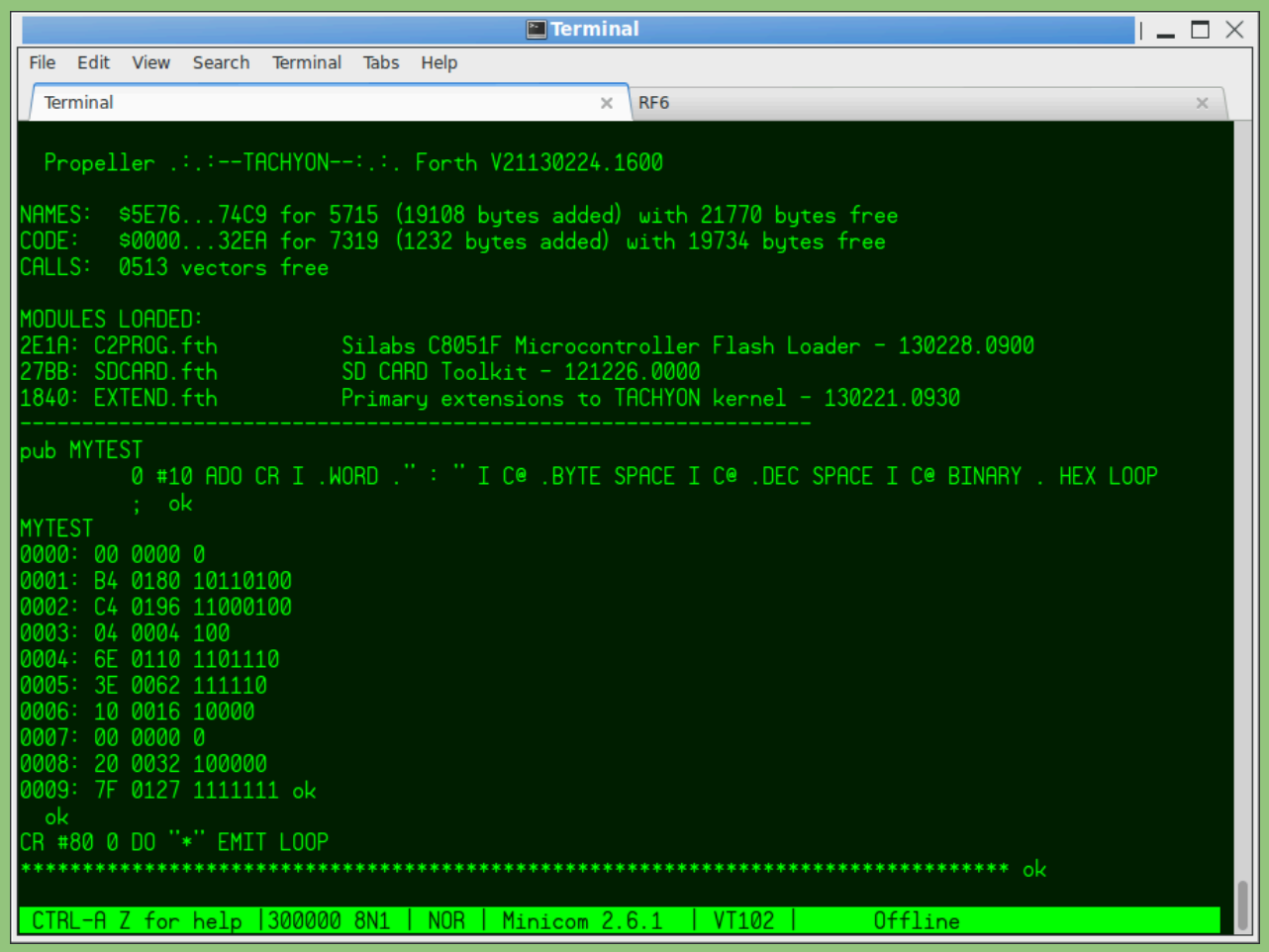
{{

.:.:-- TACHYON --:.:.

A very fast and very small forth byte code interpreter for the propeller chip.
2012 Peter Jakacki

TABLE OF CONTENTS

Sample screenshot of a Minicom terminal session (at 3Mbaud)



WORDS

COLD BOOT exttimers INFO
STATS END .free .added .VECTORS .RUNTIME BINARYDUMP FIXCKSUM IDUMP cksum INH INTMASK INTVEC
INTERRUPT CONBAUD EDUMP RESTORE ?BACKUP BACKUP 64K? EVERIFY EFILL ECOPY eebuf ELOAD ESAVEB
ESAVE ENDRD EE@ EE! EERD @EE I2CBUS SI2C@ SI2C! SI2C!? I2C@ I2C! I2C!? I2CSTOP I2CSTART
I2CPINS EEPROM SDA SCL % PWM! PWMFREQ RUNPWM PWMCOG pwmfreq pwmtbl pwmchans pwmpins
RUNTIMERS TIMERTASK ALARM TIMEOUT? TIMEOUT TIMER timercnt timers runtime MUTE HZ KHZ MHZ DAC!
FRQ BPIN APIN PLLDIV PLL DUTY CTRMODE NCO B A CTR@ CTR! CTR ctr PINS? PINLOAD? .PIN
SPRS REG$ .LAP U.N .NUM (SEP) B>L B>W W>L W>B L>W >B >W CON SERIN SEROUT SERBAUD LB
BDUMP RMAP MAP .MAP MODULES WORDS .ATRS atrs QWORDS MY LISTKEY @. C~~ C~ W~~ W~ ~~ ~
C-- C++ W-- W++ -- ++ <= => >| |< COMPARE$ COPY$ seconds second MOD LONGFILL LBIT!
MASKS PININP PINCLR PINSET TYPE ALIGN TASKS ENQ@ ATN! ENQ! ATN@ ATN? RUN TASK? COGINIT
@CNT! COGREGS @CNT @MISO @MOSI @SCK COGREG@ COGREG! CLKFREQ PFA>NFA RELEASE LOCAL X4 X3
X2 X1 @X locals ESC? BREAK U@ U! navar SPACES .INDEX == DS ORG org ... Published ok
IMMEDIATE]~ [~ LEMIT EXTEND.fth DUP OVER DROP 2DROP SWAP ROT NIP 0 1 2 3 4 5 6 7 8 ON
TRUE -1 BL 16 FALSE OFF 1+ 1- + - ADO DO LOOP +LOOP FOR NEXT <BEGIN AGAIN> UNTIL>
INVERT AND ANDN OR XOR ROL ROR SHR SHL 2/ 2* REV MASK 0= NOT = > C@ W@ @ C+! C!
C@++ W+! W! +! ! BIT! SET CLR IC! U/MOD UM* NEGATE IN@ P@ OUT! P! CLOCK OUTSET
OUTCLR OUTPUTS INPUTS WAITLOW SHROUT SHRINP LOADMOD RUNMOD RESET 0EXIT EXIT NOP 3DROP ?DUP
3RD 4TH SFR@ SPR@ COG@ COGREG COG! PASM CALL JUMP POPMARK >R R> >L L> REG DELTA
WAITCNT WAITPEQ WAITPNE (XCALL) (YCALL) (ELSE) (IF) (UNTIL) (AGAIN) (EMIT) (REG) (PUSH4)
(PUSH3) (PUSH2) (PUSH1) (VAR) CMPSTR LSTACK I CMOVE <CMOVE : pub pri IF ELSE THEN ENDIF
BEGIN UNTIL AGAIN WHILE REPEAT ; \ '' ({ } IFNDEF " ." BYTE WORD LONG TABLE CONSTANT
TO C, | || , STACKS XCALLS REBOOT STOP COGID PAR CNT INA INB OUTA OUTB DIRA DIRB
CTRA CTRB FRQA FRQB PHSA PHSB VCFG VSCL SPR SPR! [SSD] [ESPIO] [SPIO] [SPIOD] [SDRD]
[SDWR] [PWM] [PWM!] [PLOT] * !SP 2+ 2- SET? / 2DUP MIN MAX 0<> <> 0< < U< WITHIN
?EXIT BOUNDS LEAVE J K IX ERASE FILL ms us CNT@ NEWCNT LAPCNT OUT IN INPUT PIN! PIN@
KEY? KEY HEX DECIMAL BINARY .S DUMP COGDUMP .STACKS DEBUG EMIT CLS SPACE BELL CR
SPINNER .HEX .BYTE .WORD .LONG . >DIGIT NUMBER SCRUB GETWORD SEARCH FINDSTR NFA>CFA EXECUTE
V2 VER .VER TACHYON @PAD HOLD >CHAR #> <# # #S (STR) .STR STRLEN U. .DEC DISCARD
REG flags base digits delim word autorun keypoll tasks unum uemit ukey names here codes
errors baudcnt prompt find create lines ALLOT ALLOCATED HERE AUTO! MARK UNMARK >PFA NFA' '
KEYPOLL AUTORUN [COMPILE] GRAB LITERAL CREATEWORD CREATE CVARIABLE WVARIABLE VARIABLE TASK
IDLE LOOKUP VECTORS +CALL ITEM ITEM@ ITEMS NFA>CFA BUFFERS FREE TERMINAL COLD *end*

Features:

V2

Stacked backward branch references - fast and simple looping

Fast load features

- Filtered source code removes comments, extra whitespace etc.
- Large receive buffer
- ~~EEPROM dictionary temporarily buffered in RAM and released at END~~

V1

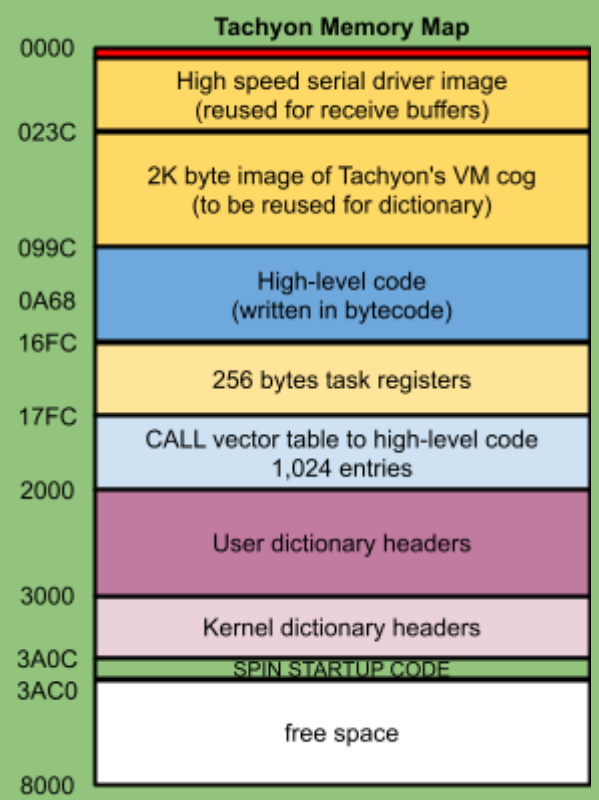
- Small memory footprint
- Low level words and run-time bytecode interpreter in PASM
- Byte codes are read from hub RAM and directly address first 256 longs of PASM code in cog
- User PASM mode via stacks
- Interpreted bytecode definitions are referenced either as:
  2 bytes - vectored CALL opcode + byte index into 512 entry table
  3 bytes - WCALL opcode + 16-bit relative address = 3 bytes

- All literals and strings are byte aligned
- Fast access Forth registers avoids deep stacks and juggling
- Fast I/O bit-bashing support for clocked and asynchronous data (2.8MHz clock speed)
  Flexible SPI or I2C PASM code support words in kernel
  Construct fast serial drivers with minimal code

- Holds Forth headers in EEPROM or SD storage (to be implemented)
  Kernel's dictionary is searched in reused RAM (from VM's cog image)
  Searches the dictionary using rapid index key searching by first character
  No hub RAM is used by headers (except cog image reuse) (after development)
  Even 32K EEPROMs can be used if the area is in RAM is normally rewritten (i.e. video memory)
  Option to hold additional information per defintion such as stack usage and description
- VM Kernel compiled in standard manner via Spin tools so other Spin objects can be combined
- Three stacks in COG RAM: Data, Return, and Loop
  Access loop indices outside of definitions allows efficient factoring
  Avoids manipulation and corruption of return stack
  Static stack arrays for direct addressing of stack items
  Intrinsically safe data stack overflow and underflow - optional error reporting
- 350ns minimum instruction cycle time (single bytecode)
  Empty loops can execute in 550ns to 875ns (absolute worst case)
  Two to one stack operations (+ * AND etc) inc opcode fetch take 900ns to 1.087us (absolute worse case) *
- Flexible number input using common symbols for prefixes and suffixes as well as allowing intermixed symbols

The header file is compiled in this file and loaded in but only needs to use it's location in EEPROM as the image in RAM can be reclaimed for user code or video memory etc. (Headers are in EEPROM automatically when source is compiled in Spin)

MEMORY MAP (121010.1200)




```
V2.1 130611
Address     Description
0000        Spin parameters
0018        Version
0020        baudrate
0024        interrupt vectors
0068        Fort registers
0168        CALL vectors (2K) - 1,024 16-bit pointers to bytecode definitions
096C        Forth bytecode kernel
182D        last
1880        EXTEND.fth code
2A00        free
6200        EXTEND.fth dictionary
69C5        Dictionary (builds down)

7500        BUFFERS (do not backup to EEPROM as cog image sits here )
7500        VM cog image
7CB0        Serial receive cog image
7EE4        Spin boot
7F38        free
```

---------- WHAT'S NEW ----------

CHANGELOG

```
}}
DAT
version         long      21_131107,1400
{{
```

131107 If "ok" has been revectored then don't emit a space after a line has been accepted
131105 Included dictionary entries for ['] [NFA'] and >VEC
131029 Added an "accept" word variable that can be used to revector the "ok" confirmation - zero is default
130910 IDLE loop includes a small delay as this cuts normal run current from 65ma to 23ma
"prompt" word variable now executes user code at prompt rather than just echo a character
130812 Add COGINIT support for PASM objects
130811 Optimized serial receive to allow full buffered throughput at 2M baud (one stop bit between characters)
Removed FILL from cog kernel and implemented as a fast CMOVE operation instead (13ms/32K vs 6.5ms)
130714 Adding support for strings
130705 COGID moved to cog kernel
130611 Improved the CASE structures - it's a little more C like now but with improved control
130611 Fixed bug with COLD which was not freeing up the vector table fully

Optimized some kernel words (- * FILL etc)
130610 Added a ?NEGATE kernel instruction to speed up some operations
130605 Modified the CLOCK instruction which is used by I2C routines to use COGREG 4 as the clock mask rather than 0 to prevent conflict with SPI routines which may be operating at the same time.
130520 Removed <BEGIN, UNITL>, and AGAIN> as they are not really used. This frees up some valuable cog space.
Added !RP as a native bytecode instruction rather than a RUNMOD and init RP in terminal loop
130412 Added !RP to allow return stack to be reset to prevent overflow etc
130411 Added CONSOLE word to allow quitting from an application back into the console mode
130404 Added missing headers for >UPPER
130227 Implement a simple CASE structure
130224 Add 16 as a fast constant but create separate code and integrate BL (32) as well.
130216 Active revectoring for KEY word using ukey
130214 Add symbol processing so that a : in a number will shift the current number left by one byte so that IP addresses can be entered like #192:168:0:1
Further improved this with a & prefix so the alternate method can use the standard period between decimal bytes - &192.168.0.1
130110 Include DEL key ($7F) as a backspace as many tablet keyboards don't support a real backspace (it seems)
130107 Added , word to compile longs (complements | and || which compile bytes and words)
121214 Fixed a small bug with DEBUG itself which also involved the STACKS entry in the dictionary.
121114 Needed a fast <CMOVE but to squeeze it in I have generalized the CMOVE (now pCMOVE) instruction to accept preconditioning
Now CMOVE passes an increment of 1 while <CMOVE adjusts the src and dst and passes an increment of -1
Moved some instructions such as > and ROT back into the first page rather than using XOPs (there's plenty of room)
V2.1 Added XOP opcode to fetch the next bytecode to directly index an "opcode" in the high 256 long bank of the cog's kernel.
Move slower and less used opcodes to high bank. Space freed up by removing jmp to high half in some opcodes
Add ESPIO module for enhanced SPIO operations. SSD module added for fast SSD2119 SPI operations
121105 Optimizing kernel
121027 Modified interrupts to use a global interrupt request in hub RAM and for each cog to have it's own interrupt mask and inhibit
Also the return stack is held in hub RAM with each cog having it's own space. Now each VM knows it's identity at boot.
121025 Added high level interrupt capability for up to 32 interrupts. See INTERRUPT for more information
121013 2 things to speed up compilation:
- Add number preprocessor so that numbers prefixed with #$% are processed immediately rather than after a dictionary search.
- Improved FINDSTR method to skip the dictionary entry using the count byte
Add BUFFERS word which points to the 2K VM image (reusable as temporary buffers - do not backup as Tachyon will not be able to boot)
121012 Modify serial break detect to check for floating inputs by pulsing the pin high for 1us then reading 1us later. If it is still low then it is counted as part of a break.

Strongly rearrange the memory allocation to place Tachyon VM image at top of RAM along with the dictionary so that the dictionary builds down and only uses memory as needed. The 2K RAM where the Tachyon VM image resides is usable as buffers etc as long as this area is not backed up.

The dictionary is being broken up into a RAM resident kernel dictionary with names arranged in some order so that frequently referenced words are located earlier. The extension and user dictionary will be separate and searched after the kernel dictionary or if an alternate method is specified then this will be used instead such as paging the dictionary from SD card.

Before searching the dictionary if the word begins with a %, #, or $ and ends in a digit then it will be processed as a number. Doing so will speed up number processing during compilation.
121010 Merge HS-SerialRxL into the Tachyon source and rearrange the memory map so that all task registers, vectors, and the dictionary etc are placed at the end for simplified backup. The serial rxbuffer has been dropped down to 512 bytes and overwrites it's image. BACKUP area is immediately after this buffer from task registers onwards.
121003 Fixed small bug (again) when executing RESET by moving intialization of task variables to a cold start
Removed WCALL and replaced with XCALL,YCALL,ZCALL,VCALL - updated compile time words
Finally got around to adding capability to handle a sign on numbers (must be the first character)
120928 Fixed small bug when executing RESET by moving intialization of task variables to a cold start
Added ZCALL and VCALL which are simple extensions of XCALL and YCALL which use 1K following the XCALLs table for vectors
- This probably means that there is no need for WCALL as it is highly unlikely that more will be needed due to memory limits
120919 Removed PLOT from cog kernel and implemented a LOADMOD version implemented in EXTEND - 9 longs now freed up in cog
Added [PWM!] module for faster PWM table updates (233us)
120917 Expanded area for dictionary (still need to optimise all this)
Modified ' (TICK) to force compilation of a literal so that it can be used inside of a definition
Added [COMPILE] word to force compilation of the next word in case it's an immediate
Added linenumber support for source code loading
120914 Modified SET and CLR to use MUXC operation rather than self-modifying code
Added BIT! which will SET or CLR depending upon the state (mask addr flg --)
120913 Added PWM module to support table based 8 channel 8-bit PWM
120912 Removed FONT from precompiled source. Leave VGA support as is although not activated.
VER now returns the address of the string of version numbers. So VER C@ will return the major version number
.VER works like the old VER in that it printed out the complete version information
120910 Modified CMPSTR so that it does not modify the dict parameter.
Modified FINDSTR and CREATEWORD to handle count byte in names.
Added dummy count bytesto all precompiled dictionary entries - run FIXDICT on cold to calculate and fix them up
Baud rate default set to 921600 for greater compatibility with terminals and Bluetooth
120908 Slight mods to adapt to HS-SerialRxL which has a larger buffer and symbol decoding and filtering
120906 V2B variant with stacked DO..LOOP and FOR..NEXT
plus added stacked branches <BEGIN..UNTIL> and <BEGIN..AGAIN> without affecting V1 structures
120905 Started Tachyon V2 which uses a MARK stack for storing loop and repeat branches.
Just using the return stack for testing LOOPs as present

<Refer to V1 for older changelogs>

To do:
Check signed operations

For the latest links:
Introduction to TACHYON Forth (including other links at the bottom of the page)

}}

{ ****** SOURCE CODE ****** }

```
DAT

baudrate        long    baud                    ' The baudrate is now stored as a variable
```

```
intreqs         long    0
intvecs         word    0[32]

CON
_clkmode = xtal1 + pll8x      ' <---------------- change to suit
_xinfreq = 10_000_000         ' <---------------- change to suit your crystal

baud            = 230400                        ' <-- change - tested from 300 baud to 3M baud
bufsize         = $200                          ' Serial rx buffer

' Standard Port assignments
scl             = 28
sda             = 29
txd             = 30
rxd             = 31

' Stack sizes
datsz           = 12            'NOTE: Doesn't need to be deep (nor should it be) - augmented by loop stack and registers
retsz           = 22
loopsz          = 8
branchsz        = 6

OBJ

'''vga    : "VGA_512x384_Bitmap"
```

```
CON

numpadsz        = 40            ' We really only need a large buffer for when long binary numbers with separators are used
wordsz          = 40
tasksz          = 8             ' 8 bytes/task RUN[2] FLAG[1]

' flags
     echo       = 1
linenums        = 2             ' prepend line number to each new line
ipmode          = 4             ' interpret this number in IP format where a "." separates bytes
sign            = $20
comp            = $40           ' force compilation of the current word - resets each time
defining        = $80


DAT

' Allocate storage memory for buffers and other variables
```

{ TASK REGISTERS }

```
' Task registers can be switched easily by setting "regptr" ( 7 COGREG! )

registers       long 0[64]                              'Variables used by kernel + general-purpose
          org  0
' register offsets within "registers". Access as    REG,delim   ...  REG,base ... etc
'
' LONG aligned registers (can be used for bytes and words also)
temp            res 16    ' general purpose
cntr            res 4
execloc         res 4     ' used by EXECUTE to store bytecodes
baudcnt         res 4
colorptr        res 4
pixelptr        res 4
uswitch         res 4     ' target parameter used in CASE structures
tasks           res tasksz*8

anumber         res 4     ' Assembled number from input
bnumber         res 4
digits          res 1     ' number of digits in current number that has just been processed
dpl             res 1     ' Position of the decimal point if encountered (else zero)
' WORD aligned registers
unum            res 2     ' User number processing routine - executed if number failed and UNUM <> 0
findvec         res 2     ' runs extended dictionary search if set after failing precompiled dictionary search
createvec       res 2     ' If set will execute user create routines rather than the kernel's
rxptr           res 2     ' Pointer to the terminal receive buffer - read & write index precedes
rxsz            res 2
autovec         res 2     ' user autostart address if non-zero - called from within terminal
keypoll         res 2     ' poll user routines - low priority background task
uemit           res 2
```

```
ukey            res 2
corenames       res 2
                res 2     ' backup of names used at start of TACHYON load
names           res 2     ' start of dictionary (builds down)
prevname        res 2     ' temp location used by CREATE
                res 2
here            res 2     ' pointer to compilation area (overwrites VM image)
codes           res 2     ' current code comilation pointer (updates "here" or is reset by it)
cold            res 2
errors          res 2
linenum         res 2

' Unaligned registers

delim           res 2     ' the delimiter used in text input and a save location
base            res 2     ' current number base + backup location during overrides

lasttwo         res 2     ' last two characters (looking for sequences)
prompt          res 2     ' pointer to code to execute when Forth prompts for a new line
accept          res 2     ' pointer to code to execute when Forth accepts a line to interpret (0=ok)
spincnt         res 1     ' Used by spinner to rotate busy symbol
flags           res 1

prefix          res 1     ' NUMBER prefix
suffix          res 1     ' NUMBER suffix

wordcnt         res 1     ' length of current word (which is still null terminated)
wordbuf         res wordsz        ' words from the input stream are assembled here

padwr           res 1             ' write index (builds characters down from lsb to msb in MODULO style)
numpad          res numpadsz      ' Number print format routines assemble digit characters here
endreg          res 0
```

