```
'' +--------------------------------------------------------------------------+
'' | Cluso's LMM_SerialDebugger for Propeller II (DE0-Nano Emulator)    v0.xx |
'' +--------------------------------------------------------------------------+
'' |  Authors:       (c)2013 "Cluso99" (Ray Rodrick)                          |
'' |  License:       MIT License - See end of file for terms of use           |
'' +--------------------------------------------------------------------------+
'' |  Acknowledgements: Bill Henning - original LMM methodology               |
'' |                    Andy (Ariba) - help with LMM call format              |
'' |                    Chip Gracey  - original P2 ROM Monitor                |
'' |                    Chip & Parallax - P2 and DE0 emulation & expansion pcb|
'' |                    Chris (Sapieha) - help with features & testing        |
'' +--------------------------------------------------------------------------+
''#############################################################################################
''##                                                                                         ##
''##    ------  Used for  CUT  and  PASTE  in Debuger window ------ After <Ctrl-L twice in PNut ##
''##                                                                                         ##
''##            p2load -b 115200 -v -s LSD_067.obj -h -T                                     ##
''##                                                                                         ##
''#############################################################################################
''
'' for: Terasic DE0-Nano & DE2-115 Prop2 Emulators
'' ------------------------------------------
'' The intended purpose is to make a simple debugger for single cog testing on DE0.
'' It is my intention to port this back to the Prop1.
''
'' RR20130317  Cluso99  Tx routine derived from Chip's P2 Monitor
'' RR20130320  v0.10    LMM & HUB_tx working
''             v0.10a   add functions
'' RR20130322  v0.10c   char/hex/hexrev/str working
''             v0.10c   ascii working
''             v0.10e   add dump (cog/hub)
'' RR20130322  v0.11e   dump working (hub only)
''             v0.20    sample release
'' RR20130323  v0.21a   Tx now can send up to 4 chars in one LmmTx call
''                  c   add cog mode to dump (not working!!)
''                  d   add Rx routine (working) <esc> to exit to monitor
'' RR20130324  v0.22    dump cog mode not working
'' RR20130327  v0.23a   fix _LmmRX branch backwards
''             v0.25    sample release
```

```
''                      WARNING: There seems to be a bug with pnut that will not
''                              compile code past $165F. It seems to use $1660-167F.
''                              DAT and ORG do not seem to matter.
''                              i.e code seems restricted to $800 longs = 2KB
''              v0.26x  add "byte 0[$E80] at the beginning to ensure correct hub addresses w/o offset
''              v0.27   use pnut to compile, and "p2load -v xxxx.obj -s -t" to load;
''              v0.30   sample release
'' Note: Rather than tidy up the code, I am proceeding on adding new features such as rxstring, etc
'' RR20130329  v0.32   add rxstring (see restrictions to be fixed within code)
''              v0.33   temporary monitor command to dump working
''              v0.34
''              v0.35   temp release
'' RR20130401  v0.36   use p2load v0.002 "p2load -v -s lsd_036.obj" to load and switch to PST quickly
''               0.37   rxstring to use lmm_ptr2 (same as dump)(not SP)
''                      DEBUG working - but single time only.
''               0.38   lmm_sp -> lmm_p3 freeup the stackpointer for use as a stack
''               0.39   use p2load v0.004a "p2load -v -s lsd_039.obj -h" or "p2load -v -s lsd039.obj -c 0,1000:0"
''                        and PST.
''               0.40   HEX: options: REV, SPACE, DIGITS 0-7 (8->0) convert routines
''               0.41   use lmm_sp (SP = @_hub_stack) for _HubXxxx calls
''               0.42   convert dump & debug to stack calls; <esc><cr> returns from debug
''               0.43   DEBUG:  xxxxx.<cr> dumps hub; xxx,<cr> dumps cog; <cr> dumps next addr;
''                              <esc><cr> returns to user program
''                      sample release
''               0.44   use __hubtable for mode decode
'' RR20130414   0.45   add _DUMP _CODE option (for Sapieha) _COG
''               0.46   add _DUMP _CODE _HUB and also add to DEBUG using ' and "
'' RR20130415   0.47   DEBUG: add xxx!<cr> for Goto Cog address (i.e. resumes user code at address xxx)
''                      add hex outputs to dump code
''                      Ctl-Z now exits to the ROM MONITOR (was <esc>)
''               0.48   DEBUG:  aaa$dddddddd<cr>   stores long "dddddddd" at cog address "aaa"
''                              aaaaa#dddddddd<cr> stores long "dddddddd" at hub address "aaaaa"
''               0.49   add _RXSTRING _PROMPT option (for debug prompt as suggested by Sapieha)
''               0.50   mark the cog code with ''##### that can be removed (it is just example code)
''                      add _MONITOR for goto rom monitor (saves cog space)
''                      example of TxString with the string residing in hub (thanks Sapieha)
'' RR20130417   0.51   Ctl-Z returns to user cog program
''                      Ctl-Q quits & goto rom monitor
'' Sapieha has verified it works on DE2 also.
