

Draft Version

NOTICE TO ALL ---

This is an
experimental

excerpt of the original

PropForth
documentation

that is created to ease

learning of the more

commonly useful
lexicon.

PropForth - Short listed

An Introduction to Interactively programming parallel
processors

Sal Sanci

[1.1](#) <#> 161

[9.1.2](#) [#>](#). 161

[9.1.3](#) [#s](#). 161

[9.1.29](#) ['](#) 165

[9.1.43](#) [+](#) 168

[9.1.44](#) [+loop](#). 168

[9.1.45](#) [-](#) 168

[9.1.46](#) [-1](#). 168

[9.1.47](#) [.](#) 169

[9.1.48](#) [."](#) 169

[9.1.49](#) [...](#) 169

[9.1.50](#) [.cstr](#). 169

[9.1.51](#) [.str](#). 169

[9.1.52](#) [.strname](#). 169

[9.1.53](#) [0](#) 170

[9.1.54](#) [0<](#). 170

[9.1.55](#) [0<>](#). 170
[9.1.56](#) [0=](#). 170
[9.1.57](#) [0>](#). 170
[9.1.58](#) [0>=](#). 170
[9.1.59](#) [0branch](#). 170
[9.1.60](#) [1](#) 171
[9.1.61](#) [1+](#). 171
[9.1.62](#) [1-](#). 171
[9.1.63](#) [2](#) 171
[9.1.64](#) [2+](#). 171
[9.1.65](#) [2-](#). 171
[9.1.66](#) [2/](#). 171
[9.1.67](#) [2>r](#). 171
[9.1.68](#) [2drop](#). 172
[9.1.69](#) [2dup](#). 172
[9.1.70](#) [2lock](#). 172
[9.1.71](#) [2unlock](#). 172
[9.1.72](#) [3drop](#). 172
[9.1.73](#) [4*](#). 172
[9.1.74](#) [4+](#). 172
[9.1.75](#) [;](#) 173
[9.1.76](#) [;](#) 173
[9.1.77](#) [≤](#) 173
[9.1.78](#) [<#](#). 173
[9.1.79](#) [≤=](#). 173
[9.1.80](#) [<>](#). 173
[9.1.81](#) [≡](#) 173
[9.1.82](#) [≥](#) 174
[9.1.83](#) [≥=](#). 174
[9.1.84](#) [>con](#). 174
[9.1.85](#) [>in](#). 174
[9.1.86](#) [>m.](#) 174
[9.1.87](#) [>out](#). 174
[9.1.88](#) [>r](#). 175
[9.1.89](#) [C!](#). 175
[9.1.90](#) [C@...](#) 175
[9.1.91](#) [C@++](#). 175

[9.1.92](#) [COG!](#). 175
[9.1.93](#) [COG@...](#) 175
[9.1.94](#) [ERR](#). 176
[9.1.95](#) [L!](#). 176
[9.1.96](#) [L@...](#) 176
[9.1.97](#) [RS!](#). 176
[9.1.98](#) [RS@...](#) 176
[9.1.99](#) [ST!](#). 176
[9.1.100](#) [ST@...](#) 177
[9.1.101](#) [W!](#). 177
[9.1.102](#) [W+!](#). 177
[9.1.103](#) [W@...](#) 177
[9.1.104](#) [\[if](#). 177
[9.1.105](#) [\[ifdef](#). 177
[9.1.106](#) [\[ifndef](#). 178
[9.1.107](#) [\.](#) 178
[9.1.108](#) [\]](#). 178

[9.1.144](#) [accept](#). 185
[9.1.145](#) [alignl](#) 185
[9.1.146](#) [alignw..](#) 185
[9.1.147](#) [allot](#). 186
[9.1.148](#) [and](#). 186
[9.1.149](#) [andn](#). 186
[9.1.150](#) [andnC!](#). 186
[9.1.151](#) [asmlabel](#) 186
[9.1.152](#) [base](#). 186
[9.1.153](#) [begin](#). 186
[9.1.154](#) [between](#). 187
[9.1.155](#) [bl](#) 187
[9.1.156](#) [bounds](#). 187
[9.1.157](#) [branch](#). 187
[9.1.158](#) [build_BootKernel](#) 187
[9.1.159](#) [build_BootOpt](#). 187
[9.1.160](#) [c"](#). 187
[9.1.161](#) [c](#). 188
[9.1.162](#) [cappend](#). 188

[9.1.163](#) [cappendn](#). 188
[9.1.164](#) [ccopy](#). 188
[9.1.165](#) [ccreate](#). 188
[9.1.166](#) [cds](#). 188
[9.1.167](#) [checkdict](#). 189
[9.1.168](#) [clearkeys](#). 189
[9.1.169](#) [clkfreq](#). 189
[9.1.170](#) [cmove](#). 189
[9.1.171](#) [cnt](#). 189
[9.1.172](#) [cogcds](#). 189
[9.1.173](#) [coghere](#). 190
[9.1.174](#) [cogid](#). 190
[9.1.175](#) [cogio](#). 190
[9.1.176](#) [cogiochan](#). 190
[9.1.177](#) [cognchan](#). 190
[9.1.178](#) [cognumpad](#). 190
[9.1.179](#) [cogpad](#). 191
[9.1.180](#) [cogreset](#). 191
[9.1.181](#) [cogstate](#). 191
[9.1.182](#) [cogstop](#). 191
[9.1.183](#) [cogx](#). 191
[9.1.184](#) [compile?](#). 191
[9.1.185](#) [cq](#). 191
[9.1.186](#) [cr](#). 192
[9.1.187](#) [create](#). 192
[9.1.188](#) [cstr=](#). 192
[9.1.189](#) [delms](#). 192
[9.1.190](#) [dictend](#). 192
[9.1.191](#) [dira](#). 192
[9.1.192](#) [do](#). 192
[9.1.193](#) [doconl](#) 193
[9.1.194](#) [doconw..](#) 193
[9.1.195](#) [doloop](#). 193
[9.1.196](#) [dothen](#). 193
[9.1.197](#) [dovarl](#) 193
[9.1.198](#) [dovarw..](#) 193
[9.1.199](#) [dq](#). 194

[9.1.200](#) [drop](#). 194
[9.1.201](#) [dup](#). 194
[9.1.202](#) [else](#). 194
[9.1.203](#) [emit](#). 194
[9.1.204](#) [exec](#). 194
[9.1.205](#) [execute](#). 194
[9.1.206](#) [execword](#). 195
[9.1.207](#) [exit](#). 195
[9.1.208](#) [femit?](#). 195
[9.1.209](#) [fill](#) 195
[9.1.210](#) [find](#). 195
[9.1.211](#) [fisnumber](#). 195
[9.1.212](#) [fkey?](#). 196
[9.1.213](#) [fl](#) 196
[9.1.214](#) [fl_in](#). 196
[9.1.215](#) [fl_lock](#). 196
[9.1.216](#) [fnumber](#). 196
[9.1.217](#) [forthentry](#). 196
[9.1.218](#) [freedict](#). 197
[9.1.219](#) [fstart](#). 197
[9.1.220](#) [here](#). 197
[9.1.221](#) [herelal](#) 197
[9.1.222](#) [herewal](#) 197
[9.1.223](#) [hex](#). 197
[9.1.224](#) [hubopf](#). 198
[9.1.225](#) [hubopr](#). 198
[9.1.226](#) [i](#) 198
[9.1.227](#) [if](#). 198
[9.1.228](#) [immediate](#). 198
[9.1.229](#) [ina](#). 198
[9.1.230](#) [init_coghere](#). 199
[9.1.231](#) [initcon](#). 199
[9.1.232](#) [interpret](#). 199
[9.1.233](#) [interpretpad](#). 199
[9.1.234](#) [io](#). 199
[9.1.235](#) [ioconn](#). 199
[9.1.236](#) [iodis](#). 200