{ ****** BYTECODE DEFINITIONS VECTOR TABLE ****** }

```
{
Kernel bytecode definitions need to be called and this table makes it easy to do so
with just a 2 byte call. Extra memory may be allocated for user definitions as well
The Spin compiler requires longs whereas we only need 16-bit words but this will do
at present. The runtime compiler can reuse the high-word of all these longs and compile
a YCALL rather than an XCALL so that the high-word is used instead
Also another table has been added to expand the call vectors up to 1024 entries.

}

DAT

          org 0           ' ensure references can be reduced to a single byte index to be called by XCALL xx
'
XCALLS
xXCALLS                   long @_XCALLS+s
xEXEC                     long @registers+execloc+s
xMIN                      long @_MIN+s
xMAX                      long @_MAX+s
xSETQ                     long @SETQ+s
xDIVIDE                   long @DIVIDE+s
xLT                       long @LT+s
xZLT                      long @ZLT+s
xULT                      long @ULT+s
xWITHIN                   long @WITHIN+s
xFILL                     long @FILL+s
xERASE                    long @ERASE+s
xMS                       long @ms+s
xus                       long @us+s
xNEWCNT                   long @NEWCNT+s
xLAPCNT                   long @LAPCNT+s
xOUT                      long @OUT+s
xAIN                      long @IN+s
xPINSTORE                 long @PINSTORE+s
xPINFETCH                 long @PINFETCH+s
xBOUNDS                   long @BOUNDS+s
xLEAVE                    long @LEAVE+s
xJ                        long @J+s
xK                        long @K+s
xIX                       long @IX+s
xInitStack long @InitStack+s
xCOMMENT                  long @COMMENT+s
xBRACE                    long @BRACE+s
```

```
xCURLY                  long @CURLY+s
xIFNDEF                 long @IFNDEF+s
xPRTHEX                 long @PRTHEX+s
xPRTBYTE                long @PRTBYTE+s
xPRTWORD                long @PRTWORD+s
xPRTLONG                long @PRTLONG+s
xPRTSTK                 long @PRTSTK+s
xPRTSTKS                long @PRTSTKS+s
xDEBUG                  long @DEBUG+s
xDUMP                   long @DUMP+s
xCOGDUMP                long @COGDUMP+s
xREBOOT                 long @_REBOOT+s
xSTOP                   long @STOP+s
'xCOGID                 long @_COGID+s
x_PAR                   long @_PAR+s
x_CNT                   long @_CNT+s
x_INA                   long @_INA+s
x_INB                   long @_INB+s
x_OUTA                  long @_OUTA+s
x_OUTB                  long @_OUTB+s
x_DIRA                  long @_DIRA+s
x_DIRB                  long @_DIRB+s
x_CTRA                  long @_CTRA+s
x_CTRB                  long @_CTRB+s
x_FRQA                  long @_FRQA+s
x_FRQB                  long @_FRQB+s
x_PHSA                  long @_PHSA+s
x_PHSB                  long @_PHSB+s
x_VCFG                  long @_VCFG+s
x_VSCL                  long @_VSCL+s
x_SPR                   long @_SPR+s
xSPRSTORE               long @SPRSTORE+s

xSSD                    long @SSD+s
xESPIO                  long @ESPIO+s
xSPIO                   long @SPIO+s
xSPIOD                  long @SPIOD+s
xSDRD                   long @SDRD+s
xSDWR                   long @SDWR+s
xPWM                    long @PWM+s
xPWMST                  long @PWMST+s
xPLOT                   long @PLOT+s

xCMOVE                  long @CMOVE+s
xRCMOVE                 long @RCMOVE+s

xCLS                    long @CLS+s
xEMIT                   long @EMIT+s
xSPACE                  long @SPACE+s
xBELL                   long @BELL+s
xCR                     long @CR+s
xOK                     long @OK+s
xSPINNER                long @SPINNER+s
xBIN                    long @BIN+s
xDECIMAL                long @DECIMAL+s
xHEX                    long @HEX+s
xKEYQ                   long @KEYQ+s
xREADBUF                long @READBUF+s
xKEY                    long @KEY+s
xQEMIT                  long @QEMIT+s
xTOUPPER                long @TOUPPER+s
xPUTCHAR                long @PUTCHAR+s
xPUTCHARPL long @PUTCHARPL+s
xSCRUB                  long @SCRUB+s
xdoCHAR                 long @doCHAR+s
xGETWORD                long @GETWORD+s
xTICK                   long @TICK+s
xATICK                  long @ATICK+s
xNFATICK                long @NFATICK+s
x_NFATICK               long @_NFATICK+s
xTOVEC                  long @TOVEC+s
xPFA                    long @PFA+s
xSETPOLL                long @SETPOLL+s
xIFEXIT                 long @IFEXIT+s

xSEARCH                 long @SEARCH+s
xFINDSTR                long @FINDSTR+s
xEXECUTE                long @EXECUTE+s
xGETVER                 long @GETVER+s
xPRTVER                 long @PRTVER+s
xTODIGIT                long @TODIGIT+s
xNUMBER                 long @NUMBER+s
```

```
xTERMINAL               long @TERMINAL+s
xCONSOLE                long @CONSOLE+s
x_NUMBER                long @_NUMBER+s
xGRAB                   long @GRAB+s
xRESFWD                 long @RESFWD+s

xATPAD                  long @ATPAD+s
xHOLD                   long @HOLD+s
xTOCHAR                 long @TOCHAR+s
xRHASH                  long @RHASH+s
xLHASH                  long @LHASH+s
xHASH                   long @HASH+s
xHASHS                  long @HASHS+s
x_STR                   long @_STR+s
xPSTR                   long @PSTR+s
xPRTSTR                 long @PRTSTR+s
xSTRLEN                 long @STRLEN+s
xUPRT                   long @UPRT+s
xPRT                    long @PRT+s
xPRTDEC                 long @PRTDEC+s

xLITCOMP                long @LITCOMP+s
xBCOMP                  long @BCOMP+s
xBCOMPILE               long @BCOMPILE+s
xCCOMP                  long @CCOMP+s
xWCOMP                  long @WCOMP+s
xLCOMP                  long @LCOMP+s

xSTACKS                 long @STACKS+s

x_STR_                  long @_STR_+s
x_PSTR_                 long @_PSTR_+s
x_IF_                   long @_IF_+s
x_ELSE_                 long @_ELSE_+s
x_THEN_                 long @_THEN_+s
x_BEGIN_                long @_BEGIN_+s
x_UNTIL_                long @_UNTIL_+s
x_AGAIN_                long @_AGAIN_+s
x_REPEAT_               long @_REPEAT_+s
xMARK                   long @MARK+s
xUNMARK                 long @UNMARK+s
xVECTORS                long @VECTORS+s
xTASK                   long @TASK+s
xIDLE                   long @IDLE+s


xALLOT                  long @ALLOT+s
xALLOCATED long @ALLOCATED+s
xCVARIABLE long @CVARIABLE+s
xWVARIABLE long @WVARIABLE+s
xVARIABLE               long @VARIABLE+s
xTABLE                  long @TABLE+s
xCONSTANT               long @ACONSTANT+s
xATO                    long @Ato+s
xATSMUDGE               long @ATSMUDGE+s
xCOLON                  long @COLON+s
xENDCOLON               long @ENDCOLON+s
xCOMPILES               long @COMPILES+s
xCREATE                 long @CREATE+s
xCREATEWORD             long @CREATEWORD+s
xCREATESTR long @CREATESTR+s
xAddACALL               long @AddACALL+s
xHERE                   long @_HERE+s
xITEM                   long @ITEM+s
xITEMFT                 long @ITEMFT+s
xITEMS                  long @ITEMS+s
'xI                     long @I+s
xNFACFA                 long @NFACFA+s
'xWORDS                 long @WORDS+s
'x_PIXELS               long @_PIXELS+s
x_TACHYON               long @_TACHYON+s
xERROR                  long @ERROR+s
xDISCARD                long @DISCARD+s
xAUTORUN                long @AUTORUN+s
xFREE                   long @FREE+s
xAUTOST                 long @AUTOST+s
xFIXDICT                long @FIXDICT+s
xBUFFERS                long @BUFFERS+s
xCOLDST                 long @COLDST+s
xSWITCH                 long @_SWITCH+s
xSWITCHFETCH            long @SWITCHFETCH+s
xISEQ                   long @ISEQ+s
```

```
xIS                     long @IS+s
xISEND                  long @ISEND+s
xISWITHIN               long @ISWITHIN+s
xCOGINIT                long @aCOGINIT+s


' NOTE: this table is limited to 256 entries but leave room for extensions and user application to use the rest of these
' 120928 - update to extend calls to 2 tables with a total of 1,024 entries
xLAST                   long 0[512-xLAST]                                ' Reserve the rest of the area possible
                        long 0                                           ' plus one extra for termination
```

{ ****** HIGH LEVEL FORTH AREA ****** }

```
DAT
                org $01D8 ' fixed address - high-level code can always assume it is here
pRUNMOD
_RUNMOD         res 18

DAT
```

{ ****** HIGH LEVEL BYTECODE DEFINITIONS ****** }

```
' Emit a character
' EMIT ( char -- )
EMIT        byte   REG,uemit,WFETCH,QDUP,_IF,01,AJMP             ' execute user EMIT if vector is non-zero
            byte   _WORD,$01,00,_OR,_SHL1,XOP,pEMIT,EXIT         ' else prep data with start and stop bits to transmit

' Print inline string
PRTSTR      byte   RPOP
pstlp       byte   CFETCHINC,QDUP,_IF,04,XCALL,xEMIT,_AGAIN,@pstret-@pstlp
pstret      byte   PUSHR,EXIT

{ debug print routines - also used by DUMP etc }

' .HEX ( n -- ) print nibble n as a hex character
PRTHEX   ' ( n -- ) print n (0..$0F) as a hex character
            byte   _BYTE,$0F,_AND
            byte   PUSH1,$30,PLUS
            byte   DUP,PUSH1,$39,GT,_IF,02,_7,PLUS                    'Adjust for A..F
PRTCH       byte   XCALL,xEMIT,EXIT

 ' .BYTE ( n -- ) print n as 2 hex characters
PRTBYTE     byte   DUP,_4,_SHR
            byte   XCALL,xPRTHEX,XCALL,xPRTHEX,EXIT

 ' .WORD ( n -- ) print n as 4 hex characters
PRTWORD     byte  DUP,_8,_SHR
            byte  XCALL,xPRTBYTE
PW1         byte  XCALL,xPRTBYTE
PW2         byte  EXIT

 ' .LONG ( n -- ) print n as 8 hex characters
PRTLONG     byte  DUP,PUSH1,16,_SHR
            byte  XCALL,xPRTWORD
            byte  _BYTE,".",XCALL,xEMIT
PRL1        byte  XCALL,xPRTWORD
PRL2        byte  EXIT
' Deprecated in favour of BDUMP in EXTEND.fth - still used internally but renamed to HDUMP to avoid conflict
' DUMP ( addr cnt -- ) Hex dump of hub RAM - (NOTE: if CFETCH is vectored then other memory can be accessed)
DUMP        byte  ADO
            byte    XCALL,xCR
            byte    I,XCALL,xPRTWORD
            byte    XCALL,xPRTSTR,": ",0
            byte    I,PUSH1,$10,ADO
            byte      I,CFETCH,XCALL,xPRTBYTE
            byte      PUSH1,$20,XCALL,xEMIT,LOOP
            byte     XCALL,xPRTSTR,"   ",0
            byte     I,PUSH1,$10,ADO
            byte       I,CFETCH,DUP,BL,XCALL,xLT,OVER,PUSH1,$7E,GT,_OR
            byte       _IF,03,DROP,PUSH1,"."
            byte       XCALL,xEMIT,LOOP
            byte    PUSH1,$10,PLOOP
            byte    XCALL,xCR,EXIT

' COGDUMP ( addr cnt -- ) Dump cog memory, but try to minimize stack usage
```

```
COGDUMP     byte  REG,temp,WSTORE,REG,temp+2,WSTORE,JUMP,@cdm2-@cdmlp
cdmlp       byte   REG,temp+2,WFETCH,_3,_AND,ZEQ,_IF,@cdm3-@cdm2
cdm2        byte  XCALL,xCR,REG,temp+2,WFETCH,XCALL,xPRTWORD,XCALL,xPRTSTR,": ",0
cdm3        byte   REG,temp+2,WFETCH,COGFETCH,XCALL,xPRTLONG,BL,XCALL,xEMIT
            byte   _1,REG,temp+2,WPLUSST,MINUS1,REG,temp,WPLUSST
            byte   REG,temp,WFETCH,ZEQ,_UNTIL,@cdm1-@cdmlp
cdm1        byte   EXIT

' .S        Print out the top four numbers of the datastack
PRTSTK      byte  XCALL,xPRTSTR,$0D,$0A,"STACK: ",0
            byte  FOURTH,XCALL,xPRTLONG,XCALL,xSPACE
            byte  THIRD,XCALL,xPRTLONG,XCALL,xSPACE
            byte  OVER,XCALL,xPRTLONG,XCALL,xSPACE
            byte  DUP,XCALL,xPRTLONG,XCALL,xSPACE
            byte  EXIT

STACKS      byte   _0,COGREG,_BYTE,18,PLUS,EXIT

PRTSTKS     ' Print stacks but avoid cluttering with data from debug routines
            byte   XCALL,xSTACKS,PUSH1,datsz
            byte   XCALL,xPRTSTR,$0D,$0A,"DATA STACK ",0
            byte   XCALL,xCOGDUMP

            byte   XCALL,xSTACKS,PUSH1,datsz,PLUS,PUSH1,retsz
            byte   XCALL,xPRTSTR,$0D,$0A,"RETURN STACK ",0
            byte   XCALL,xCOGDUMP
            byte   XCALL,xSTACKS,PUSH1,datsz+retsz,PLUS,PUSH1,loopsz
            byte   XCALL,xPRTSTR,$0D,$0A,"LOOP STACK ",0
            byte   XCALL,xCOGDUMP

            byte   XCALL,xPRTSTR,$0D,$0A,"BRANCH STACK ",0
            byte   XCALL,xSTACKS,PUSH1,datsz+retsz+loopsz,PLUS,PUSH1,branchsz
            byte   XCALL,xCOGDUMP

            byte   EXIT

' Print the stack(s) and dump the registers - also called by hitting <ctrl>D during text input
DEBUG       byte   XCALL,xPRTSTKS
            byte   XCALL,xPRTSTR,$0D,$0A,"REGISTERS",0
            byte   REG,temp,_WORD,1,00,XCALL,xDUMP
            byte   XCALL,xPRTSTR,$0D,$0A,"COMPILATION AREA",0
            byte   REG,here,WFETCH,PUSH1,$40,XCALL,xDUMP
            byte   EXIT




' I/O PORT PIN ACCESS

' PIN! ( state pin -- )
PINSTORE    byte   MASK

' Set one or more pins to high or low state
' OUT ( state pinmask -- )
OUT         byte   DUP,OUTPUTS,SWAP,SHROUT,DROP2,EXIT

' Read a single input pin
' PIN@ ( pin -- state )
PINFETCH    byte   MASK
' IN ( pinmask -- state )
IN          byte   PFETCH,_AND,ZEQ,ZEQ,EXIT

' TIMING OPERATIONS USING CNT

' NEWCNT ( -- ) Save current system CNT
NEWCNT      byte   _1,SPRFETCH,REG,cntr,STORE,EXIT

' LAPCNT ( -- time ) Elapsed time in microseconds since NEWCNT
LAPCNT      byte   _1,SPRFETCH,REG,cntr,FETCH,XCALL,xNEWCNT,MINUS,_BYTE,80,XCALL,xDIVIDE,EXIT


' COG CONTROL

_REBOOT     ' REBOOT
            byte   _BYTE,$FF
                   '      000011 00, 0i 1111 dd, ddddddd s, ssssssss
            byte   _LONG,%000011_00,%01_1111_00,%0000000_0,%00000000
            byte   JUMP,@addstk-@rb01
rb01

STOP        ' STOP ( cog -- )
```

```
                '      000011_00, 01 1111 ddddddddd
          byte   _LONG,%000011_00,%01_1111_00,%0000000_0,%00000011
          byte   _WORD,(tos+1)>>8,tos+1,_BYTE,9,_SHL,PLUS
          byte   PASM,DROP2,EXIT
addstk    byte   _WORD,(tos+1)>>8,tos+1,_BYTE,9,_SHL,PLUS
          byte   PASM,DROP,EXIT


' HERE ( -- addr ) Address of next compilation location
_HERE     byte   REG,here,WFETCH,EXIT

' ITEM ( index -- regaddr )
ITEM      byte   _SHL1,_SHL1,ATREG,EXIT
' ITEM@ ( index -- long )
ITEMFT    byte   XCALL,xITEM,FETCH,EXIT
' ITEMS ( n1..nx cnt -- ) Push n stack items into registers
ITEMS     byte   _0,DO,I,XCALL,xITEM,STORE,LOOP,EXIT
```

{ TACHYON VM CODE MODULES }

```
' There are at 24 longs reserved in the VM for a code module which can be loaded

{
''
'' Compare a null-terminated source string with a dictionary string which is 8th bit terminated.
'' This will always force a mismatch after which one is checked for a null while the other is checked
'' for the 8th bit and if verified then a match has been found.
'' The dict pointer is advanced to point to the end of the dict string on the 8th bit termination which
'' is the attribute byte as in: byte "CMPSTR",$80,CMPSTR
''
' XOP
' CMPSTR ( src dict -- src dict+ flg ) Compare strings at these two addresses
pCMPSTR

' CMPSTR ( src dict -- src dict+ flg ) Compare strings at these two addresses
_CMPSTR                   call    #_PUSHX                 ' make room for a flag
                          mov     R2,tos+1
                          mov     X,tos+2                 ' X = source
cmpstrlp                  rdbyte  R0,X                    ' read in a character from the source
                          rdbyte  R1,R2                   ' read in from the dictionary
                          cmp     R1,R0 wz                ' are they the same?
                if_nz     jmp     #nomatch
                          add     X,#1                    ' so far, so good, try next
                          add     R2,#1                   ' updates the dict pointer on the stack too
                          jmp     #cmpstrlp               ' keep at it
nomatch                   cmp     R0,#0 wz                ' was the src null terminated?
                          xor     R1,#$80
                if_z      test    R1,#$80 wz              ' set z flag if dict 8th bit set
                          jmp     #SETZ

CMPSTRMOD  byte   _WORD,(@_CMPSTR+s)>>8,@_CMPSTR+s,_BYTE,14,XOP,pLOADMOD,EXIT

}
{
                          org     _RUNMOD
' Write byte to I2C bus
' COGREGS: 0=scl 1=sda
' I2CWR ( send -- ack )  ( A: miso mosi sck )
_I2CWR                    mov     R1,#8
i2cwrlp                   andn    OUTA,sck                ' clock low
                          or      DIRA,mosi               ' make sure data is an output
                          shl     sdat,#1 wc              ' msb first
                          muxc    OUTA,mosi               ' send next bit of data out
                          call    #i2cdly
                          or      OUTA,sck                ' clock high
                          djnz    R1,#i2cwrlp
                          andn    OUTA,sck                ' get ready to read ack
                          andn    DIRA,mosi               ' float SDA
                          or      OUT,sck

                          jmp     #doNEXT
ic2dly                    nop
i2cdly_ret                ret

I2CWR      byte   _WORD,(@_I2CWR+s)>>8,@_I2CWR+s,_BYTE,16,XOP,pLOADMOD,EXIT

}
```

{ ***** ENHANCED SERIAL PERIPHERAL INPUT OUTPUT MODULE *****

- Transfers 1 to 32 bits msb first
- Transmit data does not need to be left justified to be ready for transmission
- Receive data is in correct format

- Data is shifted in and out while the clock is low
- The clock is left low between transfers unless a chip select mask is specified

}

```
                        org     _RUNMOD
' COGREGS: 0=sck 1=mosi 2=miso 3=cnt 4=cs
' ESPIO ( send -- receive )  ( A: miso mosi sck )
_ESPIO
                        andn    OUTA,REG4               ' chip select low
                        mov     R1,#32
                        sub     R1,scnt
                        shl     tos,R1                  ' left justify transmit data
                        mov     R1,scnt
ESPIOlp                 andn    OUTA,sck                ' clock low
                        shl     sdat,#1 wc              ' assume msb first
                        test    miso,INA wz             ' test data from device while clock is low
                        muxc    OUTA,mosi               ' send next bit of data out
                if_nz   or      sdat,#1                 ' now assemble data (also setup time for mosi)
                        or      OUTA,sck                ' clock high
                        djnz    R1,#ESPIOlp
                        tjnz    REG4,#ESxt              ' leave with clock high if CS mask specified
                        andn    OUTA,sck                ' leave with clock low
ESxt                    or      OUTA,REG4               ' chip select high
                        jmp     #doNEXT

ESPIO       byte   _WORD,(@_ESPIO+s)>>8,@_ESPIO+s,_BYTE,16,XOP,pLOADMOD,EXIT


                        org     _RUNMOD
' COGREGS: 0=sck 1=mosi 2=miso 3=cnt 4=mode
' SSD ( send cnt --  )  ( A: miso mosi sck )
_SSD
SSDREP                  mov     X,tos+1
                        shl     X,#7                    ' left justify but leave msb zero (control byte)
                        call    #SSD8
                        call    #SSD8D
                        call    #SSD8D
                        djnz    tos,#SSDREP
                        jmp     #DROP2

SSD8D                   or      X,#1                    ' move 1 into msb = data
                        ror     X,#1
SSD8                    andn    OUTA,REG4               ' chip select low
                        mov     R1,scnt
SSDlp                   andn    OUTA,sck                ' clock low
                        shl     X,#1 wc                 ' assume msb first
                        muxc    OUTA,mosi               ' send next bit of data out
                        or      OUTA,sck                ' clock high
                        djnz    R1,#SSDlp
                        or      OUTA,REG4               ' chip select high
SSD8D_ret
SSD8_ret                ret

SSD         byte   _WORD,(@_SSD+s)>>8,@_SSD+s,_BYTE,18,XOP,pLOADMOD,EXIT
```

```
{ FAST SPI MODE - FORUM CODE
entry

' Prepare video PLL, NCO runs @6.25MHz (4..8) which gives us 25MHz PLL clock.

                movs    ctra, #CLK              ' output pin
                movi    ctra, #%0_00010_101     ' PLL (video + pin), VCO/4
                mov     frqa, frqx              ' 6.25MHz * 16 / 4 = 25MHz

' Set frame format, 8 pixels or bits, 1 bit lasts a PLL clock.

                movd    vscl, #1 << (12 - 9)    ' 1 clock  per pixel
                movs    vscl, #8                ' 8 clocks per frame

' Start and connect video h/w.

                movd    vcfg, #vgrp             ' pin group
                movs    vcfg, #vpin             ' pins
                movi    vcfg, #%0_01_0_00_000   ' VGA, 2 colour mode

' Setup done. Do something with it.
```

```
                mov     dira, mask              ' drive outputs

' Frame time is 80/25*8 clock cycles. Too close for hub window usage but
' relaxed enough for looping.

:loop           waitvid bits, data              ' data[0]
                rol     data, #8
                waitvid bits, data              ' data[1]
                rol     data, #8
                waitvid bits, data              ' data[2]
                rol     data, #8
                waitvid bits, data              ' data[3]
                rol     data, #8

                add     data, #1
                jmp     #:loop

' initialised data and/or presets

bits            long    $0000FF00
frqx            long    $14000000               ' 6.25MHz
mask            long    |< CLK | |< DTA

' uninitialised data and/or temporaries

data            res     1                       ' requires setup

                fit

CON
  zero    = $1F0                                ' par (dst only)
  vpin    = |< (DTA & %111)                     ' pin group mask
  vgrp    = DTA / 8                             ' pin group

DAT
}

{ *** SERIAL PERIPHERAL INPUT OUTPUT MODULE ***
Transfers 1 to 32 bits msb first
Transmit data must be left justified ready for transmission
Receive data is in correct format
Data is shifted in and out while the clock is low
The clock is left low between transfers

}
                        org     _RUNMOD
' COGREGS: 0=sck 1=mosi 2=miso 3=cnt
' SPIO ( send -- receive )  ( A: miso mosi sck )
_SPIO                   mov     R1,scnt
SPIOlp                  andn    OUTA,sck                ' clock low
                        shl     sdat,#1 wc              ' assume msb first
                        test    miso,INA wz             ' test data from device while clock is low
                        muxc    OUTA,mosi               ' send next bit of data out
                if_nz   or      sdat,#1                 ' now assemble data (also setup time for mosi)
                        or      OUTA,sck                ' clock high
                        djnz    R1,#SPIOlp
                        andn    OUTA,sck                ' leave with clock low
                        jmp     #doNEXT

SPIO      byte  _WORD,(@_SPIO+s)>>8,@_SPIO+s,_BYTE,10,XOP,pLOADMOD,EXIT

                        org     _RUNMOD
' COGREGS: 0=sck 1=mosi 2=miso 3=cnt
' COGREG4 = delay
' SPIOD ( send -- receive )  ( A: miso mosi sck )
_SPIOD                  mov     R1,scnt
SPIODlp                 andn    OUTA,sck                ' clock low
                        shl     sdat,#1 wc              ' assume msb first
                        test    miso,INA wz             ' test data from device while clock is low
                        muxc    OUTA,mosi               ' send next bit of data out
                if_nz   or      sdat,#1                 ' now assemble data (also setup time for mosi)
                        or      OUTA,sck                ' clock high
                        mov     X,REG4
spiodly                 djnz    X,#spiodly
                        djnz    R1,#SPIODlp
                        andn    OUTA,sck                ' leave with clock low
                        jmp     #doNEXT
```

```
SPIOD      byte   _WORD,(@_SPIOD+s)>>8,@_SPIOD+s,_BYTE,12,XOP,pLOADMOD,EXIT

{
                           org      _RUNMOD
' L6470 ( send -- receive )  ( A: miso mosi sck )
_L6470                     mov      R1,scnt
L6470lp                    andn     OUTA,sck                  ' clock low
                           test     miso,INA wz               ' test data from device while clock is low
                           shl      sdat,#1 wc                ' assume msb first
                           muxc     OUTA,mosi                 ' send next bit of data out
                  if_nz    or       sdat,#1                   ' now assemble data (also setup time for mosi)
                           or       OUTA,sck                  ' clock high
                           djnz     R1,#L6470lp
                           jmp      #doNEXT

L6470      byte   _WORD,(@_L6470+s)>>8,@_L6470+s,_BYTE,10,XOP,pLOADMOD,EXIT
}
                           org      _RUNMOD
' Reads in a block from the SD card - 512 bytes takes 1.44ms or 2.92us/byte
' SDRD ( dst cnt --  ;read block from SD into memory )
_SDRD                      or       OUTA,mosi                 ' keep data in high
SDRDlp                     mov      R1,#8
SDRDbits                   andn     OUTA,sck                  ' clock low
                           test     miso,INA wz               ' test data from device while clock is low
                           shl      R0,#1                     ' assume msb first
                  if_nz    or       R0,#1                     ' now assemble data (also setup time for mosi)
                           or       OUTA,sck                  ' clock high
                           djnz     R1,#SDRDbits
                           andn     OUTA,sck                  ' leave with clock low
                           wrbyte   R0,tos+1
                           add      tos+1,#1
                           djnz     tos,#SDRDlp
                           jmp      #DROP2

' Load the SD read module (13us)
SDRD       byte   _WORD,(@_SDRD+s)>>8,@_SDRD+s,_BYTE,13,XOP,pLOADMOD,EXIT



' SD MODULE
                           org      _RUNMOD
_SDWR
SDWRlp                     rdbyte   R0,tos+1
                           add      tos+1,#1
                           shl      R0,#24
                           mov      R1,#8
SDWRbits                   andn     OUTA,sck                  ' clock low
                           shl      R0,#1 wc                  ' assume msb first
                           muxc     OUTA,mosi                 ' send next bit of data out
                           or       OUTA,sck                  ' clock high
                           djnz     R1,#SDWRbits
                           andn     OUTA,sck                  ' leave with clock low
                           djnz     tos,#SDWRlp
                           jmp      #DROP2

' Load the SD write module
SDWR       byte   _WORD,(@_SDWR+s)>>8,@_SDWR+s,_BYTE,12,XOP,pLOADMOD,EXIT

' PWM RUNTIME MODULE

{ PWM functions - 8 channel table based
REG0 = table pointer
REG1 = pwmpins
}

                           org      _RUNMOD

_PWM                       mov      deltaR,tos                ' use supplied delta value from tos
                           mov      target,tos
                           add      target,cnt
                           mov      R1,#0
pwmlp                      mov      X,REG0                    ' read base address of table
                           add      X,R1                      ' add in read index
                           rdbyte   X,X
                           shl      X,REG1                    ' shift up to pwmpins
                           mov      outa,X                    ' update outputs
                           add      R1,#1
                           and      R1,#$FF
                           waitcnt  target,deltaR
                           jmp      #pwmlp

PWM        byte   _WORD,(@_PWM+s)>>8,@_PWM+s,_BYTE,13,XOP,pLOADMOD,EXIT
```

```
' PWM! SETUP TABLE MODULE
                        org     _RUNMOD
' _PWM! ( duty8 mask table -- )
_PWMST                  mov     R2,#0
pwmstlp                 rdbyte  X,tos
                        andn    X,tos+1                 ' clear the bit anyway
                        cmp     R2,tos+2 wc             ' is duty8 > R2
                if_c    or      X,tos+1
                        wrbyte  X,tos
                        add     R2,#1
                        add     tos,#1
                        and     R2,#$FF
                        tjnz    R2,#pwmstlp
                        jmp     #DROP3
' [PWM!]
PWMST       byte   _WORD,(@_PWMST+s)>>8,@_PWMST+s,_BYTE,11,XOP,pLOADMOD,EXIT


' PLOT MODULE
                        org     _RUNMOD
' PLOT ( x y -- ) Setup to plot 512x384 bitmap (NEW: (pre)set COGREG 4 to 6 for 6-bit 64 bytes/line)
_PLOT                   shl     tos,pixshift            ' 64 bytes/Y line
                        mov     X,tos+1
                        shr     tos+1,#3                ' byte offset in line
                        add     tos,tos+1               ' byte offset in frame
                        add     tos,pixeladr            ' byte address in memory
                        and     X,#7                    ' get bit mask
                        mov     tos+1,#1
                        shl     tos+1,X
                        jmp     #SET
' [PLOT]
PLOT        byte   _WORD,(@_PLOT+s)>>8,@_PLOT+s,_BYTE,9,XOP,pLOADMOD,EXIT

{
                        org     _RUNMOD
' COGREGS: 0=rxpin 1=ticks 2=miso 3=cnt
' SPIO (  pin -- receive )
_SRX
                        mov     R1,#
_SRXwlp                 test    sck,ina wz
                if_nz   djnz    R1,#_SRXwlp
                if_nz   jmp     #SRXNO
                        mov     X,REG1
                        shr     X,#1
                        add     X,cnt
                        waitcnt X,REG1
                        test    sck,ina wz
                if_nz   jmp     #_SRX
                        mov     R1,#8
SRXlp

                        jmp     #doNEXT

SRX         byte   _WORD,(@_SRX+s)>>8,@_SRX+s,_BYTE,24,XOP,pLOADMOD,EXIT
}



{
                        org     _RUNMOD
' WAVE!  \ Play from the buffers via reg0
' reg0 = buffer ptr
' reg1 = rate
_PLAY
                        mov     deltaR,REG1
                        mov     target,REG1
                        add     target,cnt
                        mov     cogreg2,tos
                        sub     tos+2,#1                ' tos+2 points to sych byte
                        mov     tos+1,tos               ' tos+1 is the read pointer
                        mov     R1,dst1                 ' 512 word count
wavlp                   rdword  X,tos+1                 ' fetch a sample
                        shl     X,#16
                        add     X,msb
                        mov     frqa,X
                        add     tos+1,#2
                        wrword  #0,tos+2
                        waitcnt target,deltaR
                        djnz    R1,#wavlp
                        jmp     #_PLAY
```

```
'
PLAY      byte   _WORD,(@_PLAY+s)>>8,@_PLAY+s,_BYTE,16,XOP,pLOADMOD,EXIT

}
{
                        org     _RUNMOD
' Read a 16-bit signed sample directly from the SD card and adjust and output to FRQA
' SDRD ( dst cnt --  ;read block from SD into memory )
_SDPLAY                 or      OUTA,mosi               ' keep data in high
                        call    #GETSAMPLE
                        mov     X,R0
                        call    #GETSAMPLE
                        shl     R0,#8
                        or      X,R0

PLAY                    call    #POPX
PLAYX                   shl     X,#16
                        add     X,msb
                        waitcnt target,deltaR
                        mov     FRQA,X
                        jmp     #doNEXT

GETSAMPLE               mov     R1,#8
SDSAMlP                 andn    OUTA,sck                ' clock low
                        test    miso,INA wz             ' test data from device while clock is low
                        shl     R0,#1                   ' assume msb first
                if_nz   or      R0,#1                   ' now assemble data (also setup time for mosi)
                        or      OUTA,sck                ' clock high
                        djnz    R1,#SDSAMlp
                        andn    OUTA,sck                ' leave with clock low
GETSAMPLE_ret           ret

SDPLAY    byte   _WORD,(@_SDRD+s)>>8,@_SDRD+s,_BYTE,13,XOP,pLOADMOD,EXIT
}

BUFFERS   byte   _WORD,(@RESET+s)>>8,@RESET+s,EXIT
```