```

```
'' RR20130419   0.55    code tidy: now push/pop lmm_y; lmm_y -> lmm_f (function/mode)
''-------------------------------------------------------------------------------------------
''      Call parameters (revised definition)...
''              lmm_f   calling function/mode ( eg: mov lmm_f, #_DEBUG ) (was lmm_y)
''              lmm_x   typically a value
''              lmm_c   typically a count                                   (was lmm_2)
''              lmm_p   typically a pointer to a hub location ( eg: location of a string ) (was sometimes lmm_x)
''      Return parameters (revised definition)...
''              lmm_f   calling function/mode ( eg: mov lmm_f, #_DEBUG ) (was lmm_y)
''                              always returns unchanged (ie: "sticky")
''              lmm_x   typically a value
''                              only valid for specific calls - maybe destroyed
''              lmm_c   typically a count                                   (was lmm_2)
''                              only valid for specific calls - maybe destroyed
''              lmm_p   typically a pointer to a hub location ( eg: location of a string ) (was sometimes lmm_x)
''                              returns set/unchanged/incremented/destroyed, according to the call function/mode
''-------------------------------------------------------------------------------------------
'' RR20130420   0.56    Q quits & goto rom monitor (was Ctl-Q)
''                      revised _LIST options... (was _DUMP)
''                        _LIST         = 3 << 5        ' LIST a line (1/4 longs)   from cog/hub
''                          _ADDR2      = 1 << 4        ' 1= use lmm_c as a to-adress
''                          _COUNT      = 1 << 3        ' 1= use lmm_c to display 'n' lines (counter)
''                          _HDG        = 1 << 2        ' 1=display heading for opcode format
''                          _MON        = 00            ' \ Format 0:   4 longs, rom "MON"itor format
''                          _SMON       = 01            ' |        1:   4 longs, "S"hort rom "MON"itor format (no ascii)
''                          _CODE       = 10            ' |        2:   1 long,  code format
''                          _LONG       = 11            ' /        3:   4 longs, xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx (Sapieha)
''                      revised _LIST parameters...
''                        lmm_p = cog/hub addr (was lmm_x); lmm_2 = count(optional)
''-------------------------------------------------------------------------------------------
'' RR20130421   0.57
'' RR20130421   0.58    calling convention (lmm_p now also used for txstring & rxstring)
''                      add _RXSTRING options _ADDR and _NOLF
'' RR20130422   0.59
'' RR20130423   0.60    ParseHex skips leading spaces
''                      M command has options: M=normal(monitor) M1=short, M2=code, M3=long
''                      L is removed (replaced by M2)
''              0.61    add new parameters for M: xxxxx[.yyyyy][,cc]M[n]<cr> where yyyyy=to-addr cc=count
''                      add new parameter to _DUMP _ADDR2 for from/to address lmm_p to lmm_c
```

```
''              0.62    Change M command to L command
''                      Add code for Sapieha
''              0.64    incl code for Sapieha; change _DUMP to _LIST
''              0.65    add new command "?" for help (thanks to Sapieha for the help detail)
''              0.66    code tidy, add new ReverseByte, enable demo, fix demo bug in hex display
'' RR20130425   0.67    new command "P": displays port PINn & DIRn values (added by Sapieha)
''                      new command "M": moves memory xxxxx.yyyyy,ccM<cr>



''=============================================================================================
{{
'    Direct DEBUG commands... prompt "*"
'    ===================================
    G<cr>                               Return to user program
    0G<cr>                              Goto cog address $0 (restart without initialisation)
    xxxG<cr>                            Goto cog address $xxx

''
-------------------------------------------------------------------------------------------------------------------
--
''---------------------------- List modes suported
-------------------------------------------------------------------------------
''
-------------------------------------------------------------------------------------------------------------------
--
    xxxxxL[n]<cr>                       LIST from cog/hub ----- (Counter of lines to display programed in -- I_countL   Long    10)
    xxxxx [,cc]L[n]<cr>                 LIST from cog/hub ----- (Counter of lines to display programed in -- I_countL   Long    10)
          [,cc]                         (Autochanges Counter of lines to display programed in -- I_countL       Long    4_initialy)
         L[n]                           Repeat LIST of last address and counter
    xxxxx[.yyyyy]L[n]<cr>               LIST from cog/hub -----
                                               xxxxx is 'from addr'
                                               yyyyy is optional 'to addr'
                                               cc    is optional 'New Lines counter'
                                               n     is optional n=1/2/3 for smon/code/long else mon
''
-------------------------------------------------------------------------------------------------------------------
--
```

```
    xxxxx-hh hh hh hh ... hh<cr>        stores byte(s) at hub =OR= store 4 bytes as 1 long at cog
    xxxxx-hhhh hhhh ... hhhh<cr>        stores word(s) at hub
    xxxxx-hhhhhhhh ... hhhhhhhh<cr>     stores long(s) at cog/hub
    Q<cr>                               passes control to the Rom Monitor.

