-page" Attributes: U

DPICK (d\wn-1\...w1\w0\+n -- d\wn-1\...\w1\w0\d | 0 <= +n <= 255)
Copies the double number whose most significant cell is the nth item on the stack (0-based, not including n) to the top of the stack. An unchecked error occurs if there are fewer than +n+2 cells on the data stack. 0 DPICK is equivalent to 2DUP, 1 DPICK is equivalent to D.OVER.N, 2 DPICK is equivalent to 2OVER.
Pronunciation: "d-pick"

DR> (-- d)
Return Stack: (R: d --)
Transfers the top double number on the return stack to the data stack.
Pronunciation: "d-r-from" Attributes: C

DR>DROP (--)
Return Stack: (R: d --)
Removes the top double number on the return stack.
Pronunciation: "d-r-from-drop" Attributes: C

DR@ (-- d)
Return Stack: (R: d -- d)
Copies the top double number on the return stack to the data stack.
Pronunciation: "d-r-fetch" Attributes: C

DRANGE (d1\d2\d3 -- d1\flag)

Flag is TRUE if d1 is greater than or equal to d2 and less than or equal to d3. Otherwise flag is FALSE.
Pronunciation: "d-range"

DROP (w --)
Drops the top cell from the stack.

DSCALE (d1\n -- d2)
Arithmetically (i.e., preserving sign) shifts double number d1 by n bit places to yield signed double number result d2. If n is positive, d1 is shifted left; if n is negative, d1 is shifted right. The absolute value of n determines the number of bits of shifting. For example, 1 DSCALE is equivalent to D2* and -1 DSCALE is equivalent to D2/ . There is an unchecked error if the absolute value of n is greater than 31.
Pronunciation: "d-scale"

DU< (ud1\ud2 -- flag)
Flag is TRUE if the unsigned double number ud1 is less than the unsigned double number ud2.
Pronunciation: "d-u-less-than"

DU> (ud1\ud2 -- flag)
Flag is TRUE if the unsigned double number ud1 is greater than the unsigned double number d2.
Pronunciation: "d-u-greater-than"

DUMP (xaddr\u -- | xaddr = start address, u = number of bytes)
Displays the contents of u bytes starting at the specified xaddr. The contents are dumped as hexadecimal bytes regardless of the current number base, and the ascii equivalent contents are also displayed. For example, to display 0x40 bytes starting at address 0x1000\1, execute:

```
HEX  1000  1  40  DUMP
```

and to display the last 0x10 bytes on page 1 and the first 0x20 bytes on page 2, type:

```
        7FF0  1  30  DUMP
```

DUMP calls the word PAUSE.ON.KEY, so the dump responds to XON/XOFF handshaking and can be aborted by typing a carriage return; see PAUSE.ON.KEY.
Attributes: M, S

DUMP.INTEL (xaddr1\addr2\u --)
xaddr1 is the location of the first byte to be dumped, addr2 specifies the starting address reported in the dump, and u is the number of bytes to be dumped. Dumps the contents of u bytes starting at xaddr using the standard ascii Intel hex format which is useful for transferring data between devices. The line format is:

```
:{#bytes}{reported.addr}{00}{byte}{byte} ...{byte}{checksum}
```

All numbers are in hexadecimal base. Each line starts with a : character, followed by a 2-digit number of bytes (20, indicating that the contents of 0x20 bytes are displayed per line), followed by a 4-digit starting address for the line, followed by 00, followed by the contents of the memory locations (2 hex digits per byte), and concluding with a checksum followed by a carriage return/linefeed. The checksum is calculated by summing each of the bytes on the line into an 8-bit accumulator and negating (two's complementing) the result. The hex dump ends with the line

```
:00000001FF
```

For example, to dump 0x40 bytes starting at QED Board address 0x1000\1 so that the bytes reside at the beginning of a target memory device, execute:

```
HEX  1000  01  0000   40  DUMP.INTEL
```

which specifies 0x1000\1 as the starting address, 0000 as the reported base address in the memory device, and 0x40 as the number of bytes to be dumped. To dump the last 0x20 bytes on page 1 and the first 0x40 bytes on page 2 so that they reside at locations 0x7FE0 through 0x803F in the target memory device, execute

```
7FE0  1  7FE0   60   DUMP.INTEL
```

The complementary word RECEIVE.HEX loads QED memory starting at any location based on a received Intel or Motorola hex file. DUMP.INTEL calls the word PAUSE.ON.KEY, so the dump responds to XON/XOFF handshaking and can be aborted by typing a carriage return. See DUMP.S1, DUMP.S2, RECEIVE.HEX and PAUSE.ON.KEY.

Attributes: M, S

DUMP.REGISTERS (-- xaddr)

A user variable that holds a flag. If the flag is true, a definition that has been compiled with TRACE ON prints the contents of the registers before each instruction during a trace. If the flag is false, the register contents are not printed during a trace.

DUMP.S1 (xaddr1\addr2\u --)

xaddr1 is the location of the first byte to be dumped, addr2 specifies the starting address reported in the dump, and u is the number of bytes to be dumped. Dumps the contents of u bytes starting at xaddr using the standard ascii Motorola S1 hex format which is useful for transferring data between devices. Motorola S1 records report 16 bit addresses. (To report full 24 bit addresses, see DUMP.S2.) Outputs an S0 header record which is

```
S00900004845414445524D
```

then as many S1 data records as required, followed by an S9 termination record which is

```
S9030000FC
```

The Motorola S1 hex line format is:

```
S1{#bytes}{16bit.reported.addr}{byte}...{byte}{chksum}
```

All numbers are in hexadecimal base. Each line starts with a the record type (S1 in this case), followed by a 2-digit number of bytes (23, which equals 0x20 bytes per line plus 3 bytes for the reported address and checksum), followed by a 4-digit starting address for the line, followed by the contents of the memory locations (2 hex digits per byte), and concluding with a checksum. The checksum is calculated by summing each of the bytes on the line (excluding the record type) into an 8-bit accumulator and (one's) complementing the result.

For example, to dump 0x40 bytes starting at QED Board address 0x1000\1 so that the bytes reside at the beginning of a target memory device, execute:

```
HEX  1000  01   0000   40   DUMP.S1
```

which specifies 0x1000\1 as the starting address, 0000 as the reported base address in the memory device, and 0x40 as the number of bytes to be dumped. To dump the last 0x20 bytes on page 1 and the first 0x40 bytes on page 2 so that they reside at locations 0x7FE0 through 0x803F in the target memory device, execute:

```
    7FE0  1   7FE0    60    DUMP.S1
```

The complementary word RECEIVE.HEX loads QED memory starting at any location based on a received Motorola or Intel hex file. DUMP.S1 calls the word PAUSE.ON.KEY, so the dump responds to XON/XOFF handshaking and can be aborted by typing a carriage return. See DUMP.S2, DUMP.INTEL, RECEIVE.HEX and PAUSE.ON.KEY.
Attributes: M, S

DUMP.S2 (xaddr1\d\u --)
xaddr1 is the location of the first byte to be dumped, double number d specifies the 24 bit starting address reported in the dump, and u is the number of bytes to be dumped. Dumps the contents of u bytes starting at xaddr1 using the standard ascii Motorola S2 hex format which is useful for burning flash memory chips and transferring data between devices. Motorola S2 records report 24 bit addresses which are useful in capturing and transferring complete application programs to/from flash memory. (To report 16 bit addresses, see DUMP.S1.) Dumps an S0 header record which is

```
    S00900004845414445524D
```

then as many S2 data records as required, followed by an S9 termination record which is

```
    S9030000FC
```

The Motorola S2 hex line format is:

```
    S2{#bytes}{24bit.reported.addr}{byte}...{byte}{chksum}
```

All numbers are in hexadecimal base. Each line starts with a the record type (S2 in this case), followed by a 2-digit number of bytes (24, which equals 0x20 byte per line plus 4 bytes for the reported address and checksum), followed by a 6-digit starting address for the line, followed by the contents of the memory locations (2 hex digits per byte), and concluding with a checksum. The checksum is calculated by summing each of the bytes on the line (excluding the record type) into an 8-bit accumulator and (one's) complementing the result. DUMP.S2 calls the word PAUSE.ON.KEY, so the dump responds to XON/XOFF handshaking and can be aborted by typing a carriage return. See DUMP.S1, DUMP.INTEL, RECEIVE.HEX and PAUSE.ON.KEY.
Example of use: Assume that you have created an application program in pages 4, 5, and 6, and used PRIORITY.AUTOSTART to configure a flash-based autostart vector so that the application runs automatically upon each power-up and restart. To dump a complete application program that resides on pages 4, 5 and 6, so that the bytes reside at the beginning of a flash memory device, execute:

```
    HEX
    0000 04   DIN  000000   8000  DUMP.S2
    0000 05   DIN  008000   8000  DUMP.S2
    0000 06   DIN  010000   8000  DUMP.S2
```

Now you can edit the resulting file, concatenate the 3 dumps into 1 large S-record by removing all but the first and last S0 (header) and S9 (termination) records, and re-save the file. To transfer the application to a new QED board, simply execute

```
    DOWNLOAD.MAP
    0 4 RECEIVE.HEX       <send the captured file>
    4 PAGE.TO.FLASH
    5 PAGE.TO.FLASH
    6 PAGE.TO.FLASH
    STANDARD.MAP
```

This is a time-effective method of mass producing QED-based products running a "turnkeyed" autostart program.
Attributes: M, S

DUP (w -- w\w)
Duplicates the top cell of the data stack.
Pronunciation: "dupe"

DUP.HEAP.ITEM (xhandle1 -- [xhandle2] or [0\0])
Given the 32-bit handle xhandle1 of a source heap item, creates a duplicate heap item with identical contents in the same heap and returns its handle xhandle2. Returns 0\0 if xhandle1 is not a valid handle or if there is insufficient memory in the heap. To copy a heap item into a different heap, use TRANSFER.HEAP.ITEM.
Pronunciation: "dupe-heap-item"

DUP>R (w -- w)
Return Stack: (R: -- w)
Copies the top cell on the data stack to the return stack.
Pronunciation: "dupe-to-r" Attributes: C

ELSE (--)
Used in a colon definition to mark the beginning of the "else portion" of an IF ... ELSE ... ENDIF conditional structure (ENDIF and THEN are synonyms). Use as:

```
flag IF   words to execute if flag is true
ELSE           words to execute if flag is false
ENDIF
```

If the flag passed to IF is true, when ELSE is encountered control is passed to the word following ENDIF. If the flag passed to IF is FALSE, control is immediately passed to the word following ELSE.
Attributes: C, I

EMIT (char --)
Displays char by sending it via the serial I/O port. EMIT is a vectored routine that executes the routine whose xcfa is installed in the user variable UEMIT. The default installed routine called is EMIT1 which sends the character via the primary serial port (supported by the 68HC11's hardware UART). EMIT2 may be installed in UEMIT by USE.SERIAL2 or SERIAL2.AT.STARTUP; EMIT2 sends the character via the secondary serial port (supported by QED Forth's software UART and using pins PA3 and PA4). See EMIT1 and EMIT2.
Attributes: M, U

EMIT1 (char --)
Displays a character by sending it via the primary serial port (serial1) associated with the 68HC11's on-chip hardware UART. Before sending the character, EMIT1 waits (if necessary) for the previous character to be sent, and executes PAUSE while waiting. The most significant byte of the input data stack cell is ignored. EMIT1 is the default EMIT routine installed in the UEMIT user variable after the special cleanup mode is invoked or if SERIAL1.AT.STARTUP has been executed. If the value in SERIAL.ACCESS is RELEASE.AFTER.LINE, EMIT1 does not GET or RELEASE the SERIAL1.RESOURCE. If SERIAL.ACCESS contains RELEASE.ALWAYS, EMIT1

GETs and RELEASEs the SERIAL1.RESOURCE. If SERIAL.ACCESS contains RELEASE.NEVER, EMIT1 GETs but does not RELEASE the SERIAL1.RESOURCE. See EMIT, UEMIT, EMIT2, SERIAL.ACCESS.
Pronunciation: "emit-one" Attributes: M

EMIT2 (char --)
Writes the specified ascii character to the output buffer of the secondary serial port (serial2) for subsequent transmission. The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). If the serial2 transmitter is idle (and if the serial2 port and its interrupts have been properly initialized) then the character is transmitted immediately. Otherwise the character will be transmitted after the prior characters in the output buffer are transmitted. If the 80 character output buffer is full when EMIT2 is executed, EMIT2 PAUSEs and waits until room becomes available in the buffer (as a result of a character being sent out). The most significant byte of the input data stack cell is ignored. EMIT2 can be made the default EMIT routine installed in the UEMIT user variable after each reset or restart by executing SERIAL2.AT.STARTUP. If the value in SERIAL.ACCESS is RELEASE.AFTER.LINE, EMIT2 does not GET or RELEASE the SERIAL2.RESOURCE. If SERIAL.ACCESS contains RELEASE.ALWAYS, EMIT2 GETs and RELEASEs the SERIAL2.RESOURCE. If SERIAL.ACCESS contains RELEASE.NEVER, EMIT2 GETs but does not RELEASE the SERIAL2.RESOURCE. See EMIT, UEMIT, EMIT1, SERIAL.ACCESS.
Pronunciation: "emit-two" Attributes: M

ENABLE.INTERRUPTS (--)
Clears the interrupt mask bit (the "I bit") in the condition code register to globally enable interrupts.

END-CODE (sys -- | balances CODE)
A synonym for END.CODE
Pronunciation: "end-code"

END.CODE (sys -- | balances CODE)
Terminates an assembly code definition which started with CODE. Executes SMUDGE so that the header created by CODE can be found in the dictionary. Checks to make sure that no extra items were left on the stack during the definition process. Stores the contents of CURRENT into CONTEXT so that ASSEMBLER is no longer the search vocabulary.
Pronunciation: "end-code"

ENDCASE (n --)
Used inside a colon definition to mark the end of a CASE statement which implements a multi-decision control structure. Use as:

```
n1 CASE
        n2 OF words to be executed if n1 = n2          ENDOF
        n3 OF words to be executed if n1 = n3          ENDOF
        n4 OF words to be executed if n1 = n4          ENDOF
        words to be executed if n1 doesn't equal n2 or n3 or n4
ENDCASE
```

An error is issued if CASE and ENDCASE are not properly paired in a definition. See CASE, OF, ENDOF, RANGE.OF, and URANGE.OF.
Pronunciation: "end-case" Attributes: C, I

ENDIF (--)
Synonym for THEN. Used inside a colon definition to mark the end of an

```
     IF ... ELSE ... ENDIF
or   IF ... ENDIF
```

conditional structure. The word following ENDIF is executed after the IF or ELSE (if present) part of the conditional executes. An error is issued if ENDIF is not paired with IF or ELSE in a colon definition.
Pronunciation: "end-if" Attributes: C, I

ENDIFTRUE (--)
Marks the end of a conditional structure that is used outside a colon definition. Use as:

```
     flag IFTRUE .... OTHERWISE .... ENDIFTRUE
or
     flag IFTRUE .... ENDIFTRUE
```

The execution mode conditional structure can be used to conditionally compile portions of source code. Note that IFTRUE/OTHERWISE/ENDIFTRUE statements can not be nested. See IFTRUE and OTHERWISE.
Pronunciation: "end-if-true"

ENDOF (--)
Used inside a CASE ... ENDCASE structure to mark the end of a conditional statement that starts with OF or RANGE.OF or URANGE.OF. If the OF portion of the case statement is true, ENDOF branches to the word after ENDCASE. See CASE, ENDCASE, OF, RANGE.OF, and URANGE.OF. An error is issued if OF and ENDOF are not properly paired.
Pronunciation: "end-of" Attributes: C, I

ERASE (xaddr\u -- | u = byte count)
Stores a zero byte in each of u consecutive bytes beginning at xaddr. The specified region may cross page boundaries. Does nothing if u = 0.

EVALUATE (x$addr --)
Interprets the counted string located at x$addr. Use as:

```
     " <words to be executed>" EVALUATE
```

The string need not end with a space. EVALUATE provides one method of compiling a reference to a word that has not yet been defined. For example,

```
     : EARLY.WORD
             ...
             " INITIALIZE.EVERYTHING" EVALUATE
             ...
     ;
```

Even though the routine INITIALIZE.EVERYTHING has not yet been defined, EARLY.WORD will call it if INITIALIZE.EVERYTHING has been defined by the time EARLY.WORD is finally executed at runtime. Note that EVALUATE must search the dictionary at runtime to be able to execute INITIALIZE.EVERYTHING. For a more efficient means of implementing forward references, see REDEFINE.

Implementation detail: EVALUATE saves the contents of the input stream variables UTIB, #TIB, >IN, and BLK, then sets these variables to reference the specified x$addr, then calls INTERPRET, and finally restores the original values to the input stream variables.

EXECUTE (xcfa --)
Executes (calls) the routine whose executable machine instructions begin at the specified code field xaddress. The xcfa can be on any page.

EXIT (--)
Specifies a termination point in a definition. EXIT is an immediate word that compiles code that, when later executed, clears the locals off the return stack (if present), and pops the top value off the return stack and returns control to the calling word. When compiled into a definition, EXIT causes the word to terminate at that point when the word is later executed. An unchecked (and usually severe) error occurs if the return stack does not hold a valid return address (for example, if EXIT is used between >R and R> or inside a DO...LOOP or FOR...NEXT loop). To exit a definition from inside a single DO...LOOP, execute UNLOOP (which removes and drops 2 items from the rstack) to discard the loop index and limit from the return stack before calling EXIT. One UNLOOP is needed for each nested DO... LOOP. To exit a definition from inside a FOR...NEXT loop, call R>DROP to discard the loop index before calling EXIT. One R>DROP is needed for each nested FOR...NEXT.
Attributes: C, I

EXPECT (xaddr\+n --)
Receives a string of characters from the serial port and stores them in a buffer in memory starting at xaddr. Terminates when +n characters have been received or a carriage return occurs, whichever occurs first. Characters are echoed via the serial port. The carriage return is neither saved in the buffer nor echoed. Tab characters are echoed; however, they are saved in the buffer as TAB.WIDTH spaces. The backspace or delete character (ascii 8 or 127) removes the most recently received character from the buffer unless there are no characters in the buffer. Null characters (ascii 0) and linefeeds (ascii 10) are ignored. XON and XOFF characters are ignored except that they cause the XMIT.DISABLE flag in the user area to be set if XOFF is received or cleared if XON is received. Character case is not altered. The user variable SPAN is set equal to the number of characters received (but not including the terminating carriage return, if present). EXPECT is called by QUERY to accept serial input to the TIB.
Attributes: M

F! (r\xaddr --)
Stores a floating point number at xaddr. A synonym for 2!.
Pronunciation: "f-store"

F* (r1\r2 -- r3)
Multiplies floating point number r1 by r2 giving r3.
Pronunciation: "f-star" Attributes: S

F** (r1\r2 -- r3)
r3 equals r1 raised to the r2 power; i.e., r3 = r1^r2. Overflows if r1 is negative.

Pronunciation: "f-star-star" Attributes: S

F*/COUNTER (-- xaddr)
A user variable that holds a 16 bit counter that is incremented with each call to F* or F/. Used by the BENCHMARK: and (BENCHMARK:) routines.
Pronunciation: "f-star-slash-counter" Attributes: U

F+ (r1\r2 -- r3)
Adds floating point number r2 to r1 giving the sum r3.
Pronunciation: "f-plus" Attributes: S

F+COUNTER (-- xaddr)
A user variable that holds a 16 bit counter that is incremented with each call to F+ or F-. Used by the BENCHMARK: and (BENCHMARK:) routines.
Pronunciation: "f-plus-counter" Attributes: U

F- (r1\r2 -- r3)
Subtracts floating point number r2 from r1 giving the difference r3.
Pronunciation: "f-minus" Attributes: S

F. (r --)
Displays r using the default format as specified by the most recent execution of FIXED FLOATING or SCIENTIFIC. FLOATING is the default format after executing FP.DEFAULTS or after a cold restart. See F>FIXED$, F>SCIENTIFIC$, and F>FLOATING$.
Pronunciation: "f-dot" Attributes: M, S

F.OVER.N (r\w -- r\w\r)
Copies the floating point value r located under the top data stack cell to the top of the data stack.
Pronunciation: "f-over-n"

F/ (r1\r2 -- r3)
Divides floating point number r1 by r2 giving the result r3.
Pronunciation: "f-slash" Attributes: S

F0< (r -- flag)
Flag is TRUE if floating point number r is less than zero, FALSE otherwise.
Pronunciation: "f-zero-less-than"

F0< = (r -- flag)
Flag is TRUE if floating point number r is less than or equal to 0, FALSE otherwise.
Pronunciation: "f-zero-less-than-or-equal"

F0< > (r -- flag)
Flag is TRUE if the floating point number is not equal to zero, and FALSE otherwise.
Pronunciation: "f-zero-not-equal"

F0= (r -- flag)
Flag is TRUE if the floating point number equals zero and FALSE otherwise.

	Pronunciation: "f-zero-equal"
F0>	(r -- flag) If floating point number r is greater than zero, flag equals TRUE; otherwise, flag equals FALSE. Pronunciation: "f-zero-greater-than"
F0>=	(r -- flag) Flag is TRUE if floating point number r is greater than or equal to 0, FALSE otherwise. Pronunciation: "f-zero-greater-than-or-equal"
F2*	(r1 -- r2) Floating point number r2 equals r1 * 2. Pronunciation: "f-two-star" Attributes: S
F2/	(r1 -- r2) Floating point number r2 equals r1 divided by 2. Pronunciation: "f-two-slash" Attributes: S
F2DROP	(r1\r2 --) Drops two floating point numbers (4 cells) from the data stack. Pronunciation: "f-two-drop"
F2DUP	(r1\r2 -- r1\r2\r1\r2) Duplicates the top two floating point numbers on the data stack. Pronunciation: "f-two-dupe"
F<	(r1\r2 -- flag) Flag is TRUE if floating point number r1 is less than r2, FALSE otherwise. Pronunciation: "f-less-than"
F< =	(r1\r2 -- flag) Flag is TRUE if floating point number r1 is less than or equal to r2. Pronunciation: "f-less-than-or-equal"
F< >	(r1\r2 -- flag) Flag is TRUE if the two floating point numbers are not equal and FALSE otherwise. Pronunciation: "f-not-equal"
F=	(r1\r2 -- flag) Flag is TRUE if the two floating point numbers are equal and FALSE otherwise. Pronunciation: "f-equal"
F>	(r1\r2 -- flag) Flag is TRUE if floating point number r1 is greater than r2, FALSE otherwise. Pronunciation: "f-greater-than"
F>=	(r1\r2 -- flag) Flag is TRUE if floating point number r1 is greater than or equal to r2. Pronunciation: "f-greater-than-or-equal"

F>FIXED$ (r -- x$addr\flag)

Converts r to a counted ascii string starting at x$addr in FIXED format. If the conversion is successful, flag is TRUE. If r cannot be represented as an ascii string in FIXED format (i.e., if the values of LEFT.PLACES and RIGHT.PLACES are inappropriate) then x$addr contains the string "won'tfit" and flag is false. The FIXED format is composed of (an optional) sign, LEFT.PLACES digits, a decimal point, RIGHT.PLACES digits, and a trailing space:

    -xxx.yyy

If the NO.SPACES flag is false (the default condition), the field size equals LEFT.PLACES + RIGHT.PLACES + 3 and numbers are decimal aligned. The size of the string is clamped to a maximum of 32 characters. Setting TRAILING.ZEROS true displays all trailing zeros to the right of the decimal point to a maximum of RIGHT.PLACES.

Pronunciation: "f-to-fixed-string" Attributes: S

F>FLOATING$ (r -- x$addr\true)

Converts r to an ascii string at x$addr using FLOATING format which selects FIXED format unless the number can be displayed with greater resolution using scientific notation, in which case SCIENTIFIC format is used (see F>FIXED$ and F>SCIENTIFIC$). If FILL.FIELD if OFF (the default condition), the string is displayed using the minimum possible field size, and numbers are not decimal aligned. If FILL.FIELD is ON, the field size of the string is always equal to the scientific field size, which is MANTISSA.PLACES+8, and numbers are decimal aligned for neat display of tabular data. The string includes a trailing space unless NO.SPACES is true. The flag on top of the stack is always true because any valid floating point number can be represented in the FLOATING format.

Pronunciation: "f-to-floating-string" Attributes: S

F>R (r --)

Return Stack: (R: -- r)

Transfers the top floating point number on the data stack to the return stack.

Pronunciation: "f-to-r" Attributes: C

F>SCIENTIFIC$ (r -- x$addr\true)

Converts r into a text string at x$addr using SCIENTIFIC format. The format is: (an optional) sign, single digit, decimal point, MANTISSA.PLACES digits, E, exponent sign, 2-digit exponent, and a trailing space:

    -1.xxxxE-yy

The field size is equal to MANTISSA.PLACES + 8. The string includes a trailing space unless NO.SPACES is true. The flag on top of the stack is always true because any valid floating point number can be represented in the SCIENTIFIC format.

Pronunciation: "f-to-scientific-string" Attributes: S

F@ (xaddr -- r)

Fetches a floating point number from xaddr. A synonym for 2@.

Pronunciation: "f-fetch"

FABS (r1 -- r2)

r2 is the absolute value of r1. If floating point number r1 is negative, applies FNEGATE to r1 giving the positive number r2; otherwise, r2 = r1.
Pronunciation: "f-abs" Attributes: S

FACOS (r1 -- r2)
r2 is the arc-cosine of r1. r2 is expressed in radians.
Pronunciation: "f-a-cos" Attributes: S

FALN (r1 -- r2)
r2 equals the natural anti-log of r1; i.e., r2 = e^r1.
Pronunciation: "f-a-l-n" Attributes: S

FALOG10 (r1 -- r2)
r2 equals the base 10 anti-log of r1; i.e., r2 = 10^r1.
Pronunciation: "f-a-log-ten" Attributes: S

FALOG2 (r1 -- r2)
r2 equals the base 2 anti-log of r1; i.e., r2 = 2^r1.
Pronunciation: "f-a-log-two" Attributes: S

FALSE (-- flag | flag = 0)
Puts a boolean false flag equal to 0 on the data stack .

FASIN (r1 -- r2)
r2 is the arcsine of r1. r2 is expressed in radians.
Pronunciation: "f-a-sine" Attributes: S

FATAN (r1 -- r2)
r2 is the arctangent of r1. r2 is expressed in radians.
Pronunciation: "f-a-tan" Attributes: S

FCONSTANT (r <name> --)
Removes the next <name> from the input stream and defines a child word called <name> which when executed leaves the floating point value r on the data stack. r is stored in the definitions area of the dictionary. <name> is referred to as an "fconstant". Use as:

```
    r FCONSTANT <name>
```

Pronunciation: "f-constant" Attributes: D

FCOS (r1 -- r2)
r2 is the cosine of r1. r1 is expressed in radians.
Pronunciation: "f-cos" Attributes: S

FDROP (r --)
Drops a floating point number (two cells) from the data stack.
Pronunciation: "f-drop"

FDUP (r -- r\r)
Duplicates the top floating point number (two cells) on the data stack.
Pronunciation: "f-dupe"

FDUP>R (r -- r)
Return Stack: (R: -- r)
Copies the top floating point number on the data stack to the return stack.
Pronunciation: "f-dup-to-r" Attributes: C

FFT (matrix.xpfa --)
Computes the FFT (Fast Fourier Transform) of the complex floating point data in the source matrix specified by matrix.xpfa and stores the transformed result in the source matrix. The source matrix must be dimensioned to have 2 rows with 1 column per input data point. The number of columns must be a power of 2. Row#0 holds the floating point number representing the real part of each data point, and row#1 holds the floating point number representing the imaginary part of each data point (i.e., the part of the complex number that is multiplied by the imaginary operator which is the square root of -1). Each data point typically indicates a complex value of the input in the time or spatial domain. FFT transforms the input into a complex waveform in the frequency domain and stores the results in the source matrix. The maximum number of complex data points (columns) that can be transformed is 8192. To perform the inverse FFT, see IFFT.
Interpretation of results:
Assume that the input waveform spans T seconds. Then column#0 of the transformed signal corresponds to a frequency of 0, column #1 corresponds to a frequency of 1/T, column#2 corresponds to 2/T, etc. up to a maximum frequency of (plus or minus) N/2T where N is the number of points (columns) in the input waveform. After the midway point (N/2), the frequencies are negative. That is, the last point at column# N-1 corresponds to a frequency of -1/T, the next to last point at column# N-2 corresponds to -2/T, etc. If a given frequency component has a magnitude of A in the time domain, it is transformed into a spike with magnitude A*N in the frequency domain.
Pronunciation: "f-f-t" Attributes: S

FIELD (u <name> --)
Defines a field constant <name> which when executed adds u to the extended address on the top of the data stack. <name>'s stack picture is (xaddr -- xaddr + u). It is recommended that <name> begin with a + to suggest its runtime behavior of adding the offset u to the top data stack item. FIELD is the low level word called by all of the words that create members in structures. See MEMBER->
Attributes: D

FILL (xaddr\u\b -- | u = byte count, b = fill value)
The specified byte b is stored in each of u consecutive addresses beginning at xaddr. The specified region may cross page boundaries. Does nothing if u = 0. See FILL.MANY.

FILL.ARRAY (array.xpfa\char --)
Stores char into each byte of the specified array.

FILL.FIELD (-- xaddr)
A user variable that contains a flag. If the flag is true, floating point numbers printed in FLOATING format are padded with spaces to yield a constant field width irrespective of whether the number is printed in scientific notation or fixed notation, and numbers

printed in fixed notation are decimal aligned. This leads to neat printouts of tabular data. If the flag is false, the field width is not padded out. See F>FLOATING$
Attributes: U

FILL.MANY (xaddr\d\b -- | d = byte count, b = fill value)
The specified byte b is stored in each of d consecutive addresses beginning at xaddr. The specified region may cross page boundaries. Does nothing if d = 0.

FIND (<name> -- [xcfa\flag] or [0])
Executes BL WORD to parse the next space-delimited word from the input stream, and then searches the dictionary for a match of the parsed word. FIND first searches the CONTEXT vocabulary. Then, if the word is not found and if the CONTEXT and CURRENT vocabularies are different, it searches the CURRENT vocabulary. If the word is not found in the dictionary, FIND leaves a 0 on the stack. If the word is found, FIND leaves the word's extended code field address under a flag on the stack. The flag is +1 if the word is immediate and -1 if the word is not immediate. An error occurs if the input stream is exhausted while WORD executes. A COLD restart will occur if more than 255 page changes are made during the search through either vocabulary. This prevents the interpreter from going on an infinite search through a corrupted dictionary. Find also invokes a COLD restart if POCKET is not in common memory.

FINT (r1 -- r2)
If the absolute value of r1 is less than 2^31, r2 is the floating point representation of the integer part (truncated toward zero) of r1. For these numbers, FINT is the equivalent of DINT DFLOT. If the absolute value of r1 exceeds 2^31, FINT returns r2 = r1 and sets the OVERFLOW flag.
Pronunciation: "f-int"

FIRST (-- xaddr)
Returns the extended address of the lower boundary of the block buffers; that is, the address of the first byte of the first block buffer. Equivalent to UFIRST X@ . Initialized by BLOCK.BUFFERS.
Attributes: U

FIXED (--)
Sets the default printing format used by F. to fixed. Numbers are decimal aligned, and RIGHT.PLACES and LEFT.PLACES determine the field width. See F>FIXED$

FIXED. (r --)
Prints r using FIXED format. Does not change the default printing format. See F>FIXED$
Pronunciation: "fixed-dot" Attributes: M, S

FIXX (r -- n)
Rounds r to the nearest integer n. OVERFLOW is set if needed. See DFIXX
Attributes: S

FLN (r1 -- r2)
r2 is the natural logarithm of r1.
Pronunciation: "f-l-n" Attributes: S

FLOATING (--)
Sets the default printing format used by F. to floating. The floating format uses FIXED format if the number can be represented with the same or more significant digits as it would if it were represented in SCIENTIFIC format. Otherwise, it uses SCIENTIFIC format. See F>FLOATING$

FLOATING. (r --)
Prints r using FLOATING format. See F>FLOATING$
Pronunciation: "floating-dot" Attributes: M, S

FLOG10 (r1 -- r2)
r2 is the base 10 logarithm of r1.
Pronunciation: "f-log-ten" Attributes: S

FLOG2 (r1 -- r2)
r2 is the base 2 logarithm of r1.
Pronunciation: "f-log-two" Attributes: S

FLOOR (r1 -- r2)
r2 is the floating point representation of the greatest integer less than or equal to r1.
Attributes: S

FLOT (n -- r)
Converts the 16 bit integer n to its floating point representation r. See DFLOT
Pronunciation: "float" Attributes: S

FMAX (r1\r2 -- [r1] or [r2])
Retains the greater of r1 and r2.
Pronunciation: "f-max"

FMIN (r1\r2 -- [r1] or [r2])
Retains the lesser of r1 and r2.
Pronunciation: "f-min"

FNEGATE (r1 -- r2)
r2 is the negative of r1.
Pronunciation: "f-negate"

FNUMBER (x$addr -- [r\-1] or [0])
Attempts to convert the space-delimited counted ascii string at x$addr into a valid floating point number r. Returns r under a true flag if the conversion is successful; otherwise returns a false flag. x$addr is the address of the text string's count byte. The string at x$addr must end with a blank; the blank may or may not be included in the count. Strings parsed by the commands BL WORD obey this rule. See also NUMBER.
Pronunciation: "f-number" Attributes: S

FOR (u1 --)
Return Stack: (R: -- u1)

Used inside a colon definition to mark the beginning of a count-down loop structure that is terminated by NEXT. Sets up loop index u1 on the return stack. Use as:

```
u1      FOR  <words to be executed u1+1 times>
        NEXT
```

0 FOR...NEXT executes 1 time, 1 FOR...NEXT executes 2 times, 65,535 FOR...NEXT executes 65,536 times, etc. Because the loop index is maintained on the return stack, caution must be exercised when using the operators >R and R> inside a loop. FOR...NEXT loops may be nested as long as each FOR is matched with a corresponding NEXT in the same definition as FOR; otherwise, an error is issued. FOR ... NEXT loops execute faster than DO ... LOOP constructs. The word I may be used inside a FOR NEXT loop. J K I' and LEAVE may not be used in a FOR NEXT loop.
Attributes: C, I

FORGET (<name> --)
Removes <name> from the input stream and searches for it in the CURRENT vocabulary. If <name> is found, deletes it from the dictionary and removes all words defined after <name> by adjusting DP, NP, and the xhandle of the CURRENT vocabulary. An error occurs if <name> is not found. FORGET does not de-allocate space in the variable area (VP is not adjusted). Use of ANEW for dictionary cleanup avoids this problem. Likewise, heap space is not de-allocated by FORGET; use of ON.FORGET solves this problem. When a word is about to be forgotten, FORGET checks to see if its name is ON.FORGET. Words with the name ON.FORGET are executed before being forgotten. This allows cleanup of heap items or other cleanup actions. A warning is issued if execution of FORGET leaves any of the pointers DP, NP, or VP pointing to non-RAM locations. Use this word with caution; it is possible to FORGET the entire operating system!

FORTH (--)
Specifies FORTH as the vocabulary to be searched first during dictionary searches. Implementation detail: stores the 32-bit xhandle of the FORTH vocabulary into the user variable CONTEXT.