Table 3-1: Destination Register Fields			
31:18	17:4	3	2:0
14-bit Long address for PAR Register	14-bit Long address of code to load	New	Cog ID

```
' COGINIT ( code pars cog -- ret )
aCOGINIT  byte   SWAP,_4,_SHL,_OR,SWAP,_BYTE,18,_SHL,_OR,_COGINIT,EXIT
```

{ ****** NUMBER PRINT FORMATTING ****** }

```
' @PAD ( -- addr ) pointer to current position in number pad
ATPAD     byte   REG,padwr,CFETCH,REG,numpad,PLUS,EXIT

' HOLD ( char -- )
HOLD      byte   MINUS1,REG,padwr,CPLUSST,XCALL,xATPAD,CSTORE,EXIT

' >CHAR  ( val -- ch ) convert binary value to an ASCII character
TOCHAR    byte   PUSH1,$3F,_AND,PUSH1,"0",PLUS,DUP,PUSH1,"9"    ' convert to "0".."9"
          byte   GT,_7,_AND,PLUS                                ' convert to "A"..
          byte   DUP,PUSH1,$5D,GT,ZEXIT,_3,PLUS,EXIT            ' skip symbols to go to "a"..

' #> ( n1 -- caddr )
RHASH     byte   DROP,XCALL,xATPAD,EXIT


' <#      ' resets number pad write index to end of pad
LHASH     byte   PUSH1,numpadsz,REG,padwr,CSTORE,_0,XCALL,xHOLD,EXIT

' # ( n1 -- n2 ) convert the next ls digit to a char and prepend to number string
HASH      byte   REG,base,CFETCH,XOP,pUDIVMOD,SWAP,XCALL,xtoCHAR,XCALL,xHOLD,EXIT

' #S ( n1 -- 0 ) Convert all digits
HASHS     byte   XCALL,xHASH,DUP,ZEQ,_UNTIL,06,EXIT

STRLEN    ' ( str -- len )
          byte   DUP,CFETCHINC,DEC,_BYTE,$7E,SWAP,XCALL,xULT,_UNTIL,9,SWAP,MINUS,DEC,EXIT

' STR ( -- n ) Leave address of inline string on stack and skip to next instruction
_STR      byte   RPOP,DUP
STRlp     byte   CFETCHINC,ZEQ,_UNTIL,04,PUSHR,EXIT

' . ( n -- ) Print the number off the stack
PRT       byte   DUP,XCALL,xZLT,_IF,05,PUSH1,$2D,XCALL,xEMIT,NEGATE
' U. ( n -- ) Print an unsigned number
```

```
UPRT        byte   XCALL,xLHASH,XCALL,xHASHS,XCALL,xRHASH
' .STR ( adr -- ) Print the null or 8th bit terminated string
PSTR        byte   CFETCHINC,DUP,ZEQ,OVER,_BYTE,$7F,GT,_OR,_IF,02,DROP2,EXIT,XCALL,xEMIT,_AGAIN,16



PRTDEC      byte   _BYTE,10
' DEPPRECATED - only used by END - use .NUM from EXTEND.fth
' B. ( n base -- ) Print the number off the stack in the base specified
basePRT     byte   REG,base,CFETCH,PUSHL,REG,base,CSTORE
            byte   XCALL,xLHASH,XCALL,xHASH,XCALL,xHASH,XCALL,xHASH,XCALL,xHASHS,XCALL,xRHASH,XCALL,xPSTR
            byte   LPOP,REG,base,CSTORE,EXIT
```

****** OPERATORS ******

```
' / ( n1 n2 -- n3 ) divide
DIVIDE      byte   XOP,pUDIVMOD,SWAP,DROP,EXIT

' MIN ( n1 n2 -- n3 ) signed minumum of two items
_MIN        byte   OVER,OVER,GT,_IF,01,SWAP,DROP,EXIT

' MAX ( n1 n2 -- n3 ) signed maximum of two items
_MAX        byte   OVER,OVER,GT,_IF,01,SWAP,SWAP,DROP,EXIT

' 0< ( n -- flg )
ZLT         byte   _0,XCALL,xLT,EXIT

' < ( n1 n2 -- flg )
LT          byte   SWAP,GT,EXIT

' U<
ULT         byte   OVER,OVER,_XOR,XCALL,xZLT,_IF,05
            byte   SWAP,DROP,XCALL,xZLT,EXIT
            byte   MINUS,XCALL,xZLT,EXIT

' ( n lo hi -- flg ) true if n is within range of low and high inclusive
WITHIN      byte   INC,OVER,MINUS,PUSHR
            byte   MINUS,RPOP,XCALL,xULT
WT1         byte   ZEQ,ZEQ,EXIT

' SET? ( mask caddr -- flg ) Test single bit of a byte in memory
SETQ        byte   CFETCH,_AND,ZEQ,ZEQ,EXIT

' ?EXIT ( flg -- ) Exit if flg is true
IFEXIT      byte   _IF,02,RPOP,DROP,EXIT

'' The read and write index is stored as two words preceding the buffer, read this as a word (faster)
'' BKEY ( buffer -- ch ) ' byte size buffer is preceded with a read index, go and read the next character
'
' READBUF ( buffer -- ch )
READBUF
            byte   DUP,DEC,DEC,DUP,WFETCH                       ' point to read index ( buffer writeptr writeindex )
            byte   SWAP,DEC,DEC,WFETCH,SWAP                     ' ( buffer readindex writeindex )
            byte   OVER,EQ,_IF,03,DROP2,_0,EXIT                 ' empty, return with null
            ' ( buffer read )
            byte   OVER,PLUS,CFETCH,_WORD,$01,00,PLUS           ' get char ( buffer [buffer+read]+$100 )
            byte   SWAP,_4,MINUS   ' key readptr )
            byte   DUP,WFETCH,INC,REG,rxsz,WFETCH,DEC,_AND,SWAP,WSTORE
            byte   EXIT


' KEY? ( -- ch flg )
KEYQ        byte   REG,rxptr,WFETCH,XCALL,xREADBUF
            byte   DUP,_BYTE,$0FF,_AND,SWAP,_8,_SHR,ZEQ,ZEQ,EXIT

DOPOLL      byte   REG,keypoll,WFETCH,QDUP,_IF,01,ACALL
KEY         byte   REG,ukey,WFETCH,QDUP,_IF,01,AJMP
            byte   REG,rxptr,WFETCH,XCALL,xREADBUF,QDUP,_UNTIL,@ky1-@DOPOLL
ky1         byte   _BYTE,$0FF,_AND,EXIT


'           \      ( -- )
'           Ignore following text till the end of line.
'           IMMED
COMMENT     byte   XCALL,xKEY,_BYTE,$0D,EQ,_UNTIL,07,XCALL,xCR,EXIT
BRACE       byte   XCALL,xKEY,DUP,XCALL,xEMIT,_BYTE,")",EQ,_UNTIL,10,EXIT

IFNDEF      byte   XCALL,xTICK,ZEXIT
CURLY       byte   XCALL,xKEY,_BYTE,"}",EQ,_UNTIL,07,EXIT
```

{ ****** LOOPS ****** }

```
' BOUNDS ( from for -- to from )
BOUNDS      byte   OVER,PLUS,SWAP,EXIT

LEAVE       byte   _0,LSTACK,COGFETCH,DEC,_1,LSTACK,COGSTORE,EXIT            ' set index to the same as limit

' J ( -- index ) Index of next outer loop
J           byte   _3,LSTACK,COGFETCH,EXIT                       ' set index to the same as limit

' K ( -- index ) Index of third loop
K           byte   _5,LSTACK,COGFETCH,EXIT                       ' set index to the same as limit

' Fetch index of FOR..NEXT loop (or the limit for a DO..LOOP)
IX          byte   _0,LSTACK,COGFETCH,EXIT                       ' set index to the same as limit
```

{ ****** MOVES & FILLS ****** }

```
' CMOVE ( src dst cnt -- )
CMOVE       byte   _1,XOP,pCMOVE,EXIT

' <CMOVE ( src dst cnt -- )
RCMOVE      byte   ROT,OVER,PLUS,DEC,ROT,THIRD,PLUS,DEC,ROT,MINUS1,XOP,pCMOVE,EXIT


ERASE       byte   _0
' ( addr cnt fillch -- )
FILL        byte   THIRD,CSTORE,DEC,OVER,INC,SWAP,_1,XOP,pCMOVE,EXIT
```

{ ****** TIMING ****** }

```
' ms ( n -- ) Wait for n milliseconds
ms          byte   QDUP,ZEXIT,PUSH3,$01,$38,$80,UMMUL,DROP,DELTA,EXIT
' us ( n -- ) Wait for n microseconds
us          byte   QDUP,ZEXIT,PUSH1,80,UMMUL,DROP,DELTA,EXIT
```

{ ****** COG SPR REGISTERS ****** }

```
_PAR        byte   _WORD,$01,$F0,EXIT
_CNT        byte   _WORD,$01,$F1,EXIT
_INA        byte   _WORD,$01,$F2,EXIT
_INB        byte   _WORD,$01,$F3,EXIT
_OUTA       byte   _WORD,$01,$F4,EXIT
_OUTB       byte   _WORD,$01,$F5,EXIT
_DIRA       byte   _WORD,$01,$F6,EXIT
_DIRB       byte   _WORD,$01,$F7,EXIT
_CTRA       byte   _WORD,$01,$F8,EXIT
_CTRB       byte   _WORD,$01,$F9,EXIT
_FRQA       byte   _WORD,$01,$FA,EXIT
_FRQB       byte   _WORD,$01,$FB,EXIT
_PHSA       byte   _WORD,$01,$FC,EXIT
_PHSB       byte   _WORD,$01,$FD,EXIT
_VCFG       byte   _WORD,$01,$FE,EXIT
_VSCL       byte   _WORD,$01,$FF,EXIT
_SPR        byte   _WORD,$01,$F0,PLUS,EXIT

SPRSTORE    byte   XCALL,x_SPR,COGSTORE,EXIT

' !SP
InitStack
            byte   _0,_0,_0,_0,_0,_0,_0,_0,_0,_0,_0
            byte   _LONG,$FF,$FF,$F9,$FF
            byte   EXIT
```

{ ****** NUMBER BASE ****** }

```
' change the default number bases
BIN         byte   _2,JUMP,@SetBase-@DECIMAL
DECIMAL     byte   PUSH1,10,JUMP,@SetBase-@HEX
HEX         byte   PUSH1,16
```

```
SetBase    byte  REG,BASE,CSTORE,EXIT
```

{ ****** OUTPUT OPERATIONS ****** }

```
CLS        byte   PUSH1,$0C,XCALL,xEMIT,EXIT
SPACE      byte   BL,XCALL,xEMIT,EXIT
BELL       byte   _7,XCALL,xEMIT,EXIT
CR         byte   PUSH1,$0D,XCALL,xEMIT,PUSH1,$0A,XCALL,xEMIT,EXIT
SPINNER    byte   REG,spincnt,CFETCH,_3,_SHR,_3,_AND,XCALL,x_STR,"|/-\",0,PLUS,CFETCH
           byte   XCALL,xEMIT,_8,XCALL,xEMIT,_1,REG,spincnt,CPLUSST,_1,XCALL,xms,EXIT


' PROMPT
OK     byte  XCALL,xPRTSTR," ok",$0D,$0A,0,EXIT


' ?EMIT    ,( ch -- ch ) suppress emitting the character if echo flag is off
QEMIT   byte _BYTE,echo,REG,flags,XCALL,xSETQ,ZEXIT,DUP,XCALL,xEMIT,EXIT


' >UPPER  ( str1 --  ) Convert lower-case letters to upper-case
TULP    byte  INC
TOUPPER
     byte  DUP,CFETCH,QDUP,_IF,@TUX-@TU1                  ' end of string?
TU1     byte  _BYTE,"a",_BYTE,"z",XCALL,xWITHIN
     byte  _UNTIL,@TU2-@TULP
TU2     byte  _BYTE,-$20,OVER,CPLUSST,_AGAIN,@TUX-@TULP
TUX     byte  DROP,EXIT
```

{ ****** STRING TO NUMBER CONVERSION ****** }

```
' functional test for now - optimize later
' Convert ASCII value as a digit to a numeric value - only interested in bases up to 16 at present
'
TODIGIT    ' ( char -- val true | false )
           byte   DUP,PUSH1,"0",PUSH1,"9",XCALL,xWITHIN,_IF,@td8-@td7          ' only work with 0..9,A..F
td7        byte   PUSH1,"0",MINUS,_TRUE,EXIT                                 ' pass decimal digits
td8        byte   DUP,PUSH1,"A",PUSH1,"F",XCALL,xWITHIN,_IF,@td2-@td1
td1        byte   PUSH1,$37,MINUS,_TRUE,EXIT                                 ' pass hex digits
td2        byte   DROP,_FALSE,EXIT

{  Try to convert a string to a number
Allow all kinds of symbols but these are the rules for it to be treated as a number.
1. Leading character must be either a recognized prefix or a decimal digit
2. If trailing character is a recognized suffix then the first character must be a decimal digit
Acceptable forms are:
$1000      hex number
1000h
#1000      decimal number
1000d
%1000      binary number
1000b

Also as long as the first character and last character are valid then any symbols me be mixed in the number i.e.
11:59  11.59  #5_000_000
}
_NUMBER    ' ( str -- value digits | false )
           byte   _0,REG,0,STORE
           byte   _BYTE,sign,REG,flags,CLR
snlp       byte   DUP,CFETCH,REG,prefix,CSTORE                               ' save prefix (it may or may not be)
           byte   DUP,DEC,CFETCH,DEC,OVER,PLUS,CFETCH,REG,suffix,CSTORE      ' save suffix (assume string has count byte)
           byte   DUP,CFETCH,_BYTE,"-",EQ,_IF,@sn01-@sn00
sn00       byte   _BYTE,sign,REG,flags,SET,INC                  ',_AGAIN,@sn01-@snlp
sn01
           ' PREFIX HANDLER
           byte   DUP,CFETCH                                                 ' check prefix
           '      ( str ch )
           byte   _FALSE
           byte   OVER,PUSH1,"$",EQ,_IF,04,XCALL,xHEX,_TRUE,_OR  ' as does $ - also set hex base
           byte   OVER,PUSH1,"%",EQ,_IF,04,XCALL,xBIN,_TRUE,_OR  ' as does % - also set binary base
           byte   OVER,PUSH1,"#",EQ,_IF,04,XCALL,xDECIMAL,_TRUE,_OR          ' as does # - also set decimal base
           byte   OVER,PUSH1,"&",EQ,_IF,@pf1-@pf0                            ' as does & - also set decimal base and IP notation
pf0        byte   XCALL,xDECIMAL,_TRUE,_OR
           byte   _BYTE,$80,REG,bnumber+3,CSTORE                             ' this forces "." symbols to work the same as ":"

pf1        byte   DUP,_IF,04,ROT,INC,ROT,ROT                                 ' adjust string pointer to skip prefix
```

```
          '     ( str ch flg )
          byte  SWAP,PUSH1,"0",PUSH1,"9",XCALL,xWITHIN,_OR              ' 0..9 forces processing as a number
          '     ( str flg )
          byte  ZEQ,_IF,03,DROP,_FALSE,EXIT                           ' Give up now, it isn't a candiate
          '     ( str )                                               ' so far, so good, now check suffix
          ' SUFFIX HANDLER
          byte  REG,suffix,CFETCH
          byte  DUP,PUSH1,"0",PUSH1,"9",XCALL,xWITHIN                 ' 0..9
          byte  OVER,PUSH1,"A",PUSH1,"F",XCALL,xWITHIN,_OR            ' A..F ( str sfx flg ) true if still a digit
          byte  OVER,PUSH1,"h",EQ,_IF,04,XCALL,xHEX,_TRUE,_OR  ' h = HEX
          byte  OVER,PUSH1,"b",EQ,_IF,04,XCALL,xBIN,_TRUE,_OR  ' b = BINARY
          byte  SWAP,PUSH1,"d",EQ,_IF,04,XCALL,xDECIMAL,_TRUE,_OR     ' d = DECIMAL
          byte  ZEQ,_IF,03,DROP,_FALSE,EXIT                           ' bad suffix, no good
          ' so far the prefix and suffx have been checked prior to attempt a number conversion
          ' From here on there must be at least one valid digit for a number to be accepted
          ' DIGIT EXTRACTION & ACCUMULATION
nmlp      byte  DUP,CFETCH,DUP,_IF,@nmend-@nm1                        ' while there is another character
nm1       byte  XCALL,xTODIGIT,_IF,@nmsym-@nm2                        ' convert to a digit? or else check symbol
nm2       ' a digit has been found but is it valid for this base?     ' ( str val )
          byte  DUP,REG,BASE,CFETCH,DEC,GT,_IF,@nmok-@nm3
nm3       byte  DROP2,_FALSE,EXIT                                     ' a digit but exceeded base
nmok      byte  REG,anumber,FETCH,REG,BASE,CFETCH,UMMUL,DROP          ' shift anumber left one digit (base)
          byte  PLUS,REG,anumber,STORE                                ' and merge in new digit
          byte  _1,REG,digits,CPLUSST                                 ' update number of digits
nmnxt     byte  INC,_AGAIN,@nmsym-@nmlp                               ' update str and loop

          ' character was not a digit - check for valid symbols (keep it simple for now)
          ' SYMBOLS
nmsym     byte  DUP,CFETCH,_BYTE,":",EQ
          byte  OVER,CFETCH,_BYTE,".",EQ,REG,bnumber,FETCH,ZEQ,ZEQ,_AND,_OR
          byte  _IF,@nmsym2-@nmsym1                                   ' Use : as special byte shift for IP notation etc
nmsym1    byte  REG,bnumber,FETCH
          byte  REG,anumber,FETCH,PLUS,_8,_SHL
          byte  REG,bnumber,STORE,_0,REG,anumber,STORE
nmsym2    byte  JUMPBACK,@nmend-@nmnxt                                ' just ignore other symbols for now
          '
nmend     ' end of string - check
          byte  DROP2,REG,digits,CFETCH,DUP,ZEXIT                     ' return with false if there are no digits
          byte  REG,anumber,FETCH,REG,bnumber,FETCH,PLUS
          byte  _BYTE,sign,REG,flags,XCALL,xSETQ,QNEGATE
          byte  SWAP,EXIT                                 ' all good, return with number and true

NUMBER    ' ( str -- value digits | false )
          byte  DUP,XCALL,xSTRLEN,_2,EQ
          byte  OVER,CFETCH,_BYTE,"^",EQ,_AND,_IF,@ch01-@ctlch ' ^ch  Accept caret char as <control> char
ctlch     byte  INC,CFETCH,_BYTE,$1F,_AND,_1,EXIT
ch01      byte  DUP,XCALL,xSTRLEN,_3,EQ
          byte  OVER,CFETCH,_BYTE,$22,EQ,_AND,_IF,@ch02-@ascch ' "ch" Accept as an ASCII literal
ascch     byte  INC,CFETCH,_1,EXIT
                                                                      ' It wasn't a ASCII literal, process as a number
ch02      byte  REG,anumber,PUSH1,10,_0,XCALL,xFILL                   ' zero out assembled number (double), digits, dpl
          byte  REG,BASE,CFETCH,REG,base+1,CSTORE                     ' backup current number as it may be overridden
          byte  XCALL,x_NUMBER
nmb1      byte  REG,base+1,CFETCH,REG,BASE,CSTORE,EXIT                ' restore default base before returning
```