[9.1.237](#) [iolink](#). 200
[9.1.238](#) [iounlink](#). 200
[9.1.239](#) [isdigit](#). 200
[9.1.240](#) [isnamechar](#). 200
[9.1.241](#) [isnumber](#). 200
[9.1.242](#) [isunnumber](#). 201
[9.1.243](#) [key](#). 201
[9.1.244](#) [l](#), 201
[9.1.245](#) [l>w..](#) 201
[9.1.246](#) [lasterr](#). 201
[9.1.247](#) [lastnfa](#). 201
[9.1.248](#) [leave](#). 201
[9.1.249](#) [litl](#) 202
[9.1.250](#) [litw..](#) 202
[9.1.251](#) [lock](#). 202
[9.1.252](#) [lockdict](#). 202
[9.1.253](#) [loop](#). 202
[9.1.254](#) [lshift](#). 202
[9.1.255](#) [lxasm..](#) 202
[9.1.256](#) [max](#). 203
[9.1.257](#) [memend](#). 203
[9.1.258](#) [min](#). 203
[9.1.259](#) [name=](#). 203
[9.1.260](#) [namecopy](#). 203
[9.1.261](#) [namelen](#). 203
[9.1.262](#) [namemax](#). 204
[9.1.263](#) [negate](#). 204
[9.1.264](#) [nextword](#). 204
[9.1.265](#) [nfa>lfa](#). 204
[9.1.266](#) [nfa>next](#). 204
[9.1.267](#) [nfa>pfa](#). 204
[9.1.268](#) [nfcog](#). 205
[9.1.269](#) [nip](#). 205
[9.1.270](#) [npfx](#). 205
[9.1.271](#) [number](#). 205
[9.1.272](#) [numpad](#). 205
[9.1.273](#) [numpadsize](#). 205

[9.1.274](#) [onboot](#). 206
[9.1.275](#) [onreset](#). 206
[9.1.276](#) [or](#). 206
[9.1.277](#) [orC!](#). 206
[9.1.278](#) [orInfa](#). 206
[9.1.279](#) [outa](#). 206
[9.1.280](#) [over](#). 206
[9.1.281](#) [pad](#). 207
[9.1.282](#) [pad>in](#). 207
[9.1.283](#) [pad>out](#). 207
[9.1.284](#) [padbl](#) 207
[9.1.285](#) [padsiz](#)e. 207
[9.1.286](#) [par](#). 207
[9.1.287](#) [parse](#). 208
[9.1.288](#) [parsebl](#) 208
[9.1.289](#) [parsenw..](#) 208
[9.1.290](#) [parseword](#). 208
[9.1.291](#) [pfa>nfa](#). 208
[9.1.292](#) [prop](#). 209
[9.1.293](#) [propid](#). 209
[9.1.294](#) [r>](#). 209
[9.1.295](#) [reboot](#). 209
[9.1.296](#) [reset](#). 209
[9.1.297](#) [rot](#). 209
[9.1.298](#) [rot2](#). 209
[9.1.299](#) [rshift](#). 210
[9.1.300](#) [serial](#) 210
[9.1.301](#) [seti](#) 210
[9.1.302](#) [skipbl](#) 211
[9.1.303](#) [space](#). 211
[9.1.304](#) [spaces](#). 211
[9.1.305](#) [state](#). 211
[9.1.306](#) [swap](#). 211
[9.1.307](#) [t0](#). 212
[9.1.308](#) [t1](#). 212
[9.1.309](#) [tbuf](#). 212
[9.1.310](#) [then](#). 212

[9.1.311](#) [thens](#). 212
[9.1.312](#) [tochar](#). 213
[9.1.313](#) [todigit](#). 213
[9.1.314](#) [tuck](#). 213
[9.1.315](#) [u*](#). 213
[9.1.316](#) [u](#). 213
[9.1.317](#) [u/](#). 213
[9.1.318](#) [u/mod](#). 213
[9.1.319](#) [u>=](#). 214
[9.1.320](#) [um*](#). 214
[9.1.321](#) [um/mod](#). 214
[9.1.322](#) [unlock](#). 214
[9.1.323](#) [unlockall](#) 214
[9.1.324](#) [until](#) 214
[9.1.325](#) [unumber](#). 215
[9.1.326](#) [version](#). 215
[9.1.327](#) [w](#), 215
[9.1.328](#) [w>|](#) 215
[9.1.329](#) [wconstant](#). 215
[9.1.330](#) [wlastnfa](#). 215
[9.1.331](#) [wvariable](#). 216
[9.1.332](#) [xisnumber](#). 216
[9.1.333](#) [xnumber](#). 216
[9.1.334](#) [xor](#). 216
[9.1.335](#) [{](#). 216
[9.1.336](#) [}](#). 216

[9.2](#) [DevKernel Word Set \(BuildDevKernel\)](#). 217

[9.2.1](#) [#C](#). 217

[9.2.15](#) [_](#) 219

[9.2.16](#) [*/](#). 219

[9.2.17](#) [*/mod](#). 219

[9.2.18](#) [.byte](#). 219

[9.2.19](#) [.cogch](#). 220

[9.2.20](#) [.con.](#) 220
[9.2.21](#) [.conbyte.](#) 220
[9.2.22](#) [.concr.](#) 220
[9.2.23](#) [.concstr.](#) 220
[9.2.24](#) [.conemit.](#) 220
[9.2.25](#) [.conlong.](#) 221
[9.2.26](#) [.const?.](#) 221
[9.2.27](#) [.conwait.](#) 221
[9.2.28](#) [.conword.](#) 221
[9.2.29](#) [.long.](#) 221
[9.2.30](#) [.word.](#) 221
[9.2.31](#) [/](#) 221
[9.2.32](#) [/mod.](#) 222
[9.2.33](#) [llock.](#) 222
[9.2.34](#) [lunlock.](#) 222
[9.2.35](#) [2*.](#) 222
[9.2.36](#) [4-.](#) 222
[9.2.37](#) [4/.](#) 222
[9.2.38](#) [EC@...](#) 222
[9.2.39](#) [EW!.](#) 223
[9.2.40](#) [EW@...](#) 223

[9.2.59](#) [abs.](#) 225
[9.2.60](#) [andC!.](#) 226
[9.2.61](#) [build?.](#) 226
[9.2.62](#) [build_DevKernel](#) 226
[9.2.63](#) [cog?.](#) 226
[9.2.64](#) [cogdump.](#) 226
[9.2.65](#) [constant.](#) 226
[9.2.66](#) [decimal](#) 226
[9.2.67](#) [dump.](#) 227
[9.2.68](#) [edump.](#) 227
[9.2.69](#) [eereadpage.](#) 227
[9.2.70](#) [eewritepage.](#) 227
[9.2.71](#) [forget.](#) 227
[9.2.72](#) [free.](#) 228
[9.2.73](#) [ibound.](#) 228

[9.2.74](#) [invert.](#) 228
[9.2.75](#) [io>cogchan.](#) 228
[9.2.76](#) [j](#) 228
[9.2.77](#) [lasti?.](#) 228
[9.2.78](#) [lock?.](#) 228
[9.2.79](#) [onreset.](#) 229
[9.2.80](#) [pfa?.](#) 229
[9.2.81](#) [pinhi](#) 229
[9.2.82](#) [pinin.](#) 229
[9.2.83](#) [pinlo.](#) 229
[9.2.84](#) [pinout.](#) 229
[9.2.85](#) [px.](#) 229
[9.2.86](#) [px?.](#) 230
[9.2.87](#) [rev.](#) 230
[9.2.88](#) [revb.](#) 230
[9.2.89](#) [rnd.](#) 230
[9.2.90](#) [rndtf.](#) 230
[9.2.91](#) [rs?.](#) 230
[9.2.92](#) [saveforth.](#) 230
[9.2.93](#) [sc.](#) 231
[9.2.94](#) [serflags?.](#) 231
[9.2.95](#) [sersendbreak.](#) 231
[9.2.96](#) [sersetflags.](#) 231
[9.2.97](#) [sign.](#) 231
[9.2.98](#) [st?.](#) 231
[9.2.99](#) [u*/.](#) 232
[9.2.100](#) [u*/mod.](#) 232
[9.2.101](#) [variable.](#) 232
[9.2.102](#) [waitcnt.](#) 232
[9.2.103](#) [waitpeq.](#) 232
[9.2.104](#) [waitpne.](#) 232
[9.2.105](#) [words.](#) 233

[9.3](#) [EEPROMKernel Word Set \(build_fsr, build_fswr\).](#) 233

[9.3.13](#) [build_fsr.](#) 237

- [9.3.14](#) [build_fswr](#). 237
- [9.3.15](#) [fsbot](#). 237
- [9.3.16](#) [fsclear](#). 237
- [9.3.17](#) [fsdrop](#). 237
- [9.3.18](#) [fsfree](#). 237
- [9.3.19](#) [fsload](#). 237
- [9.3.20](#) [fsls](#). 238
- [9.3.21](#) [fsps](#). 238
- [9.3.22](#) [fsread](#). 238
- [9.3.23](#) [fstop](#). 238
- [9.3.24](#) [fswrite](#). 238
- [9.3.25](#) [onboot](#). 238