    where xxxxx = cog/hub address, cog <$1FF else hub.
    where G, M, L may be upper or lower case.

'The following commands are not currently working...
          <cr>  Display (repeat DUMP CODE/LIST) from next address.

' CALLING CONVENTION...
'-------------------------------------------------------------------------------------------------
  CALL                          lmm_f             lmm_x             lmm_c             lmm_p
'-------------------------------------------------------------------------------------------------
  LmmTx                         not-used          char(s)/???       not-used          not-used
  LmmRx                         not-used          ???/char          not-used          not-used
  LmmFn  (see below...)
'-------------------------------------------------------------------------------------------------
lmm_f                                             lmm_x             lmm_c             lmm_p
'-------------------------------------------------------------------------------------------------
_ASCII                                            char              ???               ???
_HEX[+_REV][+_SP][+n]                             byte/wd/long/???  ???               ???
_LIST[+_COUNT/ADDR2][+_HDG][+_MON/SMON/CODE/LONG] ???               count|addr2/???   addr/addr++
_TXSTRING                                         ???               ???               addr/???
_RXSTRING[_+ECHO][+_PROMPT]                       [prompt]/???      ???/count         [addr]/addr
_DEBUG                                            ???               ???               ???
_MONITOR                                          ???               ???               ???
'-------------------------------------------------------------------------------------------------
key:      not-used = returns unchanged ; ???/??? = not-used on input/invalid on return
'-------------------------------------------------------------------------------------------------




  Cog call types:
  ===============
  Type 1:Direct call to send value in lmm_x to serial port
               mov     lmm_x, [#] value                '\ 8-bit char...
```

```
              call    #LmmTx                  wz,wc   '/ ... sent to serial port

       Direct call to receive value into lmm_x from serial port
              call    #LmmRx                  wz,wc   '\ recv from serial port...
              mov     ???, lmm_x                      '/ ... 8-bit char...

  Type 2:  Call to perform various functions depending upon the parameter set in "lmm_y"
           "lmm_f" is maintained, so for subsequent identical calls, "lmm_f" need not be reset ("sticky")
           "lmm_x" is destroyed unless specifically stated
              mov     lmm_f, [#] mode + n             '\ optional "mode" and "n" parameter(s)
              mov     lmm_x, [#] value                '| parameter (typically a value)
              call    #LmmFun                 wz,wc   '/


'  ______
'  Modes:
'  ======
'  TX:
'  CHAR:
      Displays (Tx) up to 4x character(s) in long "lmm_x", lowest byte first.
        After the lower byte is sent, lmm_x will have shifted right 8 bits.
        Now if lmm_x==0, sending will terminate.
        If lmm_x==0 on entry, a single "nul" will be sent.
        e.g. tmp long "1" + "2"<<8 + $0D<<16 will send "12",<cr>.
      This permits up to 4x 8-bit characters to be sent using a single long and a single LmmTx call.
      Separate call. Does not require mode to be set and does not alter mode setting for other calls.
              mov     lmm_x, #$0D                     '\ <cr>
              call    #lmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

'  ASCII:
      Displays visible characters $20..$7E (" ".."~"). All other characters are displayed in hex between "<xx>".
      See note below regarding "stickiness" of the mode setting.
              mov     lmm_f, #_ASCII+0                '\ set ascii mode
              mov     lmm_x, #$0D                     '| <0D>
              call    #LmmFun         wz,wc           '/

'  TXSTRING:
      Displays a "nul" terminated string located in hub memory.
      See note below regarding "stickiness" of the mode setting.
              mov     lmm_f, #_STRING+0               '\ set string mode
```

```
                mov     lmm_x, ptrstring1               '| set string hub address
                call    #LmmFun         wz,wc           '/

'  HEX:
                Displays the value in "lmm_x" with the number of hex digits specified in the low nibble of "lmm_f".
                See note below regarding "stickiness" of the mode setting.
        _REV    Reverse option: reverses byte order before display
        _SP     Space option:   space after each pair of digits
        n       Digits option:  0..7 where 0=8 digits
                mov     lmm_f, #_HEX+_REV+_SP+0         '\ set hex reversed w spaces and 8(=0) digits
                mov     lmm_x, temp                     '| load some 32bit value
                call    #LmmFun         wz,wc           '/ call LmmFun routine (saves and restores Z & C flags)

'  LIST:
                LIST from hub/cog a line of hex and ascii data - 4 longs (16 bytes).