{ ****** COMPILER EXTENSIONS ****** }

```
' Most of these words are acted upon immediately rather than compiled as they are
' part of the "compiler" in that they create the necessary structures
'

' dumb compiler for literals - improve later - just needs to optimize the number of bytes needed
LITCOMP   ' ( n -- ) compile the literal according to size
          byte  DUP,PUSH1,24,_SHR
          byte  _IF,@lco1-@LITC4
          ' Compile 4 bytes - 32bits
LITC4     byte  PUSH1,PUSH4,XCALL,xBCOMP
          byte    DUP,PUSH1,24,_SHR,XCALL,xBCOMP
          byte    DUP,PUSH1,16,_SHR,XCALL,xBCOMP
          byte    DUP,_8,_SHR,XCALL,xBCOMP
          byte    XCALL,xBCOMP,EXIT
lco1
          byte  DUP,PUSH1,16,_SHR
          byte  _IF,@lco2-@LITC3
```

```
          ' Compile 3 bytes - 24bits
LITC3     byte   PUSH1,PUSH3,XCALL,xBCOMP
          byte     DUP,PUSH1,16,_SHR,XCALL,xBCOMP
          byte     DUP,_8,_SHR,XCALL,xBCOMP
          byte     XCALL,xBCOMP,EXIT
lco2
          byte   DUP,_8,_SHR
          byte   _IF,@LITC1-@LITC2
          ' Compile 2 bytes - 16bits
LITC2     byte   PUSH1,PUSH2,XCALL,xBCOMP
          byte     DUP,_8,_SHR,XCALL,xBCOMP
          byte     XCALL,xBCOMP,EXIT
          ' Compile 1 byte - 8bits
LITC1     byte   PUSH1,PUSH1,XCALL,xBCOMP
          byte     XCALL,xBCOMP,EXIT

' MARK ( addr tag -- tag&addr ) Merge tag and addr by shifting tag into hi word
MARK      byte   _BYTE,$10,_SHL,_OR,EXIT
' UNMARK   ( tag&addr -- addr tag )
UNMARK    byte   DUP,_WORD,$FF,$FF,_AND,SWAP,_BYTE,$10,_SHR,EXIT

' BEGIN as in BEGIN...AGAIN or BEGIN...UNTIL
_BEGIN_   byte   REG,codes,WFETCH,_BYTE,$BE,XCALL,xMARK                    ' generate branchs for BEGIN
bg01      byte   EXIT
' UNTIL ( flg -- )
_UNTIL_   byte   XCALL,xUNMARK
unt00     byte   _BYTE,$BE,EQ,_IF,@badthen-@unt01
unt01     byte   _BYTE,_UNTIL,XCALL,xBCOMP,JUMP,@calcback-@_REPEAT_          '
' AGAIN
_REPEAT_  byte   XCALL,xUNMARK
rp00      byte   _BYTE,$1F,EQ,_IF,@badrep-@rp02
rp02      byte   REG,codes,WFETCH,INC,INC,OVER,MINUS,SWAP,DEC,CSTORE       ' process branch of WHILE to after REPEAT
          byte   JUMP,@_AGAIN_-@badrep
badrep    byte   DROP2,JUMP,@badthen-@_AGAIN_
_AGAIN_   byte   XCALL,xUNMARK
ag00      byte   _BYTE,$BE,EQ,_IF,@badthen-@ag01                           '
ag01      byte   _BYTE,_AGAIN
          ' ( addr bc -- ) compile the bytecode and calculate the branch back
lpcalc    byte  XCALL,xBCOMP
calcback  byte   REG,codes,WFETCH,INC,SWAP,MINUS,XCALL,xBCOMP
          byte   EXIT

' IF as in IF...THEN or IF...ELSE...THEN
_WHILE_
_IF_      byte   _BYTE,_IF,XCALL,xBCOMP,_0,XCALL,xBCOMP
          byte   REG,codes,WFETCH,_BYTE,$1F,XCALL,xMARK
if01      byte   EXIT
' ELSE
_ELSE_    byte   XCALL,xUNMARK
el00      byte   _BYTE,$1F,EQ,_IF,@badthen-@el01                           ' does this match an IF?
el01      byte   _BYTE,JUMP,XCALL,xBCOMP,_0,XCALL,xBCOMP                   ' Compile a jump forward just like an IF
          byte   REG,codes,WFETCH,_BYTE,$1F,XCALL,xMARK                    ' mark the else to be processed on a THEN
el02      byte   SWAP,_BYTE,$1F,XCALL,xMARK                                 ' get the IF addr and proceed as if it were a THEN
el03
' THEN
_THEN_    byte   XCALL,xUNMARK
th00      byte   _BYTE,$1F,EQ,_IF,@badthen-@RESFWD
RESFWD    byte   REG,codes,WFETCH,OVER,MINUS,SWAP,DEC,CSTORE,EXIT          ' calculate branch and update IF's branch
badthen   byte   XCALL,xPRTSTR," Structure mismatch! ",0
          byte   DROP,EXIT

' "        Compile a literal string - no length restriction - any codes can be included except the delimiter "
_STR_     byte   _BYTE,XCALL,XCALL,xBCOMP,_BYTE,x_STR,XCALL,xBCOMP          ' compile bytecodes for string
          byte   JUMP,@COMPSTR-@_PSTR_
' ."       Compile a literal print string - no length restriction - any codes can be included except the delimiter "
_PSTR_    byte   _BYTE,XCALL,XCALL,xBCOMP,_BYTE,xPRTSTR,XCALL,xBCOMP
COMPSTR
pslp      byte   XCALL,xKEY,DUP,XCALL,xEMIT
          byte   DUP,_BYTE,$22,EQ,_IF,05,DROP,_0,XCALL,xBCOMP,EXIT
          byte   XCALL,xBCOMP,_AGAIN,@ps01-@pslp
ps01      '''
```

{ *** CASE STRUCTURE *** }

```
{ TACHYON CASE STRUCTURES
This implementation follows the C method to some degree.
Each CASE statement simply compares the supplied value with the SWITCH and executes an IF
To prevent the IF statement from falling through a BREAK is used (also acts as a THEN at CT)
The SWITCH can be tested directly and a manual CASE can be constructed with IF <code> BREAK
```

```
SWITCH ( switch -- )       \ Save switch value used in CASE structure
CASE ( val -- )            \ compare val with switch and perform an IF. Equivalent to SWITCH= IF
BREAK                      \ EXIT (return) but also completes IF structure at CT. Equivalent to EXIT THEN
\ extra functions to allow the switch to be manipulated etc
SWITCH@ ( -- switch )      \ Fetch last saved switch value
SWITCH= ( val -- flg )     \ Compare val with switch. Equivalent to SWITCH@ =
SWITCH>< ( from to -- flg )    \ Compare switch within range. Equivalent to SWITCH@ ROT ROT WITHIN

Usage:

pub CASEDEMO  ( val -- )
   SWITCH                \ use the key value as a switch in the case statement
   "A" CASE CR ." A is for APPLE " BREAK
   "H" CASE CR ." H is for HAPPY " BREAK
   "Z" CASE CR ." Z is for ZEBRA " BREAK
   $08 CASE 4 REG ~ BREAK
   \ Now accept 0 to 9 and build a number calculator style
   "0" "9" SWITCH>< IF SWITCH@ $30 - 4 REG @ #10 * + 4 REG ! ." *" BREAK
   \ On enter just display the accumulated number
   $0D CASE CR ." and our lucky number is " 4 REG @ .DEC BREAK
   \ show how we can test more than one value
   "Q" SWITCH= "X" SWITCH= OR IF CR ." So you're a quitter hey?" CR CONSOLE BREAK
   CR ." I don't know what " SWITCH@ EMIT ."  is"
    ;
pub DEMO
   BEGIN KEY UPPER CASEDEMO AGAIN
 ;


          byte   $20,"BREAK",              hd+xc+im,        XCALL,xISEND
          byte   $20,"CASE",               hd+xc+im,        XCALL,xIS
          byte   $20,"SWITCH",             hd+xc,           XCALL,xSWITCH
          byte   $20,"SWITCH@",            hd+xc,           XCALL,xSWITCHFETCH
          byte   $20,"SWITCH=",            hd+xc,           XCALL,xISEQ
          byte   $20,"SWITCH><",           hd+xc,           XCALL,xISWITHIN
}


' SWITCH
_SWITCH    byte   REG,uswitch,STORE,EXIT
' SWITCH@ ( -- switch )
SWITCHFETCH
          byte   REG,uswitch,FETCH,EXIT
' SWITCH= ( val -- flg )
ISEQ       byte   REG,uswitch,FETCH,EQ,EXIT
' CASE
IS         byte   _BYTE,XCALL,XCALL,xBCOMP,_BYTE,xISEQ,XCALL,xBCOMP,XCALL,x_IF_,EXIT
' BREAK
ISEND      byte   _BYTE,EXIT,XCALL,xBCOMP,XCALL,xALLOCATED,XCALL,x_THEN_,EXIT


' SWITCH>< ( from to -- flg )..
ISWITHIN   byte   XCALL,xSWITCHFETCH,ROT,ROT,XCALL,xWITHIN,EXIT

{
pub TC ( ch -- )
      CASE
      "4" =[ ." Knock on the door, it's a 4" ]=
      "5" =[ ." Let's jive, it's a 5" ]=
      "7" "9" ..[ ." Pick up sticks, it's more than six" ]=
;
}

{  Table vectoring -
index a table of vectors and jump to that vector
A table limit is supplied as well as a default vector

 Usage:
          <limit> VECTORS <vector if over>
          <vector0> <vector1> ..... <vectorx>)
Sample:
          4 LOOKUP BELL                                  \ an index of 4 or more will default to BELL
          INDEX0 INDEX1 INDEX2 INDEX3                    \ 0 to 3 will execute corresponding vectors

}
' VECTORS ( index range -- )
VECTORS    byte   OVER,GT,ZEQ,_IF,02,DROP,MINUS1                    ' limit index to range or -1 (.>0)
```

```
          byte   INC,_SHL1,RPOP,PLUS,CFETCHINC,SWAP,CFETCH                ' form index into vectors
EXECUTE   ' ( bytecode1 bytecode2 -- )                                     ' 2 bytecodes
          byte   _WORD,(@bcexec+s)>>8,@bcexec+s
          byte   ROT,OVER,CSTORE,INC,CSTORE
bcexec    byte   EXIT,EXIT,EXIT

' XCALLS ( -- addr ) address of XCALLS vector table
_XCALLS   byte   _WORD,(@XCALLS+s)>>8,@XCALLS+s,EXIT



' 12103 - adding a more general method for creating a call vector

' +CALL ( addr -- index opcode ) ' index = 0..$3FF
AddACALL  byte   XCALL,xXCALLS,_0
          byte   _2,PLUS,OVER,OVER,PLUS,WFETCH,ZEQ,_UNTIL,09               ' ( addr base index )
          byte   SWAP,OVER,PLUS,ROT,SWAP,WSTORE                            ' ( index )
          byte   _SHR1,DUP,_1,_AND,_BYTE,XCALL,SWAP,MINUS
          byte   OVER,_WORD,$02,00,_AND,ZEQ,ZEQ,_SHL1,PLUS
          byte   SWAP,_SHR1,_BYTE,$FF,_AND,SWAP
          byte   EXIT

' ( bytecode -- ) append this bytecode to next free code location + append EXIT (without counting)
BCOMP
          byte   REG,codes,WFETCH,CSTORE,_1,REG,codes,WPLUSST
          byte   _BYTE,EXIT,REG,codes,WFETCH,CSTORE
          byte   EXIT
```

```
' ( atradr -- ) compile bytecodes according to attribute
BCOMPILE
          byte   CFETCHINC,_3,_AND
          byte   DUP,ZEQ,_IF,05,DROP,CFETCH,XCALL,xBCOMP,EXIT
          byte   DUP,_2,EQ,_IF,08,DROP,CFETCHINC,XCALL,xBCOMP,CFETCH,XCALL,xBCOMP,EXIT
          byte   DROP2,EXIT

' byte aligned variable
CVARIABLE byte   _BYTE,",",REG,delim,CSTORE
          byte   XCALL,xCREATE,_1,XCALL,xALLOT
          byte   REG,delim+1,CFETCH,_BYTE,",",EQ,ZEQ,_UNTIL,@cva01-@CVARIABLE
cva01     byte   BL,REG,delim,CSTORE,EXIT



' word aligned variable
WVARIABLE byte   _BYTE,",",REG,delim,CSTORE
          byte   REG,codes,WFETCH,_1,_ANDN,INC,REG,codes,WSTORE ' align to a byte before a long address
          byte   XCALL,xCREATE,_2,XCALL,xALLOT
          byte   REG,delim+1,CFETCH,_BYTE,",",EQ,ZEQ,_UNTIL,@wva01-@WVARIABLE
wva01     byte   BL,REG,delim,CSTORE,EXIT

' long aligned variable
VARIABLE  byte   _BYTE,",",REG,delim,CSTORE
          byte   REG,codes,WFETCH,_3,_ANDN,_3,PLUS,REG,codes,WSTORE        ' align to a byte before a long address
          byte   XCALL,xCREATE,_4,XCALL,xALLOT
          byte   REG,delim+1,CFETCH,_BYTE,",",EQ,ZEQ,_UNTIL,@va01-@VARIABLE
va01      byte   BL,REG,delim,CSTORE,EXIT
```

```
' long aligned table - no memory allocated
TABLE     byte   REG,codes,WFETCH,_3,_ANDN,_3,PLUS,REG,codes,WSTORE        ' align to a byte before a long address
          byte   XCALL,xCREATE,EXIT

' GRAB ( -- ) \ IMMEDIATE
GRAB
          byte   PUSH1,EXIT,XCALL,xBCOMP                                   ' append an EXIT
          byte   XCALL,xHERE,DUP,REG,codes,WSTORE,ACALL                    ' execute and release preceding code in text line
          byte   EXIT


' 1234 CONSTANT <name>
ACONSTANT
          byte   XCALL,xGRAB                                               ' process literal
cn01      byte   REG,codes,WFETCH,_3,_ANDN,_3,PLUS,REG,codes,WSTORE        ' align to a byte before a long address
          byte   XCALL,xCREATE,MINUS1,XCALL,xALLOT                         ' create and step back to override VARB
          byte   _BYTE,CONL,XCALL,xBCOMP                                   ' compile a CONL instead
          byte   REG,codes,WFETCH,INC,STORE
          byte   _4,XCALL,xALLOT,EXIT

ATO
          byte   XCALL,xGRAB,XCALL,xTICK,INC,STORE,EXIT

'  create a name in the dictionary
CREATEWORD
          byte   XCALL,xGETWORD                                            ' ( -- str )
```

```
CREATESTR
          byte   REG,names,WFETCH,REG,names+2,WSTORE                        ' backup names ptr (used to change fixed fields easily)
          byte   _BYTE,hd+xc,XCALL,xPUTCHARPL                               ' add attribute byte (with smudge bit set)
          byte   REG,codes,WFETCH
          byte   XCALL,xAddACALL,XCALL,xPUTCHARPL,XCALL,xPUTCHARPL          ' write opcode + index bytecode sequence

          byte   DUP,DEC,CFETCH                                             ' ( str cnt )
          byte   DUP,NEGATE,REG,names,WPLUSST                               ' ( str cnt ) update names ptr by count
          byte   REG,names,WFETCH,SWAP,XCALL,xCMOVE                         ' copy it across
          byte   REG,names,WFETCH,DUP,XCALL,xSTRLEN                         '
          byte   MINUS1,REG,names,WPLUSST
          byte   SWAP,DEC,CSTORE
          byte   REG,names,WFETCH,XCALL,xXCALLS,_WORD,$04,10,PLUS,XCALL,xLT
          byte   _IF,@crw05-@crw04
crw04     byte   XCALL,xPRTSTR,"  Warning!!!, Dictionary space full!!! ",0
          byte   XCALL,xERROR
crw05     byte   EXIT

' CREATE <name> - Create a name in the dictionary and also a code entry
CREATE    byte   REG,createvec,WFETCH,QDUP,_IF,01,AJMP                      ' execute extended or user CREATE?
          byte   XCALL,xCREATEWORD                                          '
          byte   _BYTE,VARB,XCALL,xBCOMP,_0                                 ' set default bytecode as a VARIABLE

' ALLOT ( bytes -- )
ALLOT     byte   REG,codes,WPLUSST
ALLOCATED
          byte   REG,codes,WFETCH,REG,here,WSTORE
          byte   EXIT

' C, ( n -- )
CCOMP
          byte   XCALL,xGRAB
cc01      byte   XCALL,xBCOMP,XCALL,xALLOCATED,EXIT
' W, ( n -- )
WCOMP
          byte   XCALL,xGRAB
wc01      byte   DUP,XCALL,xBCOMP,_8,_SHR,XCALL,xBCOMP,XCALL,xALLOCATED,EXIT

' , ( n -- )
LCOMP
          byte   XCALL,xGRAB
          byte   _4,FOR,DUP,XCALL,xBCOMP,_8,_SHR,forNEXT
          byte   DROP,XCALL,xALLOCATED,EXIT


ATSMUDGE  byte   _BYTE,sm,REG,names,WFETCH,XCALL,xNFACFA,DEC,EXIT

' Create a new entry in the dictionary and also in the XCALLS table but also prevent any execution of code
' at an <enter> which would otherwise normally occur.
' unsmudge any previous name in case this is a fall-through.
' : <name>
COLON     byte   XCALL,xATSMUDGE,CLR                                              ' unsmudge any previous definition
          byte   XCALL,xCREATE                                  ' this forms an XCALL,index to this new definition
          byte   XCALL,xATSMUDGE,SET
          byte  MINUS1,XCALL,xALLOT
          byte   _BYTE,defining,REG,flags,SET,EXIT

' Update "here" pointer to point to current free position which "codes" pointer is now at
' Also unsmudge the headers tag
' ;
ENDCOLON  byte   _BYTE,EXIT,XCALL,xBCOMP
          byte   _BYTE,defining,REG,flags,CLR,XCALL,xALLOCATED
          byte   XCALL,xATSMUDGE,CLR,EXIT


' [COMPILE]
COMPILES  byte   _BYTE,comp,REG,flags,SET,EXIT

                                { ****** CONSOLE INPUT HANDLERS ****** }

{
Replaced traditional parse function with realtime stream parsing
Each word is acted upon when a delimiter is encountered and this also allows for
interactive error checking and even autocompletion.
}

SCRUB     byte   XCALL,xHERE,REG,codes,WSTORE
          byte   _0,REG,wordcnt,CSTORE,_0,REG,wordbuf,CSTORE
          byte   _BYTE,$0D,REG,delim+1,CSTORE
          byte   _BYTE,$0D,XCALL,xEMIT,_BYTE,$40,FOR,_BYTE,"-",XCALL,xEMIT,forNEXT
          byte   XCALL,xCR,EXIT
```

```
' ( ch -- ) write a character into the next free position in the word buffer
PUTCHAR    byte   REG,wordcnt,DUP,CFETCH,SWAP,INC,PLUS,CSTORE,EXIT
PUTCHARPL  byte   XCALL,xPUTCHAR,REG,wordcnt,DUP,CFETCH,INC
           byte   _BYTE,wordsz,XOP,pUDIVMOD,DROP,SWAP,CSTORE,EXIT

' As characters are accepted from the input stream, checks need to be made for delimiters,
' editing commands etc.
doCHAR     ' ( char -- flg ) Process char into wordbuf and flag true if all done
           byte   DUP,ZEXIT                                          ' NULL - ignore
           '
           byte   DUP,REG,lasttwo,DUP,CFETCH,OVER,INC,CSTORE,CSTORE    ' keep a track of this and the last character
           byte   DUP,REG,delim+1,CSTORE                             ' delimiter is always last character
           '''
           byte   _BYTE,$18,OVER,EQ,_IF,03,DROP,_TRUE,EXIT           ' ^X reeXecute previous line
           byte   _3,OVER,EQ,_IF,02,XCALL,xREBOOT                    ' ^C RESET
           byte   _4,OVER,EQ,_IF,05,DROP,XCALL,xDEBUG,_FALSE,EXIT     ' ^D DEBUG
           byte   _2,OVER,EQ,_IF,09,DROP,_0,_WORD,$80,00,XCALL,xDUMP,_FALSE,EXIT      ' ^B Block dump
           byte   _BYTE,$1A,OVER,EQ,REG,lasttwo+1,CFETCH,_BYTE,$1A,EQ,_AND
           byte   _IF,07,DROP,XCALL,xCOLDST,XCALL,xSCRUB,_FALSE,EXIT   ' ^Z^Z cold start
           byte   _BYTE,$1B,OVER,EQ,_IF,@doc2-@doc1                  ' ESC will cancel line
doc1       byte   DROP,XCALL,xSCRUB,_TRUE,EXIT
           '
doc2       byte   PUSH1,$09,OVER,EQ,_IF,03,XCALL,xEMIT,BL            ' TAB - substitute with a space
           byte   PUSH1,$1C,OVER,EQ,_IF,@dc0-@ml1                    ' ^| - multi-line interactive
ml1        byte   XCALL,xEMIT,XCALL,xCR,BL
           '''
dc0        byte   PUSH1,$0D,OVER,EQ,_IF,@dc1-@cr1                    ' CR
cr1        byte   DROP,_TRUE,EXIT                                    ' CR - Return & indicate completion
           '
dc1        byte   DUP,_BYTE,$7F,EQ,OVER                               ' allow DEL as a backspace (tablet keyboards)
           byte   _8,EQ,_OR,_IF,@tk2-@bksp1                          ' BKSP - null out last char
bksp1      byte   REG,wordcnt,CFETCH,_IF,@bksp3-@bksp2               ' don't backspace on empty word
bksp2      byte   XCALL,xEMIT,XCALL,xSPACE,_8,XCALL,xEMIT
           byte   MINUS1,REG,wordcnt,CPLUSST,_0,XCALL,xPUTCHAR       ' null previous char
           byte   _FALSE,EXIT
           '                        '
bksp3      byte   _7,XCALL,xEMIT,DROP,_FALSE,EXIT                    ' can't backspace anymore, bell
           '
tk2        byte   PUSH1,$0A,OVER,EQ,_IF,03,DROP,_FALSE,EXIT          ' LF - discard
           '
           byte   REG,delim,CFETCH,OVER,EQ,OVER,BL,EQ,_OR,_IF,@tk5-@adelim   ' delimiter? (always accept a blank)
adelim     ''byte DUP,REG,delim+1,CSTORE                             ' remember which delimter did this
           byte   XCALL,xEMIT,REG,wordcnt,CFETCH,EXIT                ' true if trailing delimiter - all done
           '
tk5        ' otherwise build text in wordbuf - null terminated with a preceding count .....
           byte   _BYTE,echo,REG,flags,XCALL,xSETQ,_IF,03,DUP,XCALL,xEMIT
           byte   XCALL,xPUTCHARPL                                   ' put a character into the word buffer
           byte   _FALSE,EXIT

' Build a delimited word and return immediately upon a valid delimiter
GETWORD    ' ( -- str ) Build a text word from character input into wordbuf for wordcnt
           byte   REG,wordcnt,PUSH1,33,_0,XCALL,xFILL                'Erase the word buffer & preceding count
gwlp       byte   XCALL,xKEY
           byte   XCALL,xdoCHAR
           byte   _UNTIL,@gw1-@gwlp                                  'continue building the next word
gw1        byte   REG,wordbuf,EXIT

{ Example of last entry in dictionary "COLD"  04 43 4F 4C 44 82  BF  A4  00 00
                                              cnt C  O  L  D  atr bc1 bc2 <end>
}
{
''
'' Compare a null-terminated source string with a dictionary string which is 8th bit terminated.
'' This will always force a mismatch after which one is checked for a null while the other is checked
'' for the 8th bit and if verified then a match has been found.
'' The dict pointer is advanced to point to the end of the dict string on the 8th bit termination which
'' is the attribute byte as in: byte "CMPSTR",$80,CMPSTR
''
CMPSTR     ' ( src dict -- src dict+ flg ) Compare strings at these two addresses
           byte   XX,_IF,03,XOP,pRUNMOD,EXIT
           byte   OVER,SWAP
cmplp      byte   OVER,CFETCH,OVER,CFETCH,EQ,_IF,@nomatch-@cmps1
cmps1      byte   INC,SWAP,INC,SWAP,_AGAIN,@nomatch,@cmplp
nomatch    byte   OVER,CFETCH,

_CMPSTR                  call     #_PUSHX                  ' make room for a flag
                         mov      R2,tos+1
                         mov      X,tos+2                  ' X = source
cmpstrlp                 rdbyte   R0,X                     ' read in a character from the source
                         rdbyte   R1,R2                    ' read in from the dictionary
                         cmp      R1,R0 wz                 ' are they the same?
```

```
                if_nz   jmp     #nomatch
                        add     X,#1                    ' so far, so good, try next
                        add     R2,#1                   ' updates the dict pointer on the stack too
                        jmp     #cmpstrlp               ' keep at it
nomatch                 cmp     R0,#0 wz                ' was the src null terminated?
                        xor     R1,#$80
                if_z    test    R1,#$80 wz              ' set z flag if dict 8th bit set
                        jmp     #SETZ
}

SEARCH
          byte  DUP,REG,corenames,WFETCH,XCALL,xFINDSTR
          byte  DUP,_IF,03,SWAP,DROP,EXIT
          byte  DROP,REG,names,WFETCH                                   ' from dictionary start

' ( src -- nfaptr | false ) Try to find the string in the dictionary(s) using CMPSTR to help (ignore smudged entries)
FINDSTR
fstlp     byte  XOP,pCMPSTR                                             ' ( src dict flg )
          byte  _IF,@nxtword-@fst1                                     ' found it  ( src dict )
fst1      byte  DUP,XCALL,xNFACFA,DEC,CFETCH
          byte  _BYTE,sm,_AND,ZEQ,_IF,@nxtword-@fst0
fst0      byte  SWAP,DROP,EXIT                                          ' ( nfaptr ) found
          '
          ' Skip the attribute byte and codes and test for end of dictionary (entry = 00)
nxtword   ' ( src dict ) advance past atr+codes to try next.  (atr(1),bytecode)
          byte  CFETCHINC,PLUS,_3,PLUS
'nwlp     byte  CFETCHINC,PUSH1,$80,_AND,_UNTIL,@nw1-@nwlp
nw1                                                                     ' ( src dict ) dict points to bc1
'         byte  INC,INC
          byte  DUP,CFETCH,ZEQ,_UNTIL,@fst2-@fstlp
          ' end of dictionary reached
fst2      byte  REG,findvec,WFETCH,QDUP,_IF,01,AJMP
fst3      byte  DROP2,_FALSE,EXIT


' The CFA is the address of the 2 bytecodes stored in the header that are executed or compiled
' Typically these bytecodes will be in the form of "XCALL,xWord"
' or in the case of COG words such as DUP "DUP,EXIT" where only DUP is compiled
' NFA>CFA ( nfa -- cfa )  BEGIN C@++ $7F > UNTIL ;
NFACFA    byte  CFETCHINC,_BYTE,$7F,GT,_UNTIL,06,EXIT

' : >PFA  ( cfa -- pfa )
PFA       byte  DUP,DEC,CFETCH,_BYTE,$80,EQ,_IF,02,CFETCH,EXIT ' COG BYTECODE - just return with it
          ' but this could be a two bytecode sequence rather than a call

          byte  XCALL,xTOVEC
          byte  WFETCH,EXIT

' >VEC ( cfa -- vecptr )
TOVEC     byte  DUP,INC,CFETCH,_SHL1,_SHL1,XCALL,xXCALLS,PLUS  ' ( cfa xcallptr )
          byte  SWAP,CFETCH,_BYTE,VCALL,MINUS                           ' calculate which vector XYZ or V we are using
          byte  _3,_XOR                                                 ' ( xcallptr mod )
          ' ( ptr indexh ) 0 = XCALL, 1 = YCALL, 2 = ZCALL, 3 = VCALL
          byte  SWAP,OVER,_1,_AND,_SHL1,PLUS                            ' point to high or low vector word
          byte  SWAP,_2,_AND,_IF,04,_WORD,$04,00,PLUS
          byte  EXIT


' NFA'
NFATICK   byte  XCALL,xGETWORD,DEC,XCALL,xSEARCH,EXIT
_NFATICK  byte  XCALL,xNFATICK,XCALL,xLITCOMP,EXIT

' ' <name>  ( -- pfa ) Find the address of the following word - zero if not found or it's pfa
TICK      byte  XCALL,xNFATICK,DUP,ZEXIT,XCALL,xNFACFA,XCALL,xPFA,EXIT

ATICK     byte  XCALL,xTICK
          byte  XCALL,xLITCOMP,EXIT

SETPOLL   byte  XCALL,xTICK,REG,keypoll,WSTORE,EXIT
{
WORDS     byte  REG,names,WFETCH
wdlp
          byte  DUP,CFETCH,ZEQ,DUP,_IF,06,SWAP,INC,DUP,CFETCH,ZEQ,ROT,_AND,_IF,@wd03-@wd02
wd02      byte  DROP,XCALL,xCR,EXIT

wd03      byte  XCALL,xCR,DUP,XCALL,xPRTWORD,XCALL,xPRTSTR,": ",0
          byte  INC,DUP,XCALL,xNFACFA,DEC,DUP,_3
          byte  ADO,I,CFETCH,XCALL,xPRTBYTE,XCALL,xSPACE,LOOP
          byte  _3,PLUS,SWAP,XCALL,xPSTR,XCALL,xSPACE,_AGAIN,@wd04-@wdlp
wd04
}
```

```
' AUTORUN <name>     - Setup Tachyon to AUTORUN the word on reset - an invalid or no name will disable AUTORUN
' <name> must be a valid user word so it must be an XCALL
AUTORUN    byte   XCALL,xTICK
' AUTO! ( addr -- ) Set the AUTORUN vector
AUTOST
setauto    byte   REG,autovec,WSTORE,EXIT



' FREE ( -- free ) Read free memory from Spin header and round up to a 64 byte page (to suit EEPROM)
'FREE      byte   _BYTE,10,WFETCH,_BYTE,$80,PLUS,_BYTE,$3F,_ANDN,EXIT
FREE       byte   _WORD,(@last+s)>>8,@last+s,_BYTE,$80,PLUS,_BYTE,$3F,_ANDN,EXIT

' correct the count byte in each entry in the dictionary as the precompiled version leaves a dummy count (too mind-numbing)
FIXDICT    byte   _WORD,(@dictionary+s)>>8,@dictionary+s
fdlp       byte   DUP,INC,XCALL,xSTRLEN,OVER,CSTORE                          ' calc length and set count byte
           byte   DUP,CFETCH,_4,PLUS,PLUS,DUP,CFETCH,ZEQ,_UNTIL,@fd01-@fdlp
fd01       byte   DROP,EXIT

' COLD     Force factory defaults
COLDST     byte   XCALL,xPRTSTR," Cold start - no user code - setting defaults ",$0D,$0A,0
           byte   XCALL,xFREE,DUP,REG,here,WSTORE,REG,here-2,WSTORE           ' free memory
           byte   _WORD,(@rxbuffers+s)>>8,@rxbuffers+s,REG,rxptr,WSTORE       ' setup saved receive buffer address
           byte   _WORD,(@dictionary+s)>>8,@dictionary+s
           byte   DUP,REG,corenames,WSTORE,DEC
           byte   DUP,REG,names,WSTORE,REG,names-2,WSTORE                     ' Reset dictionary pointer
           byte   _0,REG,autovec,WSTORE,_0,REG,keypoll,WSTORE
           byte   XCALL,xFIXDICT

           byte   _BYTE,xLAST,DUP,_SHL1,_SHL1,XCALL,xXCALLS,PLUS ' find first free XCALL memory location
           byte   _WORD,$02,00,ROT,MINUS,_SHL1,_SHL1,_0,XCALL,xFILL           ' clear all user XCALLs
           byte   XCALL,xXCALLS,INC,INC,_WORD,$08,00                          'clear YCALLs (interleaved with XCALLS)
           byte   ADO,_0,I,WSTORE,_4,PLOOP
           byte   REG,tasks,_BYTE,tasksz*8,_0,XCALL,xFILL                     ' initialize 8 task words
           ' VGA
'          byte   _WORD,(@colors+s)>>8,@colors+s,REG,colorptr,STORE           ' init pointers to VGA colors and pixels
'          byte   _WORD,Pixels>>8,Pixels,XCALL,x_PIXELS

           byte   _WORD,$A5,$5A,REG,cold,WSTORE
           byte   EXIT

' Discard the current line
DISCARD
dslp       byte   XCALL,xKEYQ,_AND,ZEQ,_UNTIL,@ds01-@dslp
ds01       byte   _BYTE,100,XCALL,xms,XCALL,xKEYQ,_AND,ZEQ,_UNTIL,@ds02-@dslp
ds02       byte   EXIT

' TASK ( cog -- addr ) Return with address of task control register in "tasks"
TASK       byte   _3,_SHL,REG,tasks,PLUS,EXIT
{
'RUN ( pfa cog -- )
RUN        byte   XCALL,xTASK,WSTORE,EXIT
}
           org             ' align the label - needs to be passed in 14-bit PAR register
' idle loop for other Tachyon cogs
IDLE       byte   XCALL,xInitStack
           byte   _1,_COGID,XCALL,xTASK,INC,INC,CSTORE                        ' task+2 = 1 to indicate Tachyon running
idlp       byte   _8,XCALL,xms                                                ' do nothing for a bit - saves power
           byte   _COGID,XCALL,xTASK,WFETCH                                   ' fetch cog's task variable
           byte   QDUP,_UNTIL,@id00-@idlp                                     ' until it is non-zero
id00       byte   DUP,_8,_SHR,_IF,01,ACALL                                    ' Execute but ignore if task address is only 8-bits
           byte   _0,_COGID,XCALL,xTASK,WSTORE                                ' clear run address only if it has returned back to idle
           byte   _AGAIN,@id01-@IDLE
id01



                    { ****** MAIN CONSOLE TERMINAL ****** }

           org     ' align the label

TERMINAL
           byte   InitRP
           byte   XCALL,xInitStack
           byte   _WORD,00,50,XCALL,xms                                        ' a little startup delay (also wait for serial cog)
           byte   _WORD,bufsize>>8,bufsize
           byte   REG,rxsz,WSTORE
           byte   _BYTE,echo,REG,flags,CSTORE                                 ' echo flag
           byte   BL,REG,delim,CSTORE,XCALL,xHEX                              ' default delimiter
'          byte   _6,_3,COGREG,COGSTORE                                        ' default bytes/line for VGA
           byte   _BYTE,txd,MASK,OUTSET
           byte   _BYTE,$00,XCALL,xEMIT,XCALL,xPRTVER                         ' VERSION with optional CLS (default null)
```

```
          byte   REG,cold,WFETCH,_WORD,$A5,$5A,EQ,ZEQ                  ' performing a check for a saved session
          byte   _IF,@warmst-@tm01                                      ' or not
tm01      byte   XCALL,xCOLDST                                          ' defaults

warmst
          byte   XCALL,xKEYQ,_AND,_1,EQ,_NOT,_IF,@termnew-@chkauto      ' abort autostart with ^A
chkauto   byte   REG,autovec,WFETCH,QDUP,_IF,@termnew-@runauto          ' check for an AUTORUN
runauto   byte   ACALL                                                  ' and execute it
CONSOLE
termnew   byte   InitRP                                                 ' init return stack in case (limited)
          byte   XCALL,xSCRUB
termcr    byte   REG,prompt,WFETCH,QDUP,_IF,01,ACALL                    ' execute user prompt code
          byte   XCALL,xHERE,REG,codes,WSTORE                           ' reset temporary code compilation pointer
          '
          ' Main console loop - read a word and process
termlp
          byte   XCALL,xGETWORD                                         ' Read a word from input stream etc

          byte   CFETCH,ZEQ,_IF,@trm1-@trm2                             ' ignore empty string
trm2      byte   REG,delim+1,CFETCH,_BYTE,$18,EQ,_NOT,_IF,@execinp-@trm3  ' if ^X then repeat last line
trm3      byte   REG,delim+1,CFETCH,_BYTE,$0D,EQ,_NOT,_IF,@chkeol-@trm1
          '
trm1      ' Preprocess prefixed numbers #$%
          byte   REG,wordbuf,CFETCH,_BYTE,"#",_BYTE,"%",XCALL,xWITHIN   ' Numeric prefixes?
          byte   REG,wordbuf-1,CFETCH,_2,GT,_AND                        ' and more than 2 characters? (inc term)
          byte   REG,wordbuf-1,DUP,CFETCH,PLUS,CFETCH                   ' and last char is a digit or hex digit?
          byte   DUP,_BYTE,"0",_BYTE,"9",XCALL,xWITHIN                  ' decimal digit?
          byte   SWAP,_BYTE,"A",_BYTE,"F",XCALL,xWITHIN,_OR,_AND        ' hex digit?
          byte   ZEQ,_IF,@tryanum-@trm4                                 ' good, process this as a number now @tryanum

          ' Search the dicitonary for a match
trm4      byte   REG,wordbuf,DEC,XCALL,xSEARCH                          ' try and find that word in the dictionary(s)
          byte   QDUP,_IF,@notfound-@foundword                          ' found it

foundword       ' found the word in the dictionary - compile or execute?
          byte   XCALL,xNFACFA,DEC
          byte   PUSH1,im,OVER,XCALL,xSETQ
          byte   _BYTE,comp,REG,flags,XCALL,xSETQ,ZEQ,_AND              ' Immediate word? and comp flag off
          byte   _IF,@compword-@immed                                   '
immed     byte   INC,CFETCHINC,SWAP,CFETCH,XCALL,xEXECUTE               ' Fetch and execute code immediately
          byte   _ELSE,@chkeol-@compword
compword  byte   XCALL,xBCOMPILE                                        ' or else compile the bytecode(s) for ths word
          byte   _BYTE,comp,REG,flags,CLR
          ' END OF LINE CHECK
chkeol    byte   REG,delim+1,CFETCH,PUSH1,$0D,EQ                        ' Was this the end of line?
          byte   DUP,_IF,@eol01-@eol00
eol00     byte   REG,accept,WFETCH,ZEQ,_IF,02,XCALL,xSPACE               ' Yes, put a space between any user input and response
          byte   _BYTE,linenums,REG,flags,XCALL,xSETQ,_IF,@eol01-@prtline
prtline   byte   _BYTE,$0D,XCALL,xEMIT
          byte   REG,linenum,WFETCH,XCALL,xPRTDEC,XCALL,xSPACE
          byte   _1,REG,linenum,WPLUSST
eol01     byte   DUP,_BYTE,defining,REG,flags,XCALL,xSETQ,_AND  ' and are we in a definition or interactive?
          byte   _IF,02,XCALL,xCR                                       ' If not interactive then CRLF (no other response)
          byte   _BYTE,defining,REG,flags,XCALL,xSETQ,ZEQ,_AND  ' do not execute if still defining
          byte   _UNTIL,@execs-@termlp                                  ' wait until CR to execute compiled codes
          ' EXECUTE CODE from user input
execs     byte   PUSH1,EXIT,XCALL,xBCOMP                                ' done - append an EXIT (minimum action on empty lines)
execinp   byte   XCALL,xHERE,ACALL                                      ' execute from beginning
          byte   REG,accept,WFETCH
          byte   QDUP,_IF,03,ACALL,_ELSE,02,XCALL,xOK
'         byte   XCALL,xOK                                              ' echo the "ok" when done
          byte   _AGAIN,@notfound-@termcr

notfound  ' NOT FOUND - before trying to convert to a number check encoding for ASCII literals (^ and ")
          ' Attempt to process this word as a number
tryanum   byte   REG,wordbuf,XCALL,xNUMBER,_IF,@unknown-@compnum
compnum   byte   XCALL,xLITCOMP
          byte   _AGAIN,@unknown-@chkeol                                ' is it a number? ( value digits )
unknown   byte   REG,unum,WFETCH,QDUP,_IF,03,ACALL,_AGAIN,@un01-@chkeol
un01
          byte   _BYTE,echo,REG,flags,XCALL,xSETQ,ZEQ,_IF,@un03-@un00
un00      byte   XCALL,xSPACE,REG,wordbuf,XCALL,xPSTR                   ' Spit out offending word
          byte   _WORD,$02,14,XOP,pEMIT,_ELSE,@un04-@un03
un03      byte   XCALL,xPRTSTR," ??? ",7,0                              ' ???
un04      byte   XCALL,xERROR
un02      byte   _AGAIN,@trmend-@termlp
trmend

ERROR
          byte   _1,REG,errors,WPLUSST                                  ' count errors
          byte   _7,XCALL,xEMIT
          byte   XCALL,xDISCARD,EXIT
```

```
' =PIXELS ( addr -- ) Set starting address of pixel buffer both in Tachyon cog and at interpreted level
'_PIXELS   byte   DUP,REG,pixelptr,WSTORE,_WORD,(pixeladr>>8),(pixeladr),COGSTORE,EXIT


' TACHYON - used to verify that source code is intended for Tachyon and also to reset load stats
_TACHYON   byte   XCALL,xPRTVER
           byte   _0,REG,errors,WSTORE
           byte   XCALL,xHERE,REG,here-2,WSTORE                              ' remember code position
           byte   _0,REG,linenum,WSTORE,_BYTE,linenums,REG,flags,SET
           byte   REG,names,WFETCH,REG,names-2,WSTORE                        ' backup dictionary pointer
           byte   XCALL,xNEWCNT,EXIT

' VER ( -- verptr )
GETVER     byte   _WORD,(@version+s)>>8,@version+s,EXIT


PRTVER     byte   XCALL,xPRTSTR,$0D,$0A
           byte     "  Propeller .:.:--TACHYON--:.:. Forth V",0
           byte   XCALL,xGETVER,DUP,FETCH,XCALL,xPRTDEC
           byte   _BYTE,".",XCALL,xEMIT,_4,PLUS,WFETCH,XCALL,xPRTDEC
           byte   XCALL,xCR,EXIT
```

```
'-------------------------------------------------------------------------------------------------------------
```

```
last

           byte   0[$5200-$]
```