FOVER (r1\r2 -- r1\r2\r1)
Places a copy of r1 on top of the data stack.
Pronunciation: "f-over"

FP&STRING.POP (--)
Return Stack: (R: 15 cells --)
Pops off the return stack the values of the scratchpad user variables placed there by FP&STRING.PUSH and restores the popped values to the user area. Also restores PAD to the value it had before FP&STRING.PUSH executed. FP&STRING.PUSH and FP&STRING.POP are intended to be used in interrupt service routines that call floating point operations and/or floating point string conversion and/or integer string conversion operations. Execution time is approximately 113 microseconds. See FP&STRING.PUSH.
Pronunciation: "f-p-and-string-pop" Attributes: S

FP&STRING.PUSH (xaddr -- | xaddr is temporary PAD)
Return Stack: (R: -- 15 cells)

Pushes onto the return stack the values of the scratchpad user variables that are modified by words that call the basic floating point operators F+ F- F* and F/ or by words that perform floating point or integer string conversion. Also installs the specified xaddr as the value of PAD. The saved items are later removed from the return stack and restored to the user area by FP&STRING.POP. FP&STRING.PUSH is intended to be used in interrupt service routines that call floating point or number/string conversion operations. FP&STRING.PUSH should be executed by the interrupt service routine before the first floating point or conversion function is called to save the scratchpad values on the return stack and to establish a temporary PAD. Executing FP&STRING.POP at the end of the interrupt service routine restores the scratchpad variables and PAD to their prior values. Using FP&STRING.PUSH and FP&STRING.POP prevents conflicts when both the foreground (i.e., non-interrupt-invoked) program and the interrupt service routine call floating point and/or string conversion operations; if the interrupt routine does not save and restore the scratchpad variables, it may corrupt a floating point or conversion operation in the foreground program. Integer and floating point string conversion use the 32 bytes below PAD for string generation, so establishing a new PAD address prevents conflicts between foreground and interrupt-invoked number conversion routines. Execution time is approximately 113 microseconds. Note that the S attribute indicates which words modify scratchpad variables; see also FP.PUSH. If FP&STRING.PUSH is called inside an interrupt service routine, there is no need to call FP.PUSH, since FP&STRING.PUSH saves all of the variables that FP.PUSH saves, plus additional variables.
Example of use: Assume that a foreground program is performing a matrix inversion and printing the results. This involves floating point additions and multiplications as well as number-to-string conversion. The following simple interrupt service routine would function correctly:

```
        FVARIABLE    LATEST.DATA
<xaddr in ram>  XCONSTANT   TEMP.PAD.XADDR
<another xaddr in ram> XCONSTANT   STRING.DESTINATION
: INTERRUPT.SERVICE         ( -- )
        TEMP.PAD.XADDR  FP&STRING.PUSH
        LATEST.DATA F@ 2.0 F*
        F>SCIENTIFIC$ DROP ( xaddr\cnt -- )
        >R STRING.DESTINATION R> CMOVE
        FP&STRING.POP       ;
```

This service routine saves the scratchpad variables on the return stack, establishes a temporary PAD, multiplies the contents of LATEST.DATA by 2.0 and converts the result to an ascii string in scientific format (this modifies the scratchpad values), moves the string to STRING.DESTINATION, and restores the scratchpad values and the original PAD before terminating so that the foreground program can proceed. Even if the foreground matrix inversion program is in the middle of a floating point or string conversion operation when an interrupt occurs, the operation is not corrupted. Note that local variables should not be used inside interrupt service routines; see LOCALS{.
Pronunciation: "f-p-and-string-push"

FP.DEFAULTS (--)

Initializes floating point user variables to default values. Sets the number base to DECIMAL, makes FLOATING the default print format, sets LEFT.PLACES = 4,

RIGHT.PLACES = 3, MANTISSA.PLACES = 3, sets TRAILING.ZEROS, NO.SPACES, and FILL.FIELD false, and initializes MAX#DIMENSIONS to 4.
Pronunciation: "f-p-defaults" Attributes: S

FP.ERROR (-- xaddr)
A user variable containing a 16-bit error flag. Set by many floating point operations and by OVERFLOW and UNDERFLOW. Can be checked after any floating point operation to detect an underflow or overflow error. Contents of 0 indicates no error, 1 indicates underflow, and -1 indicates overflow.
Pronunciation: "f-p-error" Attributes: U

FP.POP (--)
Return Stack: (R: w1\w2\w3\w4\w5 --)
Pops off the return stack the values of the 5 scratchpad user variables placed there by FP.PUSH and restores the popped values to the user area. FP.PUSH and FP.POP are intended to be used in interrupt service routines that call floating point operations. Execution time is approximately 50 microseconds. See FP.PUSH.
Pronunciation: "f-p-pop" Attributes: S

FP.PUSH (--)
Return Stack: (R: -- w1\w2\w3\w4\w5)
Pushes onto the return stack the values of the 5 scratchpad user variables that are modified by all words that call the basic floating point operators F+ F- F* and F/. The items are later removed from the return stack and restored to the user area by FP.POP. FP.PUSH is intended to be used in interrupt service routines that call floating point operations. FP.PUSH should be executed by the interrupt service routine before the first floating point function is called to save the scratchpad values on the return stack. Executing FP.POP at the end of the interrupt service routine restores the scratchpad variables to their prior values. Using FP.PUSH and FP.POP prevents conflicts when both the foreground (i.e., non-interrupt-invoked) program and the interrupt service routine call floating point operations; if the interrupt routine does not save and restore the scratchpad variables, it may corrupt a floating point operation in the foreground program. FP.PUSH saves the contents of four headerless scratchpad variables as well as the contents of FP.ERROR. Execution time is approximately 50 microseconds. Note that the S attribute indicates which words modify scratchpad variables; see also FP&STRING.PUSH.
Example of use:
Assume that a foreground program is performing a matrix inversion which involves floating point additions and multiplications. The following simple interrupt service routine would function correctly:

```
    FVARIABLE   LATEST.DATA
    : INTERRUPT.SERVICE          ( -- )
            FP.PUSH         \ save fp scratchpad user variables
            LATEST.DATA F@ 2.0 F*   LATEST.DATA F!
            FP.POP          \ restore fp scratchpad user variables
    ;
```

Even if the foreground matrix inversion program is in the middle of a floating point operation when an interrupt occurs, the floating point operation is not corrupted. INTERRUPT.SERVICE saves the fp scratchpad values, performs the F* (which modifies the scratchpad values) and then restores the original scratchpad values before

terminating so that the foreground program can proceed. Note that local variables should not be used inside interrupt service routines; see LOCALS{.
Pronunciation: "f-p-push"

FPICK (r\wn-1\...w1\w0\+n -- r\wn-1\...\w1\w0\r | 0 <= +n <= 255)
Copies the floating point number whose most significant cell is the nth item on the stack (0-based, not including n) to the top of the stack. An unchecked error occurs if there are fewer than +n+2 cells on the data stack. 0 FPICK is equivalent to FDUP, 1 FPICK is equivalent to X.OVER.N, 2 FPICK is equivalent to FOVER.
Pronunciation: "f-pick"

FR> (-- r)
Return Stack: (R: r --)
Transfers the top floating point number on the return stack to the data stack.
Pronunciation: "f-r-from" Attributes: C

FR>DROP (--)
Return Stack: (R: r --)
Removes the top floating point number from the return stack.
Pronunciation: "f-r-from-drop" Attributes: C

FR@ (-- r)
Return Stack: (R: r -- r)
Copies the top floating point number on the return stack to the data stack.
Pronunciation: "f-r-fetch" Attributes: C

FRAME.DROP ([+n bytes]\+n --)
Drops +n bytes (in addition to +n) from the data stack. If +n is odd, +n is incremented first so that an integer number of two-byte cells is dropped. FRAME.DROP is usually used to drop items placed on the stack by STACK.FRAME. See STACK.FRAME.

FRANDOM (-- r | 1.0 <= r < 2.0)
r is a pseudo-random floating point number greater than or equal to 1.0 and less than 2.0. See RANDOM# and RANDOM.GAUSSIAN.
Pronunciation: "f-random"

FREE.HANDLE (-- xaddr)
A variable that holds the 16-bit address of the next available handle in the current heap. xaddr is equal to CURRENT.HEAP - 12. Initialized by IS.HEAP.

FROM.HEAP (d -- [xhandle] or [0\0] | d = number of bytes)
If d bytes are available in the heap, allocates them and returns a 32-bit xhandle whose contents equal the base xaddress of the allocated heap item. Adjusts d upward so that it is an even multiple of 4, and allocates the heap item so that its base address is an even multiple of 4. Returns 0\0 if there is not enough heap space to perform the allocation, or if the allocated handle is within 5 bytes of the bottom of CURRENT.HEAP's page (handles must be on the same page as CURRENT.HEAP).

FROT (r1\r2\r3 -- r2\r3\r1)
Rotates the top three floating point numbers on the data stack.

Pronunciation: "f-rote"

FRTI (r1 -- r2)
Rounds r1 to the nearest integer and returns the result r2. OVERFLOW is set if necessary.
Pronunciation: "f-round-to-integer" Attributes: S

FSCALE (r1\n -- r2)
Adds the signed integer n to the (base 2) exponent of r1 to compute r2. This provides a fast way of multiplying r1 by a power of 2 (if n is positive) or dividing r1 by a power of 2 (if n is negative). OVERFLOW is set if necessary.
Pronunciation: "f-scale" Attributes: S

FSIN (r1 -- r2)
r2 is the sine of r1. r1 is expressed in radians.
Pronunciation: "f-sine" Attributes: S

FSQRT (r1 -- r2)
r2 is the square root of r1. r2 is zero if r1 is negative.
Pronunciation: "f-square-root" Attributes: S

FSWAP (r1\r2 -- r2\r1)
Exchanges the top two floating point numbers on the stack.
Pronunciation: "f-swap"

FTAN (r1 -- r2)
r2 is the tangent of r1. r1 is expressed in radians.
Pronunciation: "f-tan" Attributes: S

FVARIABLE (<name> --)
Removes the next <name> from the input stream, defines a child word called <name>, and VALLOTs 2 cells in the variable area. When <name> is executed, it leaves the extended address xaddr of the two cells reserved in the variable area to hold <name>'s contents. <name> is referred to as an "fvariable". Use as:

```
FVARIABLE <name>
```

Pronunciation: "f-variable" Attributes: D

F^N (r1\n -- r2)
Raises r1 to the nth power and returns the result r2; r2 = r1^n.
Pronunciation: "f-to-the-n" Attributes: S

GARRAY.XPFA (-- xpfa)
Returns the extended parameter field address that specifies the graphics data array. This otherwise unnamed array is dimensioned by INIT.DISPLAY if a graphics display has been specified using IS.DISPLAY. UPDATE.DISPLAY.LINE and UPDATE.DISPLAY write the contents of this array to the graphics display. DISPLAY.BUFFER returns the xaddr of the first element in this array if a graphics display is in use. See the graphics extension routines that are supplied in source code form to augment the kernel; these routines provide examples of how to access information in the graphics array.

Pronunciation: "g-array-x-p-f-a"

GET (xresource --)
Used in a multitasking system to gain access to a shared resource. PAUSEs until the resource variable whose address is xresource is available, and then GETs the resource by storing the task id (i.e., the STATUS xaddr) of the requesting task into the xresource. 0\0 in xresource indicates that the resource is available, and a non-zero value that is not equal to the requesting task's id indicates that another task controls the resource. To ensure that the state of the resource is correctly determined, GET disables interrupts for 27 to 57 cycles (6.75 to 14.25 microseconds). See ?GET, RELEASE, and RESOURCE.VARIABLE:.
Attributes: M

H.INSTANCE: (u <name> -- | u = size of heap item)
Creates a heap instance. Removes <name> from the input stream, creates a header for <name> in the dictionary, and allots and erases a 6-byte parameter field associated with <name> in the variable area. See HEAP.STRUCTURE.PF for a description of the contents of the parameter field. Does not allocate space in the heap; this is done when the command ALLOCATED is executed. When <name> is executed, its stack picture is picture is

```
     ( -- [xaddr] or [0\0] )
```

where xaddr is the base address of the item in the heap, and where 0\0 is returned if heap space has not been ALLOCATED. The size u of the item in the heap is stored in the code field of <name> for later use by the word SIZE.OF. Typical use:

```
     size.of.heap.item  H.INSTANCE:  <name>
     SIZE.OF <name>   ' <name>   ALLOCATED
      <name>          ( -- base.xaddr.of.heap.item )
```

Pronunciation: "h-instance" Attributes: D

HALT (--)
An infinite loop whose action is to put the calling task ASLEEP and execute PAUSE. Typically used to terminate a task action that is not an infinite loop.
Attributes: M

HANDLE.PTR (-- xaddr)
A variable that holds the 16-bit address of the next available location that can be allocated as a handle in the current heap. xaddr is equal to CURRENT.HEAP - 10. Initialized by IS.HEAP.
Pronunciation: "handle-pointer"

HAS.PFA (--)
Sets the "has-pfa" bit in the header of the most recently defined word in the CURRENT vocabulary. If set, this bit indicates that the word has a parameter field. HAS.PFA is executed by <DBUILDS and <VBUILDS and by defining words including VARIABLE CONSTANT INTEGER: and their double number counterparts and synonyms. The word ' (tick) checks the has-pfa bit when calculating the parameter field address of a word. ?HAS.PFA can be used to determine whether the has-pfa bit is set.
Pronunciation: "has-p-f-a"

HEAP.PTR (-- xaddr)

A variable that holds the extended address of the next available byte in the current heap. xaddr equals CURRENT.HEAP - 8. Initialized by IS.HEAP.
Pronunciation: "heap-pointer"

HEAP.STRUCTURE.PF (-- u | u = minimum size of a heap parameter field)
Places on the stack the minimum number of bytes needed to allocate a parameter field for a heap item (u = 6 bytes). HEAP.STRUCTURE.PF is a structure defined as:

```
STRUCTURE.BEGIN: HEAP.STRUCTURE.PF
        PAGE-> +HEAP.PAGE          \ page of current.heap and heap.handle
        HNDL-> +HEAP.HANDLE                \ contains base addr of heap item
        ADDR-> +CURRENT.HEAP       \ specifies which heap the item is in
STRUCTURE.END
```

See H.INSTANCE:, +HEAP.PAGE, +HEAP.HANDLE, and +CURRENT.HEAP.
Pronunciation: "heap-structure-p-f"

HERE (-- xaddr)
Places on the stack the xaddr of the next available location in the definitions area. Equivalent to DP X@
Attributes: U

HEX (--)
Sets the numeric conversion base to sixteen by storing decimal 16 into the user variable BASE.

HNDL-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a 16-bit field in the structure that will hold a handle (a handle is an address where another address is stored). Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "handle" Attributes: D

HOLD (char --)
Inserts char into the numeric output string to the left of the prior character in the string and decrements the headerless pointer #PTR. HOLD is used between <# and #> commands to create a pictured numeric string.
Attributes: S

I (-- w)
Return Stack: (R: w -- w)
Places a copy of the current (innermost) loop index on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R).
Typical use:

```
DO ... I ... LOOP
```

or

```
DO ... I ... +LOOP
```

or
    FOR ... I ... NEXT
Pronunciation: "i" Attributes: C

I' (-- w1)
Return Stack: (R: w1\w2 -- w1\w2)
Used inside a DO...LOOP or DO...+LOOP structure. Places a copy of the current (innermost) loop limit on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:
    DO ... I' ... LOOP
Pronunciation: "i-prime" Attributes: C

I+ (n1 -- n2 | n2 = n1 + I)
Return Stack: (R: w1 -- w1)
Used inside a DO...LOOP or DO...+LOOP or FOR ... NEXT structure. Adds the current (innermost) loop index I to n1 and places the result n2 on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:
    DO ... n1 I+ ... LOOP
Pronunciation: "i-plus" Attributes: C

I- (n1 -- n2 | n2 = n1 - I)
Return Stack: (R: w1 -- w1)
Used inside a DO...LOOP or DO...+LOOP or FOR ... NEXT structure. Subtracts the current (innermost) loop index I from n1 and places the result n2 on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:
    DO ... n1 I- ... LOOP
Pronunciation: "i-minus" Attributes: C

IC1.ID (-- n)
Returns the interrupt identity code for input capture 1 which is associated with port bit PA2. Used as an argument for ATTACH.
Pronunciation: "i-c-one-i-d"

IC2.ID (-- n)
Returns the interrupt identity code for input capture 2 which is associated with port bit PA1. Used as an argument for ATTACH.
Pronunciation: "i-c-two-i-d"

IC3.ID (-- n)
Returns the interrupt identity code for input capture 3 which is associated with port bit PA0. Used as an argument for ATTACH.
Pronunciation: "i-c-three-i-d"

IC4/OC5.ID (-- n)
Returns the interrupt identity code for input capture 4/ output compare 5. This interrupt can control the action of port bit PA3. Note that the optional secondary serial port uses IC4/OC5 and PA3. Used as an argument for ATTACH.
Pronunciation: "i-c-4-or-o-c-5-i-d"

ID. (xnfa --)
Prints the name of the routine associated with the extended name field address xnfa. Letters not saved in the header are printed as ___ . The name is followed by a space. If the word is smudged (for example, if an error occurred during compilation of the word), a ~ is printed before the first letter in the name. An unchecked error occurs if xnfa is not a valid name field address.
Pronunciation: "i-d-dot" Attributes: M

IF (flag --)
Used inside a colon definition to mark the beginning of a conditional structure. If flag is FALSE execution continues after the subsequent ELSE or ENDIF (THEN and ENDIF are synonyms) . If flag is TRUE execution continues with the word immediately following IF, and when ELSE is encountered (if present), control is transferred to the word following ENDIF. Use as:

```
    flag    IF      words to execute if flag is true
            ENDIF
```

or as:

```
    flag    IF      words to execute if flag is true
            ELSE    words to execute if flag is false
            ENDIF
```

Attributes: C, I

IFFT (matrix.xpfa --)
Computes the inverse Fast Fourier Transform of the frequency domain waveform in the source matrix specified by matrix.xpfa and stores the transformed result in the source matrix. The source matrix must be dimensioned to have 2 rows and 1 column per input data point. The number of columns must be a power of 2, with the maximum number of columns equal to 8192. Row#0 holds the floating point number representing the real part of each data point, and row#1 holds the floating point number representing the imaginary part of each data point. Each data point typically indicates the complex value of the input in the frequency domain. IFFT transforms the input into a complex waveform in the time or spatial domain and stores the results in the source matrix specified by matrix.xpfa. See FFT.
Pronunciation: "inverse-f-f-t" Attributes: S

IFTRUE (flag --)
Marks the start of the "true" portion of a conditional structure that is used in execution mode outside a colon definition. Use as:

```
     flag  IFTRUE .... OTHERWISE ....  ENDIFTRUE
```

If the flag passed to IFTRUE is true, the code between IFTRUE and OTHERWISE is executed, and the code between OTHERWISE and ENDIFTRUE is skipped. If the flag is false, the code between IFTRUE and OTHERWISE is skipped and execution continues with the words following OTHERWISE. Another valid syntax is:

```
    flag  IFTRUE ....  ENDIFTRUE
```

If the flag passed to IFTRUE is true, the code between IFTRUE and ENDIFTRUE is executed. If the flag is false, the code between IFTRUE and ENDIFTRUE is skipped. IFTRUE is analogous to IF but it is used outside of a colon definition. The execution mode conditional structure can be used to conditionally compile portions of source code. An unchecked error occurs if IFTRUE is used without a corresponding

ENDIFTRUE. Note that IFTRUE/OTHERWISE/ENDIFTRUE statements can not be nested.
Pronunciation: "if-true"

ILLEGAL.OPCODE.ID (-- n)
Returns the interrupt identity code for the illegal opcode interrupt. Used as an argument for ATTACH.
Pronunciation: "illegal-opcode-i-d"

IMMEDIATE (--)
Sets the "immediate bit" in the header of the most recently defined word to indicate that it should be executed immediately even in compilation mode. The command ?IMMEDIATE may be used to determine the status of the immediate bit for a given word.

INFINITY (-- r)
Pushes the largest representable floating point number onto the data stack.

INIT.A/D12&DAC (--)
Calls INIT.SPI to configure the serial peripheral interface (SPI) so that it can transfer data from the 12 bit analog to digital converter (A/D12) and to the 8 bit digital to analog converter (DAC). Sets all 8 DAC outputs to 00. See INIT.SPI.
Pronunciation: "init-A-to-D-twelve-and-dac"

INIT.DISPLAY (--)
Initializes the liquid crystal display (LCD) interface. If a graphics-style display has been specified by IS.DISPLAY, initializes the DISPLAY.HEAP and dimensions GARRAY.XPFA to point to an appropriately sized array in that heap. The base address of this array is returned by DISPLAY.BUFFER. If a character-style (alphanumeric) display has been specified by IS.DISPLAY, then the display buffer is located in the system RAM and the DISPLAY.HEAP and GARRAY.XPFA are not initialized. If the dimensions specified by IS.HEAP call for a graphics array that is larger than the available ROOM in the DISPLAY.HEAP, then INIT.DISPLAY will not dimension the array; see the glossary entry of DISPLAY.HEAP for further details. INIT.DISPLAY calls CLEAR.DISPLAY to clear the DISPLAY.BUFFER and write the blank data (ascii blanks for character displays, binary zeros for graphics displays) to the LCD display. Homes the cursor to the start of line 0, and leaves the display enabled with the cursor off and not blinking. See CLEAR.DISPLAY. Intermittently disables interrupts for 28 cycles (7 microseconds) per byte transmitted to the display.

INIT.ELAPSED.TIME (--)
Initializes the system variable TIMESLICE.COUNT to 0\0.

INIT.PIA (flag1\flag2 -- | flag1 = PPA.output?, flag2 = upper.PPC.output?)
Writes to the peripheral interface adapter (PIA) configuration register to set the specified data direction for PPA and the upper half of PPC. If flag1 is true, configures PPA as output; if flag 1 is false, configures PPA as input. Likewise, if flag2 is true, configures upper PPC as output; if flag 2 is false, configures upper PPC as input. INIT.PIA sets the direction of PPB as output and lower PPC as input and writes an initial value to PPB to ensure compatibility with the built-in keypad, display, and onboard high-current driver

interfaces. If the specified input/output configuration of the PIA is the same as the prior configuration, does not modify the PIA configuration register, and thus does not change the state of any output pins in PPA or upper PPC. If the specified PIA configuration is different than the prior configuration, INIT.PIA writes to the PIA's configuration register and this automatically zeros any outputs in PPA or upper PPC. Consult the Hardware Manual data sheet section for details of the PIA operation. INIT.PIA may interfere with accesses of the keypad, display, or high-current drivers that are in progress. INIT.PIA is called by INIT.RS485.
Pronunciation: "init-P-I-A"

INIT.RS485 (--)
Calls INIT.PIA to configure the peripheral interface adapter (PIA) so that it is consistent with operation of the RS485 circuitry, and then sets the RS485 transceiver to receive mode. Recall that INIT.PIA expects to see two flags on the stack: the first flag is true if PPA is to be an output, and the top flag on the stack is true if upper PPC is to be an output. INIT.RS485 sets the first flag to leave the data direction of PPA unchanged, and sets the top flag on the stack to TRUE to configure upper PPC as an output. PPC bit 4 controls the direction of the RS485 data transfer: when bit 4 of PPC is high, the RS485 port is in transmit mode, and when bit 4 of PPC is low, the RS485 port is in receive mode. (Make sure that the onboard RS485/RS232 jumper is properly set before attempting to use the RS485 interface). See INIT.PIA, RS485.RECEIVE, and RS485.TRANSMIT.
Pronunciation: "init-R-S-four-eighty-five"

INIT.SERIAL2 (--)
Initializes the secondary serial port (serial2) which is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). Clears the resource variable SERIAL2.RESOURCE to 0\0, initializes PARITY to OFF, initializes the transmit and receive buffers (80 characters each, located in the reserved system RAM), initializes the data directions of PA3 and PA4 as input and output, respectively, and locally enables the required interrupts associated with PA3 and PA4. Does not globally enable interrupts. The programmer must separately execute the BAUD2 command (to set the baud rate) and execute ENABLE.INTERRUPTS (to globally enable interrupts) before using the serial2 port. See USE.SERIAL2 and DISABLE.SERIAL2.
Pronunciation: "init-serial-two"

INIT.SPI (--)
Configures and enables the serial peripheral interface (SPI) so that it can transfer data from the 12 bit A/D (A/D12) and to the 8 bit digital to analog converter (DAC). The SPI uses bits 2-5 of PORTD. Initializes the 68HC11 as the SPI "master" with 2 MHz data transfer, with valid data present/sampled on the rising leading edge of the SPI clock. Initializes the contents of PORTD.DIRECTION to be compatible with being the master of the SPI (that is, PD2/MISO = input, PD3/MOSI = output, PD4/SCK = output, PD5/SS = output). Also initializes the resource variable SPI.RESOURCE to 0\0. See SPI.OFF.
Pronunciation: "init-S-P-I"

INIT.VITAL.IRQS.ON.COLD (--)
Undoes the effect of the NO.VITAL.IRQ.INIT command, and causes subsequent cold restarts to perform the default action of checking the interrupt vectors for the COP, clock monitor, illegal opcode and OC2 interrupts and initializing them if they do not

contain the standard interrupt service vectors. Implementation detail: sets location AE1B in EEPROM to 0xFF.
Pronunciation: "init-vital-i-r-qs-on-cold"

INPUT.STRING (<text> -- x$addr | x$addr = PAD)
Inputs a character string <text> to the PAD buffer, terminating when CHARS/LINE characters are received or a carriage return is received, whichever comes first. Leaves the address of the counted string on the stack. Appends a blank (not included in the count) to the end of the string.
Attributes: M

INSTALL.MULTITASKER (--)
Installs the timeslice multitasker timer by initializing the interrupt vector of the output compare 2 (OC2) timer. This command is automatically executed upon a COLD restart (unless the command NO.VITAL.IRQ.INIT has been executed) and by the command START.TIMESLICER. Because the interrupt vector is in non-volatile EEPROM, it is usually not necessary to invoke this command unless the OC2 interrupt vector has been modified.

INSTALL.REGISTER.INITS (byte1\byte2\byte3\byte4 -- | byte1=OPTION, byte2=TMSK2, byte3=BPROT, byte4 = BAUD)
Compiles a 7-byte sequence into the EEPROM that specifies the contents to be loaded into the "protected registers" plus the BAUD register after subsequent resets. The protected registers are those that must be initialized within 64 machine cycles after a reset; after that their contents cannot be changed. They are INIT, OPTION, TMSK2, and BPROT. The BAUD register controls the BAUD rate of the primary serial communications interface (serial1), and is included so that a user-specified baud rate can be set upon every restart. The INIT register controls the location of the on-chip RAM and the registers. This value is set to B8H (on-chip RAM at 0xB000, and registers at 0x8000); other values are not compatible with QED-Forth. The contents of the other 4 registers may be specified by the user. Once INSTALL.REGISTER.INITS is executed, subsequent resets will cause B8H to be stored in INIT, byte1 in OPTION, byte2 in TMSK2, byte3 in BPROT, and byte4 in BAUD. To undo the effects of this word and return to the default contents of the protected registers use the DEFAULT.REGISTER.INITS command; see its glossary entry for a list of the default values for each of the registers.
Implementation detail: INSTALL.REGISTER.INITS writes the pattern 0x13 at location 0xAE06 in the EEPROM. The five bytes following the pattern contain the specified contents of INIT (=B8H), OPTION, TMSK2, BPROT, and BAUD, respectively.

INT-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a 16-bit field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "integer" Attributes: D

INT.FLOOR (r -- n)
n is the greatest integer less than or equal to r. OVERFLOW is set if needed.
Attributes: S

INT.PART (r -- n)
n is the integer part of r. OVERFLOW is set if needed. (This function used to be named INT in QED-Forth versions up to V3.01.)
Pronunciation: "int" Attributes: S

INTEGER: (<name> --)
Defines a 16-bit self-fetching variable. Removes <name> from the input stream and creates a child word (a self-fetching variable) called <name> and VALLOTs 2 bytes in the variable area as the parameter field where the self-fetching variable's value is stored. When <name> is executed it leaves its value (a 16-bit integer) on the stack. Thus <name> behaves like a constant when executed. Unlike a constant, its parameter field is in the variable area and so can always be modified. The TO command is used to store a value into the self-fetching variable. In general, code using self-fetching variables runs faster than does similar code that uses standard variables. Use as:

```
INTEGER: <name>        <value>  TO  <name>
```

Pronunciation: "integer-colon" Attributes: D

INTERPRET (--)
Interprets the input stream as designated by BLK and >IN until the input stream is exhausted. If BLK = 0, the input stream is taken from the terminal input buffer and INTERPRET interprets a single line of input which was read into the TIB by QUERY. If BLK does not equal 0, INTERPRET interprets the specified block. INTERPRET repeatedly calls WORD (which gets the next word from the input stream) until the input stream is exhausted at the end of the current line or block. See BLK, >IN, BLOCK, WORD, LOAD, QUERY, and QUIT.

INTS-> (u1\u2 <name> -- u3)
Adds a named member to the structure being defined and reserves room for u2 integers in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "integers" Attributes: D

INVERTED (matrix.xpfa1\matrix.xpfa2 --)
Inverts the source matrix specified by matrix.xpfa1 and stores the result into the destination matrix specified by matrix.xpfa2. An error occurs if the source matrix is not square. matrix.xpfa2 may equal matrix.xpfa1.
Attributes: S

IRQ.ID (-- n)
Returns the interrupt identity code for the external interrupt request interrupt. Used as an argument for ATTACH.

Pronunciation: "i-r-q-i-d"

IS.DISPLAY (#rows\#cols\text.mode?\char.display?\hitachi? --)

Based on the specified number of rows, number of columns, and flags that indicate text or graphics mode, character versus graphics display, and Hitachi versus Toshiba graphics controller chip, this routine saves the display configuration in EEPROM so that the LCD display is properly initialized upon subsequent restarts and resets by the INIT.DISPLAY routine which is automatically executed at startup. The encoded information is accessible via the routines CHARS/DISPLAY.LINE and LINES/DISPLAY. When IS.DISPLAY is executed, #rows and #cols should be expressed as the number of 8x6- or 8x8-pixel characters that the screen can accommodate. The standard width font for Toshiba graphics displays is set by hardware inputs on the display module to either 6 or 8 pixels wide. The standard width font for Hitachi graphics displays is 8 pixels in graphics mode, and can be set by software to either 6 pixels or 8 pixels wide in text mode. The allowed values of numRows are 2, 4, 8 or 16 lines per display. The allowed values of numCols are 8, 12, 16, 20, 24, 30, and 40 characters or bytes per line. The text.mode? flag selects between text mode (if the flag is true) and graphics mode (if the flag is false) for graphics displays; character displays always operate in text mode. The character.display? flag selects between a strictly alphanumeric character display if the flag is true, and a graphics display if the flag is false. The hitachi? flag specifies the type of controller that drives the graphics display module. If the hitachi? flag is true, a Hitachi 61830 controller chip is assumed; if the hitachi? flag is false, we assume a Toshiba 6963 graphics controller chip. NOTE that if a graphics display is specified (char.display? is false) but the text mode is specified (text.mode? is true), the data buffer created by INIT.DISPLAY in the DISPLAY.HEAP will be too small to accommodate graphical data. Thus if you want to use both the text and graphics modes of a graphics display, declare a graphics mode display (i.e., with a false text.mode? flag), and use the DISPLAY.OPTIONS routine to convert to and from text mode. Then the dimensioned buffer will be large enough for either character or graphical data. The following routines specify the most commonly used displays:

```
    DECIMAL
: 4X20.CHARACTER          4 20 -1 -1  -1     IS.DISPLAY ;
: 128X240.HITACHI.GRAPHICS      16 30  0  0  -1    IS.DISPLAY ;
: 128X240.HITACHI.TEXT   16 40 -1  0  -1    IS.DISPLAY ;
: 128X240.TOSHIBA.GRAPHICS     16 40  0  0  0     IS.DISPLAY ;
: 128X240.TOSHIBA.TEXT  16 40 -1  0  0     IS.DISPLAY ;
: 128X128.HITACHI.GRAPHICS      16 16  0  0  -1    IS.DISPLAY ;
: 128X128.HITACHI.TEXT   16 20 -1  0  -1    IS.DISPLAY ;
```

The 4x20 character display is the default type that is established by the "special cleanup mode". Remember to execute INIT.DISPLAY after executing IS.DISPLAY the first time. Note that because IS.DISPLAY saves the configuration information in EEPROM, you need not execute IS.DISPLAY each time the board starts up. INIT.DISPLAY is automatically executed each time the QED Board starts up.

Implementation detail: This routine encodes the configuration information in a single byte that is saved at location AE1EH in EEPROM.

IS.DISPLAY.ADDRESS (addr --)

Configures a graphics display so that the next data write will occur at the specified RamAddress in the display RAM. This routine can be used in conjunction with (UPDATE.DISPLAY) to write data to the "off-screen" RAM that is typically present on a graphics display module. Then modifying the "home address" (upper left location) of the display allows scrolling of data across the display; see the source code of the graphics extension source code file for more details. IsDisplayAddress() has no effect if a character display is installed. See also PUT.CURSOR .

IS.HEAP (xaddr1\xaddr2 -- | xaddr1 = start, xaddr2 = end)

Initializes the heap control variables to set up a heap starting at xaddr1 and ending 1 byte below xaddr2. All of the bytes between xaddr1 and xaddr2 must be modifiable RAM. The size of the heap and of individual heap items is limited only by available memory. If the specified heap size (xaddr2 - xaddr1) is greater than or equal to 16 bytes, IS.HEAP initializes CURRENT.HEAP to xaddr2, initializes START.HEAP and HEAP.PTR to xaddr1, and initializes HANDLE.PTR and FREE.HANDLE to indicate that there are no allocated heap items. If the specified heap size (xaddr2 - xaddr1) is less than 16 bytes, only the user variable CURRENT.HEAP is initialized, and the heap control variables that are stored in the heap itself (START.HEAP, HEAP.PTR, HANDLE.PTR and FREE.HANDLE) are not initialized. This allows tasks to share a heap that has already been initialized without disturbing the values of the heap control variables. Caution: sharing a heap among tasks may lead to hard-to-diagnose multitasking failures. Consult the chapters on multitasking and re-entrant coding in the Software Manual when designing multitasking programs.
Pronunciation: "is-heap"

IS.IDENTITY (matrix.xpfa --)

Stores 1.0 in all the elements of the specified matrix that have row# equal to col# (i.e., in the diagonal elements), and stores 0.0 in all the other elements. The matrix need not be square.
Attributes: S

IS.TRACE.ACTION (xcfa --)

Installs the action referenced by the extended code field address xcfa as the first action to be performed by the trace routine. If a colon or code definition is compiled with TRACE ON, a call to the trace routine is compiled before each call to subsidiary words in the definition. The first thing that the trace routine does is to execute the code whose xcfa has been stored in the headerless user variable TRACE.ACTION by IS.TRACE.ACTION. The default action (set upon COLD restart) is NO.OP. The action must have no net effect on the data or return stacks, and must be compiled while TRACE is OFF; if trace is ON while the trace action is being defined, an infinite loop of calls to the trace routine is initiated. The word DEFAULT.TRACE.ACTION installs NO.OP (a do-nothing word) as the trace action. See also BREAK, DEBUG, DUMP.REGISTERS, and SINGLE.STEP.
Example of use: assume that a complex program contains a mysterious bug, and the programmer is trying to locate it. Further assume that the programmer knows that the bug causes a particular variable called THE.VARIABLE to be set equal to 0. One way to locate the problem would be to compile the program with TRACE ON and then execute the entire program with DEBUG ON. The disadvantage of this is that a great deal of traced output must be examined. A more efficient method is to write a word that tests the variable in question and, if it equals 0 (which signals that the bug has just occurred), turns DEBUG ON:

```
    : CHECK.FOR.ERROR    THE.VARIABLE  @  IF  DEBUG  ON  THEN ;
```

This word is then installed as the trace action by executing:

```
CFA.FOR   CHECK.FOR.ERROR   IS.TRACE.ACTION
```

Now the program is compiled with TRACE ON, and executed with DEBUG initially OFF so that the program runs at near full speed and no traced output is written to the screen. As soon as the bug manifests itself, however, the trace action word turns DEBUG ON which initiates the printout of all trace information. This shows the programmer where the bug is occurring in the program.

IXN+ (xaddr1 -- xaddr2 | xaddr2 = xaddr1 + I)
Return Stack: (R: w1 -- w1)
Used inside a DO...LOOP or DO...+LOOP or FOR ... NEXT structure. Adds the current (innermost) loop index I, interpreted as a signed integer in the range -32,768 to +32,767, to xaddr1 and places the result xaddr2 on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:

```
DO ... addr1  IXN+ ...  LOOP
```

See XN+
Pronunciation: "i-x-n-plus" Attributes: C

IXN- (xaddr1 -- xaddr2 | xaddr2 = xaddr1 - I)
Return Stack: (R: w1 -- w1)
Used inside a DO...LOOP or DO...+LOOP or FOR ... NEXT structure. Subtracts the current (innermost) loop index I, interpreted as a signed integer in the range -32,768 to +32,767, from xaddr1 and places the result xaddr2 on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:

```
DO ...xaddr1  IXN- ...  LOOP
```

See XN-
Pronunciation: "i-x-n-minus" Attributes: C

IXU+ (xaddr1 -- xaddr2 | xaddr2 = xaddr1 + I)
Return Stack: (R: w1 -- w1)
Used inside a DO...LOOP or DO...+LOOP or FOR ... NEXT structure. Adds the current (innermost) loop index I, interpreted as an unsigned integer in the range 0 to +65,535, to xaddr1 and places the result xaddr2 on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:

```
DO ... xaddr1  IXU+ ...  LOOP
```

See XU+
Pronunciation: "i-x-u-plus" Attributes: C

IXU- (xaddr1 -- xaddr2 | xaddr2 = xaddr1 - I)
Return Stack: (R: w1 -- w1)
Used inside a DO...LOOP or DO...+LOOP or FOR ... NEXT structure. Subtracts the current (innermost) loop index I, interpreted as an unsigned integer in the range 0 to +65,535, from xaddr1 and places the result xaddr2 on the data stack. Cannot be used if additional items have been placed on the return stack (for example, using >R). Typical use:

```
DO ... xaddr1  IXU- ...  LOOP
```

See XU-
Pronunciation: "i-x-u-minus" Attributes: C

I\J (-- w1\w2 | w1 = I, w2 = J)
Return Stack: (R: w3\w2\w4\w1 -- w3\w2\w4\w1)
Used inside nested DO...LOOP or DO...+LOOP. Places on the data stack a copy of the loop index for the current (innermost) loop under a copy of the loop index for the next outer loop. Equivalent to I J. May not be used if additional items have been placed on the return stack (for example, using >R). May not be used in FOR...NEXT loops. Typical use:

```
    DO
            DO  ... I\J ...
            LOOP
    LOOP
```

Pronunciation: "i-under-j" Attributes: C

J (-- w1)
Return Stack: (R: w2\w1\w3\w4 -- w2\w1\w3\w4)
Used inside nested DO...LOOP or DO...+LOOP. Places on the data stack a copy of the loop index for the next outer loop (i.e., the index of the loop nested 1 level below the loop in which J is invoked). Cannot be used if additional items have been placed on the return stack (for example, using >R). Cannot be used in FOR...NEXT loops. Typical use:

```
    DO
            DO  ... J ...
            LOOP
    LOOP
```

Attributes: C

J\I (-- w1\w2 | w1 = J , w2 = I)
Return Stack: (R: w3\w1\w4\w2 -- w3\w1\w4\w2)
Used inside nested DO...LOOP or DO...+LOOP. Places on the data stack a copy of the loop index for the next outer loop under a copy of the loop index for the current (innermost) loop. Equivalent to J I. Cannot be used if additional items have been placed on the return stack (for example, using >R). Cannot be used in FOR...NEXT loops. Typical use:

```
    DO
            DO  ... J\I ...
            LOOP
    LOOP
```

Pronunciation: "j-under-i" Attributes: C

K (-- w1)
Return Stack: (R: w2\w1\w3\w4\w5\w6 -- w2\w1\w3\w4\w5\w6)
Used inside nested DO...LOOP or DO...+LOOP. Places on the data stack a copy of the loop index for the second outer loop (i.e., the index of the loop nested 2 levels below the loop in which K is invoked). Cannot be used if additional items have been placed on the return stack (for example, using >R). Cannot be used in FOR...NEXT loops. See I and J. Typical use:

```
    DO
            DO
                    DO  ... K ...
                    LOOP
            LOOP
        LOOP
```
Attributes: C

KEY (-- char)
Waits (if necessary) for receipt of a character from the serial port and places the character on the data stack. KEY is a vectored routine that executes the routine whose xcfa is stored in the headerless user variable UKEY. The default installed routine called is KEY1 which receives the character from the primary serial port (supported by the 68HC11's hardware UART). KEY2 may be installed in UKEY by USE.SERIAL2 or SERIAL2.AT.STARTUP; KEY2 receives the character from the secondary serial port (supported by QED Forth's software UART and using pins PA3 and PA4). See KEY1 and KEY2.
Attributes: M, U

KEY1 (-- char)
Waits (if necessary) for receipt of a character from the primary serial port (serial1) and places the received character on the data stack. KEY1 does not echo the character. The serial1 port is associated with the 68HC11's on-chip hardware UART. KEY1 is the default KEY routine installed in the UKEY user variable if SERIAL1.AT.STARTUP has been executed (and after the special cleanup mode is invoked). If the value in SERIAL.ACCESS is RELEASE.AFTER.LINE, KEY1 does not GET or RELEASE the SERIAL1.RESOURCE. If SERIAL.ACCESS contains RELEASE.ALWAYS, KEY1 GETs and RELEASEs the SERIAL1.RESOURCE. If SERIAL.ACCESS contains RELEASE.NEVER, KEY1 GETs but does not RELEASE the SERIAL1.RESOURCE. See KEY, UKEY, KEY2, and SERIAL.ACCESS.
Pronunciation: "key-one" Attributes: M

KEY2 (-- char)
Waits (if necessary) for receipt of a character from the secondary serial (serial2) port, removes the character from the serial2 input buffer and places the received character on the data stack. The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). KEY2 does not echo the received character. KEY2 can be made the default KEY routine installed in the UKEY user variable after each reset or restart by executing SERIAL2.AT.STARTUP. If the value in SERIAL.ACCESS is RELEASE.AFTER.LINE, KEY2 does not GET or RELEASE the SERIAL2.RESOURCE. If SERIAL.ACCESS contains RELEASE.ALWAYS, KEY2 GETs and RELEASEs the SERIAL2.RESOURCE. If SERIAL.ACCESS contains RELEASE.NEVER, KEY2 GETs but doesn't RELEASE the SERIAL2.RESOURCE. See KEY, UKEY, KEY1, and SERIAL.ACCESS.
Pronunciation: "key-two" Attributes: M

KEYPAD (-- n | 0 <= n <= 19)
Scans the 4 X 5 keypad or touchscreen and waits for a keypress. PAUSEs while waiting. Waits until the key is released, then returns the key number on the data stack. Consult the hardware manual for a detailed description of key placement. Briefly, in the standard QED Product Design Kit or ICS orientation with 4 rows and 5 columns and the keypad connector at the bottom, key #0 is in the lower right corner, key #1 is just above it, and key #19 is in the upper left corner. Disables interrupts for 48 cycles (12 microseconds) each time a row is scanned. See ?KEYPAD and ?KEYPRESS.