                The address pointer is maintained and updated.
                Therefore, additional calls can be made without requiring an update to either the address pointer or mode.
                Cog/Hub mode is determined by the address: cog < $200, else hub
        _ADDR2  To-Addr option: display from 'addr' to 'addr2'
        _COUNT  Count option:   display 'count' lines
        _HDG    Heading option: display the appropriate heading
         _MON   Monitor       format 0: 16 hex pairs + 16 ascii chars
         _SMON  Short Monitor format 1: 16 hex pairs
         _CODE  Code          format 2: instruction format + hex rev long + hex long + 4 ascii chars
         _LONG  Long          format 3: 4 hex longs (Sapieha)
                mov     lmm_f, #_LIST                   '\ set dump mode
                mov     lmm_p, ptrdump                  '| set hub address
                mov     lmm_c, #count                   '| optional 'count' lines
                call    #LmmFun         wz,wc           '/

'  RX:
                Waits for, and receives a single serial character.
                The character is placed in lmm_x, and the call returns.
                The character is not "echoed" out the serial port.
                call    #LmmRx          wz,wc           '\ call LmmRx routine (saves and restores Z & C flags)
                mov     ?????, lmm_x                    '/ returns with char in lmm_x bits b7..0 (other bits =0)

'  RXSTRING:
                Waits for, and receives a string of characters, terminated in <cr>.
```

```
                The string is terminated in a <nul> after the inclusion of the <cr>.
                This routine does not wait for a following <lf>.
                An optional +_ECHO parameter is included
                The string hub location (pointer) is returned in lmm_x (not lmm_f because that remains unchanged for
                  future identical calls.
        _ECHO   Echoes option:      echo each recvd char
        _PROMPT Prompt option:      displays prompt char(s) in lmm_x
        _ADDR   Address option:     use hub buffer ptr in lmm_p
        _NOLF   Strip <lf> option: strips <lf>
                mov     lmm_f, #_RXSTRING+_ECHO         '\ set rxstring mode: options _ECHO
                mov     lmm_p, stringptr                '| optional hub addr (ptr)
                call    #LmmFun         wz,wc           '| call LmmFun  routine (saves and restores Z & C flags)
                mov     ?????, lmm_p                    '| returns with address of the hub string, <nul> terminated
                mov     ?????, lmm_c                    '/ returns with count of the chars entered (incl <cr>, excl <nul>)

'  DEBUG:
                Passes control to a Debug/Monitor routine which can...
                        List    HUB or COG memory
                        Move    HUB or COG memory
                        Change HUB or COG memory
                        Return to the user program
                mov     lmm_f, #_DEBUG                  '\ set debug mode
                call    #LmmFun         wz,wc           '/

'  MONITOR:
                Passes control to the Rom Monitor after displaying message
                mov     lmm_f, #_MONITOR                '\ set goto rom monitor mode
                call    #LmmFun         wz,wc           '/

'  _____________
'  Mode Setting:
'  -------------
        The mode plus options are set by storing the mode+options in "lmm_f" in bits b8..0
        When the mode is set, it remains valid until a new mode is set (i.e mode/options are "sticky")
        Therefore, the mode/options are not required to be set for any subsequent same mode/options calls.
}}

''=============================================================================================
```

```
''============================[ CON ]=================================================================
CON
  _ROM_SIZE     = $E80                          ' length of ROM (i.e. the start of Hub Ram)
  _RESERVED     = $180                          ' reserved for drivers $00E80-$00FFF
  _COG_BASE     = _ROM_SIZE+_RESERVED           ' cog code entry
  _COG_SIZE     = $1F8*4                        ' cog code size 2048-32 = ~2KB (+8 longs of special registers)
  _HUB_BASE     = $01800                        ' start of LMM HUB based code

  _HUBBUFSIZE   = 80                            ' RxString default size


  _clkmode = xinput
  _xinfreq = 60_000_000
  _baud    = 115_200
  _bitrate = _xinfreq / _baud
  _period  = 511                                ' approx bitrate for 60MHz & 115,200 baud (permits immediate operand)
  _txpin   = 90                                 ' P90=SO
  _rxpin   = 91                                 ' P91=SI


'' LMM Call 'Modes...
' order must match _hubtable
'
  _MODE         = $F << 5                       ' mode bits defining the call b8..b5 (b4..b0 are modifier options)
  _SHIFT        = 5                             ' shr # to extract mode bits
' _CHAR         = 0 << 5                        ' tx char
  _ASCII        = 1 << 5                        ' ascii <00>..<1F>," ".."~",<7F>..<FF>
  _HEX          = 2 << 5                        ' hex...