{ DICTIONARY in RAM and EEPROM* }

Although this dictionary is loaded into RAM automatically by the Prop bootloader it is not used by TACHYON and is free to be used for other purposes. Instead, the copy of the dictionary that is in EEPROM is searched and this is also where new names are appended to.

** Revision: As the image of the cog program's DAT section in RAM can be reused after boot then this would be a good place to copy the kernel's dictionary from where it is in RAM so it can reuse up some 2KB. Now the kernel's dictionary can be searched in RAM rather than EEPROM! There should be enough room left for another 200 or so entries.*

Search methods:

Structure:
1 - Count byte - This speeds up searching the dictionary both in comparing and in jumping to the next string
2- Name string
3- Attribute byte (8th bit set also terminates name string)
4- 1st bytecode, 2nd bytecode

Dictionary entries do not need a link field as they are bunched together one after another and it is very easy to find the next entry by scanning forwards and looking for the attribute byte which will have the msb set then jumping 3 bytes. A name field that begins with a null indicates end of dictionary (or link to another)

CON

```
' Dictionary header attribute flags
hd         = |<7           ' indicates this is a an attribute (delimits the start of a null terminated name)
co         = |<6           'lexicon compile only bit
im         = |<5           'lexicon immediate bit
ex         = |<4           'exec
sm         = |<3           'smudge bit - set to deactivate word during definition
```

```
sq        = |<2           'indicates the bytecode is a sequence of two instructions (as opposed to an XCALL+)

' code attributes 00 = single bytecode, 02 = XCALL bytecode (2 bytes), 03 = WCALL bytecode (3 bytes)
xc        = |<1           'XCALL bytecode
ac        = xc+|<0        'WCALL - 2 byte address - interpret header CFA as an absolute address

DAT

{ This is an 8th bit terminated string using the attribute byte so it saves one byte per entry plus it simplfies the string compare
function. Searching still proceeds from lower memory to higher memory
}

                                        { ****** DICTIONARY ******  }

'         byte  0
dictionary

          '     CNT,NAME              ATR           CODES

          byte  $20,"DUP",            hd,           DUP,EXIT
          byte  $20,"OVER",           hd,           OVER,EXIT
          byte  $20,"DROP",           hd,           DROP,EXIT
          byte  $20,"2DROP",          hd,           DROP2,EXIT
          byte  $20,"SWAP",           hd,           SWAP,EXIT
          byte  $20,"ROT",            hd,           ROT,EXIT

          byte  $20,"0",              hd,           _0,EXIT
          byte  $20,"1",              hd,           _1,EXIT
          byte  $20,"2",              hd,           _2,EXIT
          byte  $20,"3",              hd,           _3,EXIT
          byte  $20,"4",              hd,           _4,EXIT
          byte  $20,"5",              hd,           _5,EXIT
          byte  $20,"6",              hd,           _6,EXIT
          byte  $20,"7",              hd,           _7,EXIT
          byte  $20,"8",              hd,           _8,EXIT
          byte  $20,"ON",             hd,           MINUS1,EXIT
          byte  $20,"TRUE",           hd,           MINUS1,EXIT
          byte  $20,"-1",             hd,           MINUS1,EXIT
          byte  $20,"BL",             hd,           BL,EXIT
          byte  $20,"16",             hd,           _16,EXIT
          byte  $20,"FALSE",          hd,           _0,EXIT
          byte  $20,"OFF",            hd,           _0,EXIT

          byte  $20,"1+",             hd,           INC,EXIT
          byte  $20,"1-",             hd,           DEC,EXIT
          byte  $20,"+",              hd,           PLUS,EXIT
          byte  $20,"-",              hd,           MINUS,EXIT

          byte  $20,"ADO",            hd,           ADO,EXIT
          byte  $20,"DO",             hd,           DO,EXIT
          byte  $20,"LOOP",           hd,           LOOP,EXIT
          byte  $20,"+LOOP",          hd,           PLOOP,EXIT
          byte  $20,"FOR",            hd,           FOR,EXIT
          byte  $20,"NEXT",           hd,           forNEXT,EXIT
          byte  $20,"!RP",            hd,           InitRP,EXIT

'         byte  $20,"<BEGIN",         hd,           BEGINS,EXIT
'         byte  $20,"AGAIN>",         hd,           AGAINS,EXIT
'         byte  $20,"UNTIL>",         hd,           UNTILS,EXIT

          byte  $20,"INVERT",         hd,           INVERT,EXIT
          byte  $20,"AND",            hd,           _AND,EXIT
          byte  $20,"ANDN",           hd,           _ANDN,EXIT
          byte  $20,"OR",             hd,           _OR,EXIT
          byte  $20,"XOR",            hd,           _XOR,EXIT

          byte  $20,"ROL",            hd,           _ROL,EXIT
          byte  $20,"ROR",            hd,           _ROR,EXIT
          byte  $20,"SHR",            hd,           _SHR,EXIT
          byte  $20,"SHL",            hd,           _SHL,EXIT
          byte  $20,"2/",             hd,           _SHR1,EXIT
          byte  $20,"2*",             hd,           _SHL1,EXIT
          byte  $20,"REV",            hd,           _REV,EXIT
          byte  $20,"MASK",           hd,           MASK,EXIT

          byte  $20,"0=",             hd,           ZEQ,EXIT
          byte  $20,"NOT",            hd,           ZEQ,EXIT
          byte  $20,"=",              hd,           EQ,EXIT
          byte  $20,">",              hd,           GT,EXIT

          byte  $20,"C@",             hd,           CFETCH,EXIT
          byte  $20,"W@",             hd,           WFETCH,EXIT
```

```
        byte    $20,"@",                hd,             FETCH,EXIT
        byte    $20,"C+!",              hd,             CPLUSST,EXIT
        byte    $20,"C!",               hd,             CSTORE,EXIT
        byte    $20,"C@++",             hd,             CFETCHINC,EXIT
        byte    $20,"W+!",              hd,             WPLUSST,EXIT
        byte    $20,"W!",               hd,             WSTORE,EXIT
        byte    $20,"+!",               hd,             PLUSST,EXIT
        byte    $20,"!",                hd,             STORE,EXIT
        byte    $20,"BIT!",             hd,             BIT,EXIT
        byte    $20,"SET",              hd,             SET,EXIT
        byte    $20,"CLR",              hd,             CLR,EXIT
        'byte   $20,"IC!",              hd,             ICSTORE,EXIT

        byte    $20,"NIP",              hd+xc+sq,       SWAP,DROP
        byte    $20,"U/MOD",            hd+xc+sq,       XOP,pUDIVMOD
        byte    $20,"UM*",              hd,             UMMUL,EXIT
        byte    $20,"?NEGATE",          hd,             QNEGATE,EXIT
        byte    $20,"NEGATE",           hd,             NEGATE,EXIT

        byte    $20,"IN@",              hd,             PFETCH,EXIT
        byte    $20,"P@",               hd,             PFETCH,EXIT
        byte    $20,"OUT!",             hd,             PSTORE,EXIT
        byte    $20,"P!",               hd,             PSTORE,EXIT
        byte    $20,"CLOCK",            hd,             CLOCK,EXIT
        byte    $20,"OUTSET",           hd,             OUTSET,EXIT
        byte    $20,"OUTCLR",           hd,             OUTCLR,EXIT
        byte    $20,"OUTPUTS",          hd,             OUTPUTS,EXIT
        byte    $20,"INPUTS",           hd,             INPUTS,EXIT
'       byte    $20,"WAITLOW",          hd,             WAITLOW,EXIT
        byte    $20,"SHROUT",           hd,             SHROUT,EXIT
        byte    $20,"SHRINP",           hd,             SHRINP,EXIT
        byte    $20,"LOADMOD",          hd+xc+sq,       XOP,pLOADMOD
        byte    $20,"RUNMOD",           hd+xc+sq,       XOP,pRUNMOD

        byte    $20,"RESET",            hd,             RESET,EXIT
        byte    $20,"0EXIT",            hd,             ZEXIT,EXIT
        byte    $20,"EXIT",             hd,             EXIT,EXIT
        byte    $20,"NOP",              hd,             _NOP,EXIT
        byte    $20,"3DROP",            hd,             DROP3,EXIT
        byte    $20,"?DUP",             hd,             QDUP,EXIT
        byte    $20,"3RD",              hd,             THIRD,EXIT
        byte    $20,"4TH",              hd,             FOURTH,EXIT


        byte    $20,"SFR@",             hd,             SPRFETCH,EXIT
        byte    $20,"SPR@",             hd,             SPRFETCH,EXIT
        byte    $20,"COG@",             hd,             COGFETCH,EXIT
        byte    $20,"COGREG",           hd,             COGREG,EXIT
        byte    $20,"COG!",             hd,             COGSTORE,EXIT
        byte    $20,"PASM",             hd,             PASM,EXIT

        byte    $20,"CALL",             hd,             ACALL,EXIT
        byte    $20,"JUMP",             hd,             AJMP,EXIT
        byte    $20,"BRANCH>",          hd,             POPBRANCH,EXIT

        byte    $20,">R",               hd,             PUSHR,EXIT
        byte    $20,"R>",               hd,             RPOP,EXIT
        byte    $20,">L",               hd,             PUSHL,EXIT
        byte    $20,"L>",               hd,             LPOP,EXIT

        byte    $20,"REG",              hd,             ATREG,EXIT
        byte    $20,"DELTA",            hd,             DELTA,EXIT
        byte    $20,"WAITCNT",          hd,             WAITCNTS,EXIT
        byte    $20,"(WAITPEQ)",        hd,             _WAITPEQ,EXIT
        byte    $20,"(WAITPNE)",        hd,             _WAITPNE,EXIT


        byte    $20,"(XCALL)",          hd,             XCALL,EXIT
        byte    $20,"(YCALL)",          hd,             YCALL,EXIT
        byte    $20,"(ELSE)",           hd,             _ELSE,EXIT
        byte    $20,"(IF)",             hd,             _IF,EXIT
        byte    $20,"(UNTIL)",          hd,             _UNTIL,EXIT
        byte    $20,"(AGAIN)",          hd,             _AGAIN,EXIT
        byte    $20,"(EMIT)",           hd+xc+sq,       XOP,pEMIT
        byte    $20,"(REG)",            hd,             REG,EXIT
        byte    $20,"(PUSH4)",          hd,             PUSH4,EXIT
        byte    $20,"(PUSH3)",          hd,             PUSH3,EXIT
        byte    $20,"(PUSH2)",          hd,             PUSH2,EXIT
        byte    $20,"(PUSH1)",          hd,             PUSH1,EXIT
        byte    $20,"(VAR)",            hd,             VARB,EXIT
```

```
byte   $20,"COGID",              hd,             _COGID,EXIT

byte   $20,"CMPSTR",             hd+xc+sq,       XOP,pCMPSTR


byte   $20,"LSTACK",             hd,             LSTACK,EXIT
byte   $20,"I",                  hd,             I,EXIT
```