[9.4](#) [SDKernel Word Set \(build_sd\)](#). 239

- [9.4.5](#) [.num.](#). 246

- [9.4.43](#) [a_shift](#). 255
- [9.4.44](#) [build_sd](#). 255
- [9.4.45](#) [cd](#). 255
- [9.4.46](#) [cd.](#). 255
- [9.4.47](#) [cd/](#). 255
- [9.4.48](#) [cog>mem.](#). 256
- [9.4.49](#) [cog>pad](#). 256
- [9.4.50](#) [cog>tbuf7](#). 256
- [9.4.51](#) [cwd](#). 256
- [9.4.52](#) [fcreate](#). 256
- [9.4.53](#) [fload](#). 256
- [9.4.54](#) [fread](#). 256
- [9.4.55](#) [fstat](#). 257
- [9.4.56](#) [fwrite](#). 257
- [9.4.57](#) [ls](#). 257
- [9.4.58](#) [mem>cog](#). 257
- [9.4.59](#) [mkdir](#). 257
- [9.4.60](#) [onboot](#). 257
- [9.4.61](#) [pad>cog](#). 258

[9.4.62](#) [sd_append](#). 258
[9.4.63](#) [sd_appendblk](#). 258
[9.4.64](#) [sd_blockread](#). 258
[9.4.65](#) [sd_blockwrite](#). 258
[9.4.66](#) [sd_cd](#). 258
[9.4.67](#) [sd_cd.](#). 259
[9.4.68](#) [sd_cogbuf](#). 259
[9.4.69](#) [sd_cogbufclr](#). 259
[9.4.70](#) [sd_createdir](#). 259
[9.4.71](#) [sd_createfile](#). 260
[9.4.72](#) [sd_cwd](#). 260
[9.4.73](#) [sd_find](#). 260
[9.4.74](#) [sd_init](#). 260
[9.4.75](#) [sd_load](#). 260
[9.4.76](#) [sd_loadblk](#). 261
[9.4.77](#) [sd_lock](#). 261
[9.4.78](#) [sd_ls](#). 261
[9.4.79](#) [sd_mount](#). 261
[9.4.80](#) [sd_read](#). 261
[9.4.81](#) [sd_readblk](#). 261
[9.4.82](#) [sd_stat](#). 262
[9.4.83](#) [sd_trunc](#). 262
[9.4.84](#) [sd_uninit](#). 262
[9.4.85](#) [sd_unlock](#). 262
[9.4.86](#) [sd_write](#). 262
[9.4.87](#) [tbuf>cog7](#). 263
[9.4.88](#) [v_currentdir](#). 263
[9.4.89](#) [v_sd_clk](#). 263
[9.4.90](#) [v_sd_di](#) 263
[9.4.91](#) [v_sd_do](#). 263
[9.4.92](#) [v_sdbase](#). 263

9.1 Core Word Set (build_BootKernel, build_BootOpt)

9.1.1

\# (n1 -- n2) divide n1 by base and convert the remainder to a char and append to the output

9.1.2 #>

\#> (n1 -- caddr) address of a counted string representing the output, NOT ANSI

9.1.3 #s

\#s (n1 -- 0) execute # until the remainder is 0

9.1.29 ' .

\' (-- addr) returns the execution token for the next name, if not found it returns 0

9.1.43 +

\+ (n1 n2 -- n1+n2) \ sum of n1 & n2

9.1.44 +loop

\

9.1.45 -

\- (n1 n2 -- n1-n2) \ subtracts n2 from n1

9.1.46 -1

\-1 or true, used frequently

9.1.47 .

\. (n1 --) prints the signed number on the top of the stack

9.1.48 ."

\

9.1.49 **...**

\ ... followed by a CR indicates the end of file by the file system write/update word, nop at the prompt

9.1.50 **.cstr**

\ .cstr (addr --) emit a counted string at addr

9.1.51 **.str**

\ .str (c-addr u1 --) emit u1 characters at c-addr

9.1.52 **.strname**

\ .strname (c-addr --) c-addr points to a forth name field, print the name

9.1.53 **0**

\ Word constant, 0 or false, used frequently

9.1.54 **0<**

\ 0< (n1 -- t/f) true if n1 < 0

9.1.55 **0<>**

\ 0<> (n1 -- t/f) true if n1 is not zero

9.1.56 **0=**

\ 0= (n1 -- t/f) true if n1 is zero

9.1.57 **0>**

\ 0> (n1 -- t/f) true if n1 > 0

9.1.58 **0>=**

\ 0>= (n1 -- t/f) true if n1 >= 0

9.1.59 **0branch**

\ 0branch (t/f --) \ branch if top of stack value is zero 16 bit branch offset follows,

\ -2 is to itself, +2 is next word

9.1.60 **1**

\ Word constant, 1, used frequently

9.1.61 **1+**

\ 1+ (n1 -- n1+1)

9.1.62 **1-**

`\ 1- (n1 -- n1-1)`

9.1.63 **2**

`\ Word constant, 2, used frequently`

9.1.64 **2+**

`\ 2+ (n1 -- n1+2)`

9.1.65 **2-**

`\ 2- (n1 -- n1-2)`

9.1.66 **2/**

`\ 2/ (n1 -- n1>>1) n1 is shifted arithmetically right 1 bit`

9.1.67 **2>r**

`\ 2>r (n1 n2 --) \ pop top 2 stack to RS`

9.1.68 **2drop**

`\ 2drop (n1 n2 --) drop top 2 items on the stack`

9.1.69 **2dup**

`\ 2dup (n1 n2 -- n1 n2 n1 n2) copy top 2 items on the stack`

9.1.70 **2lock**

`\`

9.1.71 **2unlock**

`\`

9.1.72 **3drop**

`\ 3drop (n1 n2 n3 --) drop top 3 items on the stack`

9.1.73 **4***

`\ 4* (n1 -- n1<<2) n1 is shifted logically left 2 bits`

9.1.74 **4+**

`\ 4+ (n1 -- n1+4)`

9.1.75 :

\

9.1.76 ;

\

9.1.77 <

\ < (n1 n2 -- t/f) \ flag is true if and only if n1 is less than n2

9.1.78 <#

\ <# (--) initialize the output area

9.1.79 <=

\ <= (n1 n2 -- t/f) true if n1 <= n2

9.1.80 <>

\ <> (x1 x2 -- flag) flag is true if and only if x1 is not bit-for-bit the same as x2.

9.1.81 =

\ = (n1 n2 -- t/f) \ compare top 2 32 bit stack values, true if they are equal

9.1.82 >

\ > (n1 n2 -- t/f) \ flag is true if and only if n1 is greater than n2

9.1.83 >=

\ >= (n1 n2 -- t/f) true if n1 >= n2

9.1.84 >con

\ >con (n1 --) disconnect the current cog, and connect the console to the cog n1

9.1.85 >in

\ >in (-- addr) access as a word, addr is the var the offset in characters from the start of the input buffer to

9.1.86 >m

\ >m (n1 -- n2) produce a 1 bit mask n2 for position n1

9.1.87 >out

\ >out (-- addr) access as a word, the offset to the current output byte

9.1.88 **>r**

\>r (n1 --) \ pop stack top to RS

9.1.89 **C!**

\C! (c1 addr --) \ store 8 bit value (c1) main memory at addr

9.1.90 **C@**

\C@ (addr -- c1) \ fetch 8 bit value at main memory addr

9.1.91 **C@++**

\C@++ (c-addr -- c-addr+1 c1) fetch the character and increment the address

9.1.92 **COG!**

\COG! (n1 addr --) \ store 32 bit value (n1) at cog addr

9.1.93 **COG@**

\COG@ (addr -- n1) \ fetch 32 bit value at cog addr

9.1.94 **ERR**

\ERR (n1 --) clear the input queue, set the error n1 and reset this cog

9.1.95 **L!**

\L! (n1 addr --) \ store 32 bit value (n1) at main memory addr

9.1.96 **L@**

\L@ (addr -- n1) \ fetch 32 bit value at main memory addr

9.1.97 **RS!**

\RS! (n1 n2 --) \ store n1 at the n2th position on the return stack, 0 is the top of stack

9.1.98 **RS@**

\RS@ (addr -- n1) \ fetch n1th value down the return stack, 0 is the top of stack