    _REV                = 1 << 4                '   - reverse byte order
    _SP                 = 1 << 3                '   - space between hex output pairs
   '_DIGITS             = 7..0 where 8->0       '   - no. of digits to display
  _LIST         = 3 << 5                        ' LIST memory line (1/4 longs) from cog/hub
    _ADDR2              = 1 << 4                ' 1= use lmm_c as to-address
    _COUNT              = 1 << 3                ' 1= use lmm_c to display 'n' lines (counter)
    _HDG                = 1 << 2                ' 1=display heading for opcode format
    _MON                = 0                     ' \ Format 0:   4 longs, rom "MON"itor format
    _SMON               = 1                     ' |        1:   4 longs, "S"hort rom "MON"itor format (no ascii)
    _CODE               = 2                     ' |        2:   1 long,  code format
    _LONG               = 3                     ' /        3:   4 longs, xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx (Sapieha)
```

```
  _TXSTRING     = 4 << 5                        ' tx string (nul terminated) from hub
  _RXSTRING     = 5 << 5                        ' rx string
    _ECHO               = 1 << 4                '    - echo char
    _PROMPT             = 1 << 3                '    - prompt (lmm_x)
    _ADDR               = 1 << 2                '    - addr of string buffer supplied
    _NOLF               = 1 << 1                '    - strip <lf>
  _DEBUG        = 6 << 5                        ' debug/monitor
  _MONITOR      = 7 << 5                        ' goto rom monitor
  _MOVE         = 8 << 5                        ' MOVE memory
  _UNKNOWN      = 9 << 5                        ' this and above are invalid/unknown

''*****************************************************************************************
''======= Cluso's DE0 Constants ======[ CON ]=============================================
''*****************************************************************************************
' set $1F8 = $01555400 = PINA, $1FC = $01555400 = DIRA, turns all leds on
_LED10  = 10    ' P10 1=LED ON
_LED12  = 12
_LED14  = 14
_LED16  = 16
_LED18  = 18
_LED20  = 20
_LED22  = 22
_LED24  = 24



''*****************************************************************************************
''======= Chj_Constants ======[ CON ]=====================================================
''*****************************************************************************************
''CON
''
'OPT    FRS,BRS         forward ref.'s & branches default to short
''
CR      =       $0D             'ASCII equates
LF      =       $0A
TAB     =       $09
CTRLC   =       $03
CTRLH   =       $08
CTRLS   =       $13
```

```
CTRLX   =       $18
''
BUFLEN  =       80              'length of keyboard input buffer
'_____________________________________________________________________________ Chj_Constants
  _Buffer       =$E80
 _PINA  = $1F8                   '' COG PINA - IN/OUT Register
 _PINB  = $1F9                   '' COG PINB - IN/OUT Register
 _PINC  = $1FA                   '' COG PINC - IN/OUT Register
 _PIND  = $1FB                   '' COG PIND - IN/OUT Internal Register
 _DIRA  = $1FC                   '' COG DIRA - IO-Direction Register
 _DIRB  = $1FD                   '' COG DIRB - IO-Direction Register
 _DIRC  = $1FE                   '' COG DIRC - IO-Direction Register
 _DIRD  = $1FF                   '' COG DIRD - Special Internal Register

''------------ DE2-115 Test LED, Switch -------
'' --- Port A '---------LED --------------------
  LED15 = 29                    '' DE2-115
  LED16 = 30                    '' DE2-115
  LED17 = 31                    '' DE2-115
'' --- Port B '---------------------------------
  LED0  = 32                    '' DE2-115
  LED1  = 33                    '' DE2-115
  LED2  = 34                    '' DE2-115
  LED3  = 35                    '' DE2-115
  LED4  = 36                    '' DE2-115
  LED5  = 37                    '' DE2-115
  LED6  = 38                    '' DE2-115
  LED7  = 39                    '' DE2-115
  LED8  = 40                    '' DE2-115
  LED9  = 41                    '' DE2-115
  LED10 = 42                    '' DE2-115
  LED11 = 43                    '' DE2-115

'' --- Port A '----------SW / KEY --------------
  KEY1  = 29                    '' DE2-115
  KEY2  = 30                    '' DE2-115
  KEY3  = 31                    '' DE2-115
'' --- Port B '---------------------------------
  SW0   = 32                    '' DE2-115
```

```
  SW1    = 33                    '' DE2-115
  SW2    = 34                    '' DE2-115
  SW3    = 35                    '' DE2-115
  SW4    = 36                    '' DE2-115
  SW5    = 37                    '' DE2-115
  SW6    = 38                    '' DE2-115
  SW7    = 39                    '' DE2-115
  SW8    = 40                    '' DE2-115
  SW9    = 41                    '' DE2-115
  SW10   = 42                    '' DE2-115
  SW11   = 43                    '' DE2-115
''
''=== $01000 ======================[ DAT ]============================================================
DAT
               byte    0[_ROM_SIZE]            ' filler for ROM space $e80
               byte    "^"[_RESERVED]          ' filler for reserved HUB space $180
'$01000

               ORG     0
entry                                          'USER space

        ''#########################################################################################
        ''#             The following is example code and may be deleted.                         #
        ''#########################################################################################
        '' This is a sample program to perform various calls to the LMM_SerialDebugger...