{ INTERPRETED BYTECODE HEADERS }

```
byte   $20,"CMOVE",              hd+xc+sq,       XCALL,xCMOVE
byte   $20,"<CMOVE",             hd+xc+sq,       XCALL,xRCMOVE


byte   $20,":",                  hd+xc+im,       XCALL,xCOLON
byte   $20,"pub",                hd+xc+im,       XCALL,xCOLON
byte   $20,"pri",                hd+xc+im,       XCALL,xCOLON

byte   $20,"IF",                 hd+xc+im,       XCALL,x_IF_
byte   $20,"ELSE",               hd+xc+im,       XCALL,x_ELSE_
byte   $20,"THEN",               hd+xc+im,       XCALL,x_THEN_
byte   $20,"ENDIF",              hd+xc+im,       XCALL,x_THEN_
byte   $20,"BEGIN",              hd+xc+im,       XCALL,x_BEGIN_
byte   $20,"UNTIL",              hd+xc+im,       XCALL,x_UNTIL_
byte   $20,"AGAIN",              hd+xc+im,       XCALL,x_AGAIN_
byte   $20,"WHILE",              hd+xc+im,       XCALL,x_IF_
byte   $20,"REPEAT",             hd+xc+im,       XCALL,x_REPEAT_
byte   $20,";",                  hd+xc+im,       XCALL,xENDCOLON

byte   $20,"\",                  hd+xc+im,       XCALL,xCOMMENT
byte   $20,"''",                 hd+xc+im,       XCALL,xCOMMENT
byte   $20,"(",                  hd+xc+im,       XCALL,xBRACE
byte   $20,"{",                  hd+xc+im,       XCALL,xCURLY
byte   $20,"}",                  hd+xc+im,       _NOP,EXIT
byte   $20,"IFNDEF",             hd+xc+im,       XCALL,xIFNDEF
byte   $20,$22,                  hd+xc+im,       XCALL,x_STR_
byte   $20,$2E,$22,              hd+xc+im,       XCALL,x_PSTR_
byte   $20,"(",$2E,$22,")",      hd+xc+im,       XCALL,xPRTSTR

byte   $20,"BYTE",               hd+xc+im,       XCALL,xCVARIABLE
byte   $20,"WORD",               hd+xc+im,       XCALL,xWVARIABLE
byte   $20,"LONG",               hd+xc+im,       XCALL,xVARIABLE
byte   $20,"TABLE",              hd+xc+im,       XCALL,xTABLE
byte   $20,"CONSTANT",           hd+xc+im,       XCALL,xCONSTANT
byte   $20,"TO",                 hd+xc+im,       XCALL,xATO

byte   $20,"BCOMP",              hd+xc+im,       XCALL,xBCOMP
byte   $20,"C,",                 hd+xc+im,       XCALL,xCCOMP
byte   $20,"|",                  hd+xc+im,       XCALL,xCCOMP
byte   $20,"||",                 hd+xc+im,       XCALL,xWCOMP
byte   $20,",",                  hd+xc+im,       XCALL,xLCOMP


byte   $20,"BREAK",              hd+xc+im,       XCALL,xISEND
byte   $20,"CASE",               hd+xc+im,       XCALL,xIS
byte   $20,"SWITCH",             hd+xc,          XCALL,xSWITCH
byte   $20,"SWITCH@",            hd+xc,          XCALL,xSWITCHFETCH
byte   $20,"SWITCH=",            hd+xc,          XCALL,xISEQ
byte   $20,"SWITCH><",           hd+xc,          XCALL,xISWITHIN

byte   $20,"STACKS",             hd+xc,          XCALL,xSTACKS
byte   $20,"XCALLS",             hd+xc,          XCALL,xXCALLS
byte   $20,"REBOOT",             hd+xc,          XCALL,xREBOOT
byte   $20,"STOP",               hd+xc,          XCALL,xSTOP
byte   $20,"PAR",                hd+xc,          XCALL,x_PAR
byte   $20,"CNT",                hd+xc,          XCALL,x_CNT
byte   $20,"INA",                hd+xc,          XCALL,x_INA
byte   $20,"INB",                hd+xc,          XCALL,x_INB
byte   $20,"OUTA",               hd+xc,          XCALL,x_OUTA
byte   $20,"OUTB",               hd+xc,          XCALL,x_OUTB
byte   $20,"DIRA",               hd+xc,          XCALL,x_DIRA
byte   $20,"DIRB",               hd+xc,          XCALL,x_DIRB
byte   $20,"CTRA",               hd+xc,          XCALL,x_CTRA
byte   $20,"CTRB",               hd+xc,          XCALL,x_CTRB
byte   $20,"FRQA",               hd+xc,          XCALL,x_FRQA
byte   $20,"FRQB",               hd+xc,          XCALL,x_FRQB
byte   $20,"PHSA",               hd+xc,          XCALL,x_PHSA
byte   $20,"PHSB",               hd+xc,          XCALL,x_PHSB
byte   $20,"VCFG",               hd+xc,          XCALL,x_VCFG
byte   $20,"VSCL",               hd+xc,          XCALL,x_VSCL
```

```
        byte  $20,"SPR",              hd+xc,          XCALL,x_SPR
        byte  $20,"SPR!",             hd+xc,          XCALL,xSPRSTORE

        byte  $20,"[SSD]",            hd+xc,          XCALL,xSSD
        byte  $20,"[ESPIO]",          hd+xc,          XCALL,xESPIO
        byte  $20,"[SPIO]",           hd+xc,          XCALL,xSPIO
        byte  $20,"[SPIOD]",          hd+xc,          XCALL,xSPIOD
        byte  $20,"[SDRD]",           hd+xc,          XCALL,xSDRD
        byte  $20,"[SDWR]",           hd+xc,          XCALL,xSDWR
        byte  $20,"[PWM]",            hd+xc,          XCALL,xPWM
        byte  $20,"[PWM!]",           hd+xc,          XCALL,xPWMST
        byte  $20,"[PLOT]",           hd+xc,          XCALL,xPLOT

        byte  $20,"!SP",              hd+xc,          XCALL,xInitStack
        byte  $20,"2+",               hd+xc+sq,       _2,PLUS
        byte  $20,"2-",               hd+xc+sq,       _2,MINUS
        byte  $20,"SET?",             hd+xc,          XCALL,xSETQ
        byte  $20,"/",                hd+xc,          XCALL,xDIVIDE
        byte  $20,"2DUP",             hd+xc+sq,       OVER,OVER

        byte  $20,"*",                hd+xc+sq,       UMMUL,DROP
        byte  $20,"MIN",              hd+xc,          XCALL,xMIN
        byte  $20,"MAX",              hd+xc,          XCALL,xMAX
        byte  $20,"0<>",              hd+xc+sq,       ZEQ,ZEQ
        byte  $20,"<>",               hd+xc+sq,       EQ,ZEQ
'       byte  $20,"0>",               hd+xc+sq,       _0,GT
        byte  $20,"0<",               hd+xc,          XCALL,xZLT
        byte  $20,"<",                hd+xc,          XCALL,xLT
        byte  $20,"U<",               hd+xc,          XCALL,xULT
        byte  $20,"WITHIN",           hd+xc,          XCALL,xWITHIN
        byte  $20,"?EXIT",            hd+xc,          XCALL,xIFEXIT

        byte  $20,"BOUNDS",           hd+xc,          XCALL,xBOUNDS
        byte  $20,"LEAVE",            hd+xc,          XCALL,xLEAVE
        byte  $20,"J",                hd+xc,          XCALL,xJ
        byte  $20,"K",                hd+xc,          XCALL,xK
        byte  $20,"IX",               hd+xc,          XCALL,xIX
        byte  $20,"ERASE",            hd+xc,          XCALL,xERASE
        byte  $20,"FILL",             hd+xc,          XCALL,xFILL
        byte  $20,"ms",               hd+xc,          XCALL,xms
        byte  $20,"us",               hd+xc,          XCALL,xus
        byte  $20,"CNT@",             hd+xc+sq,       _1,SPRFETCH
        byte  $20,"NEWCNT",           hd+xc,          XCALL,xNEWCNT
        byte  $20,"LAPCNT",           hd+xc,          XCALL,xLAPCNT
        byte  $20,"OUT",              hd+xc,          XCALL,xOUT
        byte  $20,"IN",               hd+xc,          XCALL,xAIN
        byte  $20,"INPUT",            hd+xc,          XCALL,xAIN
        byte  $20,"PIN!",             hd+xc,          XCALL,xPINSTORE
        byte  $20,"PIN@",             hd+xc,          XCALL,xPINFETCH

        byte  $20,"KEY?",             hd+xc,          XCALL,xKEYQ
        byte  $20,"KEY",              hd+xc,          XCALL,xKEY

        byte  $20,"HEX",              hd+xc,          XCALL,xHEX
        byte  $20,"DECIMAL",          hd+xc,          XCALL,xDECIMAL
        byte  $20,"BINARY",           hd+xc,          XCALL,xBIN

        byte  $20,".S",               hd+xc,          XCALL,xPRTSTK
        byte  $20,"HDUMP",            hd+xc,          XCALL,xDUMP
        byte  $20,"COGDUMP",          hd+xc,          XCALL,xCOGDUMP

        byte  $20,".STACKS",          hd+xc,          XCALL,xPRTSTKS
        byte  $20,"DEBUG",            hd+xc,          XCALL,xDEBUG
        byte  $20,"EMIT",             hd+xc,          XCALL,xEMIT
        byte  $20,"CLS",              hd+xc,          XCALL,xCLS
        byte  $20,"SPACE",            hd+xc,          XCALL,xSPACE
        byte  $20,"BELL",             hd+xc,          XCALL,xBELL
        byte  $20,"CR",               hd+xc,          XCALL,xCR
        byte  $20,"SPINNER",          hd+xc,          XCALL,xSPINNER
        byte  $20,".HEX",             hd+xc,          XCALL,xPRTHEX
        byte  $20,".BYTE",            hd+xc,          XCALL,xPRTBYTE
        byte  $20,".WORD",            hd+xc,          XCALL,xPRTWORD
        byte  $20,".LONG",            hd+xc,          XCALL,xPRTLONG
        byte  $20,".",                hd+xc,          XCALL,xPRT
        byte  $20,">DIGIT",           hd+xc,          XCALL,xTODIGIT
        byte  $20,"NUMBER",           hd+xc,          XCALL,xNUMBER
        byte  $20,">UPPER",           hd+xc,          XCALL,xTOUPPER
        byte  $20,"SCRUB",            hd+xc,          XCALL,xSCRUB
        byte  $20,"GETWORD",          hd+xc,          XCALL,xGETWORD
        byte  $20,"SEARCH",           hd+xc,          XCALL,xSEARCH
        byte  $20,"FINDSTR",          hd+xc,          XCALL,xFINDSTR
```

```
        byte   $20,"NFA>CFA",            hd+xc,          XCALL,xNFACFA
        byte   $20,"EXECUTE",            hd+xc,          XCALL,xEXECUTE

        byte   $20,"V2",                 hd+xc,          XCALL,xGETVER
        byte   $20,"VER",                hd+xc,          XCALL,xGETVER
        byte   $20,".VER",               hd+xc,          XCALL,xPRTVER
        byte   $20,"TACHYON",            hd+xc,          XCALL,x_TACHYON

        byte   $20,"@PAD",               hd+xc,          XCALL,xATPAD
        byte   $20,"HOLD",               hd+xc,          XCALL,xHOLD
        byte   $20,">CHAR",              hd+xc,          XCALL,xTOCHAR
        byte   $20,"#>",                 hd+xc,          XCALL,xRHASH
        byte   $20,"<#",                 hd+xc,          XCALL,xLHASH
        byte   $20,"#",                  hd+xc,          XCALL,xHASH
        byte   $20,"#S",                 hd+xc,          XCALL,xHASHS
        byte   $20,"(STR)",              hd+xc,          XCALL,x_STR
        byte   $20,".STR",               hd+xc,          XCALL,xPSTR
        byte   $20,"STRLEN",             hd+xc,          XCALL,xSTRLEN
        byte   $20,"U.",                 hd+xc,          XCALL,xUPRT
        byte   $20,".DEC",               hd+xc,          XCALL,xPRTDEC
        byte   $20,"DISCARD",            hd+xc,          XCALL,xDISCARD
        byte   $20,"COGINIT",            hd+xc,          XCALL,xCOGINIT


' TASK REGISTERS
        byte   $20,"REG",                hd+xc,          REG,0           '
        byte   $20,"flags",              hd+xc,          REG,flags
        byte   $20,"base",               hd+xc,          REG,base
        byte   $20,"digits",             hd+xc,          REG,digits
        byte   $20,"delim",              hd+xc,          REG,delim
        byte   $20,"word",               hd+xc,          REG,wordbuf
        byte   $20,"switch",             hd+xc,          REG,uswitch
        byte   $20,"autorun",            hd+xc,          REG,autovec
        byte   $20,"keypoll",            hd+xc,          REG,keypoll
        byte   $20,"tasks",              hd+xc,          REG,tasks
        byte   $20,"unum",               hd+xc,          REG,unum
        byte   $20,"uemit",              hd+xc,          REG,uemit
        byte   $20,"ukey",               hd+xc,          REG,ukey
        byte   $20,"names",              hd+xc,          REG,names
        byte   $20,"here",               hd+xc,          REG,here
        byte   $20,"codes",              hd+xc,          REG,codes
        byte   $20,"errors",             hd+xc,          REG,errors
        byte   $20,"baudcnt",            hd+xc,          REG,baudcnt
        byte   $20,"prompt",             hd+xc,          REG,prompt
        byte   $20,"find",               hd+xc,          REG,findvec
        byte   $20,"create",             hd+xc,          REG,createvec
        byte   $20,"lines",              hd+xc,          REG,linenum


        byte   $20,"ALLOT",              hd+xc,          XCALL,xALLOT
        byte   $20,"ALLOCATED",          hd+xc,          XCALL,xALLOCATED
        byte   $20,"HERE",               hd+xc,          XCALL,xHERE
        byte   $20,"AUTO!",              hd+xc,          XCALL,xAUTOST
        byte   $20,"MARK",               hd+xc,          XCALL,xMARK
        byte   $20,"UNMARK",             hd+xc,          XCALL,xUNMARK
        byte   $20,">VEC",               hd+xc,          XCALL,xTOVEC
        byte   $20,">PFA",               hd+xc,          XCALL,xPFA
        byte   $20,"[NFA']",             hd+xc,          XCALL,xNFATICK
        byte   $20,"[']",                hd+xc,          XCALL,xTICK
' IMMEDIATE
        byte   $20,"NFA'",               hd+xc+im,       XCALL,x_NFATICK
        byte   $20,"'",                  hd+xc+im,       XCALL,xATICK
        byte   $20,"KEYPOLL",            hd+xc+im,       XCALL,xSETPOLL

        byte   $20,"AUTORUN",            hd+xc+im,       XCALL,xAUTORUN

' Building words
        byte   $20,"[COMPILE]",          hd+xc+im,       XCALL,xCOMPILES
        byte   $20,"GRAB",               hd+xc+im,       XCALL,xGRAB
        byte   $20,"LITERAL",            hd+xc+im,       XCALL,xLITCOMP

        byte   $20,"CREATE$",            hd+xc+im,       XCALL,xCREATESTR
        byte   $20,"CREATEWORD",         hd+xc+im,       XCALL,xCREATEWORD
        byte   $20,"CREATE",             hd+xc+im,       XCALL,xCREATE
        byte   $20,"CVARIABLE",          hd+xc+im,       XCALL,xCVARIABLE
        byte   $20,"WVARIABLE",          hd+xc+im,       XCALL,xWVARIABLE
        byte   $20,"VARIABLE",           hd+xc+im,       XCALL,xVARIABLE

        byte   $20,"TASK",               hd+xc,          XCALL,xTASK
        byte   $20,"IDLE",               hd+xc,          XCALL,xIDLE
```

```
          byte  $20,"LOOKUP",              hd+xc,          XCALL,xVECTORS
          byte  $20,"VECTORS",             hd+xc,          XCALL,xVECTORS
          byte  $20,"+CALL",               hd+xc,          XCALL,xAddACALL
          byte  $20,"ITEM",                hd+xc,          XCALL,xITEM
          byte  $20,"ITEM@",               hd+xc,          XCALL,xITEMFT
          byte  $20,"ITEMS",               hd+xc,          XCALL,xITEMS
'         byte  $20,"WORDS",               hd+xc,          XCALL,xWORDS
          byte  $20,"BUFFERS",             hd+xc,          XCALL,xBUFFERS
          byte  $20,"FREE",                hd+xc,          XCALL,xFREE
          byte  $20,"TERMINAL",            hd+xc,          XCALL,xTERMINAL
          byte  $20,"CONSOLE",             hd+xc,          XCALL,xCONSOLE
          byte  $20,"COLD",                hd+xc,          XCALL,xCOLDST

          byte  $20,"*end*",               hd,             _NOP,_NOP


enddict   byte  0,0,0     ' Mark the end of the kernel dictionary (2nd and 3rd byte is a pointer or null)




                        org     $10
s                       ' just an offset to be used in DAT sections rather than the distracting +$10

' Align this area to a 256 byte boundary
aln1      byte  0[$100-(@aln1+s-((@aln1+s)&$FF00))]
```

{ TACHYON VM - COG KERNEL (PASM) }

```
{ PASM KERNEL MEMORY MAP
00    RESET
01    0EXIT
03    EXIT
05    NOP
06    XOP
09    DROP3
0A    DROP2
0B    DROP
0D    ?DUP
0E    DUP
11    OVER
13    3RD
15    4TH
17    SWAP
1B    ROT
1E    NIP
20    +
22    -
24    1+
26    1-
28    UM*
29    NEGATE
2B    INVERT
2E    AND
30    ANDN
32    OR
34    XOR
36    SHR
38    SHL
3A    ROL
3C    ROR
3E    SHR
40    SHL
42    REV
44    MASK
48    NOT   0=
4A    =
4E    >
53    C@
55    W@
57    @
59    C+!
```

```
5B    C!
5D    W+!
5F    W!
61    +!
63    !
65    C@++
69    SET
6B    BIT
6D    CLR
6F    IC!
71    (LONG)
72    PUSH3
73    (WORD)
74    (
77    8
78    7
79    6
7A    5
7B    4
7C    3
7D    2
7E    1
7F    0     FALSE
80    BL
81    16
83    -1    TRUE
85    (VARB)
88    (CONL)
8B    P@
8D    P!
8F    CLOCK
91    OUTSET
93    OUTCLR
94    OUTPUTS
96    INPUTS
98    WAITLOW
9C    WAITPNE
9E    WAITPEQ
A0    SHRINP
A4    SHROUT
A7    SPR@
A8    COG@
AA    COGREG
AC    COG!
AE    PASM
B0    I
B2    LSTACK
B4    ACALL
B5    AJMP
B7    VCALL
B8    ZCALL
B9    YCALL
BA    XCALL
C1    JUMP        ELSE
C4    JZ          IF
C8    UNTIL
CC    AGAIN
CF    ADO
D2    DO
D3    FOR
D4    >L
D6    L>
D8    +LOOP
DA    LOOP
DF    AGAINS
E1    UNTILS
E4    BEGINS
E6    NEXT
EA    >R
ED    R>
EF    (REG)
F1    (@REG)
F3    DELTA
F7    WAITCNTS
```

```
F9    (RESET)
FA    doNEXT       ' Forth runtime inner interpreter loop

10E   U/MOD
128   CMPSTR
138   CMOVE
140   LOADMOD
149   (>B>BRANCH)
150   (BRANCH>)
156   (LX>)
160   (>L)
168   (POPX)
176   (PUSHX)
186   (>R)
18A   (RX>)
18E   EMIT

( VARIABLES )

0196  IP                ' Instruction pointer
0197  ACC               ' Accumulator for inline byte-aligned literals
0198  deltaR
0199  target       ' WAITCNT target

019A  R0
019B  R1
019C  R2
019D  X                 ' primary internal working registers
019E  REG0              ' user module working registers
019F  REG1
01A0  REG2
01A1  REG3
01A2  REG4

01A3  txticks           ' set transmit baud rate
01A4  txmask       ' change mask to reassign transmit
01A5  regptr       ' used by REG
01A6  Xptr              ' used by XCALL
01A7  intmask

( STACKS )
01A8  tos
01A8  datastk           [datsz]
01B4  retstk       retsz]
01CA  loopstk           [loopsz]
01D2  branch       [branchsz]

01D8  RUNMOD       24 longs
01EB  !RP               5 longs (temp)
01F0  SPRS
```

```
}

DAT

{{
Byte tokens directly address code in the first 256 longs of the cog
Rather than a jump table most functions are short or cascaded to optimize COG memory
Larger fragments of code jump to the second half of the cog's memory.
As a result of not using a jump table (there's not enough memory) there are gaps
in the bytecode values and not all values are usable.

The formatted source has bytecode instruction labels as bold white on red background.
}}


                        org     0

RESET                   jmp     #TACHYON                ' As a result of being at location 0 this bytecode = 0


' 0EXIT ( flg -- ) Exit if flg is false (or zero)  Used in place of IF......THEN EXIT as false would just end up exiting
ZEXIT                   call    #POPX
                        tjnz    X,#doNEXT
                        '''
EXIT        '           mov     R0,intmask wc,wz                    ' ignore interrupts if not enable or inhibited
          'if_nc_and_nz jmp     #INTERRUPT
```

```
EXITNOW                 call    #RPOPX
JUMPX                   mov     IP,X
_NOP                    jmp     #doNEXT

' Use next byte as an opcode that directly addresses top 256 words of cog
XOP                     call    #GETBYTE
                        add     X,#$100
                        jmp     X


                                { ****** STACK OPERATORS ****** }

DROP3                   call    #POPX
DROP2                   call    #POPX
' 1us execution time including bytecode read and execute
' DROP ( n1 -- )  Pop the top item off the datastack and discard it
DROP                    call    #POPX
                        jmp     #doNEXT

' ?DUP ( n1 -- n1 n1 | 0 ) DUP n1 if non-zero
QDUP                    tjz     tos,#doNext
' DUP ( n1 - n1 n1 ) Duplicate the top item on the stack
DUP                     mov     X,tos                   ' Read directly from the top of the data stack
PUSHX                   call    #_PUSHX                 ' Push the internal X register onto the datastack
                        jmp     #doNEXT
```

```
' OVER ( n1 n2 -- n1 n2 n1 )
OVER                    mov     X,tos+1                 'read second data item and push
                        jmp     #PUSHX
' 3RD ( n1 n2 n3 -- n1 n2 n3 n1 ) Copy the 4th item onto the stack
THIRD                   mov     X,tos+2                 ' read third data item
                        jmp     #PUSHX
' 4TH ( n1 n2 n3 n4 -- n1 n2 n3 n4 n1 ) Copy the 4th item onto the stack
FOURTH                  mov     X,tos+3
                        jmp     #PUSHX

' SWAP ( n1 n2 -- n2 n1 ) Swap the top two items
SWAP                    mov     X,tos+1
SWAPX                   mov     tos+1,tos
PUTX                    mov     tos,X
                        jmp     #doNEXT
' ROT ( a b c -- b c a )
ROT                     mov     X,tos+2
                        mov     tos+2,tos+1
                        jmp     #SWAPX


                                { ****** ARITHMETIC ****** }

MINUS                   neg     tos,tos                 ' save one long by negating and adding
' + ( n1 n2 -- n3 ) Add top two stack items together and replace with result
PLUS                    add     tos+1,tos
                        jmp     #DROP
' 1+ ( n1 -- n1+1 )
INC                     add     tos,#1
                        jmp     #doNEXT
' 1- ( n1 -- n1-1 )
DEC                     sub     tos,#1
                        jmp     #doNEXT


' ?NEGATE ( n1 flg -- n2 ) negate n1 if flg is true
QNEGATE                 tjz     tos,#DROP
                        call    #POPX
' NEGATE ( n1 -- n2 )  equivalent to  n2 = 0-n1
NEGATE                  neg     tos,tos
                        jmp     #doNEXT


                                { ****** BOOLEAN ****** }

' 400ns execution time including bytecode read and execute
' INVERT ( n1 -- n2 ) bitwise invert n1 and replace with result n2
INVERT                  neg     X,#1
                        xor     tos,X
                        jmp     #doNEXT
_AND                    and     tos+1,tos
                        jmp     #DROP
_ANDN                   andn    tos+1,tos
                        jmp     #DROP
```

```
_OR                     or      tos+1,tos
                        jmp     #DROP
_XOR                    xor     tos+1,tos
                        jmp     #DROP
' 1.2us execution time including bytecode read and execute
' SHR ( n1 cnt -- n2 ) Shift n1 right by count
_SHR                    shr     tos+1,tos
                        jmp     #DROP
_SHL                    shl     tos+1,tos
                        jmp     #DROP
_ROL                    rol     tos+1,tos
                        jmp     #DROP
_ROR                    ror     tos+1,tos
                        jmp     #DROP

' 400ns execution time including bytecode read and execute
' 2/ ( n1 -- n1 ) shift n1 right one bit (equiv to divide by 2)
_SHR1                   shr     tos,#1
                        jmp     #doNEXT
' 2* ( n1 -- n2 ) shift n1 left one bit (equiv to multiply by 2)
_SHL1                   shl     tos,#1
                        jmp     #doNEXT

' REV ( n1 bits -- n2 ) Reverse LSBs of n1 and zero-extend
_REV                    rev     tos+1,tos
                        jmp     #DROP
```

```
' 400ns execution time including bytecode read and execute
' MASK ( bitpos -- bitmask )
MASK                    mov     X,tos
                        mov     tos,#1
                        shl     tos,X
                        jmp     #doNEXT
```