KILL (xtask.id --)
Puts the task referenced by xtask.id ASLEEP and removes it from the round robin multitasking loop. The task to be killed must be installed in the round robin loop. If it isn't, or if a task attempts to KILL itself, the results are unpredictable. Aborts if xtask.id is not in common ram.

L.INVERTED (matrix.xpfa1\matrix.xpfa2 --)
Inverts a lower triangular matrix specified by matrix.xpfa1 and puts the inverse in the destination matrix.xpfa2. Ignores the upper half of the source matrix so it doesn't matter if the elements are not truly zero.
Attributes: S

LATEST (-- xnfa)
Returns the extended name field address of the last word defined in the CURRENT vocabulary.
Attributes: U

LEAST.SQUARES (X.matrix.xpfa\ Y.matrix.xpfa
\ C.matrix.xpfa\ Sample.Variance.matrix.xpfa
\ [Weight.matrix.xpfa] or [0\0] \ [Y.model.xpfa] or [0\0]
\ [Y.err.matrix.xpfa] or [0\0] \ [Covar.matrix.xpfa] or [0\0] --)
Performs general linear least squares data analysis by solving an overdetermined set of simultaneous equations in which instances of independent variables (or functions of independent variables), x, are related to instances of dependent variables (or functions of dependent variables), y, by a set of linear coefficients, c. The overdetermined system of equations is represented in matrix form as
Y = X C
Consult the software manual for a detailed example of use.
Although the matrix of dependent variables, Y, is usually only a single column, this routine can also do a number of least squares problems simultaneously, for different dependent variables or functions of dependent variables, each contained in a different column of Y. Y contains a row for each instance of the Y variable(s). X is the design matrix, a user-supplied matrix that includes a column for each independent variable or function of independent variable(s). Each row of X and Y corresponds to a single "trial", "instance", or "measurement".
C is the matrix designated by the user to contain the primary result returned by this routine, the coefficients. It contains a row for each column of X and a column for each column of Y. Most problems require only a single dependent variable so C is generally a single-column matrix.
Optionally an input matrix of weights that is parallel to the Y matrix may be included. The weights should be inversely proportional to the expected variances on the instances of Y. Weights are considered to be relative; their absolute magnitudes are ignored and only their relative proportions are used by LEAST.SQUARES. Internally to LEAST.SQUARES the user's weight matrix is normalized so that the column-averaged weight is unity, that is, the sum of weights in each column is made equal to the number of instances (rows) in the column. Uniform weighting is assumed if a weight matrix is not included.
LEAST.SQUARES computes values of the coefficient matrix C that minimizes the observed sample variance of the data points from the fitted model. Observed sample

variance is defined as the weighted sum of squares of the differences between the Y data and the modeled Y, divided by the number of degrees of freedom in order to get a variance per instance. For each column of Y the observed sample variance is

Sample.Error^2 = ∑ wi (yi - yimodel)^2 /(n-m)

where wi are the normalized weights, yi are the input y values (i.e., the contents of the Y matrix), yimodel are the predicted values of the dependent variable(s) computed from Y = X C, and (n-m) is the number of degrees of freedom in the optimization, equal to the number of instances (rows of X or Y) minus the number of coefficients determined (columns of X or rows of C). The routine reports a value for observed sample variance for each column of Y in the user's Sample.Variance.matrix. If the user provides appropriate matrix.xpfas LEAST.SQUARES also reports the predicted Y values in Y.model.matrix, the residuals or errors on Y, (Y-Ymodel), in Y.err.matrix, and the variance-covariance matrix for the determined coefficients in Covar.matrix.

If the user provides no weights, or for uniform weighting, or for all weight columns equal to one another, the returned variance-covariance matrix provides the relative covariances of the coefficients in any of the columns of the coefficient matrix, C. There are two ways to compute the actual variances and covariances for a particular column of C. If coefficient variances are to be determined using the actual deviations of the model from the best fit then the covariance matrix should be multiplied by the observed sample variance for the corresponding column of Y, found in Sample.Variance.matrix. Alternatively, if the coefficient variances are to be determined a priori from the expected experimental or measurement errors then the covariance matrix should be multiplied by the experimenter's expected variances on the Y values. After being scaled with the observed sample variance or the a priori measurement variances, the diagonal elements of the variance-covariance matrix are the variances on the coefficients of C, and their square roots are the expected errors on the coefficients.

For weights that differ from column to column the covariance matrix returned is that for only the first column of C, corresponding to the first column of Y or W.

The user's X, Y, C, and Weight matrices are not modified.

Attributes: S

LEAVE (--)

Return Stack: (R: w1\w2 -- | discards limit & index)

Forces the immediate termination of a DO...LOOP or DO...+LOOP structure. Discards the loop control parameters from the return stack and transfers control to the word following LOOP or +LOOP. An error is issued if LEAVE is used outside a DO...LOOP or DO...+LOOP structure. LEAVE cannot be used inside a CASE statement which is inside a DO LOOP or DO +LOOP structure. An unchecked error occurs if extra items are on the return stack when LEAVE executes (for example, items put on the return stack by >R). See DO, LOOP and +LOOP. Use as:

```
w1 w2  DO      ...
               flag  IF  LEAVE ENDIF
               ...
       LOOP
```

Attributes: C, I

LEFT.PLACES (-- xaddr)

A user variable that holds the number of digits to be displayed to the left of the decimal point when a floating point number is displayed in FIXED format. See F>FIXED$

Attributes: U

LIMIT (-- xaddr)

Returns the extended address of the upper boundary+1 of the block buffers; that is, the address one greater than the last byte in the last buffer. Initialized by BLOCK.BUFFERS. Equivalent to ULIMIT X@

Pronunciation: "paren-limit" Attributes: U

LINES/DISPLAY (-- n)

Returns the number of lines in the LCD display. For character displays and for graphics displays being operated in "text mode", the result n equals the number of character lines (rows) in the display (the allowed values are 2, 4, 8 or 16 lines per display). For graphics displays being operated in "graphics mode", the result n equals the number of horizontal pixels on the display (which in turn is 8 times the number of character lines on the display). The type of display and the display mode (text mode vs. graphics mode) are determined by the most recent execution of DISPLAY.OPTIONS or INIT.DISPLAY (which implements the configuration specified by IS.DISPLAY). The default value of n after executing the "special cleanup mode" is 4, corresponding to the default 4-line by 20-character display. The result returned by this routine is used by BUFFER.POSITION, PUT.CURSOR, UPDATE.DISPLAY, and UPDATE.DISPLAY.LINE.

Pronunciation: "lines-per-display"

LINK (xnfa1\xnfa2 --)

Sets the link field in the name of the word referenced by name field address xnfa2 to point to the name referenced by xnfa1. In other words, it changes the way that backward-linked name list or vocabulary is searched so that the name associated with xnfa1 is encountered just after the name associated with xnfa2 (see FIND). LINK can be used in several interesting ways. For example, if xnfa1 = xnfa2, LINK designates the associated name as the bottom name in a linked list, effectively "sealing" the linked name list or vocabulary. For example, to define a new sealed vocabulary, execute

```
VOCABULARY NEW.VOCAB
NEW.VOCAB DEFINITIONS   FORTH
\ new definitions now go into NEW.VOCAB,
\ but FORTH words can be accessed in the definitions
: #1.WORD   ....   ;
: #2.WORD   ....   ;
\ define as many words as needed in NEW.VOCAB

NFA.FOR  #1.WORD  XDUP  LINK        \ seal the vocabulary
```

Make sure that xnfa2 is in modifiable RAM memory when LINK is executed. Alternatively, instead of sealing the vocabulary by making #1.WORD the bottom name, we could execute

```
NFA.FOR  FORTH    NFA.FOR  #1.WORD    LINK
```

to link #1.WORD to FORTH which is the very first (bottom) name in the QED-Forth name list. This has the advantage of keeping the vocabulary's contents small while allowing FORTH to be recognized from within the new vocabulary. This allows graceful exit to the main FORTH vocabulary after executing NEW.VOCAB DEFINITIONS.

LINK_FILE_IO (--)

This function (present in kernel versions starting with V4.05) links in the Forth headers of the Memory Interface Board's ATA Flash Card Software Package into the linked list of Forth names. See the MIB/ATA Flash Card Users Manual for information on how to use this function. Because the Memory Interface Board has been obsoleted by the Compact Flash Wildcard and its associated kernel extension software, this function is typically not used.

LITERAL (-- w)
Compile Time: (w --)
If QED-Forth is in execution mode when LITERAL is invoked, LITERAL does nothing. If QED-Forth is in compilation mode, LITERAL removes w from the stack and compiles it into the dictionary along with code that, when later executed, pushes w to the stack. LITERAL can be used within a colon definition to compile a numeric value into the definition. For example,

```
    : <name>        ( -- n )
            [ CHARS/LINE @ ] LITERAL
    ;
```

This compiles as a literal the value of CHARS/LINE that exists when the definition is compiled. When <name> is later executed, this value will be placed on the stack.
Attributes: I

LN(2) (-- r)
Places the floating point representation of the natural logarithm of 2 (0.69314) on the stack.
Pronunciation: "l-n-of-two"

LOAD.MATRIX (xpfa <text> -- | expects numbers in input stream)
Repeatedly calls NEXT.NUMBER to get as many numbers as needed to fill the matrix specified by xpfa. The first row is filled first (starting at the left, i.e., at the first column), then the next row, etc. Carriage returns and tabs can be used to format the numbers if desired. The numbers can be valid integers or floating point numbers; NEXT.WORD automatically converts the integers to their floating point equivalents using FLOT. Non-numeric text in the input is ignored until the matrix is filled, but be careful: any valid number encountered by LOAD.MATRIX will be converted and placed in the matrix, even if it is within parentheses or after a comment delimiter such as \. Integers are converted according to the current number base. Use as

```
MATRIX: MAT.A    2 3 ' MAT.A DIMMED       \ def & dimension
' MAT.A  LOAD.MATRIX 99  3  4.2  5.3  8 3  \ init matrix
```

Attributes: M, S

LOCALS{ ([...] -- | items are removed to initialize the local variables)
Return Stack: (R: -- [...] | stack frame is placed on return stack,removed by ;)
Used within a colon definition to begin a command sequence that defines from 1 to 16 local variables. The syntax of the simplest type of local statement is:

```
    LOCALS{ <name.1> <name.2>... <name.n> }
```

The names between LOCALS{ and the terminating } are given temporary headers that remain effective for the duration of the current definition. If the first 2 characters in the name of a local are d& or D& (suggesting a 32-bit double number value) or f& or F& (suggesting a 32-bit floating point value) or x& or X& (suggesting a 32-bit extended address) then the local is defined as a 32-bit quantity. All other local variables are

defined as 16-bit quantities. At runtime (when the definition containing the LOCALS{ command is executed) the code compiled by LOCALS{ initializes the values of each of the named local variables to a corresponding item on the data stack. The top item is removed from the data stack and loaded into <name.1>, the next item on the data stack is removed and loaded into <name.2>, etc. A single cell is removed from the stack for each 16-bit local, and two cells are removed from the stack for each 32-bit local. Each local variable behaves just like a self-fetching variable. When executed, the local leaves its value on the stack. Its value can be initialized or modified by placing the desired value on the stack and invoking the TO operator followed by the name of the local variable.
An alternative syntax for the locals declaration is:

    LOCALS{ <name.1> <name.2> ... | <name.a> <name.b> ... }

where the locals defined between LOCALS{ and the | (bar) are initialized to the values on the data stack as explained above, and the locals defined between the | and the terminating } are left uninitialized. If certain local variables are to be initialized by a TO statement later in the definition, it is better to define them as uninitialized locals to save execution time. If none of the locals need to be initialized, the syntax:

    LOCALS{ | <name.a> <name.b> ... }

is acceptable. Locals can be used in recursive definitions; each execution of the word allocates a new frame on the return stack where the values of the locals are kept.
Usage rules: The locals command sequence,

    LOCALS{ ... | ... }

may occupy multiple lines in the source code; however, no comments may appear between LOCALS{ and }. If using a disk block for input, the command must reside in a single block. To avoid problems with non-uniquely named local variables the convention of using names beginning with & is recommended. Only one

    LOCALS{ ... | ... }

command sequence is allowed per definition and within the declaration only one | is allowed. The LOCALS{ declaration must not be inside a conditional structure such as an IF ... ELSE ... ENDIF structure. The number of local variables defined must be greater than 0 and less than or equal to 16. The local variable names defined in the body of a word are forgotten after the definition is finished. Colon definitions using locals should not call CREATE or other defining words and should not manipulate CURRENT. Locals cannot be used between <DBUILDS or <VBUILDS and DOES> in a defining word. Locals should not be used in interrupt service routines or in code called by interrupt service routines.
Pronunciation: "locals-start" Attributes: C, D, I

LOG10(2) (-- r)
Pushes the base-10 logarithm of 2 onto the data stack.
Pronunciation: "log-ten-of-two"

LOG2 (u -- n)
n is the integer part of the base 2 logarithm of u.
Pronunciation: "log-two"

LOOP (--)
Return Stack: (R: w1\w2 -- [w1\w2] or [] | drops w1,w2 when loop terminates)
Used inside a colon definition to mark the end of a counted loop that was started by DO. Increments the loop index on the return stack. If the incremented index is equal to the

loop limit, terminates the loop by allowing execution to continue with the word following LOOP. If the incremented index is less than the limit, transfers control to the word immediately following DO to continue looping. Use as:

```
w1 w2  DO      words to be executed (w1 - w2 - 1) times
       LOOP
```

An error is issued if DO and LOOP are not paired within a definition. See also DO I J K I' LEAVE
Attributes: C, I

LU.BACKSUBSTITUTION (residue.matrix.xpfa\matrix.xpfa\index.array.xpfa --)

Used in conjunction with LU.DECOMPOSITION to solve a set of simultaneous equations. This solution method is very efficient when solving a system of equations with multiple right hand sides (i.e., multiple residues). residue.matrix.xpfa specifies a column matrix containing the right hand side of the system of equations, matrix.xpfa specifies a matrix containing the LU decomposition of the coefficient matrix, and index.array.xpfa specifies an array that was dimensioned and initialized by LU.DECOMPOSITION. LU.BACKSUBSTITUTION replaces residue.matrix.xpfa with the solution of the equation (i.e., with the values of the unknown quantities). For application information and an example, see LU.DECOMPOSITION.
Pronunciation: "l-u-back-substitution" Attributes: S

LU.DECOMPOSITION (index.array.xpfa\matrix.xpfa -- n | n = determinant's.sign)

Finds the LU ("lower/upper") decomposition of the source matrix specified by matrix.xpfa and stores the decomposed matrix back into the source matrix. Also calculates the sign n of the determinant of the source matrix which can be passed to ?DETERMINANT to find the determinant of the source matrix. Dimensions and initializes the array associated with index.array.xpfa to contain values that are then used by LU.BACKSUBSTITUTION.
Application: Sometimes it is necessary to solve several systems of equations, each having the same left hand sides but with different right hand sides, or residue matrices. The system of equations is represented by the matrix equation

$$AX=B$$

where X is a column matrix of unknowns, A is a matrix of coefficients, and B is a column matrix known as the "residue matrix" that represents the right hand side of the equations. A second set of equations with a different residue matrix could be represented as

$$AX' = C$$

where C is the residue matrix and X' is the matrix of unknowns that solves the equation. These sets of equations are most efficiently solved by finding the LU decomposition of A using the word LU.Decomposition. "LU" refers to "lower/upper", a decomposition method that minimizes the propagation of round-off error. Now the matrix equations can be expressed as

$$LU(A)\ X= B$$
$$LU(A)\ X'= C$$

where LU(A) is the decomposition of the A matrix. These equations can be solved for the unknown matrix X with a minimum of computation using the word LU.BACKSUBSTITUTION. To solve a given equation for several residue matrices, LU(A) needs only to be computed once. LU.BACKSUBSTITUTION can then solve each residue matrix to get each solution matrix with minimal additional work.
Example of use:

```
ARRAY: INDEX.ARRAY      \ define this temporary array
MATRIX: A <initialize A here> \ define & init coefficients
MATRIX: B <initialize B here> \ define & init residue #1
MATRIX: C <initialize C here> \ define & init residue #2
' INDEX.ARRAY ' A LU.DECOMPOSITION
                        \ replace A with its LU decomposition
DROP                    \ drop the determinant sign
' B ' A ' INDEX.ARRAY LU.BACKSUBSTITUTION
            \ for residue matrix B, find the unknowns X
            \ and place them in B
CR ." The answer to AX = B is " ' B  M.        \ print solution
' C ' A ' INDEX.ARRAY LU.BACKSUBSTITUTION
                        \ for residue matrix C, find the
                        \ unknowns X' and place them in C
CR ." The answer to AX' = C is " ' C  M.       \ print solution
```

Pronunciation: "l-u-decomposition"   Attributes: S

M* (n1\n2 -- d)
Multiply signed single precision integers n1 and n2 producing signed double precision product d.
Pronunciation: "m-star"

M*MT (matrix.xpfa1\matrix.xpfa2 --)
Multiplies the source matrix.xpfa1 by its transpose to form the specified destination matrix.xpfa2. The name of the routine suggests the order of the operands. See MT*M
Pronunciation: "m-star-m-transpose"Attributes: S

M. (matrix.xpfa --)
Prints the contents of the matrix specified by matrix.xpfa using the default printing format (FIXED, FLOATING, or SCIENTIFIC). The print is performed with FILL.FIELD ON so that tabular format (constant field width, decimal alignment) is maintained. Printed elements are delimited with spaces and with carriage returns and continuance marks (...). Output line length is limited to 6 characters less than the contents of the user variable CHARS/LINE. M. calls the word PAUSE.ON.KEY, so the print responds to XON/XOFF handshaking and can be aborted by typing a carriage return; see PAUSE.ON.KEY.
Pronunciation: "m-dot"   Attributes: M, S

M.. (matrix.xpfa --)
Prints the name of the matrix specified by matrix.xpfa followed by a display of its contents as performed by M. See M.
Pronunciation: "m-dot-dot"   Attributes: M, S

M.PARTIAL (low.row#\high.row#\low.col#\high.col#\matrix.xpfa --)
Prints a portion of the matrix specified by low.row# through high.row#, inclusive and low.col# through high.col#, inclusive, using the default print format (FIXED, FLOATING, or SCIENTIFIC). The print is performed with FILL.FIELD ON so that tabular format (constant field width, decimal alignment) is maintained. Printed elements are delimited

with spaces and with carriage returns and continuance marks (...). Output line length is limited to 6 characters less than the contents of the user variable CHARS/LINE.
Typical use: Assume that MAT.A is dimensioned to have 8 rows and 9 columns. The following command prints the last 5 elements in each of the first 3 rows:

```
0 2  4 8  ' MAT.A   M.PARTIAL
```

M.PARTIAL calls the word PAUSE.ON.KEY, so the print responds to XON/XOFF handshaking and can be aborted by typing a carriage return; see PAUSE.ON.KEY.
Pronunciation: "m-dot-partial" Attributes: M, S

M/MOD (d1\n1 -- n2\d2 | n2 = remainder, d2 = quotient)
Divides double number d1 by integer n1 giving integer remainder n2 and double number quotient d2. Uses signed math. Division by zero (n1 = 0) yields n2 = 0XFFFF and d2 = FFFF0XFFFF.
Pronunciation: "m-slash-mod"

MAILBOX: (<name> --)
Removes the next <name> from the input stream, defines a child word called <name>, and VALLOTs 2 cells in the variable area. When <name> is executed, it leaves the extended address xaddr of the reserved cells that hold the mailbox's contents. <name> is referred to as a "mailbox". Use as:

```
MAILBOX: <name>
```

Mailboxes are used in multitasked systems to share information between tasks and to synchronize tasks to one another. If the mailbox's contents equal 0\0, the mailbox is empty; it contains a message if its contents are non-zero. Before its first use, the mailbox must be initialized to 0\0. After initialization to 0\0, the only operators that should access the mailbox are SEND ?SEND RECEIVE and ?RECEIVE. Consult the multitasking chapter for applications information and examples.
Attributes: D

MANTISSA.PLACES (-- xaddr)
A user variable that holds the number of digits to be displayed in the mantissa when a floating point number is displayed in SCIENTIFIC format. See F>FLOATING$
Attributes: U

MATRIX (<text> -- | expects name and numbers in input stream)
Expects the first word in the input stream following MATRIX to be the name of a dimensioned matrix. Skips the word following the matrix name (typically =) and then repeatedly calls NEXT.NUMBER to get as many numbers as needed to fill the matrix. The first row is filled first (starting at the left, i.e., at the first column), then the next row, etc. Carriage returns and tabs can be used to format the numbers if desired. The numbers can be valid integers or floating point numbers; NEXT.WORD automatically converts the integers to their floating point equivalents using FLOT. Non-numeric text in the input is ignored until the matrix is filled, but be careful: any valid number encountered by MATRIX will be converted and placed in the matrix, even if it is within parentheses or after a comment delimiter such as \. Integers are converted according to the current number base. Use as

```
MATRIX: MAT.A    2 3 ' MAT.A DIMMED \ define and dimension
MATRIX MAT.A  =  99  3  4.2  5.3  8 \ initialize matrix
```

Attributes: M, S

MATRIX* (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Performs a matrix multiplication of the two source matrices specified by matrix.xpfa1 and matrix.xpfa2, and dimensions and places the result in the destination specified by matrix.xpfa3. If matrix1 has p rows and m columns, and matrix2 has m rows and n columns, then matrix 3 will be dimensioned to have p rows and n columns. The number of columns in matrix1 must equal the number of rows in matrix2. If DEBUG is ON, aborts if the dimensions of the two sources are incompatible. The destination may be one of the sources.
Pronunciation: "matrix-star" Attributes: S

MATRIX+ (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Adds each element of the source1 matrix specified by matrix.xpfa1 to the corresponding element of the source2 matrix specified by matrix.xpfa2 and stores the result in the corresponding element in the destination specified by matrix.xpfa3. Dimensions the destination to have the proper dimensions. The destination may be one of the sources. If DEBUG is ON, aborts if source1 and source2 dimensions are incompatible.
Pronunciation: "matrix-plus" Attributes: S

MATRIX- (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Subtracts each element of the source2 matrix specified by matrix.xpfa2 from the corresponding element of the source1 matrix specified by matrix.xpfa1 and stores the result in the corresponding element in the destination specified by matrix.xpfa3. Dimensions the destination to have the proper dimensions. The destination may be one of the sources. If DEBUG is ON, aborts if source1 and source2 dimensions are incompatible.
Pronunciation: "matrix-minus" Attributes: S

MATRIX->V (matrix.xpfa -- xvaddr\sep\d.#el)
Converts a matrix specified by matrix.xpfa into the equivalent vector representation specified by xvaddr1\sep\d.#el.
Note: xvaddr is the base address of the vector, sep is the element separation expressed as a multiple of 4 bytes (e.g., sep=1 means a vector of contiguous floating point numbers, sep=2 means elements are separated by 8 bytes, etc.) and d.#el a double number that specifies the number of elements in the vector. Note that xvaddr must be 4-byte aligned (i.e., must be an even multiple of 4). The heap manager and array and matrix dimensioning words automatically perform the required 4-byte alignment.
Pronunciation: "matrix-to-v"

MATRIX.ELEMENT* (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Multiplies each element of the source1 matrix specified by matrix.xpfa1 by the corresponding element of the source2 matrix specified by matrix.xpfa2 and stores the result in the corresponding element in the destination specified by matrix.xpfa3. Dimensions the destination to have the proper dimensions. The destination may be one of the sources. If DEBUG is ON, aborts if source1 and source2 dimensions are incompatible. This operation should not be confused with a "matrix multiplication"; see MATRIX* .
Pronunciation: "matrix-element-star" Attributes: S

MATRIX.ELEMENT/ (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)

Divides each element of the source1 matrix specified by matrix.xpfa1 by the corresponding element of the source2 matrix specified by matrix.xpfa2 and stores the result in the corresponding element in the destination specified by matrix.xpfa3. Dimensions the destination to have the proper dimensions. The destination may be one of the sources. If DEBUG is ON, aborts if source1 and source2 dimensions are incompatible.
Pronunciation: "matrix-element-divide" Attributes: S

MATRIX.MAX (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Performs FMAX on each pair of corresponding elements in the source matrices specified by matrix.xpfa1 and matrix.xpfa2, and places the result in the corresponding element of the destination specified by matrix.xpfa3. Dimensions the destination to have the proper dimensions. The destination may be one of the sources. If DEBUG is ON, aborts if the dimensions of the two sources are incompatible.

MATRIX.MIN (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Performs FMIN on each pair of corresponding elements in the source matrices specified by matrix.xpfa1 and matrix.xpfa2, and places the result in the corresponding element of the destination specified by matrix.xpfa3. Dimensions the destination to have the proper dimensions. The destination may be one of the sources. If DEBUG is ON, aborts if the dimensions of the two sources are incompatible.

MATRIX.PF (-- u | u = size of a matrix parameter field)
Places on the stack the number of bytes in a matrix parameter field (u = 14 bytes). Typically used to define a stack-based temporary matrix within a definition; temporary matrices defined in this manner preserve re-entrancy (see the chapter on Designing Re-entrant Code in the Software Manual). For example:

```
: MATRIX.FUNCTION
        LOCALS{ .... | x&temp.matrix.pfa }
        MATRIX.PF PF.STACK.FRAME   TO  x&temp.matrix.pfa
        3 4 x&temp.matrix.pfa DIMMED \ dimension as 3X4
        ....                         \ use the temp matrix
        x&temp.matrix.pfa DELETED              \ delete from heap
        MATRIX.PF FRAME.DROP                   \ drop temp pf off stack
;
```

See ARRAY.PF, PF.STACK.FRAME and FRAME.DROP.
Pronunciation: "matrix-p-f"

MATRIX.SUM (matrix.xpfa -- r)
r is the sum of all of the elements in the specified matrix.
Attributes: S

MATRIX.VARIANCE (matrix.xpfa -- r)
Finds the variance r of the specified matrix. The variance is defined as the sum of the squares of all of the elements.
Attributes: S

MATRIX: (<name> --)
Removes <name> from input stream and defines <name> as a matrix. VALLOTs and clears a parameter field for <name> in the variable area; the size of the parameter field

is MATRIX.PF bytes. When executed, <name> returns the extended element address given the indices; its stack picture is:
(row#\col# -- xaddr)
The element address is also returned by the command
row# col# ' <name> M[]
MATRIX: does not allocate heap space; see DIMMED.
Pronunciation: "matrix-colon" Attributes: D

MAX (n1\n2 -- [n1] or [n2])
Retains the greater of the two signed integers n1 and n2, and drops the other. Also see UMAX.

MAX#DIMENSIONS (-- xaddr)
A user variable that holds the maximum allowable number of array dimensions. It is used to allocate space for the parameter field when an array is first defined using ARRAY:. It is also used later to compute the size of the parameter fields. The default value is 4, and the minimum value of MAX#DIMENSIONS is 2.
CAUTION: Decide on the maximum number of dimensions that will be used in the entire application program and maintain this value in MAX#DIMENSIONS throughout compilation. Unpredictable results and crashes may occur if arrays are defined (using ARRAY:) while MAX#DIMENSIONS is small, and then dimensioned or operated on when MAX#DIMENSIONS is larger.
Pronunciation: "max-number-of-dimensions" Attributes: U

MEMBER-> (u1\u2 <name> -- u3)
Adds a named member to the structure being defined and reserves room for u2 bytes in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "member" Attributes: D

MICROSEC.DELAY (u --)
Enters a software timing loop for u microseconds. At 16MHz, it can time to within 2 microseconds resolution for 16 <= u <= 65535 microseconds. Note that the elapsed time will be increased by the duration of any interrupt routines that are serviced while MICROSEC.DELAY is running. Consequently, this routine does not guarantee accurate timing when the timesliced multitasker is running.
Pronunciation: "microsecond-delay"

MIN (n1\n2 -- [n1] or [n2])
Retains the lesser of two signed integers n1 and n2, and drops the other. See UMIN.

MOD (n1\n2 -- n3 | n3 = remainder of n1/n2)
Divides n1 by n2, giving the remainder n3. The sign of n3 is the same as that of n1. If n2 is zero the result is indeterminant. See UMOD.

MOVE (xaddr1\xaddr2\u -- | xaddr1=src, xaddr2=dest, u = count)

If u is greater than 0, copies u consecutive 16-bit cells (i.e., 2*u consecutive bytes) from addresses starting at xaddr1 to addresses starting at xaddr2. The source and destination extended addresses may be located on different pages and the move may cross page boundaries. If the source and destination regions overlap and xaddr1 < xaddr2, MOVE starts at high memory and moves toward low memory to avoid propagation of the moved contents. MOVE always moves the contents in such a way as to avoid memory propagation. Speed is approximately 38 microseconds per cell. See MOVE.MANY.

MOVE.IN.PAGE (addr1\addr2\u\page -- | addr1=src, addr2=dest, u = count)

If u is greater than 0, copies u consecutive 16-bit cells (i.e., 2*u consecutive bytes) starting at addr1 to the destination addresses starting at addr2 on the specified page. If the source and destination regions overlap and addr1 < addr2, MOVE.IN.PAGE starts at high memory and moves toward low memory to avoid propagation of the moved contents. MOVE.IN.PAGE always moves the contents in such a way as to avoid memory propagation. Speed is approximately 15 microseconds per cell.
Pronunciation: "move-in-page"

MOVE.MANY (xaddr1\xaddr2\d -- | xaddr1=src, xaddr2=dest, d = count)

If 32-bit count d is greater than 0, copies d consecutive 16-bit cells (i.e., 2*d consecutive bytes) from addresses starting at xaddr1 to addresses starting at xaddr2. The source and destination extended addresses may be located on different pages and the move may cross page boundaries. If the source and destination regions overlap and xaddr1 < xaddr2, MOVE.MANY starts at high memory and moves toward low memory to avoid propagation of the moved contents. MOVE.MANY always moves the contents in such a way as to avoid memory propagation. Speed is approximately 38 microseconds per cell.

MT*M (matrix.xpfa1\matrix.xpfa2 --)

Multiplies the transpose of the source (matrix.xpfa1) by the source matrix to make the destination matrix matrix.xpfa2. The name of the routine suggests the order of the operands. The destination and the source may be the same. See M*MT
Pronunciation: "m-transpose-star-m"Attributes: S

M[] (row#\col#\matrix.xpfa -- xaddr)

Places on the stack the extended address xaddr of the element with the specified row# and column# in the matrix specified by matrix.xpfa. When using matrices this is faster than []. If DEBUG is on, ABORTs if the indices are invalid. Use as:

```
row# column#  ' <matrix.name>  M[]
```

Pronunciation: "m-brackets"

M[]! (r\row#\col#\matrix.xpfa --)

Stores r to the element at row#, column# in the specified matrix. Equivalent to M[] F!.
Pronunciation: "m-brackets-store"

M[]@ (row#\col#\matrix.xpfa -- r)

Fetches r from the element at row#, column# in the specified matrix. Equivalent to M[] F@.
Pronunciation: "m-brackets-fetch"

NDROP (wn\...\w1\+n -- | 0 <= +n <= 127)
Drops +n cells in addition to +n itself from the data stack. +n is a single cell value within the range 0...127. 1 NDROP is equivalent to DROP, 2 NDROP is equivalent to 2DROP.
Pronunciation: "n-drop"

NEEDED (+n --)
If DEBUG is true and there are fewer than +n cell units on the data stack (excluding +n), ABORTs the current word and issues a "Data stack underflow" error message.

NEGATE (n1 -- n2)
Replaces n1 with its two's complement n2. The two's complement of n1 is computed by inverting each of the bits in n1 and adding 1 to the result.

NEXT (--)
Return Stack: (R: u -- [u] or [] | drops index when loop terminates)
Used inside a colon definition to mark the end of a count-down loop structure that is begun by FOR. If the current loop index is zero, discards the index and terminates the loop, continuing execution with the word following NEXT. If the loop index is not 0, NEXT decrements the index by 1 and continues looping by transferring control to the word after FOR. Use as:

```
u1      FOR  <words to be executed u1+1 times>
        NEXT
```

0 FOR...NEXT executes 1 time, 1 FOR...NEXT executes 2 times, 65,535 FOR...NEXT executes 65,536 times, etc. FOR...NEXT loops may be nested as long as each FOR is matched with a corresponding NEXT in the same definition as FOR. An error is issued if FOR is not properly paired with NEXT inside a definition. FOR ... NEXT loops execute faster than DO ... LOOP constructs. The word I may be used inside a FOR NEXT loop. J K I' and LEAVE may not be used in a FOR NEXT loop.
Attributes: C, I

NEXT.NUMBER (<text> -- r)
Accepts text from the input stream, reading in new lines if necessary, until a space-delimited string representing a valid floating point or integer number is encountered. The floating point representation r of the number is placed on the stack. Integers are converted according to the current number BASE. See also ASK.FNUMBER and ASK.NUMBER.
Attributes: M, S

NEXT.TASK (-- xaddr)
User variable that contains the 16-bit task identifier (i.e., the base address of the task's user area) of the next task in the round-robin task list.
Attributes: U

NEXT.WORD (<name> -- addr)
Removes <name> from the input stream, inputting a new line if necessary to get it, leaves the text string in POCKET, and returns the 16-bit address of POCKET (which is in common memory) on the stack.

NFA.FOR (-- xnfa)
Compile Time: (<name> --)

Removes <name> from the input stream and returns <name>'s extended name field address xnfa. xnfa is the address of the count byte in <name>'s header. If in execution mode, leaves the xnfa on the stack. If in compilation mode, compiles the xnfa as a 2-cell literal in the current definition; the xnfa is pushed to the stack when the definition later executes. An error occurs if no <name> is given or if <name> cannot be found in the dictionary.
Pronunciation: "n-f-a-for" Attributes: I

NFA>CFA (xnfa -- xcfa)
Given the extended name field address xnfa of a header in the dictionary, returns the extended code field address xcfa of the word. xcfa is the first byte of executable machine code associated with the definition. An unchecked error occurs if xnfa is not a valid name field address. See NFA.FOR
Pronunciation: "n-f-a-to-c-f-a"

NFA>LFA (xnfa -- xaddr)
Given the extended name field address xnfa of a header in the dictionary, returns the "link field address" which is the extended xaddr in the header that contains the 3-byte link offset. The link offset consists of a page offset and an address offset which link the specified header to the previous header in the linked list of names. The page offset is a single signed byte stored at xaddr and the address offset is a 16-bit signed offset stored at xaddr+1. Adding the signed address and page offsets to the xnfa yields the xnfa of the previous word in the linked name list. See NFA.FOR
Pronunciation: "n-f-a-to-l-f-a"

NFA>PFA (xnfa -- [xpfa] or [0\0])
Given the extended name field address xnfa of a header in the dictionary, returns the extended parameter field address xpfa of the word. Returns 0\0 if the word has no pfa (see ?HAS.PFA). An unchecked error occurs if xnfa is not a valid name field address. See NFA.FOR
Pronunciation: "n-f-a-to-p-f-a"

NHERE (-- xaddr)
Places on the stack the xaddr of the next available location in the names area. Equivalent to NP X@
Pronunciation: "n-here" Attributes: U

NIP (w1\w2 -- w2)
Drops the cell below the top cell on the data stack. NIP is equivalent to SWAP DROP.