        'the following is a 0.5 sec delay mechanism only (allows PST to start)
                        getcnt  wait
                        add     wait, delay500ns                ' 0.5 sec
        '               add     wait, delay5s                   ' 5 secs
                        waitcnt wait,0
                        jmp     #entry2

''==== 777777 ========================================================================================
''-7- ###################################################################################################
''-7-------------------------------------------------------------------------------------------------
''-7----              "ReEnterDebug" Need always be placed at end of user code ---------------------------------
''-7-------------------------------------------------------------------------------------------------
''#########################################################################################
```

```
'               jmp     #ReEnterDebug           'If enabled Bypas Code after this instruction   #
''#########################################################################################
''-7-------------------------------------------------------------------------------------------
''-7-------------------------------------------------------------------------------------------
''                   "ReEnterDebug" ----- Reenter DEBUGGER                 ----------------------------------
''                    ---------------- NEEDS for bypas area not used in COG  ----------------------------------
''-7-------------------------------------------------------------------------------------------
''-7-------------------------------------------------------------------------------------------
''-7- ######################################################################################################
''==== 777777 ==========================================================================================
''
'                                                       Variables to Cluso_s demos

        wait            long    0
        delay500ns      long    _xinfreq /2                     ' 0.5 sec
        delay5s         long    _xinfreq * 5                    ' 5 sec
        temp            long    $1E0
        _LMM            long    $1E0
        tempb           long    ReEnterDebug
        temp2           long    "1" + "2"<<8 + $0D<<16

        entry2                                                  'Cluso_s demos Part 2

        ''#########################################################################################
        ''#             The following is example code and may be deleted.                         #
        ''#########################################################################################
        '' This is a sample program to perform various calls to the LMM_SerialDebugger...

        '-----------------------------------------------------------------------------------------
        '-----           jmp     #Rom_Monitor                    ' Goto 'the P2 monitor via the debugger
        '-----------------------------------------------------------------------------------------

                ' Cluso's test code can be commented out (add 2x opening braces to the start of the line)

        ' HEX
                        mov     lmm_f, #_HEX+0                  '\ set hex mode with 8 digits (8=0=default)
                        mov     lmm_x, temp                     '| load some 32bit value
                        call    #LmmFun         wz,wc           '/ call LmmFun  routine (saves and restores Z & C flags)
```

```
' TX: " "
                mov     lmm_x, #" "                     '\ " "
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

' HEX REVERSED
                mov     lmm_f, #_HEX+_REV+0             '\ set hex rev mode with 8 digits (8=0=default)
                mov     lmm_x, tempb                     '| load some 32bit value
                call    #LmmFun         wz,wc           '/ call LmmFun  routine (saves and restores Z & C flags)

' TX: <cr>
                mov     lmm_x, #$0D                     '\ <cr>
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

' display hub string
                mov     lmm_f, #_TXSTRING               '\ set string mode
                mov     lmm_p, strptr_version           '| set string hub address (string is in hub)
                call    #LmmFun         wz,wc           '/

' tx "789",<cr>
                mov     lmm_x, #"7"                     '\ "7"
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
                mov     lmm_x, #"8"                     '\ "8"
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
                mov     lmm_x, #"9"                     '\ "9"
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
                mov     lmm_x, #$0D                     '\ <cr>
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

' TX: "12",<cr> (note: we can send up to 4 chars in lmm_x using the LmmTx routine)
                mov     lmm_x, temp2                    '\ "12",<cr>
                call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

' ASCII $0D
                mov     lmm_f, #_ASCII+0                '\ set ascii mode
                mov     lmm_x, #$0D                     '| <0D>
                call    #LmmFun         wz,wc           '/
' ASCII "%"
                mov     lmm_x, #"%"                     '| "%"
                call    #LmmFun         wz,wc           '/
```

```
        ' ASCII $95
                        mov     lmm_x, #$95                     '| <95>
                        call    #LmmFun         wz,wc           '/

        ' TX: <cr>
                        mov     lmm_x, #$0D                     '\ <cr>
                        call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

        ' HEX REVERSED
                        mov     lmm_f, #_HEX+_REV+0             '\ set hex rev mode with 8 digits (8=0=default)
                        mov     lmm_x,temp                      '| load some 32bit value
                        call    #LmmFun         wz,wc           '/ call LmmFun  routine (saves and restores Z & C flags)
        ' can use an intervening call to the TX routine without corrupting lmm_f (mode)
                        mov     lmm_x, #$0D                     '\ <cr>
                        call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
        ' HEX REVERSED without reloading lmm_f
                        mov     lmm_x,ReEnterDebug              '| load some 32bit value
                        call    #LmmFun         wz,wc           '/ call LmmFun  routine (saves and restores Z & C flags)

''============== Show LMM position ========--- Needed for USER code to PNut ---======================
        ' TX: <cr>
                        mov     lmm_x, #$0D                     '\ <cr>
                        call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
        ' TX: <cr>
                        mov     lmm_x, #$0D                     '\ <cr>
                        call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
''-------------- Show LMM position ------------Needed for USER code to PNut
        ' HEX
                        mov     lmm_f, #_HEX+0                  '\ set hex mode with 8 digits (8=0=default)
                        mov     lmm_x, _LMM                     '| load some 32bit value
                        call    #LmmFun         wz,wc           '/ call LmmFun  routine (saves and restores Z & C flags)

        ' TX: <cr>
                        mov     lmm_x, #$0D                     '\ <cr>
                        call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)
''-------------- Show ReEnterDebug position ---Needed for USER code to PNut
        ' HEX
                        mov     lmm_f, #_HEX+0                  '\ set hex mode with 8 digits (8=0=default)
                        mov     lmm_x, #ReEnterDebug                    '| load some 32bit value
```

```
                        call    #LmmFun         wz,wc           '/ call LmmFun  routine (saves and restores Z & C flags)
''=============================================================================
        ' TX: <cr>
                        mov     lmm_x, #$0D                     '\ <cr>
                        call    #LmmTx          wz,wc           '/ call LmmTx routine (saves and restores Z & C flags)

        ' LIST line(s) from cog/hub (4 longs = 16 bytes)
                        mov     lmm_f, #_LIST+_HDG+_COUNT+_MON  '\ set LIST (monitor) mode
                        mov     lmm_c, #4                       '| do n lines
                        mov     lmm_p, :hubptr                  '| set hub/cog address (HUB)
                        call    #LmmFun         wz,wc           '/
                        mov     lmm_f, #_LIST+_MON              '\ set LIST (monitor) mode
                        call    #LmmFun         wz,wc           '/
        :hubptr         long    $1E0                            ' cog/hub address to LIST (executes as "NOP")

        ' LIST line(s) from cog/hub (4 longs = 16 bytes)
                        mov     lmm_f, #_LIST+_HDG+_ADDR2+_SMON '\ set LIST (short monitor) mode
                        mov     lmm_c, :cogptr02                '| from addr to addr2
                        mov     lmm_p, :cogptr                  '| set hub/cog address (COG)
                        call    #LmmFun         wz,wc           '/
        :cogptr         long    $000                            ' cog/hub address to LIST (executes as "NOP")
        :cogptr02       long    $00f                            ' cog/hub to-address

        ' LIST line(s) from cog/hub (1 long as code)
                        mov     lmm_f, #_LIST+_HDG+_CODE        '\ set LIST (code) mode
                        mov     lmm_p, :cogptr2                 '| set hub/cog address (can be immediate for cog addr)
                        call    #LmmFun         wz,wc           '/
                        call    #LmmFun         wz,wc           '/
        :cogptr2        long    $000                            ' cog/hub address to start LIST (executes as "NOP")

        ' LIST line(s) from cog/hub (4 longs as long)
                        mov     lmm_f, #_TXSTRING               '\ set string mode
                        mov     lmm_p, strptr_long              '| set hub addr of string (code header)
                        call    #LmmFun         wz,wc           '/
                        mov     lmm_f, #_LIST+_LONG             '\ set LIST (long) mode
                        mov     lmm_p, :hubptr2                 '| set hub/cog address (can be immediate for cog addr)
                        call    #LmmFun         wz,wc           '/
                        call    #LmmFun         wz,wc           '/
                        call    #LmmFun         wz,wc           '/
```

```
:hubptr2        long    $1000                           ' cog/hub address to start LIST (executes as "NOP")

'-------------------------------------------------------------------------
'' Here is an example to show a set of instructions... (thanks to Sapieha for the idea)
IStart          ''00000000000 '----------------------- To place before Instruction to Show --------------------
'the following is a 0.5 sec delay mechanism only (allows PST to start)
                getcnt  wait
                add     wait,delay500ns
                waitcnt wait,0
                add     wait,delay5s
                add     wait,delay5s
IEnd            ''11111111111 '---------------------- To place after last Instruction to Show --------------------
'' display the above instructions...