****** COMPARISON ******

```
' Basic instructions from which other comparison instructions are built from

' 0= ( n1 -- flg ) true if n1 equals 0 - same as a boolean NOT where TRUE becomes FALSE
_NOT
ZEQ                     or      tos,#0 wz
SETZ                    neg     tos, #1                 'generate a true (-1)
                if_nz   mov     tos,#0                  'force true to false
                        jmp     #doNEXT

' = ( n1 n2 -- flg ) true if n1 is equal to n2
EQ                      cmp     tos,tos+1  wz
                        neg     tos+1,#1
                if_nz   mov     tos+1,#0
                        jmp     #DROP

' > ( n1 n2 -- flg ) true if n1 > n2
GT
_GT                     cmps    tos+1,tos  wz,wc
                        call    #POPX
                if_a    neg     tos,#1
                if_be   mov     tos,#0
                        jmp     #doNEXT
```

****** MEMORY ******

```
' C@  ( caddr -- byte ) Fetch a byte from hub memory
CFETCH                  rdbyte  tos,tos
                        jmp     #doNEXT

' W@  ( waddr -- word ) Fetch a word from hub memory
WFETCH                  rdword  tos,tos
                        jmp     #doNEXT

' @  ( addr -- long ) Fetch a longfrom hub memory
FETCH                   rdlong  tos,tos
                        jmp     #doNEXT

' C+!  ( n caddr -- ) add n to byte at hub addr
CPLUSST                 rdbyte  X,tos                   ' read in word from adress
                        add     tos+1,X                 ' add to contents of address - cascade
' C!  ( n caddr -- ) store n to byte at addr
```

```
CSTORE                  wrbyte   tos+1,tos                ' write the byte using address on the tos
                        jmp      #DROP2

' W+!  ( n waddr -- ) add n to word at hub addr
WPLUSST                 rdword   X,tos                    ' read in word from adress
                        add      tos+1,X
' W!  ( n waddr -- ) store n to word at addr
WSTORE                  wrword   tos+1,tos
                        jmp      #DROP2

' +!  ( n addr -- ) add n to long at hub addr
PLUSST                  rdlong   X,tos                    ' read in long from adress
                        add      tos+1,X
' !  ( n addr -- ) store n to long at addr
STORE                   wrlong   tos+1,tos
                        jmp      #DROP2

' C@++  ( addr -- caddr+1 byte ) fetch byte character and increment address
CFETCHINC       mov     X,tos                   ' dup the address
                        call     #_PUSHX
                        add      tos+1,#1               ' inc the backup address
                        jmp      #CFETCH                ' fetch at the current byte



' SET ( mask caddr -- ) Set bit(s) in hub byte
SET                     test     $,#1  wc                'Set the carry flag (trick)
                        jmp      #BITOP                 ' mux that state according to mask

' BIT! ( mask caddr state -- ) Set or clear bit(s) in hub byte
BIT                     call     #POPX
                        tjnz     X,#SET
' CLR ( mask caddr -- ) Clear bit(s) in hub byte
CLR                     test     $,#0  wc                'clear the carry flag (trick)
                        jmp      #BITOP

{
' FILL ( src cnt ch -- )
FILL                    wrbyte   tos,tos+2
                        add      tos+2,#1
                        djnz     tos+1,#FILL
                        jmp      #DROP3
}
```

{ ****** LITERALS ****** }

```
' 3.6us execution time including bytecode read and execute
' ( -- 32bits ) Push a 32-bit literal onto the datastack by reading in the next 4 bytes (non-aligned)
_LONG
PUSH4                   call     #ACCBYTE               ' read the next byte @IP++ and shift accumulate
' 3us execution time including bytecode read and execute
' ( -- 24bits ) Push a 24-bit literal onto the datastack by reading in the next 3 bytes (non-aligned)
PUSH3                   call     #ACCBYTE
_WORD
' 2.4us execution time including bytecode read and execute
' ( -- 16bits) Push a 16-bit literal onto the datastack by reading in the next 2 bytes (non-aligned)
PUSH2                   call     #ACCBYTE
' 1.8us execution time including bytecode read and execute
' ( -- 8bits ) Push an 8-bit literal onto the datastack by reading in the next byte
_BYTE
PUSH1                   call     #ACCBYTE
PUSHACC                 call     #_PUSHACC              ' Push the accumulator onto the stack then zero it
                        jmp      #doNEXT
```

{ ****** FAST CONSTANTS ****** }

```
' Push a preset literal onto the stack using just one bytecode
'

' Use the "accumulator" to push the value which is built up by incrementing
' There is a minor penalty for the larger constants but it's still faster and more compact
' overall than using the PUSH1 method or the mov X,# method
_8                      add      ACC,#1
_7                      add      ACC,#1
_6                      add      ACC,#1
_5                      add      ACC,#1
_4                      add      ACC,#1
_3                      add      ACC,#1
_2                      add      ACC,#1
```

```
_1                      add     ACC,#1                  ' 1.4us execution time including bytecode read and execute
_FALSE
_0                      jmp     #PUSHACC                ' Push ACC and then zero ACC

BL                      mov     ACC,#16
_16                     add     ACC,#16
                        jmp     #PUSHACC                ' Push ACC and then zero ACC

_TRUE
MINUS1                  neg     X,#1
                        jmp     #PUSHX
```

{ ****** VARIABLES ****** }

```
' Byte aligned variables start with this single byte code which returns with the address of the byte variable following
' long variables just externally align this opcode a byte before the boundary
' INLINE:
VARB                    mov     X,IP
PUSHX_EXIT      call    #_PUSHX  ' push address of variable
                        jmp     #EXIT

' Long aligned constant - created with CONSTANT and already aligned
CONL                    mov     R1,IP
                        rdlong  X,R1                    ' get constant
                        jmp     #PUSHX_EXIT
```

{ ****** I/O ACCESS ****** }

```
' P@ ( -- n1 ) Read the input port A (assume it is always A for Prop 1)
PFETCH                  mov     X,INA
                        jmp     #PUSHX
' P! ( n1 -- ) Store n1 to the output port A
PSTORE                  mov     OUTA,tos
                        jmp     #DROP

' CLOCK ( COGREG4=iomask ) Toggle multiple bits on the output - use cogreg 4 (scl)
CLOCK                   xor     OUTA,REG0+4             '
                        jmp     #doNEXT

' OUTSET ( iomask -- ) Set multiple bits on the output
OUTSET                  or      OUTA,tos                'PSET ( mask -- ) \ Or mask to OUTA and make sure it's an output
                        jmp     #OUTPUTS
' OUTCLR ( iomask -- ) Clear multiple bits on the output
OUTCLR                  andn    OUTA,tos                'PCLR ( mask -- )
' OUTPUTS ( iomask -- ) Set selected port pins to outputs
OUTPUTS                 or      DIRA,tos                'PDSET ( mask -- ) \ OR mask to DIRA
                        jmp     #DROP

' INPUTS ( iomask -- ) Set selected port pins to inputs
INPUTS                  andn    DIRA,tos
                        jmp     #DROP

{
' Waits for a low on the input but with a timeout = 150ns/count (1,000,000 = 150ms)
' returns with cnt non-zero if low is detected
' WAITLOW ( cnt iomask -- cnt )
WAITLOW                 test    tos,INA wz
                if_z    jmp     #DROP
                        djnz    tos+1,#WAITLOW
                        jmp     #DROP
}


' 130910 - change the way WAITP operates in that it reads a mask from COGREGS and also captures the CNT

' (WAITPNE)       Wait until input is low - REG3 = mask, REG0 = CNT
_WAITPNE                waitpne reg3,reg3               ' use COGREG3 as the mask
                        mov     reg0,cnt                ' capture count in COGREG0
                        jmp     #doNEXT
' WAITPEQ        Wait until input is high - REG3 = mask, REG1 = CNT
_WAITPEQ                waitpeq reg3,reg3
                        mov     reg1,cnt                ' capture count in COGREG1
                        jmp     #doNEXT
```

{ *** SERIAL I/O OPERATORS *** }

{
To maximize the speed of I/O operations especially serial I/O such as ASYNCH, I2C and SPI etc there are special operators that avoid pushing and popping the stack and instead perform the I/O bit by bit and leave the latest shifted version of the data on the stack.
}

```
' Assembles with last bit received as msb - needs SHR to right justify if asynch data
' SHRINP ( iomask dat -- iomask dat*2 )
SHRINP                  shr     tos,#1
                        test    tos+1,INA wz
                if_nz   or      tos,msb
                        jmp     #doNEXT
```

{ This is optimized for when you are sending out multiple bits
as in asynchronous serial data or I2C
Shift data one bit right into output via iomask - leave mask & shifted data on stack (looping)
400ns execution time including bytecode read and execute }

```
' SHROUT ( iomask dat -- iomask dat/2 )
SHROUT                  shr     tos,#1 wc               ' Shift right and get lsb
                        muxc    OUTA,tos+1              ' reflect state to output
                        jmp     #doNEXT
```

{ ****** COG ACCESS ****** }

```
' SPR@ ( index -- long ) Fetch a cog's SPR
SPRFETCH                add     tos,#$01F0
' COG@ ( addr -- long ) Fetch a long from cog memory
COGFETCH                movs    _readsrc,tos
                        jmp     #_readsrc               ' since we need a dummy instr we may as well jump

' COGREG ( ix -- addr ) return with the address of the indexed cog register
COGREG                  add     tos,#REG0
                        jmp     #doNEXT


' COG! ( long addr -- ) Store a long to cog memory
COGSTORE                movd    _writedst,tos
                        jmp     #_writedst

' PASM ( val pasm -- result pasm )  120919 modified to not drop
PASM                    mov     pasmins,tos
                        jmp     #pasmins                ' takes care of pipeline and uses high mem for remainder of code

' I ( -- index ) The current loop index is at a fixed address just under the loop limit at top of ascending loop stack
I                       mov     X,loopstk+1
                        jmp     #PUSHX
' Loop control words such as J K LEAVE etc implemented
' LSTACK ( index -- cog_addr ) push address of the loop stack in cog memory
LSTACK                  add     tos,#loopstk
                        jmp     #doNEXT
```

{ ****** BRANCH & LOOP ****** }

```
' ACALL ( adr -- ) Call arbitrary (from the stack) address
ACALL                   call    #SAVEIP
AJMP                    mov     IP,tos
                        jmp     #DROP
```

{
Since Tachyon uses bytecodes for instructions it also uses bytes to address code.
Since bytes are limited to 256 values these are used instead to lookup the absolute address of the code from a table of vectors. To extend that beyond 256 values there are further opcodes which index the table for a total of 1,024 vectors.

Vectored call instructions form a 10-bit index with the upper 2 bits derived from the instruction so calls to other words only use 2 bytes total
Note: Originally there was only an XCALL but new opcodes were added to expand the vector table for duriing use however the kernel only uses XCALL
}

```
VCALL                   mov     ACC,#2                  ' high word of upper page
ZCALL                   add     ACC,zcalls              ' low word of upper page

' Perform a call to kernel bytecode via the XCALLS but reusing the high word of each vector
' The YCALLs are implemented by the runtime compiler to fully utilize the XCALLs table (high word of longs)
YCALL                   add     ACC,#2
' Inline call to kernel bytecode via the XCALLS vector table using the following inline byte as an index
XCALL                   add     ACC,Xptr
xycall                  call    #SETUPIP                ' Save IP and read next bytecode into X (offset in table)
                        shl     X,#2                    ' offset into longs in hub RAM
```

```
                        add     X,ACC                   ' Add to table base
                        mov     ACC,#0                  ' Always clear ACC to zero ready for reuse
                        rdword  IP,X                    ' Load IP with vector
                        jmp     #doNEXT


' Jump forward by reading the next byte inline and adding that to the IP (at that point)
JUMP
_ELSE                   call    #GETBYTE
JMPFWD                  add     IP,X
                        jmp     #doNEXT

' If flg is zero than jump forward by reading the next byte inline and adding that to the IP (at that point)
' IF R( flg -- )
' Read the next byte as a displacement and branch forward
JZ
_IF                     call    #GETBYTE                'read in next byte at IP and inc IP
                        tjnz    tos,#DROP               'test flag on stack
                        add     IP,X                    'Adjust IP forward according to flag
                        jmp     #DROP                   'discard flag

' Read the next byte as a negative displacement and jump back if TOS is zero
JZBACK
_UNTIL                  call    #GETBYTE                'read in next byte at IP and inc IP
                        tjnz    tos,#DROP               'test flag on stack
                        sub     IP,X                    'Adjust IP forward according to flag
                        jmp     #DROP                   'discard flag

' Read the next byte as a negative displacement and jump back
JUMPBACK
_AGAIN                  call    #GETBYTE
                        sub     IP,X
                        jmp     #doNEXT

' ADO = BOUNDS DO - just a quick and direct way as BOUNDS is most often never used elsewhere
' ADO ( from cnt -- )
ADO                     mov     X,tos+1
                        add     tos+1,tos
                        mov     tos,X
' DO ( to from -- )
DO                      call    #_PUSHLP                ' PUSH index onto loop stack
 ' FOR ( count -- ) Setup FOR...NEXT loop for count
FOR                     call    #_PUSHBRANCH            ' Push the IP onto the mark stack and set branch
' >L ( n -- ) Push n onto the loop stack
PUSHL                   call    #_PUSHLP
                        jmp     #doNEXT

' L> ( -- n ) Pop n from the loop stack
LPOP                    call    #LPOPX                  ' Pop loop stack into X
                        jmp     #PUSHX                  ' Push X onto the data stack as tos


' (+loop) ( n1 -- ) adds n1 to the loop index and branches back if not equal to the loop limit
PLOOP                   call    #POPX
                        jmp     #LOOPCHK

' 400ns execution time including bytecode read and execute
LOOP                    mov     X,#1
LOOPCHK                 add     loopstk+1,X wc          ' increment index,
                        cmps    loopstk,loopstk+1 wz,wc
          if_be         call    #LPOPX
          if_be         jmp     #SKIP1
' AGAIN>
AGAINS                  mov     IP,branch               'Branch to the address that is saved in branch
                        jmp     #doNEXT

{ !!! Removed to save memory - there's a limit to how much you can squeeze into 496 longs!!
' UNTIL> ( flg -- )
UNTILS                  call    #POPX
                        tjz     X,#AGAINS
                        jmp     #doNEXT
' <BEGIN - pushes a branch branch onto the mark stack - equivalent to PUSHBRANCH
BEGINS                  call    #_PUSHBRANCH
                        jmp     #doNEXT
}


' INIT STACKS
InitRP
                        movs    rpopins,#retstk
                        movd    _PUSHR,#retstk
                        jmp     #doNEXT
```

```
' Average execution time = 400ns
' NEXT ( -- ) Decrement count (on loop stack) and loop until 0, then pop loop stack
forNEXT                 djnz    loopstk,#AGAINS         ' not done yet, jump to branch
SKIP1                   call    #LPOPX
' POPBRANCH - pops a branch branch off the mark stack manually (use if forcing an exit from a stacked branch)
POPBRANCH               call    #_POPBRANCH
                        jmp     #doNEXT



' >R ( n -- ) Push n onto the return stack
PUSHR                   mov     R0,tos
                        call    #_PUSHR
                        jmp     #DROP
' R> ( -- n ) Pop n from the return stack
RPOP                    call    #RPOPX                  ' Pop return stack into R and X
                        jmp     #PUSHX                  ' Push X onto the data stack as tos


'' Registers can be used just like variables and the interpreted kernel uses some for itself
'' 128+ bytes are reserved which can be accessed as bytes/words/longs depending upon
''  the alignment.
```

```
' (REG) ( -- addr ) Read the next inline byte and return with the register byte address
REG                     call    #GETBYTE
                        call    #_PUSHX
' REG ( index -- addr ) Find the address of the register
ATREG                   add     tos,regptr
                        jmp     #doNEXT


' DELTA ( delta -- )      Use zero value to wait for the next count or use a delta to set
DELTA                   call    #POPX
                        mov     deltaR,X
                        mov     target,X
                        add     target,cnt
' WAITCNT ( -- )
WAITCNTS                waitcnt target,deltaR
                        jmp     #doNEXT
_COGID
                        cogid   X
                        jmp     #PUSHX



' _COGINIT ( dest -- cog )
_COGINIT                coginit tos wr
                        jmp     #doNEXT



' um* ( u1 u2 -- u1*u2L u1*u2H ) \ unsigned 32bit * 32bit -- 64bit result - 1us..10us
UMMUL                   mov     R0,tos+1
                        mov     R2,tos
                        cmp     tos+1,tos wc    ' try to go faster by using the lower number as the multiplier
                if_c    mov     R0,tos                  ' R0R1 = u2
                if_c    mov     R2,tos+1                ' R2R3 = u1
                        mov     R1,#0
                        mov     tos,#0                  ' zero result
                        mov     tos+1,#0
UMMULLP                 shr     R2,#1 wz,wc             ' test next bit of u1
                if_nc   jmp     #UMMUL1
                        add     tos+1,R0 wc             ' add in shifted u2
                        addx    tos,R1                  ' carry into upper long
UMMUL1                  add     R0,R0 wc                ' shift u2 left
                        addx    R1,R1                   ' carry into 64-bits
                if_nz   jmp     #UMMULLP                ' exhausted u1?
                        jmp     #doNEXT
```

********** *INTERNAL COG ROUTINES* **********

Code from here after cog address $0FF cannot be indexed directly except by an XOP.

```
' Reset Entry
TACHYON
                        mov     IP,PAR                  ' Load the IP with the address of the first instruction
```

{ ****** RUNTIME BYTECODE INTERPRETER ****** }

```
' Fetch the next byte code instruction in hub RAM pointed to by the instruction pointer IP
'
doNEXT                  rdbyte  X,IP                    'read byte code instruction
                        add     IP,#1                   'advance IP to next byte token
                        jmp     X                       'execute the code by directly indexing the first 256 long in cog

' Finalize the bit operation by read/writing the result
' ( mask caddr -- )
BITOP                   rdbyte  X,tos                   ' Read the contents of the memory location
                        muxc    X,tos+1                 ' set or clear the bit(s) specd by mask
                        wrbyte  X,tos                   ' update
                        jmp     #DROP2

' COG FETCH and STORE subs
_readsrc                mov     tos,0_0
                        jmp     #doNEXT
_writedst               mov     0_0,tos+1
                        jmp     #DROP2
pasmins                 nop
                        jmp     #doNEXT
```

{ ****** LITERALS ****** }

```
' Accumulate a literal byte by byte from 1 to 4 bytes long depending upon the number of times this routine is called.
' This allows literals to be byte aligned.
ACCBYTE                 call    #GETBYTE                ' Build a big endian literal by reading in another byte
                        shl     ACC,#8                  ' merge it into the "accumulator" byte by byte
                        or      ACC,X
ACCBYTE_ret             ret

' Read the next byte of code memory via IP
GETBYTE                 rdbyte  X,IP                    ' Simply read a byte (non-code) and advance the IP
                        add     IP,#1
GETBYTE_ret             ret
```

{ ****** ARITHMETIC ****** }

```
' u/mod ( u1 u2 -- remainder quotient) both remainder and quotient are 32 bit unsigned numbers
pUDIVMOD
                        mov     R1,#$20
                        mov     R0,#0                   ' quotient
udivmodlp               shl     tos+1, #1 wc            ' dividend
                        rcl     R0, #1                  ' hi bit from dividend
                        cmpsub  R0, tos   wc,wr         ' cmp divisor
                        rcl     R2, #1                  ' R2 - quotient
                        djnz    R1, #udivmodlp
                        mov     tos+1, R0
                        mov     tos, R2
                        jmp     #doNEXT



''
'' Compare a null-terminated source string with a dictionary string which is 8th bit terminated.
'' This will always force a mismatch after which one is checked for a null while the other is checked
'' for the 8th bit and if verified then a match has been found.
'' The dict pointer is advanced to point to the end of the dict string on the 8th bit termination which
'' is the attribute byte as in: byte "CMPSTR",$80,CMPSTR
''
' XOP
' CMPSTR ( src dict -- src dict+ flg ) Compare strings at these two addresses
pCMPSTR

' CMPSTR ( src dict -- src dict+ flg ) Compare strings at these two addresses
_CMPSTR                 call    #_PUSHX                 ' make room for a flag
                        mov     R2,tos+1
                        mov     X,tos+2                 ' X = source
cmpstrlp                rdbyte  R0,X                    ' read in a character from the source
```

```
                        rdbyte   R1,R2                  ' read in from the dictionary
                        cmp      R1,R0 wz               ' are they the same?
                if_nz   jmp      #nomatch
                        add      X,#1                   ' so far, so good, try next
                        add      R2,#1                  ' updates the dict pointer on the stack too
                        jmp      #cmpstrlp              ' keep at it
nomatch                 cmp      R0,#0 wz               ' was the src null terminated?
                        xor      R1,#$80
                if_z    test     R1,#$80 wz             ' set z flag if dict 8th bit set
                        jmp      #SETZ



' CMOVE bytes from source to destination primitive - <CMOVE conditions the parameters before calling
' XOP
' (CMOVE) ( src dst cnt inc -- ) Copy bytes from src to dst address for cnt bytes using increment for up or down
pCMOVE                  call     #POPX
CMVlp                   rdbyte   R0,tos+2               ' read source byte
                        add      tos+2,X
                        wrbyte   R0,tos+1               ' write destination byte
                        add      tos+1,X
                        djnz     tos,#CMVlp
                        jmp      #DROP3


{
```

*** INTERRUPTS ***

Interrupts are tested whenever an EXIT is executed and before the IP is restored
There are up to 31 interrupt sources which set their bit in the interrupts cog variable using COGREG 9
Interrupt vectors are stored in low hub memory as an array of 32 16-bit addresses
The interrupt request is not cleared automatically but an interrupt inhibit is set (bit 31)

This interrupt routine shouldn't be called unless one of the interrupts are set however there is a
safety fence in that when all sources have been exhausted that vector 31 will be executed which is not associated with
any 0..30 sources

REV: Made interrupts a global in hub ram and allow each cog to have a interrupt mask to determine which ones
they will service.

```
}
{  !!! DISABLED in this version - need the memory - could have one Tachyon cog that allows this perhaps??
INTERRUPT
' 121026 - changing how interrupts work - need hub RAM access
                        rdlong   R0,#@intreqs+S          ' read int requests from hub
                        and      R0,intmask wz           ' only check those interrupts we have enabled
          if_z          jmp      #EXITNOW                ' service interrupts before restoring IP from return stack
                        mov      R1,#@intvecs+s-4        ' point to interrupt vectors in hub ram (low address)
scanirqs                shr      R0,#1 wc,wz             ' test next irq bit (with a safety fence)
                        add      R1,#2                   ' point to next word vector
          if_nc_and_nz  jmp      #scanirqs
                        or       intmask,msb             ' set inhibit
                        rdword   IP,R1                   ' read in external vector
                        jmp      #doNEXT                 ' and execute  (no need to save IP as it's not yet restored)
}
```