9.1.99 **ST!**

\ST! (n1 n2 --) \ store n1 at the n2th position on the stack, 0 is the top of stack

9.1.100 **ST@**

\ST@ (addr -- n1) \ fetch n1th value down the stack, 0 is the top of stack

9.1.101 **W!**

`\ W! (h1 addr --) \ store 16 bit value (h1) main memory at addr`

9.1.102 **W+!**

`\ W+! (n1 addr --) add n1 to the word contents of address`

9.1.103 **W@**

`\ W@ (addr -- h1) \ fetch 16 bit value at main memory addr`

9.1.104 **[if**

`\ [if xxx (flag --) if flag is 0, drop all characters until], [if should be the first only only chars on the line`

9.1.105 **[ifdef**

`\ [ifdef xxx (--) if xxx is not defined drop all characters until], [ifdef xxx should be the first only only chars on the line`

9.1.106 **[ifndef**

`\ [ifndef xxx (--) if xxx is defined drop all characters until], [ifndef xxx should be the first only only chars on the line`

9.1.107 ****

`\ \ (--) moves the parse pointer >in to the end of the line`

9.1.108 **]**

`\`

9.1.109 **_accept**

`\ _accept (-- +n2) collect padsize -2 characters or until eol, convert ctl chars to space,`

`\ pad with 1 space at start & end. For parsing ease, and for the length byte when we make cstrs`

9.1.144 **accept**

`\ accept (--) uses the pad and accepts up to padsize - 2`

9.1.145 **alignl**

`\ alignl (n1 -- n1) aligns n1 to a long (32 bit) boundary`

9.1.146 **alignw**

`\ alignw (n1 -- n1) aligns n1 to a halfword (16 bit) boundary`

9.1.147 allot

\ allot (n1 --) add n1 to here, allocates space on the data dictionary or release it

9.1.148 and

\ and (n1 n2 -- n1) \ bitwise n1 and n2

9.1.149 andn

\ andn (n1 n2 -- n1) \ bitwise n1 and inverted n2

9.1.150 andnC!

\ andnC! (c1 addr --) and inverse of c1 with the contents of address

9.1.151 asmlabel

\ asmlabel (x --) skip blanks parse the next word and create an assembler entry

9.1.152 base

\ base (-- addr) access as a word, the address of the base variable

9.1.153 begin

\

9.1.154 between

\ between (n1 n2 n3 -- t/f) true if n2 <= n1 <= n3

9.1.155 bl

\ This is space constant

9.1.156 bounds

\ bounds (x n -- x+n x)

9.1.157 branch

\ branch \ 16 bit branch offset follows - -2 is to itself, +2 is next word

9.1.158 build_BootKernel

\

9.1.159 build_BootOpt

\

9.1.160 **c"**

\ c" (-- c-addr) compiles the string delimited by ", runtime return the addr of the counted string **
valid only in that line

\ comiple time, address is not left on the stack

9.1.161 **c,**

\ c, (x --) allocate 1 byte in the dictionary and copy x to that location

9.1.162 **cappend**

\ cappend (c-addr1 c-addr2 --) addpend the cstr from c-addr1 to c-addr2

9.1.163 **cappendn**

\ cappendn (n cstr --) print the number n and append to cstr

9.1.164 **ccopy**

\ ccopy (c-addr1 c-addr2 --) Copy the cstr from c-addr1 to c-addr2

9.1.165 **ccreate**

\ ccreate (cstr --) create a dictionary entry

9.1.166 **cds**

\ cds (-- addr) access as a word, the display string for this cog

9.1.167 **checkdict**

\ checkdict (n --) make sure there are at least n bytes available in the dictionary

9.1.168 **clearkeys**

\ clearkeys (--) clear the input keys

9.1.169 **clkfreq**

\ clkfreq (-- u1) the system clock frequency

9.1.170 **cmove**

\ cmove (c-addr1 c-addr2 u --) If u is greater than zero, copy u consecutive characters from the data space starting

\ at c-addr1 to that starting at c-addr2, proceeding character-by-character from lower addresses to higher addresses.

9.1.171 **cnt**

\ cnt - address of the the global cnt register for this cog

9.1.172 **cogcds**

\ cogcds (n1 -- addr) the address of the display string for cog n1

9.1.173 **coghere**

\ coghere (-- addr) access as a word, the first unused register address in this cog

9.1.174 **cogid**

\ cogid (-- n1) return id of the current cog (0 - 7)

9.1.175 **cogio**

\ cogio (n -- addr) the address of the data area for cog n

9.1.176 **cogiochan**

\ cogiochan (n1 n2 -- addr) cog n1, channel n2 ->addr

9.1.177 **cognchan**

\ cognchan (n1 -- n2) number of io channels for cog n2

9.1.178 **cognumpad**

\ cognumpad (n1 -- addr) the address of numpad for cog n1

9.1.179 **cogpad**

\ cogpad (n1 -- addr) the address of pad for cog n1

9.1.180 **cogreset**

\ cogreset (n1 --) reset the forth cog

9.1.181 **cogstate**

\ cogstate (n1 -- addr) the address of state for cog n1

9.1.182 **cogstop**

\ cogstop (n --) stop cog n

9.1.183 **cogx**

\ cogx (cstr n --) execute cstr on cog n

9.1.184 **compile?**

\ compile? (-- t/f) true if we are in a compile

9.1.185 **cq**

`\ cq (-- addr)` returns the address of the counted string following this word and increments the IP past it

9.1.186 **cr**

`\ cr (--)` emits a carriage return

9.1.187 **create**

`\ create (--)` skip blanks parse the next word and create a dictionary entry

9.1.188 **cstr=**

`\ cstr= (cstr1 cstr2 -- t/f)` case sensitive compare

9.1.189 **delms**

`\ delms (n1 --)` delay n1 milli-seconds for 80Mhz h68DB max

9.1.190 **dictend**

`\ dictend` - access as a word, the end of the total dictionary space

9.1.191 **dira**

`\ dira` - address of the the dira register for this cog

9.1.192 **do**

`\`

9.1.193 **doconl**

`\ doconl (-- n1)` \ push a 32 bit constant which follows the stack - implicit a_exit

9.1.194 **doconw**

`\ doconw (-- h1)` \ push 16 bit constant which follows on the stack - implicit a_exit

9.1.195 **doloop**

`\`

9.1.196 **dothen**

`\`

9.1.197 **dovarl**

`\ dovarl (-- addr)` \ push address of 32 bit variable which follows the stack - implicit a_exit

9.1.198 **dovarw**

`\ dovarw (-- addr)` \ push address of 16 bit variable which follows on the stack - implicit a_exit

9.1.199 **dq**

`\ dq (--)` emit a counted string at the ip, and increment the ip past it and word alignw it

9.1.200 **drop**

`\ drop (n1 --)` \ drop the value on the top of the stack

9.1.201 **dup**

`\ dup (n1 -- n1 n1)`

9.1.202 **else**

`\`

9.1.203 **emit**

`\ emit (c1 --)` emit the char on the stack

9.1.204 **exec**

`\ exec (--)` marks last entry as an eXecute word, executes always

9.1.205 **execute**

`\ execute (addr --)` execute the word - pfa address is on the stack

9.1.206 **execword**

`\ execword (-- addr)` a long, an area where the current word for execute is stored

9.1.207 **exit**

`\ exit` the current forth word, and back to the caller

9.1.208 **femit?**

`\ femit? (c1 -- t/f)` true if the output emitted a char, a fast non blocking emit

9.1.209 **fill**

`\ fill (c-addr u char --)` fill the memory with char

9.1.210 **find**

`\ find (c-addr -- c-addr 0 | xt 2 | xt 1 | xt -1)` c-addr is a counted string, 0 - not found, 2 eXecute word,