NO.AUTOSTART (--)
Undoes the effect of the AUTOSTART and PRIORITY.AUTOSTART commands and attempts to ensure that the standard QED-Forth interpreter will be entered after subsequent resets. Implementation detail: Erases the 0x1357 pattern at location 0xAE00 (put there by AUTOSTART) in EEPROM, and erases the 0x1357 pattern at location 0x7FFA\4 (put there by PRIORITY.AUTOSTART) in paged memory. Note that the priority.autostart vector at 0x7FFA\4 cannot be erased if the memory is write-protected when NO.AUTOSTART is executed. NO.AUTOSTART is invoked by the special cleanup mode.

NO.OP (--)
Does nothing. Used for redefining forward reference words. See REDEFINE.
Pronunciation: "no-op"

NO.SPACES (-- xaddr)
A user variable that contains a flag. If the flag is true, leading and trailing spaces are not printed when a floating point number is displayed. If true, the spaces are printed. See F>FIXED$, F>FLOATING$, and F>SCIENTIFIC$
Attributes: U

NO.VITAL.IRQ.INIT (--)
Writes a pattern into EEPROM so that subsequent cold restarts will not initialize the COP, clock monitor, illegal opcode, and OC2 interrupt vectors. This option is provided for programmers interested in installing their own interrupt service routines in any of these four vectors. Can be undone by INIT.VITAL.IRQS.ON.COLD. Implementation detail: Initializes location 0xAE1B in EEPROM to contain the pattern 0x13.
Pronunciation: "no-vital-i-r-q-init"

NOT (w -- flag)
flag is the boolean inverse of w. That is, if w = 0, the flag is TRUE. If w is non-zero, the flag is FALSE.

NP (-- xaddr)
User variable that contains the 32-bit pointer to the names area of the dictionary. The contents of NP are placed on the stack by NHERE and are modified by NALLOT. The command NP X@ is equivalent to NHERE; it yields the xaddr of the next available location in the names area. The command NP @ is equivalent to NPAGE; it yields the page of the names area.
Pronunciation: "n-p" Attributes: U

NPAGE (-- page)
Returns the page of the names area in the dictionary. Equivalent to NP @
Pronunciation: "n-page" Attributes: U

NUMBER (x$addr -- [n\1] or [d\2] or [0])
Converts the string whose first character is at x$addr+1 into an integer or double number in the current number base. If the number can be represented as a 16 bit signed integer, leaves it on the stack under a 1 flag. If it is convertible but cannot be represented as a 16 bit number, leaves its 32 bit representation on the stack under a 2 flag. If a non-convertible character is encountered in the string or if the string does not end with a space, leaves a 0 on the stack. x$addr must end with a space; its count is not used by NUMBER. Except for a leading + or - character and isolated embedded commas, no other punctuation is allowed in the numeric string. Also see FNUMBER.
Attributes: S

OC1.ID (-- n)
Returns the interrupt identity code for output compare 1. This interrupt can control the action of port bits PA3-PA7. Used as an argument for ATTACH.
Pronunciation: "o-c-one-i-d"

OC2.ID (-- n)
Returns the interrupt identity code for output compare 2. This interrupt can control the action of port bit PA6. Used as an argument for ATTACH. Note that the OC2 interrupt is used by the timeslice multitasker; if you wish to use it for another purpose, make sure that you do not need any of the services of the timeslicer or elapsed-time clock.
Pronunciation: "o-c-two-i-d"

OC3.ID (-- n)
Returns the interrupt identity code for output compare 3. This interrupt can control the action of port bit PA5. Used as an argument for ATTACH.
Pronunciation: "o-c-three-i-d"

OC4.ID (-- n)
Returns the interrupt identity code for output compare 4. This interrupt can control the action of port bit PA4. Used as an argument for ATTACH. Note that OC4 and PA4 are used by the optional secondary serial port supported by the QED-Forth software UART; if you are not using the secondary serial port, you may use freely use OC4 and PA4.
Pronunciation: "o-c-four-i-d"

OF (n1\n2 -- [n1] or [])
Used inside a CASE ... ENDCASE structure to mark the beginning of a conditional statement. If n1 = n2 then n1 and n2 are dropped and execution continues with the words between OF and ENDOF and then skips to the word after ENDCASE. If n1 does not equal n2, then n2 is dropped and execution continues after the next ENDOF. Use as:

```
n1 CASE
        n2 OF   words to be executed if n1 = n2           ENDOF
        n3 OF   words to be executed if n1 = n3           ENDOF
        words to be executed if n1 doesn't equal n2 or n3
ENDCASE
```

An error is issued if OF and ENDOF are not properly paired.
Attributes: C, I

OFF (xaddr --)
Stores 0 (FALSE) at xaddr.

OFFSET (-- xaddr)
A user variable that holds the 32-bit difference between the 32-bit physical block number in mass memory and the 16-bit file block number. Used by BUFFER and BLOCK. Set equal to 0\0 by IS.RAMDISK.
Attributes: U

ON (xaddr --)
Stores -1 (TRUE) at xaddr.

ON.FORGET (--)
Any word named ON.FORGET is executed by FORGET or ANEW before being forgotten. The programmer can define a word with the name ON.FORGET to de-allocate a heap item when the associated data structure is forgotten. This prevents

cluttering the heap with memory allocated to forgotten structures during debugging sessions. Typical use:

```
MATRIX: MAT.A
: DIM.MATRICES  3 3 ' MAT.A  DIMMED ;
: ON.FORGET
        ' MAT.A  DELETED
        ...other cleanup code...
;
```

If this code is later forgotten, MAT.A is deleted, thus freeing its space in the heap. If the ON.FORGET word had not been included, the definition of MAT.A would have been forgotten, but its heap space would still be allocated. Note that when this definition is compiled, the message

```
ON.FORGET isn't unique
```

will be issued. The non-uniqueness does not affect the performance of the word.

ONE (-- r)
Pushes the floating point number 1.0 onto the data stack.

OR (w1\w2 -- w3)
Performs a logical bit-wise inclusive-or of two 16-bit numbers w1 and w2 to produce the result w3.

OR.TYPE.OF: (u1\u2\u3 -- u1\max{u2,u3}\u1)
Used inside a variant field declaration started by TYPE.OF: and terminated by TYPE.END within a structure definition. Indicates that the following structure fields are a variant of the fields that followed TYPE.OF: An OR.TYPE.OF: variant should be used for variant fields which are loosely typed; i.e., during the life of the structure, the field content may change type. See TYPE.OF: for an example of use.
Pronunciation: "or-type-of" Attributes: D

OTHERWISE (--)
Marks the start of the "else" portion of a conditional structure that is used in execution mode. Use as:

```
flag  IFTRUE .... OTHERWISE ....  ENDIFTRUE
```

If the flag passed to IFTRUE is true, the code between IFTRUE and OTHERWISE is executed, and the code between OTHERWISE and ENDIFTRUE is skipped. If the flag is false, the code between IFTRUE and OTHERWISE is skipped and execution continues with the words following OTHERWISE. OTHERWISE is analogous to ELSE but it is used outside of a colon definition. The execution mode conditional structure can be used to conditionally compile portions of source code. An unchecked error occurs if OTHERWISE is used outside of IFTRUE and ENDIFTRUE. Note that IFTRUE/OTHERWISE/ENDIFTRUE statements can not be nested.

OVER (w1\w2 -- w1\w2\w1)
Places a copy of w1 on top of the stack.

OVERFLOW (--)
Sets the user variable FP.ERROR to -1 to indicate an overflow error.

PAD (-- xaddr | xaddr = start of scratchpad area)

Returns the xaddr of the start of the PAD scratchpad area. The 32 bytes below PAD are used for floating point and integer string/number conversion, and the area above PAD is available as scratchpad memory for the programmer (but note that the kernel routines ASK.NUMBER, ASK.FNUMBER, INPUT.STRING, and RECEIVE.HEX write text strings into the PAD). Equivalent to

    UPAD X@

PAD must point to modifiable RAM, and there must be at least 32 bytes of RAM below PAD for number/string conversion. PAD may be on any page, but may not cross a page boundary.
Attributes: U

PAGE-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a 1-cell (16-bit) page field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "page" Attributes: D

PAGE.TO.FLASH (page --)
Transfers the 32 Kbyte contents of the specified RAM source_page to the parallel page in flash. If the current memory map is the "download map", then valid source pages are 4, 5, or 6: page 4 RAM is transferred to page 1 flash, page 5 RAM is transferred to page 2 flash, and page 6 RAM is transferred to page 3 flash. If the current memory map is the "standard map", then valid source pages are 1, 2, or 3: page 1 RAM is transferred to page 4 flash, page 2 RAM is transferred to page 5 flash, and page 3 RAM is transferred to page 6 flash. An "invalid input parameter" error is issued if an invalid `source_page` is specified. A "can't program flash" error is issued if the flash cannot be programmed. On the QED-Flash Board, make sure that the "Write Enable Flash" DIP switch # 2 is ON and that DIP switches 3 and 4 are also ON. This function uses the 68HC11's on-chip RAM at 0xB200 to 0xB3CF to manage the write to the flash (the real-time clock and C/Forth interrupt stack reserve the bytes at 0xB3D0 to 0xB3FF). The remaining on-chip RAM at 0xB000 to 0xB1FF remains available to the user.

PAGE.TO.RAM (page --)
Transfers the 32 Kbyte contents of the specified flash source_page to the parallel page in RAM. If the current memory map is the "download map", then valid source pages are 1, 2, or 3: page 1 flash is transferred to page 4 RAM, page 2 flash is transferred to page 5 RAM, and page 3 flash is transferred to page 6 RAM. If the current memory map is the "standard map", then valid source pages are 4, 5, or 6: page 4 flash is transferred to page 1 RAM, page 5 flash is transferred to page 2 RAM, and page 6 flash is transferred to page 3 RAM. An "invalid input parameter" error is issued if an invalid source_page is specified.

PARITY (-- xaddr)
A system variable that contains a flag set by the programmer to specify the behavior of the secondary serial port (serial2) supported by QED-Forth's software UART. If the

contents of PARITY are TRUE, a parity bit is appended to each transmitted character and a parity bit is expected in each incoming character. The level of the transmitted parity bit is set by the system variable PARITY.OUT, and the value of the parity bit of the most recently received character is stored in the system variable PARITY.IN. PARITY is initialized to FALSE by INIT.SERIAL2 and USE.SERIAL2 and at each reset or restart.

PARITY.IN (-- xaddr)
A system variable that holds the value of the parity bit of the character most recently received by the secondary serial port (serial2, supported by the software UART) if PARITY is true. If the incoming parity bit was high, PARITY.IN is set to 1; otherwise it is set to 0. The contents are available to the programmer if parity checking of incoming data is required; the software UART does not check for correct parity. See PARITY.

PARITY.OUT (-- xaddr)
A system variable that holds the value of the parity bit of the character to be sent next by the secondary serial port (serial2, supported by the software UART) if PARITY is true. If the contents of PARITY are TRUE and the least significant byte of PARITY.OUT is nonzero, then the parity bit of the next outgoing character is set to one. If the contents of PARITY are TRUE and the least significant byte of PARITY.OUT is zero, then the parity bit of the next outgoing character is set to zero. The value stored in PARITY.OUT is not modified by the serial2 routines, and the application program must perform any required parity calculations. See PARITY.

PARSE (char1 -- xaddr\cnt | char1 is the terminating delimiter)
Parses a string delimited by the specified char1 from the input stream, and returns the xaddr\cnt of the parsed string, where xaddr is the address of the first character and cnt is the number of characters in the parsed string (the string may cross a page boundary). Unlike WORD, which also parses strings from the input stream, PARSE does not move the parsed string to POCKET. Thus it is suitable for parsing strings longer than 31 bytes (see WORD). If the specified delimiter char1 is a space, then leading spaces are ignored. If BLK = 0, the input stream is the terminal input buffer TIB. Otherwise PARSE executes BLOCK so that the input stream is available in a block buffer. The contents of >IN specify the offset from the start of the input stream to the first character to be parsed. PARSE leaves >IN pointing 1 byte past the terminating delimiter unless the input stream is exhausted, in which case >IN is left pointing 1 byte past the last valid location in the input stream.
Attributes: M

PAUSE (--)
Stacks the state of the current task and passes control to the next AWAKE task in the round-robin task list. You can embed calls to PAUSE in any task when you wish to give other tasks a chance to run. PAUSE may be used in multitasked systems whether or not the timeslicer is active. PAUSE switches tasks in (27 + 3.25n) microseconds, where n is the number of ASLEEP tasks encountered in the round robin task list. Of this time, interrupts are disabled for (20 + 3.25n) microseconds.
Attributes: M

PAUSE.ON.KEY (--)

Suspends execution of the calling word when a character is received and, with the exceptions noted below, resumes execution of the calling word when a second character is received. Typically coded into a loop structure to allow control of execution during debugging, or to control a data dump. PAUSE.ON.KEY checks whether a character has been received. If no character has been received, it does nothing. If a character has been received and it is a carriage return, executes ABORT which clears the stacks and returns to the interpreter. If the character received is a . (dot) executes QUIT which returns to the interpreter without clearing the data stack. If any other character is received, suspends execution until another character other than carriage return or . is received. This word effectively responds to XON/XOFF from a host terminal; a QED-Forth word that dumps data and calls PAUSE.ON.KEY repeatedly will pause when the XOFF is received by QED-FORTH, and resume when XON is received. The word does not know that the characters are special; it just stops when receiving the first and resumes after the second. The kernel words DUMP, DUMP.INTEL, DUMP.S1, DUMP.S2, M.PARTIAL, M., M.., and WORDS call PAUSE.ON.KEY.
Attributes: M

PF.STACK.FRAME (+n -- [+n bytes]\xaddr)
First performs the operation of STACK.FRAME, reserving +n bytes of room on the data stack and leaving xaddr that points to the top (lowest in memory) reserved byte in the data stack frame. PF.STACK.FRAME then zeros the first 4 bytes (lowest in memory, nearest top of stack) of the allocated stack frame. See STACK.FRAME and FRAME.DROP. PF.STACK.FRAME should be used when creating a temporary array or matrix parameter field within a definition; see ARRAY.PF and MATRIX.PF for examples of use. The first 4 bytes of a parameter field contain an xhandle to the heap, and the xhandle should always be initialized to 0\0 before the array or matrix is dimensioned. PF.STACK.FRAME performs this initialization on the temporary stack-based parameter field. There is an unchecked error if +n is less than 4.
Pronunciation: "p-f-stack-frame"

PFA>NAME (xpfa --)
Prints the name of the word associated with the specified extended parameter field address xpfa. Useful for error diagnostics to print the name of an array, matrix or other data structure given its xpfa. The name is printed as ?NAME? if no name corresponding to xpfa is found in the dictionary.
Pronunciation: "p-f-a-to-name" Attributes: M

PFA>NFA (xpfa -- [xnfa] or [0\0])
Given the extended parameter field address xpfa of a word in the dictionary, searches the dictionary and returns the extended name field address xnfa of the word. If the name associated with xpfa cannot be found in the dictionary, returns 0\0. See ' and NFA.FOR.
Pronunciation: "p-f-a-to-n-f-a"

PI (-- r)
Places the floating point value pi (= 3.1416) on the stack.

PI/2 (-- r)
Places the floating point representation of pi/2 (1.5708) on the stack.
Pronunciation: "pi-over-two"

PIA.C! (byte\xaddr --)

Stores byte at xaddr. This routine should be used in place of C! to access any PIA register on a QED board that is clocked at 16 MHz. The routine disables interrupts for 27 cycles (less than 7 microseconds).
Pronunciation: "p-i-a-c-store"

PIA.C@ (xaddr -- byte)

Fetches the byte stored at xaddr. This routine should be used in place of C@ to access any PIA register on a QED board that is clocked at 16 MHz. The routine disables interrupts for 22 cycles (less than 6 microseconds).
Pronunciation: "p-i-a-c-fetch"

PIA.CHANGE.BITS (byte1\byte2\xaddr -- | byte1 = data; byte2 = mask)

At the byte specified by xaddr, modifies the bits specified by 1's in byte2 to have the values indicated by the corresponding bits in byte1. In other words, byte2 serves as a mask which specifies the bits at xaddr that are to be modified, and byte1 provides the data which is written to the modified bits. Disables interrupts for 48 cycles (12 microseconds) to ensure an uninterrupted read/modify/write operation. This routine should be used in place of CHANGE.BITS to access any PIA register on a QED board that is clocked at 16 MHz.

PIA.CLEAR.BITS (byte1\xaddr --)

For each bit of byte1 that is set, clears the corresponding bit of the 8 bit value at xaddr. Disables interrupts for 33 cycles (about 8 microseconds) to ensure an uninterrupted read/modify/write operation. This routine should be used in place of CLEAR.BITS to access any PIA register on a QED board that is clocked at 16 MHz.

PIA.SET.BITS (byte1\xaddr --)

For each bit of byte1 that is set, sets the corresponding bit of the 8 bit value at xaddr. Disables interrupts for 31 cycles (less than 8 microseconds) to ensure an uninterrupted read/modify/write operation. This routine should be used in place of SET.BITS to access any PIA register on a QED board that is clocked at 16 MHz.

PIA.TOGGLE.BITS (byte1\xaddr --)

For each bit of byte1 that is set, reverses the state of the corresponding bit of the 8 bit value at xaddr. Disables interrupts for 31 cycles (less than 8 microseconds) to ensure an uninterrupted read/modify/write operation. This routine should be used in place of TOGGLE.BITS to access any PIA register on a QED board that is clocked at 16 MHz.

PICK (wn\...\w1\w0\+n -- wn\...\w1\w0\wn | 0 <= +n <= 255)

Copies the +nth item (not including n) to the top of the stack, where the top stack item is item#0, the next is item#1, etc. An unchecked error occurs if there are fewer than n+1 items on the data stack. 0 PICK is equivalent to DUP, 1 PICK is equivalent to OVER.

POCKET (-- xaddr | xaddr is the start of the pocket buffer)

Returns the xaddress of the start of the POCKET buffer. POCKET is a 32-byte (minimum) scratch area used by WORD. Equivalent to

```
UPOCKET @ DEFAULT.PAGE
```

FIND executes COLD if POCKET is not in the common RAM.
Attributes: U

PORTA (-- xaddr)
Returns the extended address (0x8000\0) of the 8 bit PortA register in the 68HC11. This port is available to the user and is associated with various counting and timing functions. Note that the software UART that implements the secondary serial port uses bits 3 and 4 of PORTA, so care must be taken not to alter the direction or state of these bits if the secondary serial port is in use. PORTA can be accessed with the standard C@, C!, SET.BITS, CLEAR.BITS, TOGGLE.BITS, and CHANGE.BITS operators. It is recommended that the latter four uninterruptable operators be used to modify the available port bits if the secondary serial port is in use. The PORTA.DIRECTION register sets the data direction of the pins in PORTA.
Pronunciation: "port-A"

PORTA.DIRECTION (-- xaddr)
Returns the extended address (0x8001\0) of the 8 bit DDRA register in the 68HC11 which sets the data direction of PORTA. To configure a PORTA pin to be an output, write a 1 to the corresponding bit position in PORTA.DIRECTION. Similarly, to configure a PORTA pin to be an input, write a 0 to the corresponding bit position in PORTA.DIRECTION. Note that the software UART that implements the secondary serial port uses bits 3 and 4 of PORTA, so care must be taken not to alter the direction or state of these bits if the secondary serial port is in use. The PORTA.DIRECTION register can be accessed with the standard C@, C!, SET.BITS, CLEAR.BITS, TOGGLE.BITS, and CHANGE.BITS operators.
Pronunciation: "port-A-direction"

PORTD (-- xaddr)
Returns the extended address (0x8008\0) of the 8 bit PortD register in the 68HC11. This port implements the primary serial channel on bits 0 and 1, and the serial peripheral interface (SPI) on bits 2-5. If the SPI is not in use (which implies that the 12 bit A/D and 8 bit D/A are not in use) then bits 2, 3, 4, and 5 of PORTD are available as general purpose inputs and outputs. The data direction of these bits is set by register PORTD.DIRECTION. PORTD can be accessed with the standard C@, C!, SET.BITS, CLEAR.BITS, TOGGLE.BITS, and CHANGE.BITS operators. See INIT.SPI and SPI.OFF.
Pronunciation: "port-D"

PORTD.DIRECTION (-- xaddr)
Returns the extended address (0x8009\0) of the 8 bit DDRD register in the 68HC11 which sets the data direction of bits 2-5 of PORTD. PORTD implements the primary serial channel on bits 0 and 1, and the serial peripheral interface (SPI) on bits 2-5. If the SPI is not in use (which implies that the 12 bit A/D and 8 bit D/A are not in use) then bits 2, 3, 4, and 5 of PORTD are available as general purpose inputs and outputs. To configure a PORTD pin to be an output, write a 1 to the corresponding bit position in PORTD.DIRECTION. Similarly, to configure a PORTD pin to be an input, write a 0 to the corresponding bit position in PORTD.DIRECTION. This register can be accessed with the standard C@, C!, SET.BITS, CLEAR.BITS, TOGGLE.BITS, and CHANGE.BITS operators. See INIT.SPI and SPI.OFF.
Pronunciation: "port-D-direction"

PORTE (-- xaddr)
Returns the extended address (0x800A\0) of the 8 bit PortE register in the 68HC11. This port can either be used as an 8 channel 8 bit A/D converter, or as an octal digital input port. PORTE can be accessed with the standard C@ operator. See A/D8.ON and A/D8.OFF.
Pronunciation: "port-E"

PPA (-- xaddr)
Returns the extended address (0x8080\0) of the 8 bit PPA register in the peripheral interface adapter (PIA). This port is available to the user and is initialized to all inputs upon each reset or restart. PPA can be accessed with the standard C@, C!, SET.BITS, CLEAR.BITS, TOGGLE.BITS, AND CHANGE.BITS operators. See INIT.PIA.
Pronunciation: "P-P-A"

PPB (-- xaddr)
Returns the extended address (0x8081\0) of the PPB register in the peripheral interface adapter (PIA). Note that the bottom 5 bits of this port are dedicated to the keypad interface outputs, and the top three bits control high-current outputs HC1, HC2, and HC3 (HC0 is controlled by the on-board PAL). This port is initialized to all outputs upon each reset or restart, or whenever INIT.PIA is called. See INIT.PIA.
Pronunciation: "P-P-B"

PPC (-- xaddr)
Returns the extended address (0x8082\0) of the PPC register in the peripheral interface adapter (PIA). Note that the lower 4 bits of this port are dedicated to the keypad/display interface inputs, and bit 4 (the lowest bit in the upper nibble) controls the RS485 transceiver direction if RS485 is in use. PPC is initialized as all inputs upon each reset or restart, and the lower nibble of PPC is configured as input whenever INIT.PIA is called. PPC can be accessed with the standard C@, C!, SET.BITS, CLEAR.BITS, TOGGLE.BITS, and CHANGE.BITS operators. If RS485 is in use, it is recommended that the latter four uninterruptable operators be used instead of C! to alter the state of available output bits in PPC; this makes the code more robust with respect to multitasking and interrupts. See INIT.PIA.
Pronunciation: "P-P-C"

PREV (-- xaddr)
A user variable containing the extended address of the most recently accessed block buffer.
Attributes: U

PRIORITY.AUTOSTART (xcfa --)
Compiles a 6-byte sequence at locations 0x7FFA-0x7FFF on page 4 so that upon subsequent restarts and ABORTs, the routine having the specified xcfa will be automatically executed. This allows a finished application to be automatically entered upon power up and resets. In contrast to the EEPROM-based AUTOSTART function, the PRIORITY.AUTOSTART vector is located in paged memory which is in flash memory in turnkeyed "production" boards. Thus PRIORITY.AUTOSTART facilitates the autostarting of flash-based systems. ABORT (which is called by the error handler and upon every reset or restart) checks the priority autostart vector first and executes the

specified routine (if any). If no priority autostart routine is posted or if the specified routine terminates, ABORT then checks the EEPROM-based autostart vector (see AUTOSTART) and executes the specified routine (if any). If no autostart routine is posted or if the specified routine terminates, ABORT then invokes QUIT which is the QED-Forth interpreter. PRIORITY.AUTOSTART is Flash smart; it writes to page 4 whether page 4 addresses RAM or Flash at the time. In the standard map PRIORITY.AUTOSTART writes directly to Flash in page 4. In the download memory map it also writes to page 4, now RAM. Subsequently page 4 can be copied to Flash and the Flash readdressed onto page 4 in the standard map.
Implementation detail: At location 7FFAH on page 4, PRIORITY.AUTOSTART writes the pattern 1357 followed by the four byte xcfa; make sure that page 4 is not write protected when executing PRIORITY.AUTOSTART. To undo the effects of this command and return to the default startup action, make sure that page 4 is un-write-protected RAM and call NO.AUTOSTART (which clears both the priority autostart and the EEPROM-based autostart vectors). To recover from the installation of a buggy priority autostart routine if page 4 is RAM, make sure that page 4 is not write-protected and invoke use the special cleanup mode. See AUTOSTART.

PULSE.EDGE.ID (-- n)
Returns the interrupt identity code for the pulse accumulator input edge detector which is associated with port bit PA7. Used as an argument for ATTACH.
Pronunciation: "pulse-edge-i-d"

PULSE.OVERFLOW.ID (-- n)
Returns the interrupt identity code for the pulse accumulator overflow detector which is associated with port bit PA7. Used as an argument for ATTACH.
Pronunciation: "pulse-overflow-i-d"

PUT.CURSOR (n1\n2 -- | n1 = line#, n2 = character#)
Positions the LCD display cursor at the line number specified by n1 and the character number specified by n2. For all character and Hitachi graphics displays, the next character or graphical byte sent to the display by the CHAR>DISPLAY routine will appear at the specified cursor position, and then the cursor position will automatically increment. (Note that for Toshiba graphics displays, PUT.CURSOR affects only the cursor in text mode, and you must use IS.DISPLAY.ADDRESS to specify the location of the next graphical data byte.) n1 and n2 are 0 based (that is, the top line on the display is line#0, and the left-most character on each line is character#0). PUT.CURSOR clamps n1 to one less than LINES/DISPLAY, and clamps n2 to one less than CHARS/DISPLAY.LINE. The line# n1 follows the same rules explained in the description of BUFFER.POSITION: for a graphics-style display the line# n1 is interpreted differently depending on whether the display is being used in "text mode" or "graphics mode". In text mode, n1 corresponds to the character line#; in graphics mode, n1 corresponds to the pixel line#. Note that the cursor may not be visible, and is never visible in graphics mode; see DISPLAY.OPTIONS. Also note that after the cursor reaches the end of a line it may skip to the start of a line elsewhere on the display. Finally, consider using (UPDATE.DISPLAY) instead of UPDATE.DISPLAY to avoid re-homing the cursor after executing PUT.CURSOR. This routine intermittently disables interrupts for 28 cycles (7 microseconds) per command byte to implement clock stretching.

QUERY (--)
Executes
    TIB CHARS/LINE @ EXPECT
to accept a line of up to CHARS/LINE characters from the serial port and store them in the terminal input buffer. Saves the number of characters actually received in the user variable #TIB. Sets >IN and BLK to 0 so that the next execution of WORD will parse the received line of input starting at the first character received. This is the main serial input word in the QED-Forth interpreter. Note that the terminal input buffer may be on any page, but may not cross a page boundary.
Attributes: M

QUIET (-- xaddr)
A user variable that holds a flag that controls the word BEEP. BEEP is called when an error is detected. If the flag in QUIET is false, BEEP EMITs the bell character when executed to give an audible warning. If the flag is true, BEEP does nothing. BEEP is called by the system error routine, so QUIET controls whether system errors emit an audible beep.
Attributes: U

QUIT (--)
Enters execution mode and begins an infinite loop (terminated by errors) that repeatedly executes QUERY INTERPRET to read in a new line of input and interpret it. The return stack is cleared after each line of input is interpreted while in execution mode. This is the top level word in the QED-Forth interpreter. See the Software Manual for a detailed description of this word.
Attributes: M

R0 (-- xaddr)
User variable that contains the 16-bit address which is used by RP! to initialize the return stack pointer. The first cell on the return stack occupies the 2 bytes below the address contained in R0, and the stack grows downward in memory. After changing the contents of R0, the next ABORT or restart loads the value into the return stack pointer (the S register) to change the position of the stack. The return stack is allocated in a 768 byte region in common memory after each COLD restart. See RP!.
Pronunciation: "r-zero" Attributes: U

R> (-- w)
Return Stack: (R: w --)
Transfers the top cell on the return stack to the data stack.
Pronunciation: "r-from" Attributes: C

R>DROP (--)
Return Stack: (R: w --)
Drops the top cell from the return stack.
Pronunciation: "r-from-drop" Attributes: C

R@ (-- w)
Return Stack: (R: w -- w)
Copies the top cell on the return stack to the data stack.
Pronunciation: "r-fetch" Attributes: C

RANDOM (-- n)
Generates n, a pseudo-random 16-bit integer. See RANDOM#.

RANDOM# (-- xaddr)
A user variable that holds the last 16-bit number generated by RANDOM, or the 16-bit mantissa of the last floating point random number generated by FRANDOM. Storing a specific integer (a "seed") into RANDOM# leads to the generation of a reproducible series of pseudo-random numbers by repeated calls to FRANDOM or RANDOM. This may be useful for debugging words that use random numbers.
Pronunciation: "random-number" Attributes: U

RANDOM.GAUSSIAN (-- r)
r is a random number drawn from a Gaussian distribution with unity standard deviation and zero mean. Uses the Box-Muller method.
Attributes: S

RANDOMIZED (matrix.xpfa --)
Fills the specified matrix with random numbers drawn from a Gaussian distribution with 0 mean and unity standard deviation. See RANDOM.GAUSSIAN.
Attributes: S

RANGE (n1\n2\n3 -- n1\flag)
Flag is TRUE if n1 is greater than or equal to n2 and less than or equal to n3. Otherwise flag is FALSE. See URANGE.

RANGE.OF (n1\n2\n3 -- [n1] or [])
Used inside a CASE ... ENDCASE structure to mark the beginning of a conditional statement. If n2 <= n1 <= n3 then n1, n2, and n3 are dropped and execution continues with the words between RANGE.OF and ENDOF and then skips to the word after ENDCASE. Otherwise, n2 and n3 are dropped and execution continues after the next ENDOF. Use as:

```
n1 CASE
n2 n3 RANGE.OF  executed if n1 in range (n2,n3)         ENDOF
n4 n5 RANGE.OF  executed if n1 in range (n4,n5)         ENDOF
        words to be executed if not in range (n2,n3) or (n4,n5)
ENDCASE
```

An error is issued if RANGE.OF and ENDOF are not properly paired. See OF.
Pronunciation: "range-of" Attributes: C, I

READ.ELAPSED.SECONDS (-- u\ud | u = #msec, ud = #sec)
Returns the elapsed time on the timeslice clock since it was initialized to 0\0 by INIT.ELAPSED.TIME. ud is the number of elapsed seconds, and u is the number of milliseconds since the last integral second on the timeslice clock. The resolution equals the period of the timeslice clock (the default is 5 msec). See READ.ELAPSED.TIME, TIMESLICE.COUNT, START.TIMESLICER,and 100US=TIMESLICE.PERIOD.

READ.ELAPSED.TIME (-- u1\u2\u3\u4\u5 | u1 msec, u2 sec, u3 min, u4 hrs,u5 days)
Returns the number of milliseconds, seconds, minutes, hours, and days that the timeslice clock (supported by OC2) has run since it was initialized to 0\0 by

INIT.ELAPSED.TIME. u5 is the number of days, u4 is the number of hours since the last integral day on the clock, u3 is the number of minutes since the last integral hour on the clock, u2 is the number of seconds since the last integral minute on the clock, and u1 is the number of milliseconds since the last integral second on the clock. The resolution equals the period of the timeslice clock (the default is 5 msec). The maximum time that the clock can represent is proportional to the timeslice period; the clock can represent times to over 248 days if the default 5 msec timeslice period is used. See READ.ELAPSED.TIME, TIMESLICE.COUNT, START.TIMESLICER, and *100US=TIMESLICE.PERIOD.

READ.WATCH (-- u1\u2\u3\u4\u5\u6\u7\u8)

meaning: (-- 100ths.sec\sec\min\hr\day\date\month\yr)

Reads the battery-operated real-time clock (if present), returning the time, day, and date specified by u1 through u8. The stack items and their allowed ranges are:

item	description	range (decimal)
u8	year	0 - 99
u7	month	1 - 12
u6	date	1 - 31
u5	day of week	1 - 7
u4	hour of day	0 - 23
u3	minute after the hour	0 - 59
u2	seconds after the minute	0 - 59
u1	hundredths of seconds	0 - 99

Resolution is better than +/- 1 minute per month. Once correctly set, the watch handles the differing numbers of days in each month, and correctly handles leap years. READ.WATCH uses the top 16 bytes of on-chip RAM at B3F0-B3FF as a scratchpad buffer, and disables interrupts for 0.5 msec while accessing the watch. See SET.WATCH. Note that a non-maskable interrupt such as XIRQ or RESET that occurs during a watch access can cause a crash, because the common memory cannot be accessed while the watch is being set or read.

READ/WRITE (xaddr\n\flag -- | xaddr=buffer, n = block#, flag=true to read)

Calls the routine whose xcfa is stored in the user variable UREAD/WRITE. If flag is false, the routine specified by xcfa writes a 1 Kbyte block of data from the block buffer starting at xaddr to the mass memory block specified by n plus the contents of OFFSET. If flag is true, the routine specified by xcfa reads a 1 Kbyte block of data from the mass memory block specified by n + OFFSET to the block buffer starting at xaddr. The routine should trap all errors (invalid disk block, etc.) The default routine whose xcfa is stored in UREAD/WRITE implements a "ram disk" mass memory. See IS.RAMDISK and BLOCK.BUFFERS .

Pronunciation: "read-slash-write" Attributes: M, U

REAL-> (u1 <name> -- u2)

Adds a named member to the structure being defined and reserves room for a real (floating point) number field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an

instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "real" Attributes: D

REAL: (<name> --)
REAL: is a synonym for DOUBLE: ; see its glossary entry. REAL: defines a 32-bit self-fetching variable which holds a 32-bit floating point value (a real number). Use as:

```
REAL: <name>
```

Pronunciation: "real-colon" Attributes: D

REALS-> (u1\u2 <name> -- u3)
Adds a named member to the structure being defined and reserves room for u2 real (floating point) numbers in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "reals" Attributes: D

RECEIVE (xmailbox -- wd | wd is the received message)
If xmailbox is empty (i.e., if it contains 0\0), executes PAUSE until the mailbox contains a message. If xmailbox contains a message (i.e., if it does not contain 0\0), fetches the contents of xmailbox and stores a 0\0 into xmailbox to indicate that the message has been received and that the mailbox is now empty. To ensure that the state of the mailbox is correctly determined, RECEIVE disables interrupts for 26 to 61 cycles (6.5 to 15.25 microseconds). See SEND, ?RECEIVE and MAILBOX:.
Attributes: M

RECEIVE.HEX (xaddr1 <text> --)
Accepts a download in standard Intel hex or Motorola S1 or S2 or S3 hex formats and initializes the memory locations starting at the specified xaddr1 accordingly. The first address specified in the <text> hex dump is stored in memory at xaddr1, and all subsequent bytes are stored in QED memory preserving the relative spacing of data specified in the <text> hex dump. If the specified xaddr1 is FFFF0XFFFF (that is, a 32-bit -1), the memory storage addresses are as specified in the hex dump itself. The QED paged memory is treated as a contiguous memory space; recall that the location following 7FFF on a given page is location 0000 on the following page. Accepts empty lines. If a format or checksum error is detected, emits an 'X' character to signal the error, but does not abort. Aborts with a "Missing delimiter" message if the first character on a line is not a : or S character. Terminates when an end-of-file record is received; the final line of an Intel hex dump is

```
:00000001FF
```

and the standard final line of a Motorola hex dump is

```
S9030000FC
```

although any S7, S8, or S9 termination record will terminate reception. Motorola S0 header records are accepted and ignored. Each input text line is temporarily stored at PAD. Be sure that the PAD buffer is large enough to accommodate a full line (decimal

80 bytes or more is safe). See the glossary entries for DUMP.INTEL, DUMP.S1, and DUMP.S2 for descriptions of Intel and Motorola hex formats.
Implementation detail: RECEIVE.HEX calculates an offset as the specified xaddr1 minus the first address specified in the <text> file. This offset is then added to every byte's file address (specified in the <text> file) to calculate the QED destination address. This scheme allows the data in a <text> hex dump file with arbitrary reported addresses to be loaded starting at any desired location in the QED memory space.
Pronunciation: "receive-hex"Attributes: M

RECOVER.HANDLE (xaddr -- [xhandle] or [0\0])
Searches through the handle list for a handle that contains the specified base address xaddr. If found, returns the heap item's xhandle; otherwise, returns 0\0. Useful for debugging.

