Appendix eForth_328 Commands

Additional note JPIN: these commands are copied from the full documentation.
Learning Forth is very much learning about the STACK (here the Data Stack).

The 3 colums in these pages:
The name of the command / Forth Word
What is happening on the Stack: DATA BEFORE EXECUTION -- DATA AFTER EXECUTION
I have modified this column slightly, so the 2 dashes are mostly at the same location. Easier to read.

Try a few Words!
One of the nice features of this eForth: the 4 top locations of the stack are shown all the time:
At Startup: **0 0 0 0 ok**
Store the numbers needed for the Word tried, execute the command, see what happened.
e.g. for the first line **-** type **10 (CR), 3 (CR), - (CR)**
and you see what happens on the Stack.

-	**(n1 n2 -- n3)**	Subtract n2 from n1 (n1-n2=n3).
' <name>	**(-- addr)**	Find <name> and leave its address.
!	**(n addr --)**	Store n to addr.
!IO	**(--)**	Initialize the serial I/O devices.
#	**(n -- n/base)**	Convert next digit of n and add it to output string .
#>	**(n -- addr n1)**	Terminate numeric conversion, leaving addr and count n1.
#S	**(n --)**	Convert all significant digits in n to output string.
#TIB	**(-- addr)**	Return address of variable storing number of characters received in terminal input buffer.
$" <string>	**(-- addr)**	Compile a string literal. Return its address at run time.
S"|	**(-- addr)**	Return address of following string literal at run time.
$,n	**(addr --)**	Build a new dictionary header using the string at addr.
$COMPILE	**(addr --)**	Compile string at addr to dictionary as a token or literal.
$INTERPRET	**(addr --)**	Interpret string a addr. Execute it of convert it to a number.
(<text>)	**(--)**	Ignore comment text.
(parse)	**(addr n char -- addr n delta)**	Scan string delimited by char. Return found string and its offset delta.
*****	**(n1 n2 -- n3)**	Signed multiply. Leave product.
***/**	**(n1 n2 n3 -- n4)**	Signed multiply and divide. Leave quotient of (n1*n2)/n3.
***/MOD**	**(n1 n2 n3 -- n4)**	Signed multiply and divide. Leave remainder of (n1*n2)/n3.
,	**(n --)**	Add n to parameter field of the most recently defined word.
.	**(n --)**	Display signed number with a trailing blank.
." <text>"	**(--)**	Compile <text> message. At run-time display text message.
."!	**(--)**	Display following string literal as a text message.
.(<text>)	**(--)**	Display <text> received from the input stream.
.ID	**(addr --)**	Display name of a command at addr.
.OK	**(--)**	Display ok> message.
.R	**(n n1 --)**	Display n right justified in a field of n1 character width.
/	**(n1 n2 -- quot)**	Signed division. Leave quotient of n1/n2.
/MOD	**(n1 n2 -- rem quot)**	Signed division. Leave quotient and remainder of n1/n2.
: <name>	**(--)**	Begin a compound command of <name>.
;	**(--)**	Terminate a compound command.
?	**(addr --)**	Display contents in addr.
?DUP	**(n -- n n | 0)**	Duplicate top of stack if it is not a 0.
?KEY	**(-- char T | F)**	Return input character and true, or a false if no input.

?RX	**(-- char T | F)**	Return input character and true, or a false if no input.
?UNIQUE	**(addr --)**	Display a "reDef" message if addr is an existing command.
@	**(addr -- n)**	Replace addr by number fetched from addr.
[	**(--)**	Switch from compilation to interpretation.
[COMPILE] <name>	**(--)**	Compile the word <name> in the input stream as an token.
\ <text>	**(--)**	Ignore text till end of line.
]	**(--)**	Switch from interpretation to compilation.
^H	**(bot eot cur -- bot eot cur)**	Backspace. Backup the cursor by one character.
+	**(n1 n2 -- n3)**	Add n1 and n2.
+!	**(n addr --)**	Add n to number at addr.
<	**(n1 n2 -- flag)**	True if n1 less than n2.
<#	**(--)**	Start numeric output conversion.
<MARK	**(-- addr)**	Push current program address on stack.
<RESOLVE	**(addr --)**	Compile addr to dictionary.
=	**(n1 n2 -- flag)**	True if n1 equals n2.
>	**(n1 n2 -- flag)**	True if n1 greater than n2.
>CHAR	**(n -- char)**	Convert n to a printable character char. Non-printable character is converted to an underscore character.

>IN	**(-- addr)**	Return address of a variable pointing to current character being interpreted.
>MARK	**(-- addr)**	Compile 0 to dictionary. Push its address on stack
>NAME	**(ca -- na)**	Convert a code field address to a name field address.
>R	**(n --)**	Pop top and push it on return stack.
>RESOLVE	**(addr --)**	Store address of current program word in addr.
>UPPER	**(addr --)**	Convert a count string at addr to upper case.
0<	**(n -- flag)**	True if n is negative.
0=	**(n -- flag)**	True if n is 0.
1-	**(n -- n-1)**	Decrement top.
1+	**(n -- n+1)**	Increment top.
2-	**(n -- n-2)**	Decrement top by 2.
2!	**(d addr --)**	Store a double integer to addr.
2*	**(n -- 2n)**	Multiply top by 2.
2/	**(n -- n/2)**	Divide top by 2.
2@	**(addr -- d)**	Fetch a double integer from addr.
2+	**(n -- n+2)**	Increment top by 2
2DROP	**(d --)**	Pop two numbers off stack.
2DUP	**(d -- d d)**	Duplicate a double integer on stack.
ABORT	**(--)**	Clean up stack and jump to address in 'ABORT.
'ABORT	**(-- addr)**	Return address to handle error condition.
abort"	**(flag --)**	If flag is true, display following message and ABORT.
ABS	**(n -- u)**	Return absolute value of top.
accept	**(addr n -- addr n1)**	Accept n characters to buffer at addr. Replace n with actual count n1
AFT	**(--)**	Branch to THEN to skip a branch in FOR-NEXT loop.
AHEAD	**(--)**	Branch forward to address in next word.
ALIGNED	**(n -- n1)**	Adjust n to the word boundary.
ALLOT	**(+n --)**	Add +n bytes to parameter field of the most recently word.
AND	**(n1 n2 -- n3)**	Logical bit-wise AND.
BASE	**(-- addr)**	Contain radix for numeric conversion.

BEGIN	**(--)**	Start an indefinite loop.
BL	**(-- 32)**	Push 32 on stack.
BRANCH	**(flag --)**	Branch to address in next program word if flag is 0.
C!	**(n addr --)**	Store a byte to addr.
C@	**(addr -- n)**	Fetch a byte from addr.
CHAR <string>	**(-- char)**	Push first character in the following text string.
CHARS	**(n char --)**	Send n characters char to the output device.
CMOVE	**(addr addr1 n --)**	Copy n bytes starting at addr to memory starting at addr1.
CODE <name>	**(--)**	Start a new primitive command.
COLD	**(--)**	Initialize FORTH system and start text interpreter.
COMPILE <name>	**(--)**	Retrieve address of the following command and compile it as a token.
CONSTANT <name>	**(n --)**	Define a constant. At run-time, n is pushed on the stack.
CONTEXT	**(-- addr)**	Return address of a variable pointing to name field of last word in dictionary.
COUNT	**(addr -- addr+1 n)**	Replace addr with address and count of a count string.
CP	**(-- addr)**	Return address of a variable pointing to first free space on dictionary.
CR	**(--)**	Display a new line. Carriage return and line feed.
CREATE <name>	**(--)**	Define an array. At run-time, its address is left on the stack.
DECIMAL	**(--)**	Set number base to decimal.

DIAGNOSE	**(--)**	Exercise all primitive commands for debugging.
DIGIT	**(n -- char)**	Convert digit u to a character.
DNEGATE	**(d -- d1)**	Negate a double integer on stack.
do$	**(-- addr)**	Return the address of the following compiled string.
doCON	**(-- n)**	Return contents of next program word.
doLIST	**(--)**	Start processing a new nested list.
doLIT	**(-- n)**	Push an inline literal.
doNEXT	**(--)**	Terminate a single index loop.
doVAR	**(-- addr)**	Return address of next program word.
DROP	**(n --)**	Discard top of stack.
DUMP	**(addr n --)**	Dump n bytes of memory starting from addr.
DUP	**(n1 -- n2)**	Duplicate top of stack.
ELSE	**(--)**	Terminate <true> clause, continue after the THEN.
EMIT	**(char --)**	Initialize the serial I/O devices.
ERASE	**(addr n --)**	Clear a n byte array at addr
ERROR	**(addr --)**	Display error message at addr and jump to ABORT.
EVAL	**(--)**	Interpret input stream in terminal input buffer.
'EVAL	**(-- addr)**	Return address of variable containing $INTERPRET or $COMPILE.
EXECUTE	**(addr --)**	Execute the command at addr.
EXIT	**(--)**	Terminate execution of current compound command.
EXPECT	**(addr n --)**	Accept n characters into buffer at addr.
EXTRACT	**(n base -- n/base n1)**	Extract the least significant digit n1 from n. n is divided by base.
FILL	**(addr n char --)**	Fill an array at address with n characters char.
find	**(a va -- ca na | a 0)**	Search dictionary at va for a string at a. Return ca and na if succeeded, else return a and 0.

FOR	**(n --)**	Setup loop. Repeat loop until limit n is decremented to 0.
FORGET <name>	**(--)**	Delete command <name> and all words added afterwards.
HERE	**(-- addr)**	Address of next available dictionary location.
HLD	**(-- addr)**	Return address of a variable pointing to next converted digit.
HOLD	**(char --)**	Add character char to the number string under conversion.
IF	**(flag --)**	If flag is zero, branches forward to <false> or after THEN.
IMMEDIATE	**(--)**	Set immediate bit in name field of last command added.
KEY	**(-- char)**	Get an ASCII character from the keyboard. Does not echo.
kTAP	**(bot eot cur char -- bot eot cur)**	Process a control character, CR or backspace.
LAST	**(-- char)**	Get an ASCII character from the keyboard. Does not echo.
LITERAL	**(n --)**	Compile number n. At run-time, n is pushed on the stack.
M*	**(n1 n2 -- d)**	Multiply n1 and n2. Return double integer product.
M/MOD	**(d n -- mod quot)**	Divide double integer d by n1. Return remainder and quotient.
MAX	**(n1 n2 -- n3)**	n3 is the larger of n1 and n2.
MIN	**(n1 n2 -- n3)**	n3 is the smaller of n1 and n2.
MOD	**(n1 n2 -- mod)**	Signed divide. Leaver remainder of n1/n2.
NAME?	**(addr -- ca na | a F)**	Search dictionary for name at addr. Return code field address and name field address if a command is found, else push a false.
NAME>	**(na -- ca)**	Convert a name field address to a code field address.
NEGATE	**(n1 -- n2)**	Two's complement.
NEXT	**(--)**	Decrement index and repeat loop until index is less than 0
NOT	**(n1 -- n2)**	Bit-wise one's complement.
NUMBER?	**(addr -- n T | addr F)**	Convert a string at addr to an integer and push a true flag. If it is not a number, push a false flag.
OR	**(n1 n2 -- n3)**	Logical bit-wise OR.
OVER	**(n1 n2 -- n1 n2 n1)**	Make copy of second item on stack.
OVERT	**(--)**	Change CONTEXT to add a new command to dictionary.
PACK$	**(addr n -- addr1)**	Copy a string at addr with length n, to a count string at addr1.
PAD	**(-- addr)**	Return address of a scratch pad area.
PARSE	**(char -- addr n)**	Parse terminal input buffer for a string terminated by char. Return its address and length.
PEEK	**(addr -- n)**	Fetch a byte from addr.
POKE	**(n addr --)**	Store a byte to addr.
QBRANCH	**(flag --)**	Branch to address in next word if flag is zero.
QUERY	**(-- addr)**	Leave address of a scratch area of at least 84 bytes.
QUIT	**(--)**	Return to terminal, no stack change, no message.
R@	**(-- n)**	Copy top of return stack on stack.
R>	**(-- n)**	Pop top of return stack and push it on stack.
REPEAT	**(--)**	Unconditional backward branch to BEGIN.
ROT	**(n1 n2 n3 -- n2 n3 n1)**	Rotate third item to top. "rote"
SAME?	**(addr1 addr2 n -- aadr1 addr2 flag)**	Compare two strings at addr1 and addr2 for n bytes. If string1>string2, returns a positive integer. If string1<string2, return a negative integer. If strings are identical, return a 0.
SEE <name>	**(--)**	Decompile the word <name>.
SIGN	**(n --)**	If n is negative, add a - sign to the number output string.
SPACE	**(--)**	Display a space.
str	**(n -- addr n1)**	Convert signed integer n to a numeric output string at addr, length n1.
SPACES	**(n --)**	Display n spaces.
SWAP	**(n1 n2 -- n2 n1)**	Exchange top two stack items.

TAP	**(bot eot cur char -- bot eot cur)**	Accept and echo a character and bump the cursor.
THEN	**(--)**	Terminate the IF-ELSE structure.
TIB	**(-- addr)**	Push address of terminal input buffer.
'TIB	**(-- addr)**	Return address of variable pointing to terminal input buffer.
tmp	**(-- addr)**	Return address of a temporary variable.
TOKEN	**(-- addr)**	Parse next string delimited by space into a word buffer 2 bytes above the top of dictionary.
TX!	**(char --)**	Send character c to the output device.
TYPE	**(addr +n --)**	Display a string of +n characters starting at address addr.
U.	**(n --)**	Display unsigned number with trailing blank.
U.R	**(n n1 --)**	Display unsigned number n right justified in a field of n1 characters.
U<	**(n1 n2 -- flag)**	Unsigned compare. Return true if n1<n2.
UM*	**(n1 n2 -- d)**	Unsigned multiply. Return double integer product.
UM/MOD	**(d n -- mod quot)**	Unsigned divide. Return remainder and quotient.
UM+	**(n1 n2 -- d)**	Unsigned add. Return double integer sum.
UNTIL	**(flag --)**	Repeat <loop-body> until the flag is non-zero.
UPPER	**(char -- char1)**	Convert a character to upper case.
VARIABLE <name>	**(--)**	Define a variable. At run-time, <name> leaves its address.
WHILE	**(flag --)**	Repeat <loop-body> and <true> clause while the flag is non-zero.
WITHIN	**(n1 n2 n3 -- flag)**	Return true flag if n1<=n3<n2. Else, return false flag.
WORD <text>	**(char -- addr)**	Get the char delimited string <text> from the input stream and leave as a counted string at addr.
WORDS	**(--)**	Display all commands in the dictionary.
XOR	**(n1 n2 -- n3)**	Logical bit-wise exclusive OR.