`\ 1 immediate word, -1 word NOT ANSI`

9.1.211 **fisnumber**

`\ fisnumber (--)` dummy routines for indirection when float package is loaded

9.1.212 **fkey?**

`\ fkey? (-- c1 t/f)` fast nonblocking key routine, true if c1 is a valid key

9.1.213 **fl**

`\ fl (--)` buffer the input and route to a free cog

9.1.214 **fl_in**

`\ fl_in` - pointer to next character for input

9.1.215 **fl_lock**

`\` to ensure one fast load at a time

9.1.216 **fnumber**

`\ fnumber (c-addr len -- n1)` convert string to a signed number

`\` dummy routines for indirection when float package is loaded

9.1.217 **forthentry**

`\ forthentry (--)` marks last entry as a forth word

9.1.218 **freedict**

`\ freedict (--)` free the forth dictionary

9.1.219 **fstart**

`\` this word is what the IP is set to on a reboot or a reset

`\ fstart (--)` the start word

9.1.220 **here**

`\ here` - access as a word, the current end of the dictionary space being used

9.1.221 **herelal**

`\ herelal (--)` alignw contents of here to a long boundary, 4 byte boundary

9.1.222 **herewal**

`\ herewal (--)` align contents of here to a word boundary, 2 byte boundary

9.1.223 **hex**

\ hex (--) set the base for hexadecimal

9.1.224 **hubopf**

\ hubopf (n1 n2 -- t/f) n2 specifies which hubop (0 - 7), t/f is the 'c' flag is set from the hubop

9.1.225 **hubopr**

\ hubopr (n1 n2 -- n3) n2 specifies which hubop (0 - 7), n1 is the source datcog, n3 is returned,

9.1.226 **i**

\ i (-- n1) the most current loop counter

9.1.227 **if**

\

9.1.228 **immediate**

\ immediate (--) marks last entry as an immediate word

9.1.229 **ina**

\ ina - address of the the ina register for this cog

9.1.230 **init_coghere**

\ init_coghere (--) This word can be replaced to the assembler optimizations, initializes the coghere wvariable

9.1.231 **initcon**

\ initcon (--) initialize the default serial console on this cog

9.1.232 **interpret**

\ interpret (--) the main interpreter loop

9.1.233 **interpretpad**

\ interpretpad (--) interpret the contents of the pad

9.1.234 **io**

\ io (-- addr) the address of the io channel for the cog

9.1.235 **ioconn**

\ ioconn (n1 n2 --) connect the 2 cogs, disconnect them from other cogs first

9.1.236 **iodis**

`\ iodis (n1 --)` cogid to disconnect, disconnect this cog and the cog it is connected to

9.1.237 **iolink**

`\ iolink (n1 n2 --)` links the 2 cogs, output of n1 -> input of n2, output of n2 -> old output of n1

9.1.238 **iounlink**

`\ iounlink (n1 --)` unlinks the cog n1

9.1.239 **isdigit**

`\ isdigit (c1 -- t/f)` true if is it a valid digit according to base

9.1.240 **isnamechar**

`\ isnamechar (c1 -- t/f)` true if c1 is a valif name char > \$20 < \$7F

9.1.241 **isnumber**

`\ isnumber (c-addr len -- t/f)` true if the string is numeric

9.1.242 **isunumber**

`\ isunumber (c-addr len -- t/f)` true if the string is numeric

9.1.243 **key**

`\ key (-- c1)` get a key

9.1.244 **l,**

`\ l, (x --)` allocate 1 long, 4 bytes in the dictionary and copy x to that location

9.1.245 **l>w**

`\ l>w (n1n2 -- n1 n2)` break into 16 bits

9.1.246 **lasterr**

`\ lasterr (-- addr)` access as a char, an errorcode, set by ERR, and the kernel - if 0 - no error

9.1.247 **lastnfa**

`\ lastnfa (-- addr)` gets the last NFA

9.1.248 **leave**

`\ leave (--)` exits at the next loop or +loop, i is placed to the max loop value

9.1.249 **litl**

`\ litl (-- n1) \ push a 32 bit literal on the stack`

9.1.250 **litw**

`\ litw (-- h1) \ push a 16 bit literal on the stack`

9.1.251 **lock**

`\ lock (lock# --)`

9.1.252 **lockdict**

`\ lockdict (--) lock the forth dictionary`

9.1.253 **loop**

`\`

9.1.254 **lshift**

`\ lshift (n1 n2 -- n3) \ n3 = n1 shifted left n2 bits`

9.1.255 **lxasm**

`\ lxasm (addr --) load the assembler at addr and execute it`

9.1.256 **max**

`\ max (n1 n2 -- n1) \ signed max of top 2 stack values`

9.1.257 **memend**

`\ memend - access as a word, the end of memory available to PropForth`

9.1.258 **min**

`\ min (n1 n2 -- n1) \ signed min of top 2 stack values`

9.1.259 **name=**

`\ name= (cstr1 cstr2 -- t/f) case sensitive compare`

9.1.260 **namecopy**

`\ namecopy (c-addr1 c-addr2 --) Copy the name from c-addr1 to c-addr2`

9.1.261 **namelen**

`\ namelen (c-addr -- c-addr+1 len) returns c-addr+1 and the length of the name at c-addr`

9.1.262 **namemax**

\ the maximum name length allowed must be 1F

9.1.263 **negate**

\ negate (n1 -- 0-n1) the negative of n1

9.1.264 **nextword**

\ nextword (--) increment >in past current counted string

9.1.265 **nfa>lfa**

\ nfa>lfa (addr -- addr) go from the nfa (name field address) to the lfa (link field address)

9.1.266 **nfa>next**

\ nfa>next (addr -- addr) go from the current nfa to the prev nfa in the dictionary

9.1.267 **nfa>pfa**

\ nfa>pfa (addr -- addr) go from the nfa (name field address) to the pfa (parameter field address)

9.1.268 **nfcog**

\ nfcog (-- n) returns the next valid free forth cog

9.1.269 **nip**

\ nip (x1 x2 -- x2) delete the item x1 from the stack

9.1.270 **npfx**

\ npfx (c-addr1 c-addr2 -- t/f) -1 if c-addr2 is prefix of c-addr1, 0 otherwise

9.1.271 **number**

\ number (c-addr len -- n1) convert string to a signed number

9.1.272 **numpad**

\ numpad (-- addr) the of the area used by the numeric output routines, can be used carefully by other code

9.1.273 **numpadsiz**

\ the size of the numpad, 34 bytes the largest number we can deal with is 33 digits

9.1.274 **onboot**

\ onboot (n1 -- n1) n1 - reset error code

9.1.275 onreset

\ onreset (n1 --) n1 - reset error code

9.1.276 or

\ or (n1 n2 -- n1_or_n2) \ bitwise or

9.1.277 orC!

\ orC! (c1 addr --) or c1 with the contents of address

9.1.278 orInfa

\ orInfa (c1 --) ors c1 with the nfa length of the last name field entered

9.1.279 outa

\ outa - address of the the outa register for this cog

9.1.280 over

\ over (n1 n2 -- n1 n2 n1) \ duplicate 2 value down on the stack to the top of the stack

9.1.281 pad

\ pad (-- addr) access as bytes, or words and long, the address of the pad area - used by accept for keyboard input,

\ can be used carefully by other code

9.1.282 pad>in

\ pad>in (-- addr) addr is the address to the start of the parse area.

9.1.283 pad>out

\ pad>out (-- addr) addr is the address to the the current output byte

9.1.284 padbl

\ padbl (--) fills this cogs pad with blanks

9.1.285 padsiz

\ the size of the pad area, 128 bytes

9.1.286 par

\ This is the par register, always initialized to point to this cogs section of cogdata

9.1.287 parse

\ parse (c1 -- +n2) parse the word delimited by c1, or the end of buffer is reached, n2 is the length >in

is the offset

\ in the pad of the start of the parsed word REPLACED IN VERSION 4.3 2011FEB24

9.1.288 parsebl

\ parsebl (-- t/f) parse the next word in the pad delimited by blank, true if there is a word

9.1.289 parsenw

\ parsenw (-- cstr) parse and move to the next word, str ptr is zero if there is no next word

9.1.290 parseword

\ parseword (c1 -- +n2) skip blanks, and parse the following word delimited by c1, update to be a counted string in

\ the pad

9.1.291 pfa>nfa

\ pfa>nfa (addr -- addr) gets the name field address (nfa) for a parameter field address (pfa)

9.1.292 prop

\ prop - access as a word, the address of the string identifier of this prop

9.1.293 propid

\ propid - access as a word, the numeric id of this prop

9.1.294 r>

\ r> (-- n1) \ pop top of RS to stack

9.1.295 reboot

\ reboot (--) reboot the propellor chip

9.1.296 reset

\ reset (--) reset this cog

9.1.297 rot

: rot h2 ST@ h2 ST@ h2 ST@ 3 ST! 3 ST! 0 ST! ;

9.1.298 rot2

\ rot2 (x1 x2 x3 -- x3 x1 x2)