RECURSE (--)
Compiles into the current definition a call to the word currently being defined.
Attributes: C, I

REDEFINE (xcfa1\xcfa2 -- | xcfa1 = operational word, xcfa2 = null word)
Resolves a forward reference or redefines a word by writing a call to xcfa1, the extended code field address of an operational word, into the code field specified by xcfa2. Up to 9 bytes are written into the code field of xcfa2.
To implement a forward reference (that is, to use a word before its action or operation has been defined), first define a null definition as:
```
: <null.definition.name>
        NO.OP ;
```
Then define words which call <null.definition.name> .
Then define the operational word as
```
: <operational.definition.name>
        words defining the operation   ;
```
Then execute
```
CFA.FOR <operational.definition.name>
CFA.FOR <null.definition.name>
REDEFINE
```
Now, all the words that were compiled with calls to <null.definition.name> will execute <operational.definition.name>. Of course, when REDEFINE is executed, xcfa2 must be in modifiable RAM.
REDEFINE may also be used during debugging. If xcfa2 is the code field address of a word that is found to be buggy, a bug-free version with code field address xcfa1 can be defined, and a REDEFINE command will cause all compiled calls to the buggy routine to execute the debugged version instead:
```
CFA.FOR <debugged.definition.name>
CFA.FOR <buggy.definition.name>
REDEFINE
```
Two requirements must be met: xcfa2 must be in modifiable RAM, and the code field of xcfa2 must be at least 9 bytes long so that the redefinition will not overwrite other words in the dictionary. There is no error checking. For another (though less efficient) way to implement forward references, see EVALUATE.

REDIMMED (#rows\#cols\matrix.xpfa --)

Re-writes the contents of the parameter field of the specified dimensioned matrix. Modifies the number of rows and columns in the parameter field to have the specified values without changing any data in the matrix. Thus the data in the matrix is effectively reconfigured into the new number of rows and columns. Error if the previous product of #rows times #columns is not equal to the product of the specified #rows times #cols.

REGISTER:(addr <name> --)

Typically used to define names for the HC11's hardware registers, REGISTER: removes the next <name> from the input stream and defines an XCONSTANT called <name> which when executed leaves the specified register address under a 0 (designating the default page) on the data stack. REGISTER: enforces a minimum WIDTH of 6 in the saved name to minimize non-unique names when defining registers. For example, to define a register address for the timer counter register named TCNT on the processor, execute

```
    HEX 800E  REGISTER:   TCNT
```

Pronunciation: "register-colon"   Attributes: D

RELEASE (xresource --)

If the current task owns the resource variable referenced by xresource (i.e., if xresource contains the current task's xtask.id), releases the resource by storing 0\0 in xresource. Otherwise, does nothing; this prevents a task from RELEASEing a resource controlled by another task. Interrupts are not disabled and PAUSE is not executed. See GET and RESOURCE.VARIABLE:.

RELEASE.AFTER.LINE (-- n)

A constant which is the default value stored into the SERIAL.ACCESS user variable. If stored into the SERIAL.ACCESS user variable of a task that is running the QED-Forth interpreter, prevents the low level I/O words KEY EMIT and ?KEY from executing GET or RELEASE on the active serial resource variable. Rather, the interpreter (that is, QUIT) GETs the serial resource before each line is received and RELEASEs the serial resource after each line is interpreted. This virtually eliminates the overhead required to GET and RELEASE during downloads, and allows the interpreter to run at sustainable baud rates to 19200 baud. CAUTION: In multitasking systems using both serial ports SERIAL1 and SERIAL2, the application code should include the command

```
    RELEASE.ALWAYS     SERIAL.ACCESS !
or  RELEASE.NEVER      SERIAL.ACCESS !
```

before building the tasks. This prevents contention that can occur if the default RELEASE.AFTER.LINE option is installed in the SERIAL.ACCESS user variable. See SERIAL.ACCESS, RELEASE.NEVER, and RELEASE.ALWAYS.

RELEASE.ALWAYS (-- n)

A constant. Returns a value that, when stored into the SERIAL.ACCESS user variable, causes the low level I/O words KEY EMIT and ?KEY to always RELEASE the serial resource variable after each I/O operation. This is useful if the task that has control over the serial line (for example, the task running the QED-Forth interpreter) wants to share access to the serial port. See SERIAL.ACCESS, RELEASE.NEVER, and RELEASE.AFTER.LINE.

CAUTION: Depending on which terminal program you use, you may find that storing RELEASE.ALWAYS into the QED-Forth task's SERIAL.ACCESS variable decreases the sustainable download baud rate to 9600 baud. To assure the highest sustainable download baud rate, it is recommended that RELEASE.AFTER.LINE be stored in the QED-Forth task's SERIAL.ACCESS variable during program development.

CAUTION: In multitasking systems using both serial ports SERIAL1 and SERIAL2, the application code should include the command

```
      RELEASE.ALWAYS      SERIAL.ACCESS !
or    RELEASE.NEVER       SERIAL.ACCESS !
```

before building the tasks. This prevents contention that can occur if the default RELEASE.AFTER.LINE option is installed in the SERIAL.ACCESS user variable.

RELEASE.NEVER (-- n)

A constant. Returns a value that, when stored into the SERIAL.ACCESS user variable, prevents the low level I/O words KEY EMIT and ?KEY from executing the command SERIAL RELEASE. This is useful if the task that has control over the serial line (for example, the task running the QED-Forth interpreter) does not want to share access to the serial port. See SERIAL.ACCESS, RELEASE.ALWAYS, and RELEASE.AFTER.LINE.

CAUTION: Depending on which terminal program you use, you may find that storing RELEASE.NEVER into the QED-Forth task's SERIAL.ACCESS variable decreases the sustainable download baud rate to 9600 baud. To assure the highest sustainable download baud rate, it is recommended that RELEASE.AFTER.LINE be stored in the QED-Forth task's SERIAL.ACCESS variable during program development.

CAUTION: In multitasking systems using both serial ports SERIAL1 and SERIAL2, the application code should include the command

```
      RELEASE.ALWAYS      SERIAL.ACCESS !
or    RELEASE.NEVER       SERIAL.ACCESS !
```

before building the tasks. This prevents contention that can occur if the default RELEASE.AFTER.LINE option is installed in the SERIAL.ACCESS user variable.

REPEAT (--)

Used inside a colon definition to mark the end of a BEGIN ... WHILE ... REPEAT loop structure. Use as:

```
BEGIN ...
flag WHILE    ...
REPEAT
```

See BEGIN and WHILE.

Attributes: C, I

RESERVED (u1\u2 -- u3)

Used during definition of a structure to reserve an unnamed space equal to u2 bytes within the structure. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END.

Attributes: D

RESIZE.HANDLE (xhandle\d -- flag)

Attempts to resize the heap item associated with xhandle so that it has a new size of d bytes. Retains as much data as possible in the heap item. The heap must have enough space to copy the heap item to be successful. A true flag is returned if the heap item is successfully resized. A false flag indicates failure (due to an invalid xhandle or inadequate heap space) and the original heap item is left unchanged.

RESOURCE.VARIABLE: (<name> --)

Removes the next <name> from the input stream, defines a child word called <name>, and VALLOTs 2 cells in the variable area. When <name> is executed, it leaves the extended address, xaddr, of the two cells reserved in the variable area to hold the resource variable's contents. <name> is referred to as a "resource variable". Use as:

RESOURCE.VARIABLE: <name>

Resource variables are used in multitasked systems to control access to shared resources (for example, an A/D converter, serial port, block of memory, etc.) When the resource associated with <name> is available, <name> contains 0\0. When it is controlled by a task (and hence unavailable to other tasks), it contains the task id of the controlling task. Before its first use, the resource variable must be initialized to 0\0. After initialization to 0\0, the only operators that should access the resource variable are GET ?GET and RELEASE. The following resource variables are pre-defined in the QED-Forth kernel:

DISK.RESOURCE	SPI.RESOURCE
SERIAL1.RESOURCE	SERIAL2.RESOURCE
A/D8.RESOURCE	

See their glossary entries and consult the Software Manual for further descriptions and examples of use.

Attributes: D

RESTORE (--)

Restores the memory map user variables stored by the last execution of SAVE to their respective user variables. See SAVE.

RIGHT.PLACES (-- xaddr)

A user variable that holds the number of digits to be displayed to the right of the decimal point when a floating point number is printed in FIXED format. See F>FIXED$

Attributes: U

ROLL (wn\wn-1\...\w0\+n -- wn-1\...\w0\wn | 0 <= +n <= 255)

Transfers the +nth item (not including +n) on the data stack to the top of the data stack. The top stack item is item#0, the next is item#1, etc. 0 ROLL does nothing, 1 ROLL is equivalent to SWAP, and 2 ROLL is equivalent to ROT. An unchecked error occurs if there are less than +n items on the data stack.

ROOM (-- d)

d is the number of bytes available in the HEAP.

ROT (w1\w2\w3 -- w2\w3\w1)

Rotates the top three stack cells.

Pronunciation: "rote"

ROW->V (row#\matrix.xpfa -- xvaddr\sep\d.#el)

Returns the vector representation xvaddr\sep\d.#el of the specified row in the specified matrix. xvaddr is the base address of the vector, sep is the element separation expressed as a multiple of 4 bytes (e.g., sep=1 means a vector of contiguous floating point numbers, sep=2 means elements are separated by 8 bytes, etc.) and the double number d.#el is the number of elements in the vector. In the case of a row in a matrix,

sep equals the number of rows in the matrix and d.#el equals the 32-bit equivalent of the number of columns in the matrix. Note that xvaddr must be 4-byte aligned (i.e., must be an even multiple of 4). The heap manager and array and matrix dimensioning words automatically perform the required 4-byte alignment. See also COL->V.
Pronunciation: "row-to-v"

ROW.CONCATENATE (matrix.xpfa1\matrix.xpfa2\matrix.xpfa3 --)
Concatenates the two source matrices specified by matrix.xpfa1 and matrix.xpfa2 to form a destination matrix matrix.xpfa3 with more rows. The number of columns in the two source matrices must be the same. The destination may be one of the sources.

ROW/COL* ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 --)
Multiplies each element of the source1 row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
by the corresponding element of the source2 row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
and stores the result in the corresponding element of the destination row/col specified by
[row#3\-1] or [-1\col#3]\matrix.xpfa3.
The destination row/col may be one of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources.
Pronunciation: "row-or-col-star" Attributes: S

ROW/COL*+ (r1 \ [row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 -- r1)
Multiplies the scalar r1 by each element in the source2 row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
and adds the result to the corresponding element of the source1 row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
and places the final result in the corresponding element of the destination row/col specified by
[row#3\-1] or [-1\col#3]\matrix.xpfa3
Thus for each element, dest <- src1 + r1*src2. The destination may be either of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources. The scalar r1 is left on the stack.
Pronunciation: "row-or-col-star-plus" Attributes: S

ROW/COL+ ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 --)
Adds each element of the source1 row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
to the corresponding element of the source2 row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
and stores the result in the corresponding element of the destination row/col specified by

[row#3\-1] or [-1\col#3]\matrix.xpfa3

The destination row/col may be one of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources.

Pronunciation: "row-or-col-plus" Attributes: S

ROW/COL- ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 --)

Subtracts each element of the source2 row/col specified by

[row#2\-1] or [-1\col#2]\matrix.xpfa2

from the corresponding element of the source1 row/col specified by

[row#1\-1] or [-1\col#1]\matrix.xpfa1

and stores the result in the corresponding element of the destination row/col specified by

[row#3\-1] or [-1\col#3]\matrix.xpfa3

The destination row/col may be one of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources.

Pronunciation: "row-or-col-minus" Attributes: S

ROW/COL->V ([row#\-1] or [-1\col#]\matrix.xpfa -- xvaddr1\sep\d.#el)

Converts a row/col specified by

[row#\-1] or [-1\col#]\matrix.xpfa

into the equivalent vector representation specified by

xvaddr1\sep\d.#el

To specify a row in the matrix, use row#\-1\matrix.xpfa. To specify a column in the matrix, use -1\col#\matrix.xpfa. The proper xvaddr, separation, and (32-bit) number of elements that specify the vector representation are returned.

Note: xvaddr is the base address of the vector, sep is the element separation expressed as a multiple of 4 bytes (e.g., sep=1 means a vector of contiguous floating point numbers, sep=2 means elements are separated by 8 bytes, etc.) and double number d.#el is the number of elements in the vector. Note that xvaddr must be 4-byte aligned (i.e., must be an even multiple of 4). The heap manager and array and matrix dimensioning words automatically perform the required 4-byte alignment.

Pronunciation: "row-or-col-to-v"

ROW/COL.ALL= ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2 -- flag)

Compares each element of the source1 row/col specified by

[row#1\-1] or [-1\col#1]\matrix.xpfa1

with the corresponding element of the source2 row/col specified by

[row#2\-1] or [-1\col#2]\matrix.xpfa2

and returns a true flag if all of the corresponding elements are equal. Otherwise returns a false flag.

Pronunciation: "row-or-col-all-equal"

ROW/COL.ANY= ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2 -- flag)

Compares each element of the source1 row/col specified by

[row#1\-1] or [-1\col#1]\matrix.xpfa1
with the corresponding element of the source2 row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
and returns a true flag if any of the corresponding elements are equal. Otherwise returns a false flag.
Pronunciation: "row-or-col-any-equal"

ROW/COL.CENTERED ([row#\-1] or [-1\col#]\matrix.xpfa -- r)
Finds the arithmetic mean r of the specified row or column in the matrix, and subtracts this value from each element in the row or column, leaving the mean value on the data stack.
Pronunciation: "row-or-col-centered" Attributes: S

ROW/COL.COPY ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2 --)
Copies the source row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
to the destination row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2.
Pronunciation: "row-or-col-copy"

ROW/COL.DELETED ([row#1\-1] or [-1\col#1]\matrix.xpfa1\matrix.xpfa2 --)
Copies all but the specified row or column of the source matrix matrix.xpfa1 to the destination specified by matrix.xpfa2. The destination may be the source.

ROW/COL.DOT.PRODUCT ([row#1\-1] or [col#1\-1]\matrix.xpfa1\
[row#2\-1] or [col#2\-1]\matrix.xpfa2 -- r)
r is the dot product of the 2 specified row/cols. The dot product is calculated by multiplying each element in the row/col specified by
[row#1\-1] or [-1\col#1]\ matrix.xpfa1
by the corresponding element in the row/col specified by
[row#2\-1] or [-1\col#2]\ matrix.xpfa2
and summing the results to produce r.
Pronunciation: "row-or-col-dot-product" Attributes: S

ROW/COL.FILL (r\[row#\-1] or [-1\col#]\matrix.xpfa --)
Stores r into each element of the specified row or column in the matrix.
Pronunciation: "row-or-col-fill"

ROW/COL.INSERTED ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2 --)
Inserts the source row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
into the destination matrix specified by matrix.xpfa2 as the row or column specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
The source row/col is unchanged unless it is also the destination. The source may be within the destination matrix.

ROW/COL.IS.UNITY.LENGTH ([row#\-1] or [-1\col#]\matrix.xpfa -- r)

Calculates the length r of the specified row or column and divides each element in the row/column by r so that the row/column has unity length. The old length is left on the stack. The length is defined as the square root of the sum of the squares of the elements in the row or column.
Pronunciation: "row-or-col-is-unity-length" Attributes: S

ROW/COL.MAX ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 --)

Performs the function FMAX on each pair of corresponding elements in the two source row/cols specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
and [row#2\-1] or [-1\col#2]\matrix.xpfa2
and places the result in the destination row/col specified by
[row#3\-1] or [-1\col#3]\matrix.xpfa3
The destination may be either of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources.
Pronunciation: "row-or-col-max"

ROW/COL.MIN ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 --)

Performs the function FMIN on each pair of corresponding elements in the two source row/cols specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
and [row#2\-1] or [-1\col#2]\matrix.xpfa2
and places the result in the destination row/col specified by
[row#3\-1] or [-1\col#3]\matrix.xpfa3
The destination may be either of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources.
Pronunciation: "row-or-col-min"

ROW/COL.SUM ([row#\-1] or [-1\col#]\matrix.xpfa -- r)

r is the sum of all of the elements in the specified row or column in the matrix.
Pronunciation: "row-or-col-sum" Attributes: S

ROW/COL.SWAP ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2 --)

Swaps the contents of the row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
with the contents of the row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2.
Pronunciation: "row-or-col-swap"

ROW/COL.TRANSFORMED ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\ xcfa --)

xcfa specifies a function that transforms a single floating point number into a floating point result. ROW/COL.TRANSFORMED applies this transformation to each element of the source row or column specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1

and places the result in the destination row or column specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
The source row/col can equal the destination.
Pronunciation: "row-or-col-transformed" Attributes: S

ROW/COL.VARIANCE ([row#\-1] or [-1\col#]\matrix.xpfa -- r)
r is the variance, or length squared of the specified row or column in the matrix. It is calculated by performing the dot product of the row or column with itself, which squares each element and sums the squares to produce r.
Pronunciation: "row-or-col-variance" Attributes: S

ROW/COL/ ([row#1\-1] or [-1\col#1]\matrix.xpfa1\
[row#2\-1] or [-1\col#2]\matrix.xpfa2\
[row#3\-1] or [-1\col#3]\matrix.xpfa3 --)
Divides each element of the source1 row/col specified by
[row#1\-1] or [-1\col#1]\matrix.xpfa1
by the corresponding element of the source2 row/col specified by
[row#2\-1] or [-1\col#2]\matrix.xpfa2
and stores the result in the corresponding element of the destination row/col specified by
[row#3\-1] or [-1\col#3]\matrix.xpfa3
The destination row/col may be one of the sources. If the destination is in the same matrix as one or both of the sources, the destination should not intersect either of the sources.
Pronunciation: "row-or-col-slash" Attributes: S

RP! (--)
Return Stack: (R: [...] --)
Initializes the return stack pointer to be equal to the value in the user variable R0, thus clearing the return stack. The first return stack item will be stored in the two bytes below the value in R0, and the stack grows downward in memory. For example, if R0 = 0x9000, the first stack item is at memory locations 0x8FFE and 0x8FFF. Forces a COLD restart if R0 is not in common RAM.
Pronunciation: "r-p-store"

RS485.RECEIVE (--)
Clears bit 4 in PPC (of the PIA) to the logic 0 state. If upper PPC has been configured as an output port, this places the RS485 transceiver in the receive mode. (Make sure that the onboard RS485/RS232 jumper is properly set before attempting to use the RS485 interface). See INIT.RS485 and RS485.TRANSMIT.
Pronunciation: "R-S-four-eighty-five-receive"

RS485.TRANSMIT (--)
Sets bit 4 in PPC (of the PIA) to the logic 1 state. If upper PPC has been configured as an output port, this places the RS485 transceiver in the transmit mode. (Make sure that the onboard RS485/RS232 jumper is properly set before attempting to use the RS485 interface). See INIT.RS485 and RS485.RECIEVE.
Pronunciation: "R-S-four-eighty-five-transmit"

RTI.ID (-- n)

Returns the interrupt identity code for the real time interrupt. Used as an argument for ATTACH.
Pronunciation: "r-t-i-i-d"

S*MATRIX (r\matrix.xpfa1\matrix.xpfa2 --)
Multiplies the scalar r by each element in the source matrix specified by matrix.xpfa1 and stores the result in the corresponding element of the destination matrix specified by matrix.xpfa2. The source and destination matrices may be the same.
Pronunciation: "s-star-matrix" Attributes: S

S+MATRIX (r\matrix.xpfa1\matrix.xpfa2 --)
Adds the scalar r to each element in the source matrix specified by matrix.xpfa1 and stores the result in the corresponding element of the destination matrix specified by matrix.xpfa2. The source and destination matrices may be the same.
Pronunciation: "s-plus-matrix" Attributes: S

S-MATRIX (r\matrix.xpfa1\matrix.xpfa2 --)
Subtracts each element in the source matrix specified by matrix.xpfa1 from the scalar r and stores the result in the corresponding element of the destination matrix specified by matrix.xpfa2. The source and destination matrices may be the same.
Pronunciation: "s-minus-matrix" Attributes: S

S.ROW/COL* (r\[row#\-1] or [-1\col#]\matrix.xpfa --)
Multiplies r by each element in the specified row or column in the matrix and stores the result back into the element.
Pronunciation: "s-row-or-col-star" Attributes: S

S.ROW/COL+ (r\[row#\-1] or [-1\col#]\matrix.xpfa --)
Adds the scalar r to each element in the specified row or column in the matrix and stores the result back into the element.
Pronunciation: "s-row-or-col-plus" Attributes: S

S.ROW/COL- (r\[row#\-1] or [-1\col#]\matrix.xpfa --)
Subtracts each element in the specified row or column in the matrix from r and stores the result back into the element.
Pronunciation: "s-row-or-col-plus" Attributes: S

S.ROW/COL/ (r\[row#\-1] or [-1\col#]\matrix.xpfa --)
Divides r by each element in the specified row or column in the matrix and stores the result back into the element.
Pronunciation: "s-row-or-col-divide" Attributes: S

S.ROW/COL< (r\[row#\-1] or [-1\col#]\matrix.xpfa -- flag)
flag is true if r is less than each element in the specified row or column in the matrix. Alternate interpretation: flag is true if all elements in the row or column are greater than or equal to r.
Pronunciation: "s-row-or-col-less-than"

S.ROW/COL> (r\[row#\-1] or [-1\col#]\matrix.xpfa -- flag)

flag is true if r is greater than each element in the specified row or column in the matrix. Alternate interpretation: flag is true if all elements in row or column are less than or equal to r.
Pronunciation: "s-row-or-col-greater-than"

S.V* (r\xvaddr\sep\d.#el --)
Multiplies r (a scalar) by each element in the vector specified by xvaddr\sep\d.#el.
Pronunciation: "s-v-star" Attributes: S

S.V+ (r\xvaddr\sep\d.#el--)
Adds r (a scalar) to each element in the vector specified by xvaddr\sep\d.#el.
Pronunciation: "s-v-plus" Attributes: S

S.V- (r\xvaddr\sep\d.#el--)
Subtracts each element in the vector specified by xvaddr\sep\d.#el from the scalar r.
Pronunciation: "s-v-minus" Attributes: S

S.V.ALL= (r\xvaddr\sep\d.#el -- flag)
Compares the scalar r to each element of the vector specified by xvaddr\sep\d.#el. flag is TRUE if r is equal to each of the elements in the vector; otherwise, flag is FALSE.
Pronunciation: "s-v-all-equal"

S.V.ANY= (r\xvaddr\sep\d.#el -- flag)
Compares the scalar r to each element of the vector specified by xvaddr\sep\d.#el. flag is TRUE if r is equal to any of the elements in the vector; otherwise, flag is FALSE.
Pronunciation: "s-v-any-equal"

S.V/ (r\xvaddr\sep\d.#el --)
Divides the scalar r by each element in the vector specified by xvaddr\sep\d.#el.
Pronunciation: "s-v-slash" Attributes: S

S.V< (r\xvaddr\sep\d.#el -- flag)
Compares the scalar r to each element of the vector specified by xvaddr\sep\d.#el. flag is true if r is less than each of the elements in the vector; otherwise, flag is FALSE.
Pronunciation: "s-v-less-than"

S.V> (r\xvaddr\sep\d.#el -- flag)
Compares the scalar r to each element of the vector specified by xvaddr\sep\d.#el. flag is true if r is greater than each element in the vector; otherwise, flag is false.
Pronunciation: "s-v-greater-than"

S/MATRIX (r\matrix.xpfa1\matrix.xpfa2 --)
Divides the scalar r by each element in the source matrix specified by matrix.xpfa1 and stores the result in the corresponding element of the destination matrix specified by matrix.xpfa2. The source and destination matrices may be the same.
Pronunciation: "s-slash-matrix" Attributes: S

S0 (-- xaddr)
User variable that contains the 16-bit address which is used by SP! to initialize the data stack pointer. The first cell on the data stack occupies the 2 bytes below the address

contained in S0, and the stack grows downward in memory. After changing the contents of S0, the next ABORT or restart loads the value into the stack pointer (the Y register) to change the position of the stack. The data stack is allocated in a 768 byte region in common memory after each COLD restart. See SP!.
Pronunciation: "s-zero" Attributes: U

S>D (n -- d)
Sign-extends a single precision integer to a double precision equivalent.
Pronunciation: "s-to-d"

SAVE (--)
Saves the current memory map so that it may be restored later. Saves the DP, NP, VP, last xnfa in the FORTH vocabulary, and CURRENT.HEAP in a reserved area in EEPROM (0xAE0C to 0xAE1A). RESTORE fetches these quantities and places them in the appropriate user variables to restore the saved state. Useful for dictionary management and for recovery from crashes. Consult the "Program Development Techniques" chapter in the Software Manual for a detailed description of use. See (EEC!).

SCALE (n1\n2 -- n3)
Arithmetically (i.e., preserving sign) shifts n1 by n2 bit places to yield signed result n3. If n2 is positive, n1 is shifted left; if n2 is negative, n1 is shifted right. The absolute value of n2 determines the number of bits of shifting. For example, 1 SCALE is equivalent to 2* and -1 SCALE is equivalent to 2/ . There is an unchecked error if the absolute value of n2 is greater than 15.

SCAN (xaddr1\u1\char -- xaddr2\u2)
xaddr2 is the extended address of the first instance of the specified char in the u1 bytes following xaddr1. u2 is the count remaining in the string after the first non-char bytes have been skipped:

u2 = u1 - (xaddr2 - xaddr1)

u1 and u2 are 16 bit counts. The string may cross a page boundary. SCAN is used by WORD to locate the trailing delimiter of the next word in the input stream.

SCI.ID (-- n)
Returns the interrupt identity code for the asynchronous serial communications interface. Used as an argument for ATTACH.
Pronunciation: "s-c-i-i-d"

SCIENTIFIC (--)
Sets the default printing format used by F. to scientific. See F>SCIENTIFIC$

SCIENTIFIC. (r --)
Prints r using SCIENTIFIC format. See F>SCIENTIFIC$
Pronunciation: "scientific-dot" Attributes: M, S

SCR (-- xaddr)
A user variable containing the block number of the last block (also known as a screen) listed. The name is derived from the first 3 letters of screen. See LIST.
Pronunciation: "s-c-r" Attributes: U

SELECT.COLUMNS (n1\...\nN\#cols\matrix.xpfa1\matrix.xpfa2 --)

Dimensions the destination matrix specified by matrix.xpfa2 to have #cols columns and the same number of rows as the source specified by matrix.xpfa1. Copies #cols columns having column indices n1\...\nN from the source matrix to the destination. The destination may be the source.

SEND (wd\xmailbox --)

PAUSEs until the mailbox with extended address xmailbox is empty (i.e., contains 0\0) and then stores the 32-bit message wd in xmailbox. The message wd can be any 32-bit quantity except 0\0. For example, the message can be an xaddress that points to a block of data. To ensure that the state of the mailbox is correctly determined, SEND disables interrupts for 16 to 50 cycles (4 to 12.5 microseconds). See ?SEND, RECEIVE, ?RECEIVE, and MAILBOX:.
Attributes: M

SERIAL (-- xresource)

A resource variable associated with the primary serial I/O port. A synonym for SERIAL1.RESOURCE. See SERIAL1.RESOURCE.

SERIAL.ACCESS (-- xaddr)

A user variable containing a flag that controls when a task GETs and RELEASEs access to the serial resource. If more than one task needs access to the serial I/O port, this flag can help specify which task (if any) gets priority use. If SERIAL.ACCESS contains the value RELEASE.ALWAYS, then each I/O operation by KEY EMIT or ?KEY GETs the active serial resource before each I/O operation and RELEASEs the active serial resource after each character I/O operation is complete. If SERIAL.ACCESS contains the value RELEASE.NEVER, then I/O operations called by the task always GET but never RELEASE the serial resource variable. If SERIAL.ACCESS contains the value RELEASE.AFTER.LINE, then KEY EMIT and ?KEY never GET or RELEASE the serial resource. Rather, the QED-Forth interpreter (that is, QUIT) GETs the serial resource before each line is received and RELEASEs the serial resource after each line is interpreted. This virtually eliminates the overhead required to GET and RELEASE during downloads, and allows the interpreter to run at sustainable baud rates to 19200 baud. The default value stored in SERIAL.ACCESS after a COLD restart is RELEASE.AFTER.LINE.
CAUTION: In multitasking systems using both serial ports SERIAL1 and SERIAL2, the application code should include the command

```
     RELEASE.ALWAYS  SERIAL.ACCESS !
or   RELEASE.NEVER  SERIAL.ACCESS !
```

before building the tasks. This prevents contention that can occur if the default RELEASE.AFTER.LINE option is installed in the SERIAL.ACCESS user variable. See SERIAL1.RESOURCE, SERIAL2.RESOURCE, GET, RELEASE, KEY, EMIT, ?KEY, and QUIT.
Attributes: U

SERIAL1.AT.STARTUP (--)

Initializes a flag in EEPROM which installs the primary serial port (serial1) as the default serial port used by the QED-Forth interpreter after each reset or restart. The serial1 port is supported by the 68HC11's on-chip hardware UART.

Implementation detail: Sets the contents of address AE1DH in EEPROM to 0xFF. Upon each reset or restart, the QED-Forth startup routine checks this byte, and contents of 0xFF cause the USE.SERIAL1 routine to be executed. See USE.SERIAL1 and SERIAL2.AT.STARTUP.
Pronunciation: "serial-one-at-startup"

SERIAL1.RESOURCE (-- xaddr)

A resource variable that mediates access to the primary serial port (serial1) associated with the 68HC11's on-chip hardware UART. Should be accessed only by the words GET ?GET and RELEASE. Initialized to 0\0 by USE.SERIAL1 and USE.SERIAL2 and at each reset or restart. See RESOURCE.VARIABLE:.
Pronunciation: "serial-one-resource"

SERIAL2.AT.STARTUP (u -- | u = baud.rate)

Initializes a flag in EEPROM which installs the secondary serial port (serial2) at the specified baud rate u as the default serial port used by the QED-Forth interpreter after each reset or restart. The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). The specified baud rate u must a power of 2 times 75 baud up to a maximum of 9600 baud. Thus the allowed baud rates for this routine are 75, 150, 300, 600, 1200, 2400, 4800, and 9600 baud. The effect of this routine is canceled by executing SERIAL1.AT.STARTUP. Note that the serial2 port can support many more baud rates, but the options have been limited to facilitate setting a reasonable startup baud rate based on a simple implementation as described below. Note also that the maximum baud rate that can be sustained by the serial2 port is less than 9600 baud; see the glossary entry for BAUD2.
Implementation detail: Sets the contents of address AE1DH in EEPROM equal to u/75. Upon each reset or restart, the QED-Forth startup routine checks this byte, and contents equal to an exact power of two cause the USE.SERIAL2 routine to be executed before control is passed to the interpreter or to an autostart routine. Note that USE.SERIAL2 globally enables interrupts during the startup process. If you wish to use the secondary serial port while avoiding this side-effect and maintaining control over the global enabling of interrupts, don't execute SERIAL2.AT.STARTUP. Rather, have your autostart routine explicitly call USE.SERIAL2 after ensuring that all interrupt service routines are properly initialized.
Pronunciation: "serial-two-at-startup"

SERIAL2.RESOURCE (-- xaddr)

A resource variable that mediates access to the secondary serial port (serial2). The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). Should be accessed only by the words GET ?GET and RELEASE. Initialized to 0\0 by USE.SERIAL1 and USE.SERIAL2 and at each reset or restart. See RESOURCE.VARIABLE:.
Pronunciation: "serial-two-resource"

SET.BITS (byte1\xaddr --)

For each bit of byte1 that is set, sets the corresponding bit of the 8 bit value at xaddr. Disables interrupts for ten cycles (2.5 microseconds) to ensure an uninterrupted read/modify/write operation. See also (SET.BITS) and CLEAR.BITS.

SET.HIGH.CURRENT (byte --)

For each bit of the input mask byte that is set, turns the corresponding high current driver ON (so that it is sinking current). Bits 0-3 in the input mask byte control the high current drivers named HC0-HC3, respectively. Disables interrupts for 31 cycles (less than 8 microseconds). See also CLEAR.HIGH.CURRENT.

SET.WATCH (u1\u2\u3\u4\u5\u6\u7\u8 --)

meaning: (100ths.sec\sec\min\hr\day\date\month\yr --)

Sets the battery-operated real-time clock (if present) to the time, day, and date specified by u1 through u8. The stack items and their allowed ranges are:

item	description	range (decimal)
u8	year	0 - 99
u7	month	1 - 12
u6	date	1 - 31
u5	day of week	1 - 7
u4	hour of day	0 - 23
u3	minute after the hour	0 - 59
u2	seconds after the minute	0 - 59
u1	hundredths of seconds	0 - 99

Resolution is better than +/- 1 minute per month. Once correctly set, the watch handles the differing numbers of days in each month, and correctly handles leap years. SET.WATCH uses the top 16 bytes of on-chip RAM at B3F0-B3FF as a scratchpad buffer, and disables interrupts for 0.45 msec while accessing the watch. See READ.WATCH.

SIGN (n --)

If n is negative, inserts a minus sign into the pictured output to the left of the previous character. Used between <# and #>.

Attributes: S

SIGNED.D>S (d -- n)

n is the signed 16-bit representation of d. FP.ERROR is set if d cannot be represented as a 16-bit signed integer.

Pronunciation: "signed-d-to-s" Attributes: S

SINGLE.STEP (-- xaddr)

A user variable that holds a flag. If the flag is true, a definition that has been compiled with TRACE ON stops and enters a special BREAK interpreter before each instruction during the trace. If the flag is false, the BREAK interpreter is not automatically entered before each instruction during the trace. (Even if the flag is false, a keystroke from the terminal may still be used to enter the BREAK mode while the trace is in progress.) See BREAK.

Attributes: U

SIZE.OF (-- u)

Compile Time: (<name> --)

Removes <name> from the input stream, where <name> is a heap structure instance defined using D.INSTANCE: or H.INSTANCE: or V.INSTANCE: (an unchecked error occurs if <name> was not created by one of these three defining words). SIZE.OF returns the size in bytes of <name>. If executing, leaves u on the stack. If compiling, SIZE.OF compiles u as a literal in the current definition.

Attributes: I

SKIP (xaddr1\u1\char -- xaddr2\u2)
Skips the leading specified chars in the string whose first character is at xaddr1 and whose count is u1, returning the string specification xaddr2\u2. xaddr2 is the address of the first byte not equal to char found after searching from xaddr1 to at most xaddr1+u1 . u2 is the count remaining after the leading chars have been skipped:
u2 = u1 - (xaddr2 - xaddr1)
u1 and u2 are 16 bit counts. The string may cross a page boundary. This routine is used by WORD to skip leading spaces when parsing the input stream.

SKIP> (xaddr\u1\char -- xaddr\u2)
Strips the trailing characters specified by char from the string located at xaddr by adjusting the count of the string. Returns the new character count, u2, of the text string with trailing chars removed. The string may cross a page boundary.
Pronunciation: "skip-back"

SMUDGE (--)
Toggles (i.e., reverses the state of) the smudge bit in the header of the most recently defined word in the CURRENT vocabulary. If the smudge bit is set, the word cannot be found during a search of the dictionary. The smudge bit is used to prevent execution of incomplete definitions. Used by : ; CODE and END.CODE.

SOLVE.EQUATIONS
(coefficient.matrix.xpfa\residue.matrix.xpfa\solution.matrix.xpfa --)
Solves a set of simultaneous linear equations.
Example of use:
If the system of equations is represented by the matrix equation
M X = B
where M is a matrix of coefficients, X is a column matrix of unknown quantities (to be solved for), and B is a column matrix representing the right hand side (residue) of the equations, first define matrices M B and X using the MATRIX: command, initialize M and B, and then execute
' M ' B ' X SOLVE.EQUATIONS
which dimensions the matrix X and stores into it the solution for the unknown quantities.
Attributes: S

SP! ([...] --)
Initializes the data stack pointer to be equal to the value in the user variable S0, thus clearing all items off the data stack. The first stack item will be stored in the two bytes below the value in S0, and the stack grows downward in memory. For example, if S0 = 0x9000, the first stack item will be at memory locations 0x8FFE and 0x8FFF. Forces a COLD restart if S0 is not in common RAM.
Pronunciation: "s-p-store"

SPACE (--)
Emits one space character. Equivalent to BL EMIT
Attributes: M

SPACES (+byte --)

Emits +byte spaces. Does nothing if +byte is negative.
Attributes: M

SPAN (-- xaddr)
A user variable that contains the number of characters received by the last execution of EXPECT.
Attributes: U

SPEED.TO.DUTY (steps_per_second\ticks_per_second -- duty_cycle)
Returns an integer representation of a duty cycle which specifies the step rate of the stepper motor. The first input parameter is the integer number of steps per second if full stepping, or the number of halfsteps per second if half stepping. The second input parameter is the integer number of clock ticks per second; the default is 1000 ticks per second. The integer output parameter can be interpreted as a fraction with the radix point to the left of the most significant bit. A 100% duty cycle is represented by 0xFFFF, and this tells the STEP.MANAGER to output a new step pattern on every tick of the interrupt clock (e.g., once per millisecond, corresponding to 1000 (half) steps per second). A duty cycle of 0x8000 means a new step pattern is written to the motor port every other clock tick; a duty cycle of 0x0100 dictates one step every 256 clock ticks; and a duty cycle of 0000 means corresponds to a stopped state with no step pattern updates. See the high level source file steppers.4th in the Demos_and_Drivers directory of the distribution.

SPI.ID (-- n)
Returns the interrupt identity code for the synchronous serial peripheral interface (SPI). Used as an argument for ATTACH. Note that the SPI communicates with the onboard 12 bit A/D and 8 bit D/A if they are installed. See INIT.SPI, SPI.ON, and SPI.OFF.
Pronunciation: "s-p-i-i-d"

SPI.OFF (--)
Disables the serial peripheral interface (SPI) by clearing the SPI enable (SPE) bit in the SPI control register (SPCR). After execution of this routine, PORTD pins PD2-PD5 may be used as standard digital I/O subject to the data direction specified in the PORTD.DIRECTION register. See INIT.SPI.
Pronunciation: "init-S-P-I"

SPI.RESOURCE (-- xaddr)
A resource variable associated with the serial peripheral interface (SPI) which is used for data transfer to and from the 12 bit analog to digital converter and 8 bit digital to analog converter. Should be accessed only by the words GET ?GET and RELEASE. Initialized to 0\0 by INIT.SPI and INIT.A/D12&DAC and at each reset or restart. SPI.RESOURCE is automatically invoked by many of the A/D12 and DAC device driver routines. See RESOURCE.VARIABLE:.
Pronunciation: "S-P-I-resource"

SQRT(2) (-- r)
Places the floating point representation of the square root of 2 (1.41421) on the stack.
Pronunciation: "square-root-of-two"

STACK.FRAME (+n -- [+n bytes]\xaddr)

Reserves +n bytes of room on the data stack and leaves xaddr that points to the top (lowest in memory) reserved byte of the data stack frame. xaddr is equal to the stack pointer before the xaddr is placed on the stack. xaddr is the base address of the stack frame. If +n is odd it is incremented to reserve an integer number of cells (of two bytes each) on the stack. STACK.FRAME is typically used to create a temporary variable space within colon definitions so that re-entrant code may be written. FRAME.DROP is used to drop the stack frame off the data stack.

STANDARD.MAP (--)

Sets a flag in EEPROM and changes the state of a latch in the onboard PALs to put the standard memory map into effect on flash-equipped QED-FLASH Boards. After execution of this routine, and upon each subsequent reset or restart, pages 1, 2, and 3 are addressed in the S2 RAM, and pages 4, 5, and 6 are addressed in the S1 flash memory. After code is downloaded to RAM and transferred to flash using the PAGE.TO.FLASH function, establishing the standard map allows code resident on pages 4, 5 and 6 to be executed. To establish the download memory map, see the glossary entry for DOWNLOAD.MAP. Note that the standard map is active after a "factory cleanup" operation.

STANDARD.RESET (--)

Undoes the effect of the COLD.ON.RESET command so that subsequent resets will result in the standard warm-or-cold startup sequence. Implementation detail: sets the flag at location AE1CH in EEPROM to 0xFF.

START.HEAP (-- xaddr)

A variable that holds the extended address of the start of the current heap. The xaddr left on the stack by START.HEAP is equal to CURRENT.HEAP - 4. Initialized by IS.HEAP.

START.TIMESLICER (--)

Starts the timeslice clock and begins timeslice multitasking. Initializes the OC2 interrupt vector (if it wasn't already initialized) so that the multitasking executive/elapsed-time clock routine services the interrupt. Enables the OC2 interrupt mask and globally enables interrupts by clearing the I bit in the condition code register of each built task.
Notes:
1. The default timeslice clock period of 5 msec can be changed with the command *100US=TIMESLICE.PERIOD.
2. START.TIMESLICER does not initialize the value in TIMESLICE.COUNT; execute INIT.ELAPSED.TIME if you wish to initialize the clock count to 0\0.
3. After a restart, the system is configured so that timeslice multitasking can begin at any time; if no other tasks have been built, the main QED-Forth task is the only task in the task loop.
4. The timeslice clock must be running to use the BENCHMARK: function.
5. The timeslicer's interrupt service routine disables interrupts for the duration of a task switch which requires 25 microseconds plus 3.25 microseconds for each ASLEEP task encountered in the task list.

STATE (-- xaddr)

A user variable that indicates the compilation state. If the contents of STATE equal 0, the system is in execution mode. If the contents equal -1, the system is in compilation mode. STATE is modified by the commands [and].
Attributes: U

STATUS (-- xaddr | xaddr is also the task's xtask.id)
A user variable that contains the status of a task. Typically contains one of the 16-bit constants AWAKE or ASLEEP. An ASLEEP task does not run as the multitasking executive goes around the round robin task list. STATUS is the first user variable in the user area, so the extended address returned by executing STATUS is also the base address of the user area. This base address is also referred to as the "task identifier" or "task id"; it is the address in common memory used to identify a particular task. See also (STATUS).
Attributes: U

STEP.MANAGER (--)
Expects the base address of the STATUS.ARRAY in the Y register. Based on the information in the STATUS.ARRAY and the RAMP.ARRAY, for each enabled motor STEP.MANAGER writes the appropriate step pattern at the specified duty cycle to the motor port to attain the speed specified in the motor's RAMP.ARRAY. This function is meant to be called from a periodic interrupt service routine typically associated with an output compare (OC) interrupt; the default time base is once per millisecond generated by the OC3 interrupt, with a resulting maximum speed of 1000 full- or half-steps per second. This assembly coded routine executes in approximately 120 µs per enabled stepper motor. Thus running four stepper motors at a maximum speed of 1000 full- or half-steps per second requires approximately half of the 68HC11's available time (480 µs interrupt service time every 1000 µs). See the high level source file steppers.4th in the Demos_and_Drivers directory of the distribution. CAUTION: The presence of other interrupt service routines can affect the timing of the step manager, and may affect the smoothness of the stepper motor operation.

STOP.TIMESLICER (--)
Stops the multitasker's timeslice clock by disabling the local OC2 timer interrupt mask. Cooperative (PAUSE-invoked) task switching is not affected. See START.TIMESLICER. Note that this command also stops QED-Forth's elapsed-time clock, and that the timing feature of the BENCHMARK: command cannot be used unless the timeslice clock is running.

STRING-> (u1\u2 -- u3)
Adds a named member to the structure being defined and reserves room for a counted string in the structure. u2 is the number of characters in the string; u2+1 bytes are reserved to allow room for the string's count byte. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "string" Attributes: D

STRUCT-> (u1\u2 <name> -- u3)

Adds a named member to the structure being defined and reserves room for a (sub)structure of size u2 bytes in the structure being defined. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.

Pronunciation: "struct" Attributes: D

STRUCTS-> (u1\u2\u3 <name> -- u4)

Adds a named member to the structure being defined and reserves room for u2 (sub)structures in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the size of the (sub)structure, and u4 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.

Pronunciation: "structs" Attributes: D

STRUCTURE.BEGIN: (<name> -- xpfa\0)

Begins a structure definition and creates a named constant <name> which, when executed, returns the size of the structure in bytes. See the "Structures" chapter in the Software Manual for a detailed description and examples of use.

Implementation detail: <name>'s parameter field is at xpfa, and its initial contents equal 0. STRUCTURE.BEGIN: leaves the xpfa and the initial size on the stack. The size is incremented throughout the structure's definition. STRUCTURE.END stores the total size of the structure into the xpfa of the structure constant <name>.

Pronunciation: "structure-begin" Attributes: D

STRUCTURE.END (xpfa\u -- | u is the structure's size)

Marks the end of a structure definition. Stores u, the total number of bytes in the structure being defined, into the xpfa of the structure constant created by STRUCTURE.BEGIN:. See STRUCTURE.BEGIN:.

Attributes: D

SWAP (w1\w2 -- w2\w1)

Exchanges the top two stack cells.

SWAP.ARRAYS (array.xpfa1\array.xpfa2 --)

Interchanges the contents of the parameter fields of the two specified arrays and leaves the heap undisturbed, thus rapidly swapping the two arrays.

SWAP.MATRIX (matrix.xpfa1\matrix.xpfa2 --)

Interchanges the contents of the parameter fields of the two specified matrices and leaves the heap undisturbed, thus rapidly swapping the two matrices.

SWI.ID (-- n)

Returns the interrupt identity code for the software interrupt (SWI). Used as an argument for ATTACH. See the SWI instruction in the assembler glossary.
Pronunciation: "s-w-i-i-d"

TAB.WIDTH (-- xaddr)
A user variable that contains the number of spaces that EXPECT places in the TIB to replace each incoming TAB character (ascii 09). Replacing tabs with spaces ensures that tab-delimited words can be interpreted. The default is 4. See EXPECT.
Pronunciation: "tab-width" Attributes: U

TASK'S.USER.VAR (xaddr1\xtask.id -- xaddr2)
Converts the xaddress of a specified user variable xaddr1 in the current task to the xaddress of the equivalent user variable xaddr2 in the task specified by xtask.id. Facilitates the modification of user variables in other tasks. Use with care. For example, to put a task named OTHER.TASK asleep, execute

```
ASLEEP  STATUS  OTHER.TASK  TASK'S.USER.VAR  !
```

Pronunciation: "task's-user-variable"

TASK: (xtask.id <name> --)
Removes the next <name> from the input stream and creates an XCONSTANT that, when executed, leaves the task identifier xtask.id on the stack. xtask.id is the base xaddress of the user area of the new task being defined. An error is issued if xtask.id is not in common RAM. xtask.id is also referred to as the task's STATUS address. See STATUS.
Attributes: D

TEN (-- r)
Pushes the floating point number 10. onto the data stack.

THEN (--)
Synonym for ENDIF . Used inside a colon definition to mark the end of an IF ... ELSE ... THEN or IF ... THEN conditional structure. The word following THEN is executed after the IF or ELSE (if present) part of the conditional executes. An error is issued if THEN is not paired with IF or ELSE in a colon definition. See IF and ELSE.
Attributes: C, I

THIS.PAGE (-- byte)
Returns the contents of the page latch which indicates the current page. THIS.PAGE is equivalent to (PAGE.LATCH) (C@)

TIB (-- xaddr | xaddr is the start of the terminal input buffer)
Returns the starting xaddr of the terminal input buffer. Equivalent to UTIB X@. The terminal input buffer may be on any page, but may not cross a page boundary. The default size of the terminal input buffer is 96 bytes. See QUERY.
Attributes: U

TIMER.OVERFLOW.ID (-- n)
Returns the interrupt identity code for the free-running timer overflow interrupt. Used as an argument for ATTACH.
Pronunciation: "timer-overflow-i-d"

TIMESLICE.COUNT (-- xaddr)

Returns the extended address of the system variable TIMESLICE.COUNT. It contains a 32-bit count of the number of clock ticks on the timeslicer clock. The period of the clock is set by *100US=TIMESLICE.PERIOD. See READ.ELAPSED.SECONDS and READ.ELAPSED.TIME.

TO ([n] or [r] or [xaddr] or [d] -- | depends on type of self fetchers or locals)

Compile Time: (<name> --)

Stores a value into the named self-fetching or local variable. Removes <name> from the input stream. If in compilation mode, compiles code that, when later executed, will store the value on top of the stack into the self-fetching or local variable. If in execution mode, stores the value on top of the stack into the self-fetching or local variable. If <name> represents a 32-bit self-fetching variable (i.e., created by REAL: DOUBLE: or XADDR:) or a 32-bit local variable (i.e., one whose name begins with D& or F& or X&), then a 32-bit value is removed from the stack and stored; otherwise, a 16-bit value is removed from the stack and stored.

Attributes: I

TO.FLASH (xaddr1\xaddr2\u -- success_flag | xaddr1=src, xaddr2=dest, u = byte count)

Transfers num bytes (0 <= num <= 65,535) starting at the specified source extended address, to the specified destination extended address in flash. The source may be anywhere in memory; it may even be in the flash which is being programmed. The destination must be in flash. Returns a flag equal to -1 if the programming was successful, or 0 if the programming failed. Reasons for failure include improper DIP switch settings, or a destination that is not in a programmable page in flash memory. Recall that, on a QED-Flash Board, DIP switches number 2, 3, and 4 must be ON to allow writes to the flash. (If any locations in the flash are programmed more than 10,000 times, the cell may wear out causing a failure flag to be returned). Assuming that the standard 256 Kbyte flash is present on the board, writable flash pages include pages 4, 5 and 6 for the standard map, and pages 1, 2, and 3 for the download memory map. Page 7 is always in flash and writable; it provides an excellent location for data or graphics storage. Page 0x0D is also writeable flash, and is often used to hold kernel extension code. This function uses the 68HC11's on-chip RAM at 0xB200 to 0xB3CF to manage the write to the flash (the real-time clock and C/Forth interrupt stack reserve the bytes at 0xB3D0 to 0xB3FF). The remaining on-chip RAM at 0xB000 to 0xB1FF remains available to the user. Caution: the prolonged disabling of interrupts by TO.FLASH can adversely affect real-time servicing of interrupts including those associated with the secondary serial line. See PAGE.TO.FLASH.

TO.HEAP (xhandle -- flag)

If xhandle is a valid 32-bit handle in the current heap, the heap item associated with the xhandle is returned to the heap (de-allocated), the heap is compacted, and a true flag is returned. If xhandle is not a valid handle in the current heap, no action is taken and a false flag is returned.

TOGGLE.BITS (byte1\xaddr --)

For each bit of byte1 that is set, reverses the state of the corresponding bit of the 8 bit value at xaddr. Disables interrupts for ten cycles (2.5 microseconds) to ensure an uninterrupted read/modify/write operation. See also (TOGGLE.BITS).

TRACE (-- xaddr)

A user variable that contains a flag. If true, this flag causes a call to a trace routine to be compiled before each compiled word in a colon or code definition. The trace instruction (a headerless routine called DO.TRACE) can facilitate debugging. If a definition has been compiled while TRACE is ON, then when the word is executed (if DEBUG is ON) the compiled trace routine prints out the name of each called subroutine in the definition as it executes, along with the stack picture after that step in the definition. If DUMP.REGISTERS is ON, the contents of the 68HC11's registers are printed; this aids in the debugging of assembly coded routines. If SINGLE.STEP is ON, the BREAK mode is entered after each step in the definition. The BREAK mode is also entered if any character is received by QED-Forth while a trace is in progress. To add even more flexibility and power to the debugger, the first thing that DO.TRACE does is to execute the code whose xcfa is stored in the user variable TRACE.ACTION. The default action is NO.OP, but the programmer can define any action and install it using IS.TRACE.ACTION; see IS.TRACE.ACTION for further details. See also BREAK, DEBUG, DUMP.REGISTERS, and SINGLE.STEP.

Attributes: U

TRAILING.ZEROS (-- xaddr)

A user variable that contains a flag. If the flag is false, trailing zeroes are not printed when a floating point number is displayed in fixed or floating format. If true, trailing zeros are displayed. See F>FIXED$ and F>FLOATING$.

Attributes: U

TRANSFER.HEAP.ITEM (xhandle1\xaddr -- [xhandle2] or [0\0])

Copies the heap item specified by xhandle1 in the current heap into the heap whose CURRENT.HEAP is equal to xaddr. If the operation is successful, returns the 32-bit handle xhandle2 of the new heap item; if unsuccessful, does nothing and returns 0\0. To copy a heap item within a single heap, see DUP.HEAP.ITEM.

TRANSFORM.MATRIX (matrix.xpfa1\matrix.xpfa2\xcfa --)

xcfa specifies a unary operator that transforms a floating point number into another floating point number. TRANSFORM.MATRIX applies the specified transformation to each element of the source matrix specified by matrix.xpfa1 and places the result in the destination matrix specified by matrix.xpfa2. The source and destination matrices may be the same.

For example, to replace each element in a matrix named MAT.A with its base 10 logarithm, execute:

```
    ' MAT.A  XDUP  CFA.FOR  FLOG10  TRANSFORM.MATRIX
```

Pronunciation: "transform-matrix" Attributes: S

TRANSMITTING (-- xaddr)

Returns the extended address of a system variable that holds a flag. The flag is true if the secondary serial port (serial2) is in the process of transmitting a character. If the serial2 transmitter is active, the TRANSMITTING flag stays true until the serial2 output buffer is empty. The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output).

TRANSPOSED (matrix.xpfa1\matrix.xpfa2 --)

Properly dimensions the destination matrix and places the transpose of the source matrix.xpfa1 in the destination matrix.xpfa2. (Matrix transposition changes rows in the source into columns in the destination, and columns in the source into rows in the destination). The source and the destination may be the same.
Attributes: S

TRUE (-- flag | flag = -1)
Puts a boolean true flag equal to -1 on the data stack.

TUCK (w1\w2 -- w2\w1\w2)
Copies the top data stack cell to below the next data stack cell. TUCK is equivalent to SWAP OVER.

TYPE (xaddr\cnt --)
If cnt is greater than zero, emits cnt characters beginning at location xaddr. xaddr is typically the beginning of a text string and cnt is the text string's character count. There is an unchecked error if cnt is outside the range 0 to 255. The string may cross a page boundary. See COUNT.TYPE.
Attributes: M

TYPE.END (u1\u2\u3 -- max{u1,u2,u3})
Marks the end of a TYPE.OF: construct within a structure definition. See TYPE.OF: and OR.TYPE.OF:
Attributes: D

TYPE.OF: (u -- u\u\u)
Marks the beginning of a TYPE.OF: construct within a structure definition. This allows fields to be defined for variant data types. Each variant type is designated by an OR.TYPE.OF: declaration, and the TYPE.OF: construct is terminated by TYPE.END. For example, the following structure allows a member to be referred to either as a 32-bit xhandle or as a separate handle and page:

```
STRUCTURE.BEGIN: HEAP.STRUCTURE.PF
        TYPE.OF:
                XHNDL->     +XHANDLE
        OR.TYPE.OF:
                PAGE->      +HNDL.PAGE
                ADDR->      +HNDL.ADDR
        TYPE.END
                ADDR->      +END.HEAP
STRUCTURE.END
```

Pronunciation: "type-of" Attributes: D

U*/MOD (u1\u2\u3 -- u4\u5 | do u1*u2/u3; u4 = remainder; u5 = quotient)
Multiplies two unsigned integers u1 and u2 producing an intermediate unsigned double number result which is divided by unsigned integer u3 to yield an integer remainder u4 and quotient u5. An unchecked error occurs on overflow. Division by zero (u2=0) yields u4 = u5 = -1 . See */MOD.
Pronunciation: "u-star-slash-mod"

U. (u --)

Prints unsigned integer u with no leading spaces and 1 trailing space.
Pronunciation: "u-dot" Attributes: M, S

U.INVERTED (matrix.xpfa1\matrix.xpfa2 --)
Inverts an upper triangular matrix specified by matrix.xpfa1 and puts the inverse in the destination matrix.xpfa2. Ignores the lower half of the source matrix so it doesn't matter if the elements are not truly zero. The source and destination may be the same.
Pronunciation: "u-inverted" Attributes: S

U/ (u1\u2 -- u3 | u3 = u1/u2)
Divides unsigned integer u1 by unsigned integer u2, giving the unsigned integer quotient u3. Division by 0 (u2 = 0) results in a quotient of -1. This is the fastest integer divide command. See /.
Pronunciation: "u-slash"

U/MOD (u1\u2 -- u3\u4 | u3 = remainder, u4 = quotient)
Divides unsigned integer u1 by u2, giving the unsigned integer quotient u4 and remainder u3. Division by 0 (u2 = 0) results in a quotient of -1 and an indeterminant remainder. This is the fastest integer divide with remainder. See /MOD.
Pronunciation: "u-slash-mod"

U2/ (u1 -- u2 | u2 = u1 / 2)
Divides the unsigned number u1 by 2 giving u2. See 2/.
Pronunciation: "u-two-slash"

U< (u1\u2 -- flag)
Flag is TRUE if unsigned integer u1 is less than unsigned integer u2 and FALSE otherwise.
Pronunciation: "u-less-than"

U> (u1\u2 -- flag)
Flag is TRUE if unsigned integer u1 is greater than unsigned integer u2 and FALSE otherwise.
Pronunciation: "u-greater-than"

U>D (u -- ud)
Converts unsigned integer u to its double number equivalent ud by placing a 0 on the data stack above u.
Pronunciation: "u-to-d"

U?KEY (-- xaddr)
A user variable that contains the extended code field address of the ?KEY routine. See ?KEY.
Pronunciation: "u-question-key" Attributes: U

UABORT (-- xaddr)
A user variable that contains the extended code field address of the user-supplied abort routine that is executed if the CUSTOM.ABORT flag is TRUE. If CUSTOM.ABORT is FALSE, ABORT executes the default (ABORT) routine. UABORT is initialized by COLD to contain the xcfa of (ABORT). See (ABORT) and CUSTOM.ABORT.

Pronunciation: "u-abort" Attributes: U

UD*S (ud1\u -- ud2 | ud2 = ud1 * u)
Multiplies unsigned double number ud1 by unsigned single precision number u giving the unsigned double number result ud2. An unchecked error occurs on overflow.
Pronunciation: "u-d-star-s"

UD.R (ud\+byte -- | +byte = width)
Prints the unsigned double number ud right-justified in a field of +byte characters. If +byte is less than or equal to the number of characters to be printed, the number is printed with no extra spaces. See D.R
Pronunciation: "u-d-dot-r" Attributes: M, S

UEMIT (-- xaddr)
A user variable that contains the extended code field address of the EMIT routine. See EMIT.
Pronunciation: "u-emit" Attributes: U

UERROR (-- xaddr)
A user variable that contains the extended code field address of the error routine that is executed if the CUSTOM.ERROR flag is TRUE. If CUSTOM.ERROR is FALSE, all system errors call the default (ERROR) routine which prints descriptive error messages. UERROR is initialized by COLD to contain the xcfa of a simple default error handler that prints the hexadecimal system error number and executes ABORT. See ((ERROR)), (ERROR), CUSTOM.ERROR, and ABORT, and consult the error message appendix in the Software Manual.
Pronunciation: "u-error" Attributes: U

UFIRST (-- xaddr)
A user variable that holds the extended address of the first byte of the first block buffer. Executing UFIRST X@ places the xaddress of the first byte of the block buffers on the stack. Initialized by BLOCK.BUFFERS.
Pronunciation: "u-first" Attributes: U

UFIXX (r -- u)
Rounds the positive floating point number r to the nearest unsigned integer u. Equivalent to DFIXX D>S. Overflow errors are not checked. See DFIXX.
Attributes: S

UFLOT (u -- r)
Converts the 16 bit unsigned integer u to its floating point representation r. See DFLOT and FLOT.
Pronunciation: "u-float" Attributes: S

UKEY (-- xaddr)
A user variable that contains the extended code field address of the KEY routine. See KEY.
Pronunciation: "u-key" Attributes: U

ULIMIT (-- xaddr)

A user variable that contains the extended address of the last+1 byte in the last block buffer. Initialized by BLOCK.BUFFERS.
Pronunciation: "u-limit" Attributes: U

UM* (u1\u2 -- ud | ud = u1 * u2)
Multiplies unsigned integers u1 and u2 giving the unsigned double precision product ud.
Pronunciation: "u-m-star"

UM/MOD (ud1\u1 -- u2\ud2 | u2 = remainder, ud2 = quotient)
Divides unsigned double number ud1 by unsigned integer u1 to give an unsigned single-precision remainder u2 and an unsigned double number quotient ud2. Division by 0 (u1=0) yields u2 = -1 and ud2 = -1.
Pronunciation: "u-m-slash-mod"

UMAX (u1\u2 -- [u1] or [u2])
Retains the greater of two unsigned integers and drops the other. See MAX.
Pronunciation: "u-max"

UMIN (u1\u2 -- [u1] or [u2])
Retains the lesser of two unsigned integers and drops the other. See MIN.
Pronunciation: "u-min"

UMOD (u1\u2 -- u3 | u3 = remainder of u1/u2)
Divides unsigned integer u1 by unsigned integer u2, giving the unsigned remainder u3. Division by zero results in an indeterminant remainder. This is the fastest modulus function. See MOD.
Pronunciation: "u-mod"

UNDERFLOW (--)
Sets the user variable FP.ERROR to 1 to indicate an underflow error.

UNIQUE.MSG (-- xaddr)
A user variable that contains a flag. If the flag is true (the default condition), BEEP is executed and a warning message is printed each time a word is defined that already exists in the CURRENT or CONTEXT vocabularies. If the flag is false, no warning is issued when non-unique words are added to the dictionary.
Pronunciation: "unique-message"

UNLOOP (--)
Return Stack: (R: w1\w2 -- | discards the loop limit and index)
Removes the top two cells from the return stack. If you need to immediately EXIT a word definition from inside a DO...LOOP, call UNLOOP to discard the loop index and limit before executing EXIT to exit the definition. To exit a definition from within nested do loops, execute one UNLOOP for each level of nesting. Make sure that there are no additional items on the return stack (e.g., resulting from a >R command). UNLOOP is a synonym for XR>DROP which drops two cells from the return stack.
Attributes: C

UNTIL (flag --)

Used inside a colon definition to mark the end of a BEGIN ... UNTIL loop structure which terminates based on the value of flag. If flag is true, the loop terminates and execution continues with the word following UNTIL. If flag is false, looping continues and execution passes to the word following BEGIN. Use as:

```
BEGIN ... words to be executed ...
flag UNTIL
```

An error is issued if BEGIN and UNTIL are not properly paired within a definition.
Attributes: C, I

UP (-- xhandle | xhandle contains 16-bit user pointer)
Places on the stack the 32-bit extended address that contains the 16-bit base address of the current task's user area. Executing UP @ is equivalent to executing (STATUS) ; both return the 16-bit base address (in common memory) of the current task.
Pronunciation: "u-p"

UPAD (-- xaddr)
User variable that holds the 32 bit xaddr of PAD. The contents of UPAD must point to modifiable RAM, and there must be at least 32 bytes of RAM below PAD for number/string conversion. PAD may be on any page, but may not cross a page boundary. To change the location of PAD, store the new xaddress into UPAD using X!. See PAD.
Pronunciation: "u-pad" Attributes: U

UPDATE (--)
Marks the current buffer as updated.
Implementation detail: Sets the top bit in the buffer's 32-bit status flag. The current buffer is pointed to by PREV.

UPDATE.DISPLAY (--)
Writes the contents of the DISPLAY.BUFFER to the LCD display. When finished, leaves the display cursor pointing at the first position in the first line. For character displays, the cursor is turned off during the write to the display and is restored to its prior state after the update is complete, thus avoiding "flickering" of the cursor. Intermittently disables interrupts for 28 cycles (7 microseconds) per byte to implement clock stretching. See also (UPDATE.DISPLAY) and UPDATE.DISPLAY.LINE.

UPDATE.DISPLAY.LINE (n -- | n = line#)
Writes the contents of the specified line number n in the DISPLAY.BUFFER to the LCD display. n is zero-based, and is clamped to a maximum of 1 less than LINES/DISPLAY. Writes CHARS/DISPLAY.LINE characters to the display. When finished, leaves the display cursor pointing at the first position in the line following n. For character displays, the cursor is blanked during the write to the display and is restored to its prior state after the update is complete, thus avoiding "flickering" of the cursor. The line# n1 follows the same rules explained in the description of BUFFER.POSITION: for a graphics-style display the line# n1 is interpreted differently depending on whether the display is being used in "text mode" or "graphics mode". In text mode, n1 corresponds to the character line#; in graphics mode, n1 corresponds to the pixel line#. Intermittently disables interrupts for 28 cycles (7 microseconds) per byte to implement clock stretching.

UPOCKET (-- xaddr)

User variable that holds the 16-bit addr of POCKET which is in common memory. To change the location of POCKET, store the new xaddress into UPOCKET using !. Note that FIND executes COLD if POCKET is not in the common RAM.
Pronunciation: "u-pocket" Attributes: U

UPPER.CASE (x$addr -- x$addr)
Converts all of the characters in the counted string at x$addr to upper case letters. The string may not cross a page boundary.

URANGE (u1\u2\u3 -- u1\flag)
Flag is TRUE if u1 is greater than or equal to u2 and less than or equal to u3. Otherwise flag is FALSE. Uses unsigned comparison. See RANGE.
Pronunciation: "u-range"

URANGE.OF (u1\u2\u3 -- [u1] or [])
Used inside a CASE ... ENDCASE structure to mark the beginning of a conditional statement. If u2 <= u1 <= u3 (using unsigned math) then u1, u2, and u3 are dropped and execution continues with the words between URANGE.OF and ENDOF and then skips to the word after ENDCASE. Otherwise, u2 and u3 are dropped and execution continues after the next ENDOF. Use as:

```
n1 CASE
u2 u3 URANGE.OF executed if n1 in range (u2,u3)        ENDOF
u4 u5 URANGE.OF executed if n1 in range (u4,u5)        ENDOF
words to be executed if not in range (u2,u3) or (u4,u5)
ENDCASE
```

An error is issued if URANGE.OF and ENDOF are not properly paired. See CASE, OF, RANGE.OF.
Pronunciation: "u-range-of" Attributes: C, I

UREAD/WRITE (-- xaddr)
A user variable that contains the extended code field address of the mass memory read/write word called by the command READ/WRITE. Its default contents equal the xcfa of the ram disk utility. See READ/WRITE.
Pronunciation: "u-read-slash-write" Attributes: U

USE (-- xaddr)
A user variable containing the extended address of the next block buffer to use (that is, the least most recently accessed buffer).
Attributes: U

USE.PAGE (page --)
Sets up a useful default memory map in the specified page of RAM, and places the heap area in the top 14.5K of page fifteen and the variable area in common RAM. The locations of the user area, data stack, return stack, TIB, POCKET and PAD are not changed (they are typically in the common RAM). The memory map is initialized as follows:

20 Kbyte definitions area starts at 0x0000 on the specified page
12 Kbyte name area starts at 0x5000 on the specified page
Variable area starts at 0x8E00 in common memory
14.5 Kbyte heap occupies 0x4600\F-0x7FFF\F

The dictionary and names areas are on the specified page and may be write-protectable (pages 4, 5, 6, and 7 are write-protectable). The heap is on page 0x0F (non-write-protectable RAM) and the variable area is in common RAM, so the variable and heap areas will never be mistakenly ROMmed or write protected. This routine is provided as a convenience to users who need to set up a reasonable memory map immediately after start up. It is strongly recommended that the programmer execute

```
    ANEW <name>
```

immediately after using this word. Doing so will properly reset the necessary pointers when forgetting compiled words during debugging. Implementation detail example: Executing 4 USE.PAGE is equivalent to:

```
    HEX    0000  04  DP  X!          5000  04  NP X!
           8E00  00  VP  X!          4600  0F  7FFF  0F  IS.HEAP
```

USE.SERIAL1 (--)

Installs the primary serial port (serial1) as the serial link used by the QED-Forth interpreter and called by EMIT, ?KEY, and KEY. The serial1 port is associated with the 68HC11's on-chip hardware UART. Stores the xcfa of KEY1 in UKEY, the xcfa of ?KEY1 in U?KEY, and the xcfa of EMIT1 in UEMIT. Thus the vectored routines KEY, ?KEY, and EMIT will automatically execute the serial1 routines KEY1, ?KEY1, and EMIT1 respectively. Initializes the resource variable SERIAL1.RESOURCE to 0\0, and initializes the resource variable associated with the prior serial channel in use (typically either SERIAL1.RESOURCE or SERIAL2.RESOURCE) to 0\0. Does not disable the serial2 port.
Pronunciation: "use-serial-one"

USE.SERIAL2 (--)

Installs the secondary serial port (serial2) as the serial link used by the QED-Forth interpreter and called by EMIT, ?KEY, and KEY, calls INIT.SERIAL2 to initialize the serial2 port, and globally enables interrupts to allow the serial2 port to operate. The serial2 port is supported by QED-Forth's software UART using hardware pins PA3 (input) and PA4 (output). USE.SERIAL2 stores the xcfa of KEY2 in UKEY, the xcfa of ?KEY2 in U?KEY, and the xcfa of EMIT2 in UEMIT. Thus the vectored routines KEY, ?KEY, and EMIT will automatically execute the serial2 routines KEY2, ?KEY2, and EMIT2 respectively. Initializes the resource variable SERIAL2.RESOURCE to 0\0, and initializes the resource variable associated with the prior serial channel in use (typically either SERIAL1.RESOURCE or SERIAL2.RESOURCE) to 0\0. Does not disable the serial1 port. See BAUD2.
Pronunciation: "use-serial-two"

USER (+byte <name> -- | +byte = offset from user area base address)

Removes the next <name> from the input stream and creates a user variable called <name> which when executed places on the stack an extended address equal to the user base address (i.e., the value in UP) plus the specified offset +byte. Use as:

```
    +byte USER <name>
```

The user area holds system variables. The kernel word #USER.BYTES returns the number of bytes in the user area that are already used by the QED-Forth system. Thus when defining a new user variable, +byte should be greater than or equal to #USER.BYTES and less than 255 (the maximum size of the user area).
Attributes: D

UTIB (-- xaddr)
User variable that holds the 32 bit xaddr of the TIB (terminal input buffer). To change the location of TIB, store the new xaddress into UTIB using X!. The terminal input buffer may be on any page, but may not cross a page boundary.
Pronunciation: "u-t-i-b" Attributes: U

V* (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el --)
Multiplies each element of the source1 vector specified by xvaddr1\sep1\d.#el with the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el and places the result in the corresponding element of the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be one of the source vectors.
Pronunciation: "v-star" Attributes: S

V*+ (r1\xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el -- r1)
Multiplies the scalar r1 by each element in the source2 vector specified by xvaddr2\sep2\d.#el and adds the result to the corresponding element in the source1 vector specified by xvaddr1\sep1\d.#el, and places the final result in the corresponding element of the destination vector specified by xvaddr3\sep3\d.#el. Thus for each element,

dest <- src1 + r1*src2

The destination vector may be one of the source vectors. The scalar r1 is left on the stack.
Pronunciation: "v-star-plus" Attributes: S

V+ (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el --)
Adds each element of the source1 vector specified by xvaddr1\sep1\d.#el with the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el and places the result in the corresponding element of the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be one of the source vectors.
Pronunciation: "v-plus" Attributes: S

V, (w --)
Stores w at the next available location in the variable area and increments the variable pointer VP by 2. An error occurs if w is not correctly stored; e.g. if VP does not point to RAM. An error occurs if the V, operation causes VP to be incremented across the boundary between 0x7FFF (the last valid address in a given page) and 0x8000 (the start of the register area).
Pronunciation: "v-comma"

V- (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el --)
Subtracts each element of the source2 vector specified by xvaddr2\sep2\d.#el from the corresponding element in the source1 vector specified by xvaddr1\sep1\d.#el and places the result in the corresponding element of the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be one of the source vectors.
Pronunciation: "v-minus" Attributes: S

V.ALL= (xvaddr1\sep1\xvaddr2\sep2\d.#el -- flag)
flag is true if each element in the source1 vector specified by xvaddr1\sep1\d.#el is equal to the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el.

Pronunciation: "v-all-equal"

V.ANY= (xvaddr1\sep1\xvaddr2\sep2\d.#el -- flag)
flag is true if any element in the source1 vector specified by xvaddr1\sep1\d.#el is equal to the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el.
Pronunciation: "v-any-equal"

V.COPY (xvaddr1\sep1\xvaddr2\sep2\d.#el --)
Copies the contents of the source vector specified by xvaddr1\sep1\d.#el to the destination vector specified by xvaddr2\sep2\d.#el.
Pronunciation: "v-copy"

V.FILL (r\xvaddr\sep\d.#el --)
Stores r into each element of the vector specified by xvaddr\sep\d.#el.
Pronunciation: "v-fill"

V.INSTANCE: (u <name> -- | u is the size of the structure)
Removes <name> from the input stream, creates a structure instance called <name>, and allocates u bytes in the variable area starting at VHERE for the structure instance (the "V" in "V.INSTANCE:" refers to the Variable area where the instance is allocated). Compare with D.INSTANCE:. When <name> is executed, the base address of the allocated structure instance is placed on the data stack. Typical use:

    <structure.name> V.INSTANCE: <name>

where <structure.name> was defined using

    STRUCTURE.BEGIN: <structure.name> ... STRUCTURE.END

Executing <structure.name> leaves the structure size n on the stack, and V.INSTANCE: <name> allocates and names the instance. Executing

    SIZE.OF <name>

places the allocated size of the instance on the stack. Note that the instance may cross page boundaries, and may increment the variable pointer VP so that it points to a new page.
Pronunciation: "v-instance" Attributes: D

V.MAX (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el --)
Performs the function FMAX on each pair of corresponding elements in the two source vectors specified by xvaddr1\sep1\d.#el and xvaddr2\sep2\d.#el and places the result in the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be one of the sources.
Pronunciation: "v-max"

V.MIN (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el --)
Performs the function FMIN on each pair of corresponding elements in the two source vectors specified by xvaddr1\sep1\d.#el and xvaddr2\sep2\d.#el and places the result in the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be one of the sources.

V.SUM (xvaddr\sep\d.#el -- r | r = sum of elements in vector)
Returns the sum of elements r in the vector specified by xvaddr\sep\d.#el.
Pronunciation: "v-sum" Attributes: S

V.SWAP (xvaddr1\sep1\xvaddr2\sep2\d.#el --)
Exchanges the contents of the vector specified by xvaddr1\sep1\d.#el with the contents of the vector specified by xvaddr2\sep2\d.#el.
Pronunciation: "v-swap"

V.TRANSFORM (xvaddr1\sep1\xvaddr2\sep2\d.#el\xcfa --)
xcfa specifies a unary operation that transforms one floating point number into another floating point number. V.TRANSFORM performs the unary operation indicated by xcfa on each of the elements in the source vector specified by xvaddr1\sep1\d.#el and stores the result in the destination vector specified by xvaddr2\sep2\d.#el. The source vector may equal the destination vector.
Pronunciation: "v-transform" Attributes: S

V/ (xvaddr1\sep1\xvaddr2\sep2\xvaddr3\sep3\d.#el --)
Divides each element of the source1 vector specified by xvaddr1\sep1\d.#el by the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el and places the result in the corresponding element of the destination vector specified by xvaddr3\sep3\d.#el. The destination vector may be one of the source vectors.
Pronunciation: "v-divide" Attributes: S

V< (xvaddr1\sep1\xvaddr2\sep2\d.#el -- flag)
flag is true if every element in the source1 vector specified by xvaddr1\sep1\d.#el is less than the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el.
Pronunciation: "v-less-than"

V> (xvaddr1\sep1\xvaddr2\sep2\d.#el -- flag)
flag is true if every element in the source1 vector specified by xvaddr1\sep1\d.#el is greater than the corresponding element in the source2 vector specified by xvaddr2\sep2\d.#el.
Pronunciation: "v-greater-than"

VALLOT (n --)
Reserves n bytes in the variable area by incrementing the variable pointer VP by n. An error occurs if the VALLOT operation causes VP to be incremented across the boundary between 0x7FFF (the last valid address in a given page) and 0x8000 (the start of the register area).
Pronunciation: "v-allot"

VARIABLE (<name> --)
Removes the next <name> from the input stream, defines a child word called <name>, and VALLOTs a cell in the variable area. When <name> is executed, it leaves the extended address xaddr of a the cell reserved in the variable area to hold the variable's contents. <name> is referred to as a "variable". Use as:

```
VARIABLE <name>
```

Attributes: D

VC, (byte --)
Stores byte at the next available location in the variable area and increments the variable pointer VP by 1. An error occurs if byte is not correctly stored; e.g. if VP does not point to RAM. An error occurs if the VC, operation causes VP to be incremented

across the boundary between 0x7FFF (the last valid address in a given page) and 0x8000 (the start of the register area).
Pronunciation: "v-c-comma"

VFORTH (-- xaddr)
A user variable that contains the xnfa (extended name field address) of the top word in the FORTH vocabulary, which is the default vocabulary to which the programmer's definitions are typically appended. In turnkeyed (autostarted) applications that require the interpreter to run in the final application, the autostart word should initialize VFORTH to contain the xnfa of the last word defined. For an example of use, see the definition of the application's autostart routine in the "Putting It All Together" chapter in the Software Manual. See LATEST, CONTEXT, CURRENT, FIND, and NFA.FOR.
Attributes: U

VHERE (-- xaddr)
Places on the stack the xaddr of the next available location in the variable area. Equivalent to VP X@.
Pronunciation: "v-here" Attributes: U

VOCABULARY (<name> --)
Creates and initializes to LATEST a 32 bit xhandle in the variable area. When <name> executes, this xhandle is stored into the user variable CONTEXT so that the <name> vocabulary branch is searched first during dictionary searches. See FIND.
Attributes: D

VP (-- xaddr)
User variable that contains the 32-bit Variable Pointer. The contents of VP are placed on the stack by VHERE and are modified by VALLOT. The command VP X@ is equivalent to VHERE; it yields the xaddr of the next available location in the variable area. The command VP @ is equivalent to VPAGE; it yields the page of the definitions area.
Pronunciation: "v-p" Attributes: U

WARM (--)
Restarts the QED-Forth system and clears the data and return stacks and executes ABORT. Unlike COLD, WARM does not initialize all of the user variables to their default values. See the "Program Development Techniques" chapter in the Software Manual for a detailed discussion of WARM and COLD restarts.

WHICH.MAP (-- [0] or [1])
Returns a 0 if the current memory map is the "standard map", and returns a 1 if the current map is the "download map" on flash-carrying boards. If the standard map is active, pages 4, 5, and 6 are addressed in the S1 flash, and pages 1, 2, and 3 are addressed in the S2 RAM. If the download map is active, pages 4, 5, and 6 are addressed in the S2 RAM, and pages 1, 2, and 3 are addressed in the S1 flash memory. This routine allows a user or program to verify which map is currently being used. After a "factory cleanup" operation, the standard map is active. See STANDARD.MAP and DOWNLOAD.MAP.

WHILE (flag --)

Used inside a colon definition to mark the beginning of the "while true" portion of a BEGIN ... WHILE ... REPEAT loop. If flag is TRUE, the loop continues and the words between WHILE and REPEAT are executed, after which control is transferred to the word following BEGIN. If flag is FALSE, the loop terminates and execution continues with the word following REPEAT. Use as:

```
    BEGIN
                    words to be iteratively executed
    flag WHILE      words to be iteratively executed
    REPEAT
```

An error is issued if BEGIN WHILE and REPEAT are not properly paired inside a colon definition.
Attributes: C, I

WIDTH (-- xaddr)
A user variable that contains the number of characters saved in a name entry by CREATE. Minimum value is 2, maximum is 31.
Attributes: U

WORD (char1 -- xaddr | char1 is delimiter, xaddr = POCKET)
Parses the next word delimited by the specified char1 from the input stream. Moves the parsed word as a counted string to the buffer at POCKET in common memory, and places the xaddr of POCKET on the data stack. If the word has more than 31 characters, only the first 31 characters are moved and the count is clamped to 31. WORD appends a space to the counted string in POCKET; the space is not included in the string's count. If the input stream is exhausted when WORD executes, the count left at POCKET equals 0. If the specified delimiter char1 is a space, then leading spaces are ignored. If BLK = 0, the input stream is the terminal input buffer TIB. Otherwise WORD executes BLOCK so that the input stream is available in a block buffer. The value of >IN specifies the offset from the start of the input stream to the first character to be parsed. WORD leaves >IN pointing 1 byte past the terminating delimiter unless the input stream is exhausted, in which case >IN is left pointing 1 byte past the last valid location in the input stream. The interpreter executes BL WORD to parse input commands, and then calls (#FIND) to search for the parsed command in the dictionary. To parse strings longer than 31 characters, use PARSE.
Attributes: M

WORDS (--)
Prints all words in the CURRENT vocabulary. WORDS incorporates PAUSE.ON.KEY, so the printout can be terminated by typing a carriage return or . (dot); it can be suspended and resumed by typing other characters, and it responds to XON/XOFF handshaking (see PAUSE.ON.KEY). Each word is printed left justified in a field of 16 or 32 characters, 3 names per line. Characters that are not saved in the headers are represented by the appropriate number of _ characters.
Attributes: M

X! (xaddr1\xaddr2 -- | xaddr1 is stored at xaddr2)
Stores the extended address xaddr1 at xaddr2. A synonym for 2!.
Pronunciation: "x-store"

X.OVER.N (xaddr\w -- xaddr\w\xaddr)

Copies the extended address located under the top data stack cell to the top of the data stack.
Pronunciation: "x-over-n"

X1-X2>D (xaddr1\xaddr2 -- d)
Subtracts xaddr2 from xaddr1 to yield the signed double number result d. There is an unchecked error if one of the xaddresses is in common memory (addr >= 0x8000) and the other is in paged memory (addr <= 0x7FFF). Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-one-minus-x-two-to-d"

X1-X2>N (xaddr1\xaddr2 -- n)
Subtracts xaddr2 from xaddr1 to yield the signed integer result n. There is an unchecked error if one of the xaddresses is in common memory (addr >= 0x8000) and the other is in paged memory (addr <= 0x7FFF), or if the difference can not be represented as a signed 16-bit integer. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-one-minus-x-two-to-n"

X2DROP (xaddr1\xaddr2 --)
Drops two extended addresses (4 cells) from the data stack.
Pronunciation: "x-two-drop"

X2DUP (xaddr1\xaddr2 -- xaddr1\xaddr2\xaddr1\xaddr2)
Duplicates the top two extended addresses on the data stack.
Pronunciation: "x-two-dupe"

X< > (xaddr1\xaddr2 -- flag)
Flag is TRUE if the two extended addresses are not equal and FALSE otherwise.
Pronunciation: "x-not-equal"

X= (xaddr1\xaddr2 -- flag)
Flag is TRUE if the two extended addresses are equal and FALSE otherwise.
Pronunciation: "x-equals"

X>R (xaddr --)
Return Stack: (R: -- xaddr)
Transfers the top extended address on the data stack to the return stack.
Pronunciation: "x-to-r" Attributes: C

X@ (xaddr1 -- xaddr2)
Fetches an extended address xaddr2 from memory location xaddr1. A synonym for 2@.
Pronunciation: "x-fetch"

XADDR-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a 32-bit extended address field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining

word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "x-address" Attributes: D

XADDR: (<name> --)
XADDR: is a synonym for DOUBLE:. It defines a 32-bit self-fetching variable called <name> which holds a 32-bit extended address. See the glossary entry for DOUBLE:. Use as:

```
    XADDR:  <name>
```

Pronunciation: "x-address-colon" Attributes: D

XADDRS-> (u1\u2 <name> -- u3)
Adds a named member to the structure being defined and reserves room for u2 extended addresses in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u3 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "x-addresses" Attributes: D

XALIGN (xaddr1 -- xaddr2)
If xaddr1 is not an even multiple of 4 bytes, increases xaddr1 by 1, 2, or 3 bytes to yield xaddr2 which is an even multiple of 4 bytes. The vector and matrix operations require data structures to be 4-byte aligned; the heap manager and DIM.CONSTANT.MATRIX: and DIM.CONSTANT.ARRAY: use XALIGN to ensure that heap items, arrays, and matrices are 4-byte aligned.
Pronunciation: "x-align"

XCONSTANT (xaddr <name> --)
Removes the next <name> from the input stream and defines a child word called <name> which when executed leaves the specified extended address xaddr on the data stack. xaddr is stored in the definitions area of the dictionary. <name> is referred to as an "xconstant". Use as:

```
    xaddr XCONSTANT <name>
```

Pronunciation: "x-constant" Attributes: D

XD+ (xaddr1\d -- xaddr2)
Adds signed double number d to xaddr1 to yield xaddr2. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-d-plus"

XD- (xaddr1\d -- xaddr2)
Subtracts signed double number d from xaddr1 to yield xaddr2. Note that in paged memory, the address immediately preceding 0000 is address 0x7FFF on the preceding page.
Pronunciation: "x-d-minus"

XDROP (xaddr --)
Drops an extended address (two cells) from the data stack.
Pronunciation: "x-drop"

XDUP (xaddr -- xaddr\xaddr)
Duplicates the top extended address (two cells) on the data stack.
Pronunciation: "x-dupe"

XDUP>R (xaddr -- xaddr)
Return Stack: (R: -- xaddr)
Copies the top extended address on the data stack to the return stack.
Pronunciation: "x-dup-to-r" Attributes: C

XHNDL-> (u1 <name> -- u2)
Adds a named member to the structure being defined and reserves room for a 32-bit extended handle field in the structure. Removes <name> from the input stream and creates a structure field called <name>. u1 is the structure offset initialized by STRUCTURE.BEGIN:. u2 is the updated offset to be used by the next member defining word or by STRUCTURE.END. When <name> is later executed, it adds its offset u1 to the extended address found on the data stack which is typically the start xaddress of an instance of the data structure; the result is the xaddress of the desired member in the structure.
Pronunciation: "x-handle" Attributes: D

XIRQ.ID (-- n)
Returns the interrupt identity code for the external non-maskable interrupt called XIRQ. Used as an argument for ATTACH. The XIRQ interrupt is activated by an active-low signal on the XIRQ input pin and is enabled by the X bit in the condition code register.
Pronunciation: "x-i-r-q-i-d"

XMIT.DISABLE (-- xaddr)
A user variable that contains a flag that is set each time that EXPECT receives an XOFF character (ascii 19) and is cleared upon startup and each time EXPECT receives the XON character (ascii 17). XON and XOFF are standard handshaking characters used to control the flow of data between computers. The XMIT.DISABLE flag is not used by QED-Forth; it is provided in case the programmer needs to implement an XON/XOFF handshaking protocol. Note that inserting PAUSE.ON.KEY in the inner loop of a data dump word provides a very easy way to emulate XON/XOFF control of data dumps. See EXPECT and PAUSE.ON.KEY.
Pronunciation: "transmit-disable" Attributes: U

XN+ (xaddr1\n -- xaddr2)
Adds signed integer n to xaddr1 to yield xaddr2. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-n-plus"

XN- (xaddr1\n -- xaddr2)
Subtracts signed integer n from xaddr1 to yield xaddr2. Note that in paged memory, the address immediately preceding 0000 is address 0x7FFF on the preceding page.

Pronunciation: "x-n-minus"

XOR (w1\w2 -- w3)
w3 is the result of a logical bit-by-bit exclusive-or of w1 and w2.
Pronunciation: "x-or"

XOVER (xaddr1\xaddr2 -- xaddr1\xaddr2\xaddr1)
Places a copy of xaddr1 on top of the data stack.
Pronunciation: "x-over"

XPICK (xaddr\wn-1\...w1\w0\+n -- xaddr\wn-1\...\w1\w0\xaddr)
Copies to the top of the stack the extended address whose most significant cell is the +nth item on the stack (0-based, not including +n). An unchecked error occurs if there are fewer than +n+2 cells on the data stack or if +n is outside the range 0 to 255, inclusive. 0 XPICK is equivalent to XDUP, 1 XPICK is equivalent to X.OVER.N, 2 XPICK is equivalent to XOVER.
Pronunciation: "x-pick"

XR> (-- xaddr)
Return Stack: (R: xaddr --)
Transfers the top extended address on the return stack to the data stack.
Pronunciation: "x-r-from" Attributes: C

XR>DROP (--)
Return Stack: (R: xaddr --)
Removes the top extended address from the return stack.
Pronunciation: "x-r-from-drop" Attributes: C

XR@ (-- xaddr)
Return Stack: (R: xaddr -- xaddr)
Copies the top extended address on the return stack to the data stack.
Pronunciation: "x-r-fetch" Attributes: C

XRANGE (xaddr1\xaddr2\xaddr3 -- xaddr1\flag)
Flag is TRUE if xaddr1 is greater than or equal to xaddr2 and less than or equal to xaddr3. Otherwise flag is FALSE. There is an unchecked error if one or two of the xaddresses are in common memory (addr >= 0x8000) and other(s) are in paged memory (addr <= 0x7FFF). In other words, XRANGE works properly only if all three xaddresses are in paged memory, or if all three are in common memory. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-range"

XROT (xaddr1\xaddr2\xaddr3 -- xaddr2\xaddr3\xaddr1)
Rotates the top three extended addresses on the data stack.
Pronunciation: "x-rote"

XSWAP (xaddr1\xaddr2 -- xaddr2\xaddr1)
Exchanges the top two extended addresses on the data stack.
Pronunciation: "x-swap"

XU+ (xaddr1\u -- xaddr2)
Adds unsigned integer u to xaddr1 to yield xaddr2. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-u-plus"

XU- (xaddr1\u -- xaddr2)
Subtracts unsigned integer u from xaddr1 to yield xaddr2. Note that in paged memory, the address immediately preceding 0000 is address 0x7FFF on the preceding page.
Pronunciation: "x-u-minus"

XU< (xaddr1\xaddr2 -- flag)
Flag is TRUE if xaddr1 is less than xaddr2. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-u-less-than"

XU> (xaddr1\xaddr2 -- flag)
Flag is TRUE if xaddr1 is greater than xaddr2. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "x-u-greater-than"

XVARIABLE (<name> --)
Removes the next <name> from the input stream, defines a child word called <name>, and VALLOTs 2 cells in the variable area. When <name> is executed, it leaves the extended address xaddr of the two cells reserved in the variable area to hold <name>'s contents. <name> is referred to as an "xvariable". Use as:

```
XVARIABLE <name>
```

Pronunciation: "x-variable" Attributes: D

ZERO (-- r | r = 0.0)
Puts the floating point representation for zero (= 0\0) on the data stack.

ZERO.ARRAY (array.xpfa --)
Stores 0 into each byte of the specified array.

ZERO.MATRIX (matrix.xpfa --)
Stores 0 into each byte of the specified matrix.

[(--)
Sets STATE equal to 0 and enters the execution mode so that subsequent text from the input stream is executed. See] and STATE.
Pronunciation: "left-bracket" Attributes: I

[0] (array.xpfa -- [xaddr] or [0\0] | xaddr is array base address)
Returns the base address (that is, the address of the first element) of the array or matrix indicated by array.xpfa. Returns 0\0 if the array or matrix is undimensioned. No error checking is performed.
Pronunciation: "bracket-zero"

[COMPILE] (<word> --)

Removes the next <name> from the input stream and compiles a call to <name> into the current definition. Use as:

```
: <namex>
        ... [COMPILE] <name> ...
;
```

where <namex> is typically not immediate and <name> is typically immediate. [COMPILE] forces the immediate word <name> to be compiled into the definition of <namex> instead if executing while <namex> is being defined. Consult the Advanced Topics chapter of the Software Manual for further description. See COMPILE.
Pronunciation: "bracket-compile" Attributes: C, I

[] (n1...nn\array.xpfa -- xaddr | n1...nn = indices)
Returns the extended address xaddr of the element whose indices are n1...nn in the array specified by array.xpfa. If DEBUG is on, ABORTs if the indices are invalid. Typical use:

```
<indices> ' <array.name> []
```

Pronunciation: "brackets"

\ (--)
Ignores all remaining input on the current line. Very useful for inserting descriptive comments into QED-Forth source code. See (.
Pronunciation: "back-slash" Attributes: I

] (--)
Sets STATE equal to -1 and enters the compilation mode so that subsequent text from the input stream is compiled instead of being executed. See [and STATE.
Pronunciation: "right-bracket" Attributes: I

|2!| (w1\w2\xaddr -- | [xaddr] gets w2, [xaddr+2] gets w1)
Stores two 16 bit integers at xaddr. Disables interrupts during the store to ensure that an interrupting routine or task will read valid data. w2 is stored at xaddr and w1 is stored at xaddr+2. Can also be used to store a double number at xaddr. Disables interrupts for 28 cycles (7 microseconds) unless the specified 4 bytes straddle a page boundary, in which case interrupts are disabled for approximately 260 cycles. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "bar-two-store"

|2@| (xaddr -- w1\w2)
Fetches two 16 bit integers from xaddr. Disables interrupts during the fetch to ensure that an interrupting routine or task does not modify the contents while the fetch is in process. w2 is taken from xaddr and w1 is from xaddr+2. Can also be used to fetch a double number from xaddr. Disables interrupts for 28 cycles (7 microseconds) unless the specified 4 bytes straddle a page boundary, in which case interrupts are disabled for approximately 260 cycles. Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "bar-two-fetch"

|F!| (r\xaddr --)

Stores a floating point number at xaddr. Disables interrupts during the store to ensure that an interrupting routine or task will read valid data. A synonym for |2!|.
Pronunciation: "bar-f-store"

|F@| (xaddr -- r)
Fetches a floating point number from xaddr. Disables interrupts during the fetch to ensure that an interrupting routine or task does not modify the contents while the fetch is in process. A synonym for |2@|.
Pronunciation: "bar-f-fetch"

|X!| (xaddr1\xaddr2 --)
Stores the extended address xaddr1 at xaddr2. Disables interrupts to ensure that an interrupting routine or task will read valid contents. A synonym for |2!|.
Pronunciation: "bar-x-store"

|X1-X2|>U (xaddr1\xaddr2 -- n)
Subtracts xaddr2 from xaddr1 and takes the absolute value of the result to yield the unsigned integer difference u. There is an unchecked error if one of the xaddresses is in common memory (addr >= 0x8000) and the other is in paged memory (addr <= 0x7FFF). Note that in paged memory, the address immediately following 0x7FFF is address 0000 on the following page.
Pronunciation: "abs-of-x-one-minus-x-two-to-u"

|X@| (xaddr1 -- xaddr2)
Fetches an extended address xaddr2 from memory location xaddr1. Disables interrupts during the fetch to ensure that an interrupting routine or task does not modify the contents while the fetch is in process. A synonym for |2@|.
Pronunciation: "bar-x-fetch"

Assembler Glossary

(Alphabetized in ASCII Order)

>ASSM (--)
Consult the main glossary.

>FORTH (--)
Consult the main glossary.

ABA (--)
Compiles the opcode sequence for the ABA instruction into the dictionary. When later executed, this code adds the contents of accumulator A and accumulator B, and stores the result in accumulator A.
Pronunciation: "add-b-to-a"

ABX (--)
Compiles the opcode sequence for the ABX instruction into the dictionary. When later executed, this code adds the unsigned 8 bit contents of accumulator B to the contents of register X and leaves the result in X.
Pronunciation: "add-b-to-x"

ABY (--)
Compiles the opcode sequence for the ABY instruction into the dictionary. When later executed, this code adds the unsigned 8 bit contents of accumulator B to the contents of register Y and leaves the result in Y.
Pronunciation: "add-b-to-y"

ADCA (arg\mode--)
Compiles the opcode sequence for the ADCA instruction into the dictionary. When later executed, this code adds the carry bit to the sum of the operand (specified by arg and mode) and the contents of accumulator A, and places the result in accumulator A.

Pronunciation: "add-with-carry-to-a"

ADCB (arg\mode--)
Compiles the opcode sequence for the ADCB instruction into the dictionary. When later executed, this code adds the carry bit to the sum of the operand (specified by arg and mode) and the contents of accumulator B, and places the result in accumulator B.

Pronunciation: "add-with-carry-to-b"

ADDA (arg\mode --)
Compiles the opcode sequence for the ADDA instruction into the dictionary. When later executed, this code adds the operand (specified by arg and mode) and the contents of accumulator A, and places the result in accumulator A.
Pronunciation: "add-a"

ADDB (arg\mode--)

Compiles the opcode sequence for the ADDB instruction into the dictionary. When later executed, this code adds the operand (specified by arg and mode) and the contents of accumulator B, and places the result in accumulator B.
Pronunciation: "add-b"

ADDD (arg\mode--)
Compiles the opcode sequence for the ADDD instruction into the dictionary. When later executed, this code adds the operand (specified by arg and mode) and the contents of accumulator D, and places the result in accumulator D.
Pronunciation: "add-d"

AGAIN, (--)
Used within a code definition to designate the end of an assembly coded infinite loop. Use as:

```
    BEGIN,          <code to be iterated>
    AGAIN,
```

The words between BEGIN, and AGAIN, are executed indefinitely. AGAIN, is equivalent to NEVER UNTIL,
Pronunciation: "again-comma"

ALWAYS (-- condition)
Within a code definition, leaves a condition flag on the data stack. Indicates an "always true" condition.

ANDA (arg\mode--)
Compiles the opcode sequence for the ANDA instruction into the dictionary. When later executed, this code performs the logical AND of the operand (specified by arg and mode) and the contents of accumulator A, and places the result in accumulator A.
Pronunciation: "and-a"

ANDB (arg\mode--)
Compiles the opcode sequence for the ANDB instruction into the dictionary. When later executed, this code performs the logical AND of the operand (specified by arg and mode) and the contents of accumulator B, and places the result in accumulator B.
Pronunciation: "and-b"

ANY.BITS.CLR (-- condition)
Within a code definition, leaves a condition flag on the data stack. Used in conjunction with a single-byte argument (which we'll designate as P) and a bit mask (designated as Q). The condition is true if any bits are clear in the result obtained by performing the logical AND operation P AND Q. This condition code is associated with the test and branch instructions BRSET and BRCLR.
Pronunciation: "any-bits-clear"

ANY.BITS.SET (-- condition)
Within a code definition, leaves a condition flag on the data stack. Used in conjunction with a single-byte argument (which we'll designate as P) and a bit mask (designated as Q). The condition is true if any bits are set in the result obtained by performing the logical AND operation P AND Q. This condition code is associated with the test and branch instructions BRSET and BRCLR.

ASL (arg\mode--)
Compiles the opcode sequence for the ASL instruction into the dictionary. When later executed, this code causes an arithmetic shift left of the 8-bit operand which is specified by arg and mode. The C (carry) bit in the CCR is loaded from the most significant bit of arg, and 0 is shifted into the least significant bit.
Pronunciation: "arithmetic-shift-left"

ASLA (--)
Compiles the opcode sequence for the ASLA instruction into the dictionary. When later executed, this code causes an arithmetic shift left of the contents of accumulator A. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator A, and 0 is shifted into the least significant bit.
Pronunciation: "arithmetic-shift-left-a"

ASLB (--)
Compiles the opcode sequence for the ASLB instruction into the dictionary. When later executed, this code causes an arithmetic shift left of the contents of accumulator B. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator B, and 0 is shifted into the least significant bit.
Pronunciation: "arithmetic-shift-left-b"

ASLD (--)
Compiles the opcode sequence for the ASLD instruction into the dictionary. When later executed, this code causes an arithmetic shift left of the contents of accumulator D. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator D, and 0 is shifted into the least significant bit.
Pronunciation: "arithmetic-shift-left-d"

ASR (arg\mode--)
Compiles the opcode sequence for the ASR instruction into the dictionary. When later executed, this code causes an arithmetic shift right of the operand specified by arg and mode. The C (carry) bit in the CCR is loaded from the least significant bit of arg. The most significant bit is held constant.
Pronunciation: "arithmetic-shift-right"

ASRA (--)
Compiles the opcode sequence for the ASRA instruction into the dictionary. When later executed, this code causes an arithmetic shift right of the contents of accumulator A. The C (carry) bit in the CCR is loaded from the least significant bit of accumulator A. The most significant bit is held constant.
Pronunciation: "arithmetic-shift-right-a"

ASRB (--)
Compiles the opcode sequence for the ASRB instruction into the dictionary. When later executed, this code causes an arithmetic shift right of the contents of accumulator B. The C (carry) bit in the CCR is loaded from the least significant bit of accumulator B. The most significant bit is held constant.
Pronunciation: "arithmetic-shift-right-b"

ASSEMBLER (--)
Consult the main glossary.

BCC (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BCC instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if C in CCR is clear.
Pronunciation: "branch-if-carry-clear"

BCLR (byte1\arg\mode-- | byte1 = mask)
Compiles the opcode sequence for the BCLR instruction into the dictionary. When later executed, this code clears bits in the operand (specified by arg and mode) which are set in the mask byte1.
Pronunciation: "bit-clear"

BCS (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BCS instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if C in CCR is set.
Pronunciation: "branch-if-carry-set"

BEGIN, (--)
Used within a code definition to designate the beginning of a looping structure. Use as:

```
     BEGIN, . . . . condition.flag UNTIL,
or
     BEGIN, . . . . condition.flag WHILE, . . . . REPEAT,
or
     BEGIN, . . . . AGAIN,
```

The words after UNTIL, or REPEAT, are executed after the loop structure terminates. BEGIN, ... AGAIN, is an infinite loop.
Pronunciation: "begin-comma"

BEQ (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BEQ instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if Z in CCR is set.
Pronunciation: "branch-if-equal"

BGE (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BGE instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if either N and V in the CCR are both set, or N and V are both clear. Typically used after a subtraction or comparison of signed numbers, causes a branch if the result is greater than or equal to 0.
Pronunciation: "branch-if-greater-than-or-equal-to"

BGT (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BGT instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if Z is clear, and either N and V in the CCR are both set, or N and V are both clear. Typically

used after a subtraction or comparison of signed numbers, causes a branch if the result is greater than 0.
Pronunciation: "branch-if-greater-than"

BHI (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BHI instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if C and Z in CCR are both clear. Typically used after a subtraction or comparison of unsigned numbers, causes a branch if the result is greater than 0.
Pronunciation: "branch-if-higher"

BHS (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BHS instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if C in CCR is clear. Typically used after a subtraction or comparison of unsigned numbers, causes a branch if the result is greater than or equal to 0.
Pronunciation: "branch-if-higher-or-same"

BITA (arg\mode--)
Compiles the opcode sequence for the BITA instruction into the dictionary. When later executed, this code modifies the bits in the condition code register according to the result of performing a logical AND between the operand (specified by arg and mode) and accumulator A. Neither the contents of accumulator A nor the operand are affected.
Pronunciation: "bit-test-a"

BITB (arg\mode--)
Compiles the opcode sequence for the BITB instruction into the dictionary. When later executed, this code modifies the bits in the condition code register according to the result of performing a logical AND between the operand (specified by arg and mode) and accumulator B. Neither the contents of accumulator B nor the operand are affected.
Pronunciation: "bit-test-b"

BLE (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BLE instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if Z is set, or N is set and V is clear, or N is clear and V is set in the CCR. Typically used after a subtraction or comparison of signed numbers, causes a branch if the result is less than or equal to 0.
Pronunciation: "branch-if-less-than-or-equal-to"

BLO (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BCS instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if C in CCR is set. Typically used after a subtraction or comparison of unsigned numbers, causes a branch if the result is less than 0.
Pronunciation: "branch-if-lower or same"

BLS (arg\mode-- | mode must be REL)

Compiles the opcode sequence for the BLS instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if either C or Z in CCR is set. Typically used after a subtraction or comparison of unsigned numbers, causes a branch if the result is less than or equal to 0.
Pronunciation: "branch-if-lower-or-same"

BLT (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BLT instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if either N is set and V is clear in the CCR, or N is clear and V is set. Typically used after a subtraction or comparison of signed numbers, causes a branch if the result is less than 0.
Pronunciation: "branch-if-less-than"

BMI (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BMI instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if N in CCR is set.
Pronunciation: "branch-if-minus"

BNE (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BNE instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if Z in CCR is clear.
Pronunciation: "branch-if-not-equal"

BPL (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BPL instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if N in CCR is clear.
Pronunciation: "branch-if-plus"

BRA (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BRA instruction into the dictionary. When later executed, this code executes an unconditional branch to the address equal to PC + arg + 2.
Pronunciation: "branch-always"

BRCLR (byte1\mode1\byte2\arg\mode2-- | byte1=offset, mode1=REL, byte2 =mask)
Compiles the opcode sequence for the BRCLR instruction into the dictionary. When later executed, this code causes a branch to an address if the bits set in the mask byte2 are clear in the operand specified by arg and mode2. The address branched to is equal to

PC + 4 + byte1

when mode2 is DIR or IND,X. When mode2 is IND,Y the address branched to is

PC + 5 + byte1

Pronunciation: "branch-if-bits-clear"

BRN (arg\mode-- | mode must be REL)

Compiles the opcode sequence for the BRN instruction into the dictionary. When later executed, this code acts as a two byte NOP instruction.
Pronunciation: "branch-never"

BRSET (byte1\mode1\byte2\arg\mode2-- | byte1=offset, mode1=REL, byte2=mask)
Compiles the opcode sequence for the BRSET instruction into the dictionary. When later executed, this code causes a branch to an address if the bits set in the mask byte2 are also set in the operand specified by arg and mode2. The address branched to is equal to
PC + 4 + byte1
when mode2 is DIR or IND,X. When mode2 is IND,Y the address branched to is
PC + 5 + byte1
Pronunciation: "branch-if-bits-set"

BSET (byte1\arg\mode-- | byte1 = mask)
Compiles the opcode sequence for the BSET instruction into the dictionary. When later executed, this code sets bits in the operand (specified by arg and mode) which are set in the mask byte1.
Pronunciation: "bit-set"

BSR (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BSR instruction into the dictionary. When later executed, this code causes the PC to be incremented by 2 and pushed onto the return stack, and then branches to the address equal to PC + arg.
Pronunciation: "branch-to-subroutine"

BVC (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BVC instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if V in CCR is clear.
Pronunciation: "branch-if-overflow-clear"

BVS (arg\mode-- | mode must be REL)
Compiles the opcode sequence for the BVS instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if V in CCR is set.
Pronunciation: "branch-if-overflow-set"

CALL (<name> --)
Consult the main glossary.

CBA (--)
Compiles the opcode sequence for the CBA instruction into the dictionary. When later executed, this code modifies the bits in the condition code register according to the comparison A - B. The contents of accumulators A and B are not affected.
Pronunciation: "compare-b-to-a"

CC (-- condition)
Used within a code definition, leaves a condition flag on the data stack. The condition is true when the carry flag in the condition code register is 0. Note: if you wish to use the

hex number CC within an assembly definition, type it as 0CC to avoid confusion with this condition code.
Pronunciation: "carry-clear"

CLC (--)
Compiles the opcode sequence for the CLC instruction into the dictionary. When later executed, this code clears the C (carry) bit in the condition code register.
Pronunciation: "clear-carry"

CLI (--)
Compiles the opcode sequence for the CLI instruction into the dictionary. When later executed, this code clears the I (global interrupt enable) bit in the condition code register. Maskable interrupts are enabled when the I bit is clear.
Pronunciation: "clear-interrupt-mask"

CLR (arg\mode--)
Compiles the opcode sequence for the CLR instruction into the dictionary. When later executed, this code clears the contents of the operand specified by arg and mode.
Pronunciation: "clear"

CLRA (--)
Compiles the opcode sequence for the CLRA instruction into the dictionary. When later executed, this code clears the contents of accumulator A.
Pronunciation: "clear-a"

CLRB (--)
Compiles the opcode sequence for the CLRB instruction into the dictionary. When later executed, this code clears the contents of accumulator B.
Pronunciation: "clear-b"

CLV (--)
Compiles the opcode sequence for the CLV instruction into the dictionary. When later executed, this code clears the V bit (the 2s complement overflow bit) in the CCR.
Pronunciation: "clear-overflow-bit"

CMPA (arg\mode--)
Compiles the opcode sequence for the CMPA instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator A and sets the condition code register bits accordingly. The operand and accumulator A are unaffected.
Pronunciation: "compare-a"

CMPB (arg\mode--)
Compiles the opcode sequence for the CMPB instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator B and sets the condition code register bits accordingly. The operand and accumulator B are unaffected.
Pronunciation: "compare-b"

CODE (<name> --)

Consult the main glossary.

COM (arg\mode --)
Compiles the opcode sequence for the COM instruction into the dictionary. When later executed, this code replaces the operand specified by arg and mode with its ones complement.
Pronunciation: "complement"

COMA (--)
Compiles the opcode sequence for the COMA instruction into the dictionary. When later executed, this code replaces the contents of accumulator A with its ones complement (which is formed by complementing the state of each bit).
Pronunciation: "complement-a"

COMB (--)
Compiles the opcode sequence for the COMB instruction into the dictionary. When later executed, this code replaces the contents of accumulator B with its ones complement (which is formed by complementing the state of each bit).
Pronunciation: "complement-b"

CPD (arg\mode--)
Compiles the opcode sequence for the CPD instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator D and sets the condition code register bits accordingly. The operand and accumulator D are unaffected.
Pronunciation: "compare-d"

CPX (arg\mode--)
Compiles the opcode sequence for the CPX instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator X and sets the condition code register bits accordingly. The operand and accumulator X are unaffected.
Pronunciation: "compare-x"

CPY (arg\mode--)
Compiles the opcode sequence for the CPY instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator Y and sets the condition code register bits accordingly. The operand and accumulator Y are unaffected.
Pronunciation: "compare-y"

CS (-- condition)
Used within a code definition, leaves a condition flag on the data stack. The condition is true when the carry flag in the condition code register is 1.
Pronunciation: "carry-set"

DAA (--)
Compiles the opcode sequence for the DAA instruction into the dictionary. When later executed, this code decimal adjusts the contents of accumulator A to facilitate binary-

coded-decimal calculations. For a detailed description of operation, consult the DAA entry in the Motorola HC11 manual.
Pronunciation: "decimal-adjust-a"

DEC	(arg\mode--) Compiles the opcode sequence for the DEC instruction into the dictionary. When later executed, this code subtracts one from the operand specified by arg and mode. Pronunciation: "decrement"
DECA	(--) Compiles the opcode sequence for the DECA instruction into the dictionary. When later executed, this code subtracts one from the contents of accumulator A. Pronunciation: "decrement-a"
DECB	(--) Compiles the opcode sequence for the DECB instruction into the dictionary. When later executed, this code subtracts one from the contents of accumulator B. Pronunciation: "decrement-b"
DES	(--) Compiles the opcode sequence for the DES instruction into the dictionary. When later executed, this code subtracts one from the contents of the S register which is the return stack pointer. Pronunciation: "decrement-s"
DEX	(--) Compiles the opcode sequence for the DEX instruction into the dictionary. When later executed, this code subtracts one from the contents of the index register X. Pronunciation: "decrement-x"
DEY	(--) Compiles the opcode sequence for the DEY instruction into the dictionary. When later executed, this code subtracts one from the contents of the index register Y. Pronunciation: "decrement-y"
DIR	(-- mode) Used within a code definition, leaves a constant on the stack indicating that the direct addressing mode should be used by an instruction opcode. Pronunciation: "direct"
ELSE,	(--) ELSE, is used in assembly coded routines to mark the beginning of the "else portion" of a conditional structure. Use as: condition IF, ... ELSE, ... ENDIF, (or THEN,) When executed, ELSE, causes a branch instruction to be compiled into the dictionary and resolves IF,'s branch. When the compiled code is later executed, the code between ELSE, and ENDIF, is executed if the condition is not met. Pronunciation: "else-comma"

END-CODE (sys -- | balances CODE)
A synonym for END.CODE. Consult the main glossary.

END.CODE (sys -- | balances CODE)
Consult the main glossary.

ENDIF, (--)
ENDIF, is used in assembly coded routines to mark the end of a conditional IF, structure. ENDIF, and THEN, are synonyms. Use as:

```
    condition IF,      . . . .
    ELSE,  . . . .
    ENDIF,
```

When executed, ENDIF, resolves the branch instructions used in the conditional structure. When the code compiled by the control structure is executed, the code between IF, and ELSE, is executed if the condition is true, and then control passes to the code following ENDIF,. If the condition is false, the code between ELSE, and ENDIF, is executed, and execution continues with the code following ENDIF,. An alternate form is

```
    condition IF, . . . . ENDIF,
```

When the code compiled by this control structure is executed, the code between IF, and ENDIF, is executed if the condition is true, and is not executed if the condition is false.
Pronunciation: "end-if-comma"

EORA (arg\mode--)
Compiles the opcode sequence for the EORA instruction into the dictionary. When later executed, this code performs a logical exclusive or between the contents of accumulator A and the operand specified by arg and mode. The result is stored in accumulator A.
Pronunciation: "exclusive-or-a"

EORB (arg\mode--)
Compiles the opcode sequence for the EORB instruction into the dictionary. When later executed, this code performs a logical exclusive or between the contents of accumulator B and the operand specified by arg and mode. The result is stored in accumulator B.
Pronunciation: "exclusive-or-b"

EQ (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P = Q. Alternate interpretation: Condition is true if the Z bit in the condition code register is set.
Pronunciation: "equal"

EXT (-- mode)
Used within a code definition, leaves a constant on the stack indicating that the extended addressing mode should be used by an instruction opcode.
Pronunciation: "extended"

FDIV (--)
Compiles the opcode sequence for the FDIV instruction into the dictionary. When later executed, this code performs an unsigned fractional divide of the 16-bit numerator in

accumulator D by the 16-bit denominator in the index register X. The quotient is stored in index register X and the remainder is stored in accumulator D.
Pronunciation: "fractional-divide"

GE (-- condition)
Within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P >= Q using signed math. Alternate interpretation: Condition is true if the N and V bits in the condition code register are either both set or both clear.
Pronunciation: "greater-than-or-equal-to"

GT (-- condition)
Within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P > Q using signed math. Alternate interpretation: Condition is true if the Z bit is clear, and the N and V bits are either both set or both clear in the condition code register.
Pronunciation: "greater-than"

HI (-- condition)
Within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P > Q using unsigned math. Alternate interpretation: Condition is true if C and Z in the condition code register are zero.
Pronunciation: "higher"

HS (-- condition)
Within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P >= Q using unsigned math. Alternate interpretation: Condition is true if the C (carry) bit in the condition code register is clear.
Pronunciation: "higher-or-same"

IDIV (--)
Compiles the opcode sequence for the IDIV instruction into the dictionary. When later executed, this code performs an unsigned integer divide of the 16-bit numerator in accumulator D by the 16-bit denominator in the index register X. The quotient is stored in index register X and the remainder is stored in accumulator D.
Pronunciation: "integer-divide"

IF, (condition --)
IF, is used in assembly coded routines to mark the start of a conditional structure. Use as:

```
    condition IF,      . . . .
    ELSE,  . . . .
    ENDIF,   ( or THEN,)
```

or

```
    condition IF,      . . . .
    ENDIF,   ( or THEN,)
```

When the branch compiled by IF, is executed, the code following IF, is executed if the condition is met, otherwise control is transferred to the code following ELSE, or ENDIF,. (THEN, and ENDIF, are synonyms).
Pronunciation: "if-comma"

IMM (-- mode)
Used within a code definition, leaves a constant on the stack indicating that the immediate addressing mode should be used by an instruction opcode.
Pronunciation: "immediate"

INC (arg\mode--)
Compiles the opcode sequence for the INC instruction into the dictionary. When later executed, this code adds one to the operand specified by arg and mode.
Pronunciation: "increment"

INCA (--)
Compiles the opcode sequence for the INCA instruction into the dictionary. When later executed, this code adds one to the contents of accumulator A.
Pronunciation: "increment-a"

INCB (--)
Compiles the opcode sequence for the INCB instruction into the dictionary. When later executed, this code adds one to the contents of accumulator B.
Pronunciation: "increment-b"

IND,X (-- mode)
Used within a code definition, leaves a constant on the stack indicating that the index register X addressing mode should be used by an instruction opcode.
Pronunciation: "indexed-x"

IND,Y (-- mode)
Used within a code definition, leaves a constant on the stack indicating that the index register Y addressing mode should be used by an instruction opcode.
Pronunciation: "indexed-y"

INH (--)
Defined as a no-op. May be used within a code definition if desired to indicate the inherent addressing mode.
Pronunciation: "inherent"

INS (--)
Compiles the opcode sequence for the INS instruction into the dictionary. When later executed, this code adds one to the S register which is the return stack pointer.
Pronunciation: "increment-s"

INX (--)
Compiles the opcode sequence for the INX instruction into the dictionary. When later executed, this code adds one to index register X.
Pronunciation: "increment-x"

INY (--)

Compiles the opcode sequence for the INY instruction into the dictionary. When later executed, this code adds one to index register Y.
Pronunciation: "increment-y"

JMP (arg\mode--)
Compiles the opcode sequence for the JMP instruction into the dictionary. When later executed, this code transfers control to the instruction stored at the effective address specified by arg and mode.
Pronunciation: "jump"

JSR (arg\mode--)
Compiles the opcode sequence for the JSR instruction into the dictionary. When later executed, this code increments PC properly and pushes it onto the return stack. Program control is then transferred to the effective address specified by arg and mode.
Pronunciation: "jump-to-subroutine"

LDAA (arg\mode--)
Compiles the opcode sequence for the LDAA instruction into the dictionary. When later executed, this code loads the operand specified by arg and mode into accumulator A.
Pronunciation: "load-accumulator-a"

LDAB (arg\mode--)
Compiles the opcode sequence for the LDAB instruction into the dictionary. When later executed, this code loads the operand specified by arg and mode into accumulator B.
Pronunciation: "load-accumulator-b"

LDD (arg\mode--)
Compiles the opcode sequence for the LDD instruction into the dictionary. When later executed, this code loads the operand specified by arg and mode into accumulator D.
Pronunciation: "load-d"

LDS (arg\mode--)
Compiles the opcode sequence for the LDS instruction into the dictionary. When later executed, this code loads the operand specified by arg and mode into the S register which is the return stack pointer.
Pronunciation: "load-s"

LDX (arg\mode--)
Compiles the opcode sequence for the LDX instruction into the dictionary. When later executed, this code loads the contents of the operand specified by arg and mode into index register X.
Pronunciation: "load-x"

LDY (arg\mode--)
Compiles the opcode sequence for the LDY instruction into the dictionary. When later executed, this code loads the contents of the operand specified by arg and mode into index register Y.
Pronunciation: "load-y"

LE (-- condition)

Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P <= Q using signed math. Alternate interpretation: Condition is true if the Z bit is set, or the N bit is set and V bit is clear, or the N bit is clear and the V bit is set in the condition code register.
Pronunciation: "less-than-or-equal-to"

LO (-- condition)
Used inside a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P < Q using unsigned math. Alternate interpretation: Condition is true if the C (carry) bit in the condition code register is set.
Pronunciation: "lower"

LS (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P <= Q using unsigned math. Alternate interpretation: Condition is true if either C or Z in the condition code register is set.
Pronunciation: "lower-or-same"

LSL (arg\mode--)
Compiles the opcode sequence for the LSL instruction into the dictionary. When later executed, this code causes a logical shift left of the 8-bit operand which is specified by arg and mode. The C (carry) bit in the CCR is loaded from the most significant bit of arg, and 0 is shifted into the least significant bit.
Pronunciation: "logical-shift-left"

LSLA (--)
Compiles the opcode sequence for the LSLA instruction into the dictionary. When later executed, this code causes a logical shift left of the contents of accumulator A. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator A, and 0 is shifted into the least significant bit.
Pronunciation: "logical-shift-left-a"

LSLB (--)
Compiles the opcode sequence for the LSLB instruction into the dictionary. When later executed, this code causes a logical shift left of the contents of accumulator B. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator B, and 0 is shifted into the least significant bit.
Pronunciation: "logical-shift-left-b"

LSLD (--)
Compiles the opcode sequence for the LSLD instruction into the dictionary. When later executed, this code causes a logical shift left of the contents of accumulator D. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator D, and 0 is shifted into the least significant bit.
Pronunciation: "logical-shift-left-d"

LSR (arg\mode--)

Compiles the opcode sequence for the LSR instruction into the dictionary. When later executed, this code causes a logical shift right of the contents of the operand specified by arg and mode. The C (carry) bit in the CCR is loaded from the least significant bit of arg. A 0 is shifted into the most significant bit.
Pronunciation: "logical-shift-right"

LSRA (--)
Compiles the opcode sequence for the LSRA instruction into the dictionary. When later executed, this code causes a logical shift right of the contents of accumulator A. The C (carry) bit in the CCR is loaded from the least significant bit of accumulator A. A 0 is shifted into the most significant bit.
Pronunciation: "logical-shift-right-a"

LSRB (--)
Compiles the opcode sequence for the LSRB instruction into the dictionary. When later executed, this code causes a logical shift right of the contents of accumulator B. The C (carry) bit in the CCR is loaded from the least significant bit of accumulator B. A 0 is shifted into the most significant bit.
Pronunciation: "logical-shift-right-b"

LSRD (--)
Compiles the opcode sequence for the LSRD instruction into the dictionary. When later executed, this code causes a logical shift right of the contents of accumulator D. The C (carry) bit in the CCR is loaded from the least significant bit of accumulator D. A 0 is shifted into the most significant bit.
Pronunciation: "logical-shift-right-d"

LT (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P < Q using signed math. Alternate interpretation: Condition is true if the N bit is set and V bit is clear, or the N bit is clear and the V bit is set in the condition code register.
Pronunciation: "less-than"

MI (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q = R, indicates the condition R < 0 using signed math. Alternate interpretation: Condition is true if the N bit in the condition code register is set.
Pronunciation: "minus"

MUL (--)
Compiles the opcode sequence for the MUL instruction into the dictionary. When later executed, this code multiplies the contents of accumulator A by the contents of accumulator B and stores the result in accumulator D.
Pronunciation: "multiply"

NE (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P not equal to Q. Alternate interpretation: Condition is true if the Z bit in the condition code register is clear.

Pronunciation: "not-equal"

NEG (arg\mode--)
Compiles the opcode sequence for the NEG instruction into the dictionary. When later executed, this code replaces the contents of the operand specified by arg and mode with its twos complement which is formed by inverting the state of each bit and adding 1.
Pronunciation: "negate"

NEGA (--)
Compiles the opcode sequence for the NEGA instruction into the dictionary. When later executed, this code replaces the contents of accumulator A with its twos complement which is formed by inverting the state of each bit and adding 1.
Pronunciation: "negate-a"

NEGB (--)
Compiles the opcode sequence for the NEGB instruction into the dictionary. When later executed, this code replaces the contents of accumulator B with its twos complement which is formed by inverting the state of each bit and adding 1.
Pronunciation: "negate-b"

NEVER (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Indicates a "never true" condition.

NOP (--)
Compiles the opcode sequence for the NOP instruction into the dictionary. When later executed, this code does nothing.
Pronunciation: "no-op"

ORAA (arg\mode--)
Compiles the opcode sequence for the ORAA instruction into the dictionary. When later executed, this code performs a logical OR operation between the contents of accumulator A and the contents of the operand specified by arg and mode. The result is stored in accumulator A.
Pronunciation: "or-accumulator-a"

ORAB (arg\mode--)
Compiles the opcode sequence for the ORAB instruction into the dictionary. When later executed, this code performs a logical OR operation between the contents of accumulator B and the contents of the operand specified by arg and mode. The result is stored in accumulator B.
Pronunciation: "or-accumulator-b"

PL (-- condition)
Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q = R, indicates the condition R >= 0 using signed math. Alternate interpretation: Condition is true if the N bit in the condition code register is clear.
Pronunciation: "plus"

PSHA (--)
Compiles the opcode sequence for the PSHA instruction into the dictionary. When later executed, this code stores the contents of accumulator A at the next available location on the return stack pointed to by the S register and decrements the S register.
Pronunciation: "push-a"

PSHB (--)
Compiles the opcode sequence for the PSHB instruction into the dictionary. When later executed, this code stores the contents of accumulator B at the next available location on the return stack pointed to by the S register and decrements the S register.
Pronunciation: "push-b"

PSHX (--)
Compiles the opcode sequence for the PSHX instruction into the dictionary. When later executed, this code stores the contents of index register X at the next available locations on the return stack pointed to by the S register and decrements the S register by 2.
Pronunciation: "push-x"

PSHY (--)
Compiles the opcode sequence for the PSHY instruction into the dictionary. When later executed, this code stores the contents of index register Y at the next available locations on the return stack pointed to by the S register and decrements the S register by 2.
Pronunciation: "push-y"

PULA (--)
Compiles the opcode sequence for the PULA instruction into the dictionary. When later executed, this code pops the top value off the return stack (pointed to by the S register) into accumulator A and increments the S register.
Pronunciation: "pull-a"

PULB (--)
Compiles the opcode sequence for the PULB instruction into the dictionary. When later executed, this code pops the top value off the return stack (pointed to by the S register) into accumulator B and increments the S register.
Pronunciation: "pull-b"

PULX (--)
Compiles the opcode sequence for the PULX instruction into the dictionary. When later executed, this code pops the top 2 bytes off the return stack (pointed to by the S register) into index register X and increments the S register by 2.
Pronunciation: "pull-x"

PULY (--)
Compiles the opcode sequence for the PULY instruction into the dictionary. When later executed, this code pops the top 2 bytes off the return stack (pointed to by the S register) into index register Y and increments the S register by 2.
Pronunciation: "pull-y"

REL (-- mode)
Used within a code definition, leaves a constant on the stack indicating that the relative addressing mode should be used by an instruction opcode.
Pronunciation: "relative"

REPEAT, (--)
REPEAT, is used to designate the end of an assembly coded

```
     BEGIN,                          . . . .
     condition.flag WHILE,    . . . .
     REPEAT,
```

structure. See BEGIN, and WHILE, .
Pronunciation: "repeat-comma"

ROL (arg\mode--)
Compiles the opcode sequence for the ROL instruction into the dictionary. When later executed, this code shifts the bits in the operand specified by arg and mode left by one bit. The C (carry) bit is shifted into the least significant bit of arg, and the most significant bit in arg is shifted into C.
Pronunciation: "rotate-left"

ROLA (--)
Compiles the opcode sequence for the ROLA instruction into the dictionary. When later executed, this code shifts the bits in accumulator A left by one bit. The C (carry) bit is shifted into the least significant bit of accumulator A, and the most significant bit in accumulator A is shifted into C.
Pronunciation: "rotate-left-a"

ROLB (--)
Compiles the opcode sequence for the ROLB instruction into the dictionary. When later executed, this code shifts the bits in accumulator B left by one bit. The C (carry) bit is shifted into the least significant bit of accumulator B, and the most significant bit in accumulator B is shifted into C.
Pronunciation: "rotate-left-b"

ROR (arg\mode--)
Compiles the opcode sequence for the ROR instruction into the dictionary. When later executed, this code shifts the bits in the operand specified by arg and mode right by one bit. The C (carry) bit is shifted into the most significant bit of arg, and the least significant bit in arg is shifted into C.
Pronunciation: "rotate-right"

RORA (--)
Compiles the opcode sequence for the RORA instruction into the dictionary. When later executed, this code shifts the bits in accumulator A right by one bit. The C (carry) bit is shifted into the most significant bit of accumulator A, and the least significant bit in accumulator A is shifted into C.
Pronunciation: "rotate-right-a"

RORB (--)

Compiles the opcode sequence for the RORB instruction into the dictionary. When later executed, this code shifts the bits in accumulator B right by one bit. The C (carry) bit is shifted into the most significant bit of accumulator B, and the least significant bit in accumulator B is shifted into C.
Pronunciation: "rotate-right-b"

RTI (--)
Compiles the opcode sequence for the RTI instruction into the dictionary. When later executed, this code restores accumulators A and B, and registers X, Y and PC with values pulled from the stack.
Pronunciation: "return-from-interrupt"

RTS (--)
Compiles the opcode sequence for the RTS instruction into the dictionary. When later executed, this code restores the program counter with a value pulled from the stack, thereby resuming execution just after the point where the subroutine was called.
Pronunciation: "return-from-subroutine"

SBA (--)
Compiles the opcode sequence for the SBA instruction into the dictionary. When later executed, this code subtracts the contents of accumulator B from accumulator A and stores the result in accumulator A. The contents of accumulator B are not affected.
Pronunciation: "subtract-b-from-a"

SBCA (arg\mode--)
Compiles the opcode sequence for the SBCA instruction into the dictionary. When later executed, this code subtracts the contents of the operand specified by arg and mode and the contents of C from accumulator A and stores the result in accumulator A.
Pronunciation: "subtract-with-carry-a"

SBCB (arg\mode--)
Compiles the opcode sequence for the SBCB instruction into the dictionary. When later executed, this code subtracts the contents of the operand specified by arg and mode and the contents of C from accumulator B and stores the result in accumulator B.
Pronunciation: "subtract-with-carry-b"

SEC (--)
Compiles the opcode sequence for the SEC instruction into the dictionary. When later executed, this code sets the C (carry) bit in the condition code register.
Pronunciation: "set-carry"

SEI (--)
Compiles the opcode sequence for the SEI instruction into the dictionary. When later executed, this code sets the I (global interrupt enable) bit in the condition code register. When the I bit is set, all maskable interrupts are disabled.
Pronunciation: "set-interrupt-mask"

SEV (--)

Compiles the opcode sequence for the SEV instruction into the dictionary. When later executed, this code sets the V (2's complement overflow) bit in the condition code register.
Pronunciation: "set-overflow-bit"

STAA (arg\mode--)
Compiles the opcode sequence for the STAA instruction into the dictionary. When later executed, this code stores the contents of accumulator A into the effective address specified by arg and mode.
Pronunciation: "store-accumulator-a"

STAB (arg\mode--)
Compiles the opcode sequence for the STAB instruction into the dictionary. When later executed, this code stores the contents of accumulator B into the effective address specified by arg and mode.
Pronunciation: "store-accumulator-b"

STD (arg\mode--)
Compiles the opcode sequence for the STD instruction into the dictionary. When later executed, this code stores the contents of accumulator D into the effective address specified by arg and mode.
Pronunciation: "store-d"

STOP (--)
Compiles the opcode sequence for the STOP instruction into the dictionary. When later executed, this code halts all system clocks and places the system in a minimum power consumption mode if the S bit in the CCR register is clear. If the S bit is set, STOP is disabled and acts like a NOP. Recovery is accomplished by a reset, or an active low signal on XIRQ, or a non-masked IRQ interrupt.

STS (arg\mode--)
Compiles the opcode sequence for the STS instruction into the dictionary. When later executed, this code stores the contents of the S register (which is the return stack pointer) at the effective address specified by arg and mode.
Pronunciation: "store-s"

STX (arg\mode--)
Compiles the opcode sequence for the STX instruction into the dictionary. When later executed, this code stores the contents of index register X at the effective address specified by arg and mode.
Pronunciation: "store-x"

STY (arg\mode--)
Compiles the opcode sequence for the STY instruction into the dictionary. When later executed, this code stores the contents of index register Y at the effective address specified by arg and mode.
Pronunciation: "store-y"

SUBA (arg\mode--)

Compiles the opcode sequence for the SUBA instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator A and places the result in accumulator A.
Pronunciation: "subtract-a"

SUBB (arg\mode--)
Compiles the opcode sequence for the SUBB instruction into the dictionary. When later executed, this code subtracts the operand specified by arg and mode from the contents of accumulator B and places the result in accumulator B.
Pronunciation: "subtract-b"

SUBD (arg\mode--)
Compiles the opcode sequence for the SUBD instruction into the dictionary. When later executed, this code subtracts the contents of the operand specified by arg and mode from the contents of accumulator D and places the result in accumulator D.
Pronunciation: "subtract-d"

SWI (--)
Compiles the opcode sequence for the SWI instruction into the dictionary. When later executed, this code invokes a software interrupt. It stores the CCR, accumulators A and B, and registers X, Y and PC on the stack. The stack pointer is decremented by 1 for each byte stored on the stack. The I bit in the CCR register is set. The PC register is then loaded with the address located at the interrupt vector for SWI, and instruction execution resumes at this location.
Pronunciation: "software interrupt"

TAB (--)
Compiles the opcode sequence for the TAB instruction into the dictionary. When later executed, this code transfers the contents of accumulator A to accumulator B.
Pronunciation: "transfer-a-to-b"

TAP (--)
Compiles the opcode sequence for the TAP instruction into the dictionary. When later executed, this code transfers the contents of accumulator A to the condition code register.
Pronunciation: "transfer-a-to-condition-code-register"

TBA (--)
Compiles the opcode sequence for the TBA instruction into the dictionary. When later executed, this code transfers the contents of accumulator B to accumulator A.
Pronunciation: "transfer-b-to-a"

TEST (--)
Compiles the opcode sequence for the TEST instruction into the dictionary. When later executed, this code causes the PC to be continuously incremented. This instruction can only be executed when the CPU is configured to operate in test mode. If executed in any mode other than test, the instruction is treated as an illegal opcode.

THEN, (--)

THEN, is used in assembly coded routines to mark the end of a conditional IF, structure. It is a synonym for ENDIF,. See ENDIF,.
Pronunciation: "then-comma"

TPA (--)
Compiles the opcode sequence for the TPA instruction into the dictionary. When later executed, this code transfers the contents of the condition code register into accumulator A.
Pronunciation: "transfer-condition-code-register-to-a"

TST (arg\mode--)
Compiles the opcode sequence for the TST instruction into the dictionary. When later executed, this code subtracts zero from the contents of the operand specified by arg and mode and sets the condition code register's bits accordingly. The operand is unaffected by the subtraction.
Pronunciation: "test"

TSTA (--)
Compiles the opcode sequence for the TSTA instruction into the dictionary. When later executed, this code subtracts zero from the contents of accumulator A and sets the condition code register's bits accordingly. Accumulator A is unaffected.
Pronunciation: "test-a"

TSTB (--)
Compiles the opcode sequence for the TSTB instruction into the dictionary. When later executed, this code subtracts zero from the contents of accumulator B and sets the condition code register's bits accordingly. Accumulator B is unaffected.
Pronunciation: "test-b"

TSX (--)
Compiles the opcode sequence for the TSX instruction into the dictionary. When later executed, this code loads index register X with one plus the contents of the S register (which is the return stack pointer). After this operation the X register points to the top item on the return stack.
Pronunciation: "transfer-stack-to-x"

TSY (--)
Compiles the opcode sequence for the TSY instruction into the dictionary. When later executed, this code loads index register Y with one plus the contents of the S register (which is the return stack pointer). After this operation the Y register points to the top item on the return stack.
Pronunciation: "transfer-s-to-y"

TXS (--)
Compiles the opcode sequence for the TXS instruction into the dictionary. When later executed, this code loads the S register (which is the return stack pointer) with the contents of the index register X minus one.
Pronunciation: "transfer-x-to-s"

TYS (--)

Compiles the opcode sequence for the TYS instruction into the dictionary. When later executed, this code loads the S register (which is the return stack pointer) with the contents of the index register Y minus one.
Pronunciation: "transfer-y-to-s"

UNTIL, (condition --)
UNTIL, designates the end of an assembly coded looping structure and resolves branch instructions according to the specified condition. Use as:

```
BEGIN,              . . . code to be executed . . .
condition UNTIL,
```

If the condition is true, the loop terminates and execution continues with the code following UNTIL,. If the condition is false, looping continues and execution passes to the code following BEGIN,.
Pronunciation: "until-comma"

VC (-- condition)
Used within a code definition, leaves a condition flag on the data stack. The condition is true when the V (2's complement overflow) flag in the condition code register is 0.
Pronunciation: "overflow-clear"

VS (-- condition)
Used within a code definition, leaves a condition flag on the data stack. The condition is true when the V (2's complement overflow) flag in the condition code register is 1.
Pronunciation: "overflow-set"

WAI (--)
Compiles the opcode sequence for the WAI instruction into the dictionary. When later executed, this code saves the condition code register, accumulators A and B, and register X, Y and PC on the stack. The stack pointer is decremented for each byte stored. The CPU the enters a lower power consumption mode, while waiting for an interrupt which has not been masked. When an interrupt is recognized, the CPU sets the I bit in the CCR, and execution resumes at the address stored in the appropriate interrupt vector.
Pronunciation: "wait-for-interrupt"

WHILE, (condition --)
WHILE, is used in assembly coded routines to mark the beginning of a the "while true" portion of a BEGIN, . . . WHILE, . . . REPEAT, loop. Use as:

```
BEGIN,              . . . .
condition WHILE,    . . . .
REPEAT,
```

When executed, WHILE, causes a branch instruction to be compiled into the dictionary.
When the compiled code is executed:
If the condition is true, the loop continues and the code between WHILE, and REPEAT, are executed, after which control is transferred to the code following BEGIN,. If the condition is false, the loop terminates and execution continues with the code following REPEAT,.
Pronunciation: "while-comma"

XGDX (--)

Compiles the opcode sequence for the XGDX instruction into the dictionary. When later executed, this code exchanges the contents of accumulator D with the contents of index register X.
Pronunciation: "exchange-d-and-x"

XGDY	(--) Compiles the opcode sequence for the XGDY instruction into the dictionary. When later executed, this code exchanges the contents of accumulator D with the contents of index register Y. Pronunciation: "exchange-d-and-y"

C Debugger Glossary

This glossary summarizes functions and keywords that assist those who are programming the QED Board using the Control C language. These keywords can be interactively typed at the terminal and interpreted by the QED-Forth debugger and operating system to assist in the calling and debugging of C functions. In general, those who program exclusively in the QED-Forth language will not use these functions.

Entries (Alphabetized in ASCII Order):

=CHAR ([addr] or [xaddr] <name> --)

=CHAR is a QED-Forth function that acts as an interactive assignment operator. It is used in the form:

    <destination> =CHAR <char_specifier>

where <destination> is a 16-bit address left on the data stack by a variable name, or a 32-bit xaddress left on the data stack by a FORTH_ARRAY element. <char_specifier> is either a valid number or a variable name or FORTH_ARRAY element that contains a byte. =CHAR assigns the byte specified by the right-hand-side (RHS) to the memory location specified by the left-hand-side (LHS). The syntax is similar to a C assignment statement.

=FLOAT ([addr] or [xaddr] <name> --)

=FLOAT is a QED-Forth function that acts as an interactive assignment operator. It is used in the form:

    <destination> =FLOAT <float_specifier>

where <destination> is a 16-bit address left on the data stack by a variable name, or a 32-bit xaddress left on the data stack by a FORTH_ARRAY element. <float_specifier> is either a valid floating point number or a variable name or FORTH_ARRAY element that contains a floating point number. =FLOAT assigns the float specified by the right-hand-side to the memory location specified by the left-hand-side. The syntax is similar to a C assignment statement.

=INT ([addr] or [xaddr] <name> --)

=INT is a QED-Forth function that acts as an interactive assignment operator. It is used in the form:

    <destination> =INT <integer_specifier>

where <destination> is a 16-bit address left on the data stack by a variable name, or a 32-bit xaddress left on the data stack by a FORTH_ARRAY element. <integer_specifier> is either a valid number or a variable name or FORTH_ARRAY element that contains an integer. =INT assigns the integer specified by the right-hand-side to the memory location specified by the left-hand-side. The syntax is similar to a C assignment statement.

=LONG ([addr] or [xaddr] <name> --)

=LONG is a QED-Forth function that acts as an interactive assignment operator. It is used in the form:

    <destination> =LONG <long_specifier>

where <destination> is a 16-bit address left on the data stack by a variable name, or a 32-bit xaddress left on the data stack by a FORTH_ARRAY element. <long_specifier>

is either a valid number or a variable name or FORTH_ARRAY element that contains an long. =LONG assigns the long specified by the right-hand-side to the memory location specified by the left-hand-side. The syntax is similar to a C assignment statement.

C$>COUNTED$ (xaddr1 -- x$addr2)

Converts the specified null-terminated string at xaddr1 into a Forth-style counted string at x$addr2 with the count in the first byte and the non-null-terminated string in the remaining bytes. x$addr2 is the 32-bit address of PAD which is where the converted counted string is located. Note that the size of the PAD buffer puts a limit on the string size; the input string length should be less than 86 bytes. See STRINGMOVE() and PAD.
Pronunciation: "c-string-to-counted-string"

CALL.CFN (xaddr <input_parameter_list> --)

A low-level function inserted by the "Make" utility in the .TXT download file created by the Control C compiler. When properly inserted in a QED-Forth function, enables interactive calls to C functions that were declared using the _Q keyword. CALL.CFN expects on the data stack a 32-bit xaddress representing the execution address of the function to be called. CALL.CFN removes from the input stream a list of comma-delimited parameters terminated by the) character. It then sets up the proper stack frame for a "pascal" type function (i.e., a function declared using the _Q or _pascal keyword) that has been compiled by the Control C compiler. CALL.CFN calls the designated function, then prints the return values (passed in the D and Y registers):

in the current number base as two 16-bit integers;
as a 32-bit hexadecimal number; and,
as a floating point number.

It is up to the programmer to decide which (if any) of these return value summaries is relevant based on the declared type of the called function's return value.
Pronunciation: "call-c-function"

CHAR (<name> -- char)

CHAR is a QED-Forth function that examines the next token; if it is a valid number such as 5 or 3.2, CHAR simply converts it to the nearest 8-bit byte. There is an unchecked error if the input is not in the range 0-255 (unsigned char) or -128 to _127 (signed char). If the next token is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), CHAR extracts the 8-bit contents stored at the specified memory location. CHAR is also used to specify the type of an input parameter when interactively calling a function.

CHAR* (<name> -- char)

CHAR* is a QED-Forth function that examines the next token; if it is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), CHAR* extracts the 16-bit pointer stored at the specified memory location, and in turn extracts the 8-bit byte pointed to by the pointer.

DO[] (addr <input_parameter_list> -- xaddr)

A low-level function inserted by the "Make" utility in the .TXT download file by the Control C compiler. When properly inserted in a QED-Forth function, enables interactive examination and modification of FORTH_ARRAY elements. Expects on the data stack a 16-bit address representing the pfa (parameter field address) of a

FORTH_ARRAY. DO[] removes from the input stream a row specifier, a comma, a column specifier, and a terminating]. It leaves on the stack the 32-bit xaddress of the specified element in the specified FORTH_ARRAY.
Pronunciation: "do-brackets"

FLOAT (<name> -- r | r is an ANSI-C floating point number)
FLOAT is a QED-Forth function that examines the next token; if it is a valid integer or QED-formatted floating point number such as 5 or 3.2, FLOAT simply converts it to an ANSI-C-formatted floating point number. If the next token is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), FLOAT extracts the 32-bit (float) contents stored at the specified memory location. FLOAT is also used to specify the type of an input parameter when interactively calling a function.

FLOAT* (<name> -- r | r is an ANSI-C floating point number)
FLOAT* is a QED-Forth function that examines the next token; if it is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), FLOAT* extracts the 16-bit pointer stored at the specified memory location, and in turn extracts the 32-bit float pointed to by the pointer.

FPtoString (r -- addr | r is an ANSI-C floating point number)
Converts the specified ansi input floating point number to a null-terminated ascii string, and returns the 16-bit address of the string. If the conversion fails, returns 0. The specified number is converted into one of three formats: FIXED, SCIENTIFIC, or FLOATING. FLOATING format is the default after a COLD restart.

FP_CtoQ (r1 -- r2 | r1 is in ANSI C format; r2 is in QED format)
Converts the ANSI/IEEE-standard formatted input floating point number into the QED-Forth floating point format. Converts denormalized input numbers to zero; that is, if the biased exponent = 0, the returned QED-formatted floating point number = zero. NAN (not a number) inputs are converted to +/- infinity depending on their sign bit. The least significant bit (lsb) of the mantissa is not rounded, resulting in up to 1 lsb error during the conversion.

FP_QtoC (r1 -- r2 | r1 is in QED format; r2 is in ANSI C format)
Converts the QED-Forth formatted input floating point format into an ANSI/IEEE-standard formatted floating point number.

INT (<name> -- n)
INT is a QED-Forth function that examines the next token; if it is a valid integer or floating point number such as 5 or 3.2, INT simply converts it to the nearest integer. If the next token is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), INT extracts the 16-bit contents stored at the specified memory location. INT is also used to specify the type of an input parameter when interactively calling a function.

INT* (<name> -- n)
INT* is a QED-Forth function that examines the next token; if it is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY

	element xaddress), INT* extracts the 16-bit pointer stored at the specified memory location, and in turn extracts the 16-bit integer pointed to by the pointer.
LONG	(<name> -- d) LONG is a QED-Forth function that examines the next token; if it is a valid number such as 5 or 1234567 or 453.2, LONG simply converts it to the nearest 32-bit long number. If the next token is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), LONG extracts the 32-bit (long) contents stored at the specified memory location. LONG is also used to specify the type of an input parameter when interactively calling a function.
LONG*	(<name> -- d) LONG* is a QED-Forth function that examines the next token; if it is a named 16-bit address (such as a variable name) or a 32-bit xaddress (such as a FORTH_ARRAY element xaddress), LONG* extracts the 16-bit pointer stored at the specified memory location, and in turn extracts the 32-bit long pointed to by the pointer.
MAIN	(--) Executes the main() function which is located at address 0x0000 on page 0x04. Each compiled C program must contain one and only one definition of the main() function.
PrintFP	(r -- | r is an ANSI-C floating point number) Prints the input ANSI-C floating point parameter using the format specified by the most recent execution of FIXED, SCIENTIFIC, or FLOATING.

Source Form Glossary

This glossary describes some little-used routines that used to reside in QED-Forth V2.xx Kernel PROM, but are now provided instead as source code files on diskette.

Entries (Alphabetized in ASCII Order):

--> (--)
When encountered while interpreting a block, causes the next block to be interpreted. Sets >IN = 0 and increments BLK.
Pronunciation: "to-next-block"

.LINE (n1\n2 -- | n1 = line#, n2 = block#)
Prints the specified line in the specified block.
Pronunciation: "dot-line" Attributes: M

2xN.MATRIX* (2x2.matrix.xpfa\2xN.matrix.xpfa --)
Multiplies a 2x2 source1 matrix specified by 2x2.matrix.xpfa times a 2xN source2 matrix specified by 2xN.matrix.xpfa to produce a destination 2xN matrix that is stored into the source2 matrix specified by 2xN.matrix.xpfa. No error checking is performed, so the source matrices must be properly dimensioned. Multiplies two 2x2 matrices in just 2.7 msec.
Pronunciation: "two-by-n-matrix-star" Attributes: S

3xN.MATRIX* (3x3.matrix.xpfa\3xN.matrix.xpfa --)
Multiplies a 3x3 source1 matrix specified by 3x3.matrix.xpfa times a 3xN source2 matrix specified by 3xN.matrix.xpfa to produce a destination 3xN matrix that is stored into the source2 matrix specified by 3xN.matrix.xpfa. No error checking is performed, so the source matrices must be properly dimensioned. Multiplies two 3x3 matrices in just 13.5 msec.
Pronunciation: "three-by-n-matrix-star" Attributes: S

>L (n <text> -- | n = line#)
Removes remaining text from the input line and puts it in the block (screen) designated by the user variable SCR at the specified line number n. Intended for use from terminal only (not from inside a block)!
Pronunciation: "to-l"

BLOCK.BUFFERS (xaddr\n -- | xaddr = start, n = #blocks)
Allocates a contiguous region of memory for use as block buffers. This command must be executed before using the mass memory interface. xaddr is the extended starting address of the first buffer, and n is the number of block buffers. n is typically equal to 2, although allocating more buffers reduces the time to load a new block when a slow mass memory device is used. BLOCK.BUFFERS restricts n to the range 2 <= n <= 63. The size of each buffer is 1028 (hex 404) bytes, and the buffers may cross page boundaries. BLOCK.BUFFERS initializes FIRST, LIMIT, PREV, and USE. See also IS.RAMDISK.

EMPTY.BUFFERS (--)
Unassigns and blanks all block buffers. Does not write to mass memory.

FLUSH (--)

Unassigns all block buffers, writing to mass memory any UPDATEd blocks.

INIT.UREAD/WRITE (--)

Revectors READ/WRITE to execute the RAMDISK function which treats a specified block of RAM as a mass memory device. To use the RAMDISK, execute INIT.UREAD/WRITE in your AUTOSTART or PRIORITY.AUTOSTART routine so it is installed after each restart. To use an external mass memory device, define a routine with the same stack picture as READ/WRITE (see the main glossary) that accesses the mass memory device, and install its extended code field address (xcfa) in the UREAD/WRITE user variable.

IS.RAMDISK (xaddr\u -- | xaddr = start, u = number of 1K blocks)

Declares an area of memory starting with xaddr and totaling (1024*u) bytes as a ram disk, and stores 0\0 into OFFSET. The specified ram disk memory may cross page boundaries and may occupy multiple contiguous pages. This word must be executed before using the ram disk feature of the mass memory (blocks) interface. Note that BLOCK.BUFFERS must also be executed before using the blocks interface.
Implementation detail: Initializes the headerless system variables RAMDISK.START and #RAMDISK.BLOCKS appropriately, and zeroes the user variable OFFSET.
Pronunciation: "is-ram-disk"

LINE>$ (n1\n2 -- xaddr\cnt | n1 = line#, n2 = block#)

Converts line# n1 and block# (screen#) n2 into a string specification xaddr\cnt.
Pronunciation: "line-to-string"

LIST (n -- | n = block#)

Displays the specified block and stores n into the user variable SCR. The block is printed as 16 lines of 64 characters per line.
Attributes: M, S

LOAD (n -- | n = block#)

Loads (i.e., interprets) the specified block. Block 0 cannot be loaded.

SAVE.BUFFERS (--)

Transfers the contents of all UPDATEd block buffers to mass storage. Marks all buffers as unmodified.

SUBSTRING (xaddr1\cnt1\xaddr2\cnt2--xaddr3\cnt3\flag)

Searches for a substring specified by xaddr1\cnt (i.e., the address of the first character under the character count) in the larger string xaddr2\cnt2, and returns the closest match xaddr3\cnt3 in the larger string. When checking for a match, comparison starts with the first character in the smaller string. flag is TRUE if a full match is found, and FALSE if a partial or no match is found. If no match is found, flag is FALSE and cnt3 equals 0.

THRU (u1\u2 -- | u1 = starting.block#, u2 = ending.block#)

Loads (i.e., interprets) from the starting block u1 through the ending block u2, inclusive.
Pronunciation: "through"