{ ****** INTERNAL STACKS ****** }

```
{
As well as the data and return stack, a loop stack is also employed
The return stack should normally only used for return addresses
V2 adds a branch stack to speed up loops by storing backward branch addresses used by DO, FOR, <BEGIN etc
}
```

{ ****** BRANCH STACK HANDLER ****** }

```
{
The branch stack is used for fast loop branching and so holds the branch address
It is constructed as a fixed top-of-stack location which physically moves data when it's pushed or popped
This means also that it is not possible to corrupt other memory by over or underflow.
}
_PUSHBRANCH
                        mov      branch+5,branch+4
                        mov      branch+4,branch+3
                        mov      branch+3,branch+2
                        mov      branch+2,branch+1
                        mov      branch+1,branch
                        mov      branch,IP
_PUSHBRANCH_ret         ret
```

```
_POPBRANCH      mov     branch,branch+1
                mov     branch+1,branch+2
                mov     branch+2,branch+3
                mov     branch+3,branch+4
                mov     branch+4,branch+5
_POPBRANCH_ret          ret
```

{ ****** LOOP STACK HANDLER ****** }

```
{
The loop stack holds the loop limit and current index. It is constructed as a fixed top-of-stack location which physically moves data when it's pushed or popped
This means also that it is not possible to corrupt other memory by over or underflow.
}
' pop the loop stack into X
LPOPX
                mov     X,loopstk
                mov     loopstk,loopstk+1
                mov     loopstk+1,loopstk+2
                mov     loopstk+2,loopstk+3
                mov     loopstk+3,loopstk+4
                mov     loopstk+4,loopstk+5
                mov     loopstk+5,loopstk+6
                mov     loopstk+6,loopstk+7
                mov     loopstk+7,#0
LPOPX_ret       ret

' Push tos onto the loop stack and drop tos
_PUSHLP
                mov     loopstk+7,loopstk+6
                mov     loopstk+6,loopstk+5
                mov     loopstk+5,loopstk+4
                mov     loopstk+4,loopstk+3
                mov     loopstk+3,loopstk+2
                mov     loopstk+2,loopstk+1
                mov     loopstk+1,loopstk
                mov     loopstk,tos
'
' falls through into a DROP to remove the tos
'
```

{ ****** DATA STACK HANDLER ****** }

```
{ iterative pop method - about 3 times slower than brute force code
POPX            movd    popxins,#tos
                movs    popxins,#tos+1
                mov     X,tos
                mov     R1,#datsz-1
popxins         mov     _0,_0
                add     popxins,dstsrc          ' increment dst and src
                djnz    R1,#popxins
_PUSHLP_ret
POPX_ret        ret
dstsrc          long    $0201
'}

' Pop the data stack using fixed size stack in COG memory (allows fast direct access for operations)
POPX            mov     X,tos                   ' pop old tos into X (temporary)
                mov     tos,tos+1
                mov     tos+1,tos+2
                mov     tos+2,tos+3
                mov     tos+3,tos+4
                mov     tos+4,tos+5
                mov     tos+5,tos+6
                mov     tos+6,tos+7
                mov     tos+7,tos+8
                mov     tos+8,tos+9
                mov     tos+9,tos+10
                mov     tos+10,tos+11
POPX_ret
_PUSHLP_ret     ret


_PUSHACC        mov     X,ACC                   ' Accumulator operation used for fast constants
_PUSHX          mov     ACC,#0                  ' clear it for next operation
_PUSHX1         mov     tos+11,tos+10
                mov     tos+10,tos+9
                mov     tos+9,tos+8
                mov     tos+8,tos+7
                mov     tos+7,tos+6
                mov     tos+6,tos+5
```

```
                        mov     tos+5,tos+4
                        mov     tos+4,tos+3
                        mov     tos+3,tos+2
                        mov     tos+2,tos+1
                        mov     tos+1,tos
                        mov     tos,X                   ' replace tos with X (DEFAULT)
_PUSHACC_ret
_PUSHX_ret      ret
```

{ ****** RETURN STACK HANDLER ****** }

```
{
Return stack builds up in memory
Return stack items do not need to be directly addressed
This indexed method does not use movd and movs methods but directly inc/decs the
source and destination fields of the instruction.(retstk) so it must stay synchronized - Use !RP if needed
}
SETUPIP                 call    #GETBYTE                ' read the next byte into X and save the current IP
SAVEIP                  mov     R0,IP
_PUSHR                  mov     retstk,R0               ' save it on the stack (dest modified)
                        add     rpopins,#1              ' update source for popping
                        add     _PUSHR,dst1             ' update dest for pushing (points to next free)
SETUPIP_ret                     '
SAVEIP_ret              '
_PUSHR_ret      ret

RPOPX                   sub     rpopins,#1              ' decrement rpop's source field
                        sub     _PUSHR,dst1
rpopins                 mov     X,retstk
RPOPX_ret               ret



{
' RETURN STACK in hub RAM method (must ensure other tasks have their own space)
SETUPIP                 call    #GETBYTE                ' read the next byte into X and save the current IP
SAVEIP                  mov     R0,IP
_PUSHR
                        sub     RP,#4
                        wrlong  R0,RP
SETUPIP_ret                     '
SAVEIP_ret              '
_PUSHR_ret      ret

RPOPX
                        rdlong  X,RP
                        add     RP,#4
RPOPX_ret               ret

RP                      long    @retstks+s
'}
```

{ ****** CONSOLE SERIAL OUTPUT ****** }

```
{
Transmit the character that is in the tos register. This character is preconditioned with start and stop bits and the output direction is also set (to save cog space)
}
' EMIT ( char -- )
pEMIT

TRANSMIT
                        mov     R1,#10                  ' 8 data bits + 1 start + stop bits
txdat                   mov     R0,cnt
                        add     R0,txticks
txbit                   shr     tos,#1 wc               ' lsb first
                        muxc    outa,txmask             ' output bit
                        waitcnt R0,txticks              ' bit timing
                        djnz    R1,#txbit               'another bit to transmit?
                        jmp     #DROP
```

{ ****** PASM MODULE LOADER ****** }

```
{
16 longs (or more) are reserved for a selectable module as a helper for specialized functions such as SPI etc.
LOADMOD loads the longs into this area to be executed with RUNMOD
```

```
130811 - Added feature so that a large module can be loaded directly into address 0 and run automatically if cnt >24
}
' LOADMOD ( src cnt -- ) Load a PASM module of up to 16 bytes into the cog
pLOADMOD
                        cmp     tos,#24 wc
          if_nc         movd    lcdst,#0                ' this is a full cog load - must run automatically as well
          if_c          movd    lcdst,#_RUNMOD          ' Init dst pointer - just loading a small module alongside Tachyon
ldmlp                   add     lcdst,dst1              ' post-increment long destination (pipelined after rdlong)
lcdst                   rdlong  0_0,tos+1               ' read a long from hub ram into cog's destination
                        add     tos+1,#4                ' increment hub memory long address
                        djnz    tos,#ldmlp
          if_nc         jmp     #0                      ' run from address 0
                        jmp     #DROP2



dst1                    long    $200                    ' instruction's destination field increment
msb                     long    $8000_0000
zcalls                  long    $0400-2                         ' 1K offset after XCALLs for ZCALL table


'****************** COG VARIABLES *****************
'


' Registers used by PASM modules to hold parameters such as I/O masks and bit counts etc
REG0            long 0
REG1            long 0
REG2            long 0
REG3            long 0
REG4            long 0

txticks         long (80_000_000 / baud )               ' set transmit baud rate
txmask          long |<txd                              ' change mask to reassign transmit
regptr          long @registers+s                       ' used by REG
Xptr            long @XCALLS+s                          ' used by XCALL
intmask         long 0

' rearranged these register to follow REG0 so that they can be directly accessed by COGREG
IP              long 0    ' Instruction pointer
ACC             long 0    ' Accumulator for inline byte-aligned literals
X               long 0    ' primary internal working registers
R0              long 0
R1              long 0
R2              long 0
deltaR          long 0
target          long 0    ' WAITCNT target



' *** STACKS ***
tos
datastk         long 0[datsz]
retstk          long 0[retsz]
loopstk         long 0[loopsz]

branch          long 0[branchsz]


_RUNMOD_   ' this is a dummy symbol but the org must be equal to _RUNMOD (or less)
                long 0[18]

                fit 496

                long 0[32]        'The kernel image becomes a general-purpose buffer and this expands it to at least 2K for coginits

' define some constants used by this cog
' The CLKDAT parameters are defined here so that the method can be changed easily

                org tos
sdat            res 1

                org REG0
sck             res 1
mosi            res 1
miso            res 1
pixshift
scnt            res 1
pixeladr        res 1

endofkernel     res 0     ' just a branch to indentify the end of the kernel in the listing (BST)
```

```
                res 0
                res 0
                res 0
                res 0


' If hub ram is selected for return stacks then the space here down is used. Each cog has 32 longs.
retstks
```

{ SERIAL RECEIVE }

```
'************************************ SERIAL RECEIVE ****************************************

{ This is a dedicated serial receive routine that runs in it's own cog }

DAT
                      long 0[2]   ' read and write           ' this hub space is used for rxwr & rxrd at runtime
rxbuffers                                                    ' hub ram gets reused as the receive buffer
                         org

HSSerialRx      mov      rxwr,#0  ' init write index
                         wrword   rxwr,hubrxwr               ' clear rxrd in hub
                         wrword   rxwr,hubrxrd               ' make rxwr = rxrd
'                        cmp      rxticks,#26 wz             ' is it 3M baud?
'                 if_z   jmp      #receive3                  ' use special receive routine for 3M
                         mov      stticks,rxticks            ' calculate start bit timing
                         shr      stticks,#1                 ' half a bit time less
                         sub      stticks,#4                 ' compensate timing - important at high speeds
                         mov      breakcnt,#200              ' reset break count

------------------------------------------------------------------------------------------------------------------------

receive                  mov      rxcnt,stticks              ' Adjusted bit timing for start bit
                         waitpne  rxpin,rxpin                ' wait for a low = start bit
'
' START BIT DETECTED
'                                                            ' time sample for middle of start bit
                         add      rxcnt,cnt                  ' uses special start bit timing
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina       wz         ' sample middle of start bit
rxcond2          if_nz   jmp      #receive                   ' restart if false start otherwise bit time from center

'
' START bit validated
' Read in data bits lsb first
' No point in looping as we have plenty of code to play with
' and inlining (don't call) and unrolling (don't loop) can lead to higher receive speeds
'
                         waitcnt  rxcnt,rxticks              ' wait until middle of first data bit
                         test     rxpin,ina wc               ' sample bit 0
                         muxc     rxdata,#01                 ' and assemble rxdata
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 1
                         muxc     rxdata,#02
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 2
                         muxc     rxdata,#04
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 3
                         muxc     rxdata,#08
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 4
                         muxc     rxdata,#$10
 ' data bit 5
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 5
                         muxc     rxdata,#$20
                         mov      X1,rxbuf                   'squeeze in some overheads, calc write pointer
                         add      X1,rxwr                    ' X points to buffer location to store
' data bit 6
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 6
                         muxc     rxdata,#$40
                         add      rxwr,#1                    ' update write index
                         and      rxwr,wrapmask
' last data bit 7
                         waitcnt  rxcnt,rxticks
                         test     rxpin,ina wc               ' sample bit 7
                         muxc     rxdata,#$80
                         wrbyte   rxdata,X1                  ' save data in buffer - could take 287.5ns worst case
' stop bit
```

```
stopins                 waitcnt  rxcnt,rxticks          ' check stop bit (a little earlier) (need to detect errors and breaks)
                        test     rxpin,ina wc           ' sample stop bit
                if_nc   jmp      #frmerror              ' framing error - check for break - disregard loss of timing integrity now
                if_c    wrword   rxwr,hubrxwr           ' update hub index for code reading the buffer if all good
                        jmp      #receive


frmerror
                        sub      rxwr,#1
                        cmp      rxdata,#0 wz           'if it's a null it could be part of a break
                if_nz   mov      breakcnt,#200          'reset the break count (compromise here to keep main loop tight)
                if_nz   jmp      #receive               'ignore normal framing error
                        or       outa,rxpin
                        or       dira,rxpin             'unintentional? make sure the input is not floating
                        mov      rxcnt,#16
                        djnz     rxcnt,$
                        andn     dira,rxpin             'restore input and delay
                        mov      rxcnt,#16
                        djnz     rxcnt,$
                        test     rxpin,ina wz           'if it's still low then it is being intentionally driven
                if_nz   jmp      #receive               'ignore floating input
                        djnz     breakcnt,#receive
aboot                   mov      rxcnt,#$80
                        clkset   rxcnt
                        jmp      $

{
'
' 3M baud receive
receive3

' 333.33ns/bit with 50ns/instruction or 26.66 clocks

                        waitpne  rxpin,rxpin            ' wait for a low = start bit
 '
' START BIT DETECTED
'                                                       ' time sample for middle of start bit
                        nop
                        nop
                        test     rxpin,ina       wz     ' sample middle of start bit
                if_nz   jmp      #receive3              ' restart if false start
' 200ns
' START bit validated
'
                        nop
                        nop
' @300ns
' bit 0
                        nop
                        nop
                        nop
                        test     rxpin,ina wc
                        muxc     rxdata,#01
' bit 1
                        nop
                        nop
                        nop
                        test     rxpin,ina wc
                        muxc     rxdata,#02
' bit 2
                        nop
                        nop
                        nop
                        nop
                        nop
                        test     rxpin,ina wc
                        muxc     rxdata,#04
' bit 3
                        nop
                        nop
                        nop
                        nop
                        test     rxpin,ina wc
                        muxc     rxdata,#08
' bit 4
                        nop
                        nop
                        nop
                        nop
                        nop
                        test     rxpin,ina wc
```

```
                        muxc    rxdata,#$10
' bit 5
                        nop
                        nop
                        mov     X1,rxbuf
                        add     X1,rxwr                 ' X points to buffer location to store
                        test    rxpin,ina wc
                        muxc    rxdata,#$20
' bit 6
                        nop
                        nop
                        nop
                        add     rxwr,#1
                        and     rxwr,wrapmask
                        test    rxpin,ina wc
                        muxc    rxdata,#$40
' bit 7
                        nop
                        nop
                        nop
                        nop
                        test    rxpin,ina wc
                        muxc    rxdata,#$80
                        wrbyte  rxdata,X1               ' save data in buffer - could take 287.5ns worst case
' stop bit
                        test    rxpin,ina wc            ' stop bit
                if_nc   sub     rxwr,#1                 ' restore rxwr on error - no need to worry yet about wrap
                if_c    wrword  rxwr,hubrxwr
                        jmp     #receive3
}

'-------------------------------------------------------------------------------

rxpin                   long    |<rxd                   ' mask of rx pin
hubrxrd                 long    @rxbuffers+s-4          ' ptr to rxrdin hub
hubrxwr                 long    @rxbuffers+s-2          ' word address of rxwr in hub (after init)
rxbuf                   long    @rxbuffers+s            ' pointer to rxbuf in hub memory
mode                    long    0
rxticks                 long    0
stticks                 long    0
spticks                 long    0
breakcnt                long    40
wrapmask                long    bufsize-1
rxcnt                   long    0
rxdata                  long    0                       'assembled character
lastch                  long    0
X1                      long    0
savdat                  long    0
rxwr                    long    0                       'cog receive buffer write index - copied to hub ram
end                     res     0

'-------------------------------------------------------------------------------

{ SERIAL COM PORT DRIVER - specify pins,mode,speed,buffers
DAT
                        org
comport                 mov     t1,par                  'get structure address
                        add     t1,#4 << 2              'skip past heads and tails
                        rdlong  t2,t1                   'get rx_pin bit# (or tr)
                        mov     rxpin,#1
                        shl     rxpin,t2                ' rxpin has 1 bit set

                        add     t1,#4                   'get tx_pin  (or te)
                        rdlong  t2,t1
                        mov     txpin,#1
                        shl     txpin,t2                '

                        add     t1,#4                   'get rxtx_mode
                        rdlong  config,t1

                        mov     tepin,#0                'prepare tepin
                        test    config,#|<rs485  wz     'rs485?
                if_nz   mov     tepin,txpin             ' the second parameter is actually the transmit enable
                if_nz   mov     txpin,rxpin             ' rx and tx pin are shared so tx is same as rx
                if_nz   andn    outa,tepin              ' make sure te is low
                if_nz   or      dira,tepin              ' activate te

                        add     t1,#4                   'get bit_ticks
                        rdlong  bitticks,t1

                        add     t1,#4                   'get buffer_ptr
                        rdlong  rxbuff,t1
```

```
                        mov     txbuff,rxbuff
                        add     txbuff,_rxsz                'txbuff is = rxbuff+rxsz

                        test    config,#|<open  wz          'init tx pin according to mode
                        test    config,#|<invtx wc
          if_z_ne_c     or      outa,txpin
          if_z          or      dira,txpin


                        mov     bits,#databits+1
                        test    config,#|<parity wz
          if_nz         add     bits,#1                     'allow an extra bit for parity

                        mov     txcode,#transmit            'initialize ping-pong multitasking
'                       mov     breakcnt,#40                ' break duration must be at least 40 characters
'                       mov     breakcnt+1,#2               '
                        mov     gaptimer,#0

'' ************************ RECEIVE ***************************

receive                 add     gaptimer,#1

                        jmpret  rxcode,txcode               'run a chunk of transmit code, then return
                        test    config,#|<invrx  wz         'wait for start bit on rx pin
                        test    rxpin,ina                   wc
          if_z_eq_c     jmp     #receive

                        'time sample for middle of start bit
                        mov     rxcnt,bitticks              'read baudrate delay
'                       sub     rxcnt,rxstcomp
                        shr     rxcnt,#1                    'generate 1/2 bit start delay
                        add     rxcnt,cnt                   'set rxcnt as target sample time
:rxstart                'wait for middle of start bit
          if_z          jmpret  rxcode,txcode
                        mov     t1,rxcnt                    ' check if bit receive period done
                        sub     t1,cnt
                        cmps    t1,#0                       wc
          if_nc         jmp     #:rxstart
                        'sample middle of start bit
                        test    config,#|<invrx  wz         'if rx inverted, invert data
                        test    rxpin,ina                   wc          'sample middle of start bit
          if_c_and_z    jmp     #receive
          if_nc_and_nz  jmp     #receive

'
' START bit validated
'
                        mov     rxbits,bits
                        call    #rxchar                     'shift in character+parity+stopbits
                        test    config,#|<invrx  wz         'if rx inverted, invert byte
          if_nz         xor     rxdata,ones

                        mov     r0,#1
                        shl     r0,bits                     ' generate mask for STOP bit
                        shr     r0,#1

                        test    rxdata,r0 wz                ' test stop
          if_z          jmp     #receive
                        mov     rxparbit,rxdata
                        shr     rxparbit,#(databits)        ' get rx parity into bit 0
                        and     rxdata,#(|<databits)-1  'mask to suit bit size

                        '!!! what are we going to do with received parity???
:rxstore
                        rdlong  t2,par                      'save received byte and inc head
                        add     t2,rxbuff
                        wrbyte  rxdata,t2
                        sub     t2,rxbuff
                        add     t2,#1
                        and     t2,_rxmask
                        wrlong  t2,par
                        mov     breakcnt,#40                'reset any break detection in progress
                        jmp     #receive                    'byte done, receive next byte
{
:cntbreak               djnz    breakcnt,#receive
                        djnz    breakcnt+1,#receive
                        rdbyte  t2,#4
                        movd    :clkwr1,t2
                        or      t2,#$80
                        movd    :clkwr,t2
                        nop
:clkwr                  clkset  %11101110                   'reset the cpu (manually set CLK)
```

```
:clkwr1                 clkset  %11101110               'reset the cpu (manually set CLK)
                        jmp     $                       'takes a while to reset, meanwhile..
}
' receive one character + <parity> + stop
'
rxchar                  mov     rxhold,#1               'shifting mask starting from lsb
                        mov     rxdata,#0
:rxlp                   add     rxcnt,bitticks          'ready next bit period
'                       sub     rxcnt,rxbitcomp

:rxwait                 jmpret  rxcode,txcode           'if fullduplex run a chuck of transmit code
                        mov     t1,rxcnt                ' check if bit receive period done
                        sub     t1,cnt
                        cmps    t1,#0                   wc
          if_nc         jmp     #:rxwait

                        test    rxpin,ina               wc          'receive bit on rx pin
          if_c          or      rxdata,rxhold           ' merge next data bit (rxhold) if 1 (c)
                        shl     rxhold,#1               'promote data mask to next highest bit (0..7)
                        djnz    rxbits,#:rxlp
rxchar_ret              ret



'
'' ***************************** TRANSMIT ****************************

transmit                jmpret  txcode,rxcode           'run a chunk of receive code, then return

                        mov     t1,par                  'check for head <> tail  (hub memory)
                        add     t1,#2 << 2
                        rdlong  t2,t1
                        add     t1,#1 << 2
                        rdlong  t3,t1
                        cmp     t2,t3                   wz
          if_nz         jmp     #transmitbyte           ' not empty

''TX EMPTY
                        andn    outa,tepin              'release TE and assert 'RE
                        test    config,#|<rs485 wz
          if_nz         andn    dira,txpin              'release TX
                        test    config,#|<open wz
          if_nz         andn    dira,txpin              'release TX
                        jmp     #transmit

transmitbyte
                        or      dira,txpin
                        or      outa,tepin              'assert TE (if set)

                        add     t3,txbuff               'get byte and inc tail
                        rdbyte  txhold,t3
                        sub     t3,txbuff               ' convert back to an index
                        add     t3,#1                   ' inc txread index
                        and     t3,_txmask              ' wrap
                        wrlong  t3,t1                   ' update
                        mov     txdata,txhold           ' load transmit register
                        shl     txdata,#1               ' add a start bit

'
' Send start bit and data bits
'
                        mov     txbits,#databits+1
                        call    #txdat                  ' transmit start bit + data bits

'' Send optional PARITY
'                               !!!! could use TEST to generate parity in C
                        test    config,#|<parity wz
          if_z          jmp     #txstop
                        mov     txdata,#0
                        mov     txbits,#databits
:par                    shr     txhold,#1 wc
          if_c          add     txdata,#1
                        djnz    txbits,#:par
                        test    config,#|<odd wz
          if_nz         xor     txdata,#1               'invert parity
                        mov     txbits,#1
                        call    #txdat

'' Send STOP bits

txstop                  mov     txbits,#stopbits        'stop bits ( 1 or more )
```

```
                        mov     txdata,#$1FF
                        call    #txdat

                        mov     tecnt,bitticks
                        add     tecnt,tecnt
                        add     tecnt,cnt
                        jmp     #transmit                   'byte done, transmit next byte

'' Send START + DATA

txdat                   mov     txcnt,cnt
                        sub     txcnt,txbitcomp
txbit
                        test    config,#%10  wc
          if_c          xor     txdata,#1                   ' invert
                        shr     txdata,#1                   wc          ' lsb first
                        muxc    outa,txpin                  ' output bit
                        add     txcnt,bitticks              ' ready next cnt
'                       sub     txcnt,txbitcomp

:txwait
                        test    config,#|<half wz
          if_z          jmpret  txcode,rxcode               'run a chunk of receive code, then return
                        mov     t1,txcnt                    'check if bit transmit period done
                        sub     t1,cnt
                        cmps    t1,#0                       wc
          if_nc         jmp     #:txwait

                        djnz    txbits,#txbit               'another bit to transmit?
txdat_ret               ret



ones                    long    -1
rxstcomp                long    25
rxbitcomp               long    25
txbitcomp               long    40
_rxsz                   long    rxsz
_rxmask                 long    rxsz-1
_txmask                 long    txsz-1
'
'
' Uninitialized data
'
t1                      res     1
t2                      res     1
t3                      res     1
r0                      res     1
bits                    res     1

config                  res     1
bitticks                res     1

rxpin                   res     1                           'mask of rx pin
rxbuff                  res     1                           'pointer to rxbuf in hub memory
rxdata                  res     1                           'assembled character
rxbits                  res     1                           'counter
rxcnt                   res     1
rxcode                  res     1
rxhold                  res     1

txpin                   res     1                           'mask of tx pin
txbuff                  res     1
txdata                  res     1
txhold                  res     1
txcnt                   res     1
txcode                  res     1

tepin                   res     1
tecnt                   res     1
txbits                  res     1

breakcnt                res     2
lastchar                res     1
gaptimer                res     1
rxparbit                res     1


}
```

```
{ Boot-time Spin startup - launches Tachyon into all the remaining cogs and starts serial receive in this cog }


PUB Start
  long[@rxticks] := long[@txticks] := clkfreq / long[@baudrate] ' Force VM transmit routine to correct baud
  cognew(@RESET, @TERMINAL)                                     ' Tachyon terminal task

  cognew(@RESET, @IDLE)                                                 ' clone Tachyon kernel into remaining cogs
  cognew(@RESET, @IDLE)
  cognew(@RESET, @IDLE)
  cognew(@RESET, @IDLE)
  cognew(@RESET, @IDLE)
  cognew(@RESET, @IDLE)

  coginit(0,@HSSerialRx, @rxbuffers+s)                                  ' finally reassign the Spin cog for serial coms

  '''vga.start(16, @Colors, Pixels, @vgasync)                  ' VGA
```

{ Extend this kernel by pasting or sending EXTEND.fth to the Prop running the kernel - use 20ms line delay }