9.1.299 **rshift**

\ rshift (n1 n2 -- n3) \ n3 = n1 shifted right logically n2 bits

9.1.300 **serial**

\ serial (n1 n2 n3 --)

\ n1 - tx pin

\ n2 - rx pin

\ n3 - baud rate

\ h00 - h04 -- io channel

\ h04 - h84 -- the receive buffer

\ h84 - hC4 -- the transmit buffer

\ hC4 - breaklength (long), if this long is not zero the driver will transmit a break breaklength cycles,

\ the minmum lenght is 16 cycles, at 80 Mhz this is 200 nanoSeconds

\ hC8 - flags (long)

\ h_0000_0001 - if this bit is 0, CR is transmitted as CR LF

\ - if this bit is 1, CR is transmitted as CR

9.1.301 **seti**

\ seti (n1 --) set the most current loop counter

9.1.302 **skipbl**

\ skipbl (--) increment >in past blanks or until it equals padsize

9.1.303 **space**

\ space (--) emits a space

9.1.304 **spaces**

\ spaces (n --) emit n spaces

9.1.305 **state**

\ state (-- addr) access as a char

\ bit 0 - 0 - interpret mode / 1 - forth compile mode

\ bit 1 - 0 - prompts and errors on / 1 - prompts and errors off

\ bit 2 - 0 - Other / 1 - PropForth cog

\ bit 3 - 0 - accept echos chars on / 1 - accept echos chars off

\ bit 4 - 0 - accept echos line off / 1 - accept echos line on

\ bit 5 - 7 - number of io channels - 1

9.1.306 **swap**

\ swap (n1 n2 -- n2 n1) \ swap top 2 stack values

9.1.307 **t0**

\ these are temporary variables, and by convention are only used within a word

\ caution, make sure you know what words you are calling

\ t0 - access as a word, temp variable

9.1.308 **t1**

\ these are temporary variables, and by convention are only used within a word

\ caution, make sure you know what words you are calling

\ t1 - access as a word, temp variable

9.1.309 **tbuf**

\ these are temporary variables, and by convention are only used within a word

\ caution, make sure you know what words you are calling

\ tbuf - access as a chars, words, or longs. Temp array of 32 bytes

9.1.310 **then**

\

9.1.311 **thens**

\

9.1.312 **tochar**

\ tochar (n1 -- c1) convert c1 to a char

9.1.313 **todigit**

`\ todigit (c1 -- n1)` converts character to a number

9.1.314 **tuck**

`\ tuck (x1 x2 -- x2 x1 x2)`

9.1.315 **u***

`\ u* (u1 u2 -- u1*u2)` $u1*u2$ must be a valid 32 bit unsigned number

9.1.316 **u.**

`\ u. (n1 --)` prints the unsigned number on the top of the stack

9.1.317 **u/**

`\ u/ (u1 u2 -- u1/u2)` $u1$ divided by $u2$

9.1.318 **u/mod**

`\ u/mod (u1 u2 -- remainder quotient)` both remainder and quotient are 32 bit unsigned numbers

9.1.319 **u>=**

`\ u>= (u1 u2 -- t/f)` \ flag is true if and only if $u1$ is greater or equal to than $u2$

9.1.320 **um***

`\ um* (u1 u2 -- u1*u2L u1*u2H)` \ unsigned 32bit * 32bit -- 64bit result

9.1.321 **um/mod**

`\ um/mod (u1lo u1hi u2 -- remainder quotient)` \ unsigned divide & mod $u1$ divided by $u2$

9.1.322 **unlock**

`\ unlock (lock# --)`

9.1.323 **unlockall**

`\ unlockall (--)` unlocks everything this cog has locked

9.1.324 **until**

`\`

9.1.325 **unnumber**

`\ unnumber (c-addr len -- u1)` convert string to an unsigned number

9.1.326 **version**

\ **version** - access as a word, the address of the string version of PropForth

9.1.327 **w,**

\ **w,** (x --) allocate 1 halfword 2 bytes in the dictionary and copy x to that location

9.1.328 **w>l**

\ **w>l** (n1 n2 -- n1n2) consider only lower 16 bits of each source word

9.1.329 **wconstant**

\ **wconstant** (x --) skip blanks parse the next word and create a constant, allocate a word, 2 bytes

9.1.330 **wlastnfa**

\ **wlastnfa** - access as a word, the address of the last nfa

9.1.331 **wvariable**

\ **wvariable** (--) skip blanks parse the next word and create a variable, allocate a word, 2 bytes

9.1.332 **xisnumber**

\ **xisnumber** (c-addr len -- t/f) true if the string is numeric

9.1.333 **xnumber**

\ **xnumber** (c-addr len -- n1) convert string to a signed number

9.1.334 **xor**

\ \ **xor** (n1 n2 -- n1_xor_n2) \ bitwise xor

9.1.335 **{**

\ { (--) discard all the characters between { and }

\ open brace **MUST** be the first and only character on a new line, the close brace must be on another line

9.1.336 **}**

\ } (--)

9.2 DevKernel Word Set (BuildDevKernel)

\

9.2.1 #C

\#C (c1 --) prepend the character c1 to the number currently being formatted

9.2.15 *

* (n1 n2 -- n1*n2) n1 multiplied by n2

9.2.16 */

*/ (n1 n2 n3 -- n4) n4 = (n1*n2)/n3. Uses a 64bit intermediate result.

9.2.17 */mod

*/mod (n1 n2 n3 -- n4 n5) n5 = (n1*n2)/n3, n4 is the remainder. Uses a 64bit intermediate result.

9.2.18 .byte

\.byte (n1 --) output a byte

9.2.19 .cogch

\.cogch (n1 n2 --) print as x(y)

9.2.20 .con

\.con (n1 --) print n1 to the console, non blocking, may get overwritten

9.2.21 .conbyte

\.con (c1 --) print byte c1 to the console, non blocking, may get overwritten

9.2.22 .concr

\.concr (--) emit a cr to the console, non blocking, may get overwritten

9.2.23 .conctr

\.conctr (cstr --) emit cstr to console, non blocking, may get overwritten

9.2.24 .conemit

\.conemit (c1 --) emit cr to console, non blocking, may get overwritten

9.2.25 **.conlong**

\ **.conlong** (n1 --) print n1 to console, non blocking, may get overwritten

9.2.26 **.const?**

\ **.const?** (--) prints out the stack to the console, non blocking, may get overwritten

9.2.27 **.conwait**

\ **.conwait** (--) if con is not ready for a char, wait long enough for a char to transmit

9.2.28 **.conword**

\ **.conword** (n1 --) print n1 to console, non blocking, may get overwritten

9.2.29 **.long**

\ **.long** (n1 --) output a long

9.2.30 **.word**

\ **.word** (n1 --) output a word

9.2.31 **/**

\ / (n1 n2 -- n1/n2) n1 divided by n2

9.2.32 **/mod**

\ /mod (n1 n2 -- n3 n4) \ signed divide & mod n4 = n1/n2, n3 is the remainder

9.2.33 **1lock**

\ **1lock**(--) equivalent to 1 lock

9.2.34 **1unlock**

\ **1unlock**(--) equivalent to 1 unlock

9.2.35 **2***

\ **2*** (n1 -- n1<<1) n2 is shifted logically left 1 bit

9.2.36 **4-**

\ **4-** (n1 -- n1-4)

9.2.37 **4/**

\ **4/** (n1 -- n1>>2) n2 is shifted arithmetically right 2 bits

9.2.38 **EC@**

\ **EC@** (eeAddr -- c1) read a byte from the eeprom

9.2.39 **EW!**

\ **EW!** (n1 eeAddr --) write n1 to the eeprom

9.2.40 **EW@**

\ **EW@** (eeAddr -- n1) read a word from the eeprom

9.2.59 **abs**

\ **abs** (n1 -- abs_n1) absolute value of n1

9.2.60 **andC!**

\ **andC!** (c1 addr --) and c1 with the contents of address

9.2.61 **build?**

\ **build?** (--) print out build information

9.2.62 **build_DevKernel**

\ **Word constant**

9.2.63 **cog?**

\ **cog?** (--) print out cog information

9.2.64 **cogdump**

\ **cogdump** (adr cnt --) dump cog memory

9.2.65 **constant**

\ **constant** (x --) skip blanks parse the next word and create a constant, allocate a long, 4 bytes

9.2.66 **decimal**

\ **decimal** (--) set the base for decimal

9.2.67 **dump**

\ **dump** (adr cnt --) dump main memory, uses tbuf

9.2.68 **edump**

\ **edump** (adr cnt --) dump eeprom, uses tbuf

9.2.69 eereadpage

\ the eereadpage and eewritePage words assume the eeprom are 64kx8 and will address up to

\ 8 sequential eeproms

\ eereadpage (eeAddr addr u -- t/f) return true if there was an error, use lock 1

9.2.70 eewritepage

\ the eereadpage and eewritePage words assume the eeprom are 64kx8 and will address up to

\ 8 sequential eeproms

\ eewritepage (eeAddr addr u -- t/f) return true if there was an error, use lock 1

9.2.71 forget

\ forget (--) wind the dictionary back to the word which follows - caution

9.2.72 free

\ free (--) display free main bytes and current cog longs

9.2.73 ibound

\ ibound (-- n1) the upper bound of i

9.2.74 invert

\ invert (n1 -- n2) bitwise invert n1

9.2.75 io>cogchan

\ io>cogchan (addr -- n1 n2) addr -> n1 cogid, n2 channel

9.2.76 j

\ j (-- n1) the second most current loop counter

9.2.77 lasti?