                mov     lmm_f, #_LIST+_CODE+_COUNT+_HDG '\ set LIST cog code
                mov     lmm_c, #(IEnd-IStart)           '| set n lines
                mov     lmm_p, #IStart                  '| set cog address
                call    #LmmFun         wz,wc           '/
'-------------------------------------------------------------------------
'-------------------------------------------------------------------------
'-------------------------------------------------------------------------
'-------------------------------------------------------------------------
'-------------------------------------------------------------------------

''{***
'''Receive a string, then display the string
''              mov     lmm_f, #_RXSTRING               '\ set rx string mode (no echo)
''              call    #LmmFun         wz,wc           '/
''              mov     strptr, lmm_x                   '  save the string pointer
''              mov     lmm_p, lmm_x                    '  x points to hub string
''                mov     lmm_f, #_TXSTRING             '\ set string mode
''                call    #LmmFun         wz,wc           '/
''***}
''
''
''{***
''' RX: wait and receive a serial char
'':rxloop
''                call    #LmmRx          wz,wc           '> call LmmRx routine (saves and restores Z & C flags)
''                cmp     lmm_x, #$1B     wz              '    <esc> ?
```

```
        ''        if_z    jmp     #:done                          '    <esc>  y: exit to monitor
        ''                call    #LmmTx          wz,wc           '  echo it back
        ''                jmp     #:rxloop
        ''
        '':done
        ''                getcnt  wait                            ' delay 1s
        ''                add     wait,delta
        ''                waitcnt wait,0
        ''***}
'''-------------------------------------------------------------------------
''
        ' Cluso's test code can be commented out (add 2x closing braces to the start of the line)
''==== 777777 ==========================================================================================================
''-7- ##################################################################################################
''-7-------------------------------------------------------------------------------------------------------
''-7---- Need always be placed at end of user code -----------------------------------------------------
''-7-------------------------------------------------------------------------------------------------------
''#########################################################################################
'               jmp     #ReEnterDebug           'If enabled Bypas Code after this instruction   #
''#########################################################################################
''-7-------------------------------------------------------------------------------------------------------
''-7-------------------------------------------------------------------------------------------------------
''                    "ReEnterDebug" ----- Reenter DEBUGGER                  ----------------------------------
''                     ---------------- NEEDS for bypas area not used in COG  ----------------------------------
''-7-------------------------------------------------------------------------------------------------------
''-7-------------------------------------------------------------------------------------------------------
''-7- ##################################################################################################
''==== 777777 ==========================================================================================================
''
''#########################################################################################
''##    Hard coded to   $1E0-26 = $1B4                  ' (fixed location for Sapieha)          ##
''
                long    $3A3A3A3A[$1CC-$]               ' fill with "::::"
'               BYTE    "--- LMM Exec ---"
''#########################################################################################
''              Byte    "Free user area ----- $000 to $1C0",$0D
''#########################################################################################
''##                                                                                       ##
''##    ------  Used for  CUT  and  PASTE  in Debuger window ------ After <Ctrl-L twice in PNut ##
```

```
''##                                                                                    ##
''##                    p2load -b 115200 -v -s LSD_067.obj -h -T                        ##
''##                                                                                    ##
''############## the following is example code and may be deleted ##################################
''=============================================================================================
''      Displays RUN, C, Z flags        on DE2-115                                      ==
''      For DE0-NANO You can add LEDs to any port and change LEDxx definitions          ==
''=============================================================================================
''
ReEnterDebug    ORG     $                                       ''ENDS
''
                setp    #led17                                  'Code have be executed else HANG if LED not active
''
        if_z    setp    #led15                                  'Set Z flag returned
        if_nz   clrp    #led15                                  'Set NZ flag returned
        if_c    setp    #led16                                  'Set C flag returned
        if_nc   clrp    #led16                                  'Set NC flag returned
''              ==============
''''=========================================================================================
'''' Return to Debuger                                                                  ====
''''=========================================================================================
''Sapieha - I hAVE DISABLED THIS BECAUSE THE FOLLOWING FALLS THRU AFTER DEBUG RETURNS           ????????????????
''          SO I ADDED A CALL TO THE ROM MONITOR IN CASE OF THIS.                               ????????????????
''          YOU NEED TO RECALC THE ADDRESS!                                                     ????????????????
'' ------------------------------------------------          $3A3A3A3A[$1CC-$]             ' fill with ":::::"

'Cluso - LOOK IF I ADDED ALL THAT NEEDS IN THIS PART
'      - