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.

186	WORDS	(--)	Display all commands in the dictionary.	1
42	+	(n1 n2 -- n3)	Add n1 and n2.	2
1	-	(n1 n2 -- n3)	Subtract n2 from n1 (n1-n2=n3).	3
16	*	(n1 n2 -- n3)	Signed multiply. Leave product.	4
27	/	(n1 n2 -- quot)	Signed division. Leave quotient of n1/n2.	5
17	*/	(n1 n2 n3 -- n4)	Signed multiply and divide. Leave quotient of (n1*n2)/n3.	6
28	/MOD	(n1 n2 -- rem quot)	Signed division. Leave quotient and remainder of n1/n2.	7
18	*/MOD	(n1 n2 n3 -- n4)	Signed multiply and divide. Leave remainder of (n1*n2)/n3.	8
137	MOD	(n1 n2 -- mod)	Signed divide. Leaver remainder of n1/n2.	8
72	ABS	(n -- u)	Return absolute value of top.	9
135	MAX	(n1 n2 -- n3)	n3 is the larger of n1 and n2.	10
136	MIN	(n1 n2 -- n3)	n3 is the smaller of n1 and n2.	11
48	=	(n1 n2 -- flag)	True if n1 equals n2.	12
44	<	(n1 n2 -- flag)	True if n1 less than n2.	13

49	>	(n1 n2 -- flag)	True if n1 greater than n2.	14
58	0=	(n -- flag)	True if n is 0.	15
57	0<	(n -- flag)	True if n is negative.	16
78	AND	(n1 n2 -- n3)	Logical bit-wise AND.	17
144	OR	(n1 n2 -- n3)	Logical bit-wise OR.	18
187	XOR	(n1 n2 -- n3)	Logical bit-wise exclusive OR.	19
140	NEGATE	(n1 -- n2)	Two's complement.	20
142	NOT	(n1 -- n2)	Bit-wise one's complement.	21
54	>R	(n --)	Pop top and push it on return stack.	22
156	R>	(-- n)	Pop top of return stack and push it on stack.	23
155	R@	(-- n)	Copy top of return stack on stack.	24
109	DUP	(n1 -- n2)	Duplicate top of stack.	25
107	DROP	(n --)	Discard top of stack.	26
165	SWAP	(n1 n2 -- n2 n1)	Exchange top two stack items.	27
158	ROT	(n1 n2 n3 -- n2 n3 n1)	Rotate third item to top. "rote"	28
145	OVER	(n1 n2 -- n1 n2 n1)	Make copy of second item on stack.	29
21	." <text>"	(--)	Compile <text> message. At run-time display text message.	30
39	\ <text>	(--)	Ignore text till end of line.	31
14	(<text>)	(--)	Ignore comment text.	32
80	BEGIN	(--)	Start an indefinite loop.	33
180	UNTIL	(flag --)	Repeat <loop-body> until the flag is non-zero.	34
29	: <name>	(--)	Begin a compound command of <name>.	35
30	;	(--)	Terminate a compound command.	36
150	PEEK	(addr -- n)	Fetch a byte from addr.	37
151	POKE	(n addr --)	Store a byte to addr.	38

2	' <name>	(-- addr)	Find <name> and leave its address.
3	!	(n addr --)	Store n to addr.
4	!IO	(--)	Initialize the serial I/O devices.
5	#	(n -- n/base)	Convert next digit of n and add it to output string .
6	#>	(n -- addr n1)	Terminate numeric conversion, leaving addr and count n1.
7	#S	(n --)	Convert all significant digits in n to output string.
8	#TIB	(-- addr)	Return address of variable storing number of characters received in terminal input buffer.
9	$" <string>	(-- addr)	Compile a string literal. Return its address at run time.
11	$,n	(addr --)	Build a new dictionary header using the string at addr.
12	$COMPILE	(addr --)	Compile string at addr to dictionary as a token or literal.
13	$INTERPRET	(addr --)	Interpret string a addr. Execute it of convert it to a number.
15	(parse)	(addr n char -- addr n delta)	Scan string delimited by char. Return found string and its offset delta.
19	,	(n --)	Add n to parameter field of the most recently defined word.
20	.	(n --)	Display signed number with a trailing blank.
22	."!	(--)	Display following string literal as a text message.
23	.(<text>)	(--)	Display <text> received from the input stream.
24	.ID	(addr --)	Display name of a command at addr.
25	.OK	(--)	Display ok> message.
26	.R	(n n1 --)	Display n right justified in a field of n1 character width.
31	?	(addr --)	Display contents in addr.
32	?DUP	(n -- n n | 0)	Duplicate top of stack if it is not a 0.
33	?KEY	(-- char T | F)	Return input character and true, or a false if no input.
34	?RX	(-- char T | F)	Return input character and true, or a false if no input.
35	?UNIQUE	(addr --)	Display a "reDef" message if addr is an existing command.
36	@	(addr -- n)	Replace addr by number fetched from addr.
37	[	(--)	Switch from compilation to interpretation.

38	[COMPILE] <name>	(--)	Compile the word <name> in the input stream as an token.
40	]	(--)	Switch from interpretation to compilation.
41	^H	(bot eot cur -- bot eot cur)	Backspace. Backup the cursor by one character.
43	+!	(n addr --)	Add n to number at addr.
45	<#	(--)	Start numeric output conversion.
46	<MARK	(-- addr)	Push current program address on stack.
47	<RESOLVE	(addr --)	Compile addr to dictionary.
50	>CHAR	(n -- char)	Convert n to a printable character char. Non-printable character is converted to an underscore character.
51	>IN	(-- addr)	Return address of a variable pointing to current character being interpreted.
52	>MARK	(-- addr)	Compile 0 to dictionary. Push its address on stack
53	>NAME	(ca -- na)	Convert a code field address to a name field address.
55	>RESOLVE	(addr --)	Store address of current program word in addr.
56	>UPPER	(addr --)	Convert a count string at addr to upper case.
59	1-	(n -- n-1)	Decrement top.
60	1+	(n -- n+1)	Increment top.
61	2-	(n -- n-2)	Decrement top by 2.
62	2!	(d addr --)	Store a double integer to addr.
63	2*	(n -- 2n)	Multiply top by 2.
64	2/	(n -- n/2)	Divide top by 2.
65	2@	(addr -- d)	Fetch a double integer from addr.
66	2+	(n -- n+2)	Increment top by 2
67	2DROP	(d --)	Pop two numbers off stack.
68	2DUP	(d -- d d)	Duplicate a double integer on stack.
69	ABORT	(--)	Clean up stack and jump to address in 'ABORT.
70	'ABORT	(-- addr)	Return address to handle error condition.
71	abort"	(flag --)	If flag is true, display following message and ABORT.
73	accept	(addr n -- addr n1)	Accept n characters to buffer at addr. Replace n with actual count n1

74	AFT	(--)	Branch to THEN to skip a branch in FOR-NEXT loop.
75	AHEAD	(--)	Branch forward to address in next word.
76	ALIGNED	(n -- n1)	Adjust n to the word boundary.
77	ALLOT	(+n --)	Add +n bytes to parameter field of the most recently word.
79	BASE	(-- addr)	Contain radix for numeric conversion.
81	BL	(-- 32)	Push 32 on stack.
82	BRANCH	(flag --)	Branch to address in next program word if flag is 0.
83	C!	(n addr --)	Store a byte to addr.
84	C@	(addr -- n)	Fetch a byte from addr.
85	CHAR <string>	(-- char)	Push first character in the following text string.
86	CHARS	(n char --)	Send n characters char to the output device.
87	CMOVE	(addr addr1 n --)	Copy n bytes starting at addr to memory starting at addr1.
88	CODE <name>	(--)	Start a new primitive command.
89	COLD	(--)	Initialize FORTH system and start text interpreter.
90	COMPILE <name>	(--)	Retrieve address of the following command and compile it as a token.
91	CONSTANT <name>	(n --)	Define a constant. At run-time, n is pushed on the stack.
92	CONTEXT	(-- addr)	Return address of a variable pointing to name field of last word in dictionary.
92	COUNT	(addr -- addr+1 n)	Replace addr with address and count of a count string.
94	CP	(-- addr)	Return address of a variable pointing to first free space on dictionary.
95	CR	(--)	Display a new line. Carriage return and line feed.
96	CREATE <name>	(--)	Define an array. At run-time, its address is left on the stack.
97	DECIMAL	(--)	Set number base to decimal.
98	DIAGNOSE	(--)	Exercise all primitive commands for debugging.
99	DIGIT	(n -- char)	Convert digit u to a character.

100	DNEGATE	(d -- d1)	Negate a double integer on stack.
101	do$	(-- addr)	Return the address of the following compiled string.
102	doCON	(-- n)	Return contents of next program word.
103	doLIST	(--)	Start processing a new nested list.
104	doLIT	(-- n)	Push an inline literal.
105	doNEXT	(--)	Terminate a single index loop.
106	doVAR	(-- addr)	Return address of next program word.
108	DUMP	(addr n --)	Dump n bytes of memory starting from addr.
110	ELSE	(--)	Terminate <true> clause, continue after the THEN.
111	EMIT	(char --)	Initialize the serial I/O devices.
112	ERASE	(addr n --)	Clear a n byte array at addr
113	ERROR	(addr --)	Display error message at addr and jump to ABORT.
114	EVAL	(--)	Interpret input stream in terminal input buffer.
115	'EVAL	(-- addr)	Return address of variable containing $INTERPRET or $COMPILE.
116	EXECUTE	(addr --)	Execute the command at addr.
117	EXIT	(--)	Terminate execution of current compound command.
118	EXPECT	(addr n --)	Accept n characters into buffer at addr.
119	EXTRACT	(n base -- n/base n1)	Extract the least significant digit n1 from n. n is divided by base.
120	FILL	(addr n char --)	Fill an array at address with n characters char.
121	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.
122	FOR	(n --)	Setup loop. Repeat loop until limit n is decremented to 0.
123	FORGET <name>	(--)	Delete command <name> and all words added afterwards.
124	HERE	(-- addr)	Address of next available dictionary location.
125	HLD	(-- addr)	Return address of a variable pointing to next converted digit.
126	HOLD	(char --)	Add character char to the number string under conversion.
127	IF	(flag --)	If flag is zero, branches forward to <false> or after THEN.
128	IMMEDIATE	(--)	Set immediate bit in name field of last command added.
129	KEY	(-- char)	Get an ASCII character from the keyboard. Does not echo.

130	kTAP	(bot eot cur char -- bot eot cur)	Process a control character, CR or backspace.
131	LAST	(-- char)	Get an ASCII character from the keyboard. Does not echo.
132	LITERAL	(n --)	Compile number n. At run-time, n is pushed on the stack.
133	M*	(n1 n2 -- d)	Multiply n1 and n2. Return double integer product.
134	M/MOD	(d n -- mod quot)	Divide double integer d by n1. Return remainder and quotient.
138	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.
139	NAME>	(na -- ca)	Convert a name field address to a code field address.
141	NEXT	(--)	Decrement index and repeat loop until index is less than 0
143	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.
146	OVERT	(--)	Change CONTEXT to add a new command to dictionary.
147	PACK$	(addr n -- addr1)	Copy a string at addr with length n, to a count string at addr1.
148	PAD	(-- addr)	Return address of a scratch pad area.
149	PARSE	(char -- addr n)	Parse terminal input buffer for a string terminated by char. Return its address and length.
152	QBRANCH	(flag --)	Branch to address in next word if flag is zero.
153	QUERY	(-- addr)	Leave address of a scratch area of at least 84 bytes.
154	QUIT	(--)	Return to terminal, no stack change, no message.
157	REPEAT	(--)	Unconditional backward branch to BEGIN.
10	S"|	(-- addr)	Return address of following string literal at run time.
159	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.
160	SEE <name>	(--)	Decompile the word <name>.
161	SIGN	(n --)	If n is negative, add a - sign to the number output string.
162	SPACE	(--)	Display a space.
164	SPACES	(n --)	Display n spaces.
163	str	(n -- addr n1)	Convert signed integer n to a numeric output string at addr, length n1.

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

3	1
18	2
22	3
23	4
26	5
24	6
17	7
25	8
22	8
19	9
20	10
21	11
13	12
12	13

14	14
16	15
15	16
17	17
20	18
21	19
18	20
19	21
25	22
27	23
26	24
30	25
28	26
33	27
32	28
31	29
34	30
35	31
36	32
	33
55	34
1	35
2	36
23	
24	

4

5

36

37

6

7

45

8

46

47

48

49

9

10

29

50

51

52

53

54

11

56