\ lasti? (-- t/f) true if this is the last value of i in this loop, assume an increment of 1

9.2.78 lock?

\ lock? (--) displays the status of the locks

9.2.79 onreset

\ onreset (n1 --) reset message and error n1 on a reset, echo to the console

9.2.80 pfa?

\ pfa? (addr -- t/f) true if addr is a pfa

9.2.81 pinhi

\ pinhi (n1 --) set pin # n1 to hi

9.2.82 pinin

\ pinin (n1 --) set pin # n1 to an input

9.2.83 pinlo

\ pinlo (n1 --) set pin # n1 to lo

9.2.84 pinout

\ pinout (n1 --) set pin # n1 to an output

9.2.85 px

\ px (t/f n1 --) set pin # n1 to h - true or l false

9.2.86 px?

\ px? (n1 -- t/f) true if pin n1 is hi

9.2.87 rev

\ rev (n1 n2 -- n3) n3 is n1 with the lower 32-n2 bits reversed and the upper bite cleared

9.2.88 revb

\ revb (n1 -- n2) n2 is the lower 8 bits of n1 reversed

9.2.89 rnd

\ rnd (-- n1) n1 is a random number from 00 - FF

9.2.90 rndtf

\ rndtf (-- t/f) true or false randomly

9.2.91 rs?

\ rs? (--) prints out the return stack

9.2.92 saveforth

\ saveforth(--) write the running image to eeprom UPDATES THE CURRENT VERSION STR

9.2.93 **sc**

`\ sc (--)` clears the stack

9.2.94 **serflags?**

`\ serflags? (n1 -- n2)` n2 are the serial flags for the serial driver running on cog n1

9.2.95 **sersendbreak**

`\ sersendbreak (n2 n1 --)` for the serial driver running on cog n1, send a break of n2 clock cycles

9.2.96 **serflags**

`\ serflags (n2 n1 -- 0)` for the serial driver running on cog n1, set the flags to n2

9.2.97 **sign**

`\ sign (n1 n2 -- n3)` n3 is the xor of the sign bits of n1 and n2

9.2.98 **st?**

`\ st? (--)` prints out the stack

9.2.99 **u*/**

`\ u*/ (u1 u2 u3 -- u4)` u4 = (u1*u2)/u3 Uses a 64bit intermediate result.

9.2.100 **u*/mod**

`\ u*/mod (u1 u2 u3 -- u4 u5)` u5 = (u1*u2)/u3, u4 is the remainder. Uses a 64bit intermediate result.

9.2.101 **variable**

`\ variable (--)` skip blanks parse the next word and create a variable, allocate a long, 4 bytes

9.2.102 **waitcnt**

`\ waitcnt (n1 n2 -- n1)` \ wait until n1, add n2 to n1

9.2.103 **waitpeq**

`\ waitpeq (n1 n2 --)` \ wait until state n1 is equal to inanded with n2

9.2.104 **waitpne**

`\ waitpne (n1 n2 --)` \ wait until state n1 is not equal to inanded with n2

9.2.105 **words**

`\ words (--)` prints the words in the forth dictionary, if the pad has another string following, with that prefix

9.3 **EEpromKernel Word Set (build_fsr, build_fswr)**

\ A very simple file system for eeprom. The goal is not a general file system, but a place to put text

\ (or code) in eeprom so it can be dynamically loaded by propforth

\

\ the eeprom area start is defined by fsbot and the top is defined by fstop

\ The files are in eeprom memory as such:

\ 2 bytes - length of the contents of the file

\ 1 byte - length of the file name (this is a normal counted string, counted strings can be up

\ 255 bytes in length, the length here is limited for space reasons)

\ 1 - 31 bytes - the file name

\ 0 - 65534 bytes - the contents of the file

\

\ the last file has a length of 65535 (0hFFFF)

\

\ the start of every file is aligned with eeprom pages, for efficient read and write, this is 64 bytes

\

\ Status: 2010NOV24 Beta

\

\ 2011FEB03 - fix to align page reads to page addresses so crossing eeproms does not cause a problem
_fsread

\

\ 2011MAY31 - stable, reformat, updated error codes, added error message for file not found, added
RO option

\

\ main routines

\

\ fsload filename - reads filename and directs it to the next free forth cog, every additional nested fsload

\ requires an additional free cog

\ fsread filename - reads the file and echos it directly to the terminal

\ fswrite filename - writes the file to the filesystem - takes input from the input until ...\0h0d is encountered

\ - ie 3 dots followed by a carriage return

\ fsls - lists the files

\ fsclear - erases all files

\ fsdrop - erases the last file

9.3.13 build_fsr

\ Word constant

9.3.14 build_fswr

\ Word constant

9.3.15 fsbot

\ Long constant - the start adress in eeprom for the file system

9.3.16 fsclear

\ fsclear (--) erase all files and initialize the eeprom file system

9.3.17 fsdrop

\ fsdrop (--) deletes last file

9.3.18 fsfree

\ fsfree (--) print out free bytes in the eeprom file system

9.3.19 fsload

\ fsload filename (--) send the file to the next free forth cog

9.3.20 **fsls**

\ **fsls (--) list the files**

9.3.21 **fsps**

\ **fsps** - word constant, the page size set to 64, a page size which should work with 32kx8 & 64kx8 eeproms and should work with larger as well. MUST BE A POWER OF 2

9.3.22 **fsread**

\ **fsread filename (--) prints filename to the output**

9.3.23 **fstop**

\ **Long constant** - the end address in the eeprom for the file system

9.3.24 **fswrite**

\ **fswrite filename (--) writes a file until ... followed immediately by a cr is encountered**

9.3.25 **onboot**

\ **onboot (n1 -- n1) execute file boot.f if it exists**

9.4 **SDKernel Word Set (build_sd)**

sdfs is meant to be a very simple fast file system on top of sd_driver.

The design criteria are around small code size and speed, as opposed to generality and versatility.

Main concepts:

1. A file system occupies n contiguous 512 byte blocks on the SD card. If it is desired that the card should be FAT32 formatted, it should be possible to format card, create a very large file, figure out which blocks the file occupies, and mount sdfs to use those blocks.

This means sdfs could be manipulated on a pc by writing some code. This is NOT a goal of the initial implementation.

The file system must not start at block 0, block 1 is the first valid block number.

This also means multiple file systems can be defined on one sd card, useful for isolating functions like logging.

An sd card can be partitioned into multiple "disks", each which can contain one file system. There is no partition table or disk routines, when a filesystem is created, the start and end inherently defines the partition.

2. Files are 2 - n contiguous blocks, the maximum file size is 2Gig, (max positive 32 bit integer.) When a file is created, the space that is allocated to the file (the space is allocated as blocks of 512 bytes) is the maximum size the file can grow to. This trades space efficiency for a very simple and fast allocation.

3. File names must be 1 to 26 characters in length, and can only contain characters 0h30 - 0h7D. There are no other restrictions on file names. It is recommended that names do not use special characters. The reason for this is mapping urls to file names gets more complicated.

4. There is directory support, a directory is a fixed length file, whose name ends with a / There is no other differentiator. A directory can contain up to 2048 entries. A simple hashing mechanism makes navigation quick. (This hash function should be tested more thoroughly for at some point, but initial testing seems ok.) This optimizes for opening files, the result is that to list a directory the whole directory must be traversed, so directory "listing" is slower.

However, assuming the hash function performs reasonably, finding a file, and opening the file file be fast, and not slow down as there are more entries in the directory.

The root directory is /

Pathnames are simple concatenated directory and file names.

The maximum total length of a path name is 120 characters, this should make mapping urls to files very easy, and allow the use of the pad for parsing and file name manipulation.

To access a file, or directory, the current directory must be set to the parent of the file or directory. File access is only in that directory. This means no relative navigation or fully qualified file names. Reasons are 2 fold, 1 simplicity, 2 security. This will become evident when the http server is using the file system.

5. There is no limit on directory depth other than that which is imposed by the maximum path name length. The deeper a file is in the structure, the longer it will take to navigate to it.

6. Each file has a header block, immediately followed by all the data blocks.

7. The header block number is used by routines to reference files and directories.

8. Block numbers are absolute, this makes debugging, repair, and verification much simpler.

Unfortunately it means you cannot easily move the file system around.

9. The main interface routines to sdfs are:

```
\_sd_CrEaTe ( n1 n2 -- ) n1 - starting block, n2 - last block + 1, CREATE a file system, WIPES OUT DATA
```

```
\sd_mount ( n1 -- ) mount the file system, n1 - the starting block of the file system,
```

```
\ the file system must be mounted before it is used
```

\sd_createdir (cstr -- n1) cstr is the name of the directory to create, n1 is the header block
\ of the directory. If this directory already exists, it returns the root block of the existing
\ directory

\sd_cd.. (--) make the parent directory the current directory

\sd_cd (cstr --) make cstr the current directory, if it does not exist, nothing happens
\ the directory name in cstr must have a / at the end of it

\sd_cwd (--) get the pathname of the current directory and copy it to pad

\sd_ls (--) list the current directory

\sd_createfile (cstr n1 -- n2) allocate n1 blocks, this includes the header block

\sd_find (filename -- blocknumber/0) search for filename in the current directory

\sd_stat (filename --) prints stats for the file

\sd_write (numblocks filename --) writes a new or existing file until ... followed immediately by a cr
is encountered

\ if the file exists, writing will be truncated to the existing maximum file size allocated when the file
was created

\sd_trunc (length filename --) sets the number of bytes used in the file to length

\sd_appendblk(addr size blk --)

\sd_append(addr size filename --)

\sd_readblk (n1 --) n1 - header block number of file,

\ read the file and emit the char

\sd_read (filename --) read the file and emit the chars

\sd_loadblk (n1 --) n1 - the header block for the file loads the file,

\ routes the file to the next free forth cog

\sd_load (cstr --) loads the file, routes the file to the next free forth cog

\

\ These are provided for command line convenience

\

\ls (--)

\cd dirname (--)

\cd.. (--)

\cd/ (--)

\cwd (--) print the current directory

\mkdir dirname (--)

`\ fread filename (--)`

`\ fcreate filename (numblocks_to_allocate --)`

`\ fwrite filename (numblocks_to_allocate --)`

`\ fstat filename (--)`

9.4.5 .num

`\ .num (n1 --) print n1 as a fixed format number`

9.4.43 a_shift

`\ Internal word, used to invoke the assembler routines`

9.4.44 build_sd

`\ Word constant`

9.4.45 cd

`\ cd dirname (--)`

9.4.46 cd..

`\ cd.. (--)`

9.4.47 cd/

`\ cd/ (--)`

9.4.48 cog>mem

`\ cog>mem (memaddr cogaddr numlongs --) memaddr must be long aligned`

9.4.49 cog>pad

`\ cog>pad (n1 --) the cog address to start reading 32 longs`

9.4.50 cog>tbuf7

`\ tbuf>cog7 (n1 --) the cog address to start writing 7 longs`

9.4.51 cwd

`\ cwd (--) print the current directory`

9.4.52 **fcreate**

`\fcreate filename (numblocks_to_allocate --)`

9.4.53 **fload**

`\fload filename (--)` load the file using the next free cog

9.4.54 **fread**

`\fread filename (--)` print the file to the output

9.4.55 **fstat**

`\fstat filename (--)` print the file stats to the output

9.4.56 **fwrite**

`\fwrite filename (numblocks_to_allocate --)` allocate blocks and write until a ... followed by a cr is encountered

9.4.57 **ls**

`\ls (--)` list the files

9.4.58 **mem>cog**

`\mem>cog (memaddr cogaddr numlongs --)` memaddr must be long aligned

9.4.59 **mkdir**

`\mkdir dirname (--)` create the directory

9.4.60 **onboot**

`\onboot (n1 -- n1)` load the file sd_boot.f

9.4.61 **pad>cog**

`\pad>cog (n1 --)` the cog address to start writing 32 longs

9.4.62 **sd_append**

`\sd_append(addr size filename --)` append buffer of size to the file

9.4.63 **sd_appendblk**

`\sd_appendblk(addr size headerblk --)` append buffer of size to the file at headerblk

9.4.64 **sd_blockread**

`\sd_blockread (n1 --)` n1 - the block number. Reads a 512 byte block into _sd_buf

9.4.65 **sd_blockwrite**

\sd_blockwrite (n1 --) n1 - the block number. Writes 512 byte block from _sd_buf

9.4.66 **sd_cd**

\sd_cd (cstr --) make cstr the current directory, if it does not exists, nothing happens

\ the directory name in cstr must have a / at the end of it

9.4.67 **sd_cd..**

\sd_cd.. (--) make the parent directory the current directory

9.4.68 **sd_cogbuf**

\ set the data area for the buffer at the end of the assembler code, the allocation is done in sd_init

\sd_cogbuf is the beginning of the buffer

9.4.69 **sd_cogbufclr**

\sd_cogbufclr (--) initialize the buffer to zeros

9.4.70 **sd_createdir**

\sd_createdir (cstr -- n1) cstr is the name of the directory to create, n1 is the header block

\ of the directory. If this directory already exists, it returns the root block of the existing

\ directory

9.4.71 **sd_createfile**

\sd_createfile (cstr n1 -- n2) allocate n1 blocks, this includes the header block,

\ create a directory entry, and write the file header,

\ n2 is the block number of the file header

9.4.72 **sd_cwd**

\sd_cwd (--) get the pathname of the current directory and copy it to pad

9.4.73 **sd_find**

\sd_find (filename -- blocknumber/0) search for filename in the current directory

9.4.74 **sd_init**

\sd_init (--) call for every cog using the sd card

9.4.75 **sd_load**

\sd_load (cstr --) loads the file, routes the file to the next free forth cog

9.4.76 **sd_loadblk**

\sd_loadblk (n1 --) n1 - the header block for the file loads the file,

\ routes the file to the next free forth cog

9.4.77 **sd_lock**

\sd_lock (--) lock the sd card so no one else can use it

9.4.78 **sd_ls**

\sd_ls (--) list the current directory

9.4.79 **sd_mount**

\sd_mount (n1 --) mount the file system, n1 - the starting block of the file system,

\ the file system must be mounted before it is used

9.4.80 **sd_read**

\sd_read (filename --) read the file and emit the chars

9.4.81 **sd_readblk**

\

\sd_readblk (n1 --) n1 - header block number of file,

\ read the file and emit the chars

9.4.82 **sd_stat**

\sd_stat (filename --) prints stats for the file

9.4.83 **sd_trunc**

\sd_trunc (length filename --) sets the number of bytes used in the file to length

9.4.84 **sd_uninit**

\sd_uninit (--) "releases" the cog memory

9.4.85 **sd_unlock**

\sd_unlock (--) unlock the sd card

9.4.86 **sd_write**

\sd_write (numblocks filename --) writes a new or existing file until ... followed immediately by a cr is encountered

\ if the file exists, writing will be truncated to the existing maximum file size allocated when the file was created

9.4.87 **tbuf>cog7**

\ tbuf>cog7 (n1 --) the cog address to start writing 7 longs

9.4.88 **v_currentdir**

\ Address to a long in cog memory, the block number of the current directory

9.4.89 **v_sd_clk**

\ Address to a long in cog memory, a mask with the for sd_clk

9.4.90 **v_sd_di**

\ Address to a long in cog memory, a mask with the for sd_di

9.4.91 **v_sd_do**

\ Address to a long in cog memory, a mask with the for sd_do

9.4.92 **v_sdbase**

\ Address to a long in cog memory, the base for all the cog memory used by thesd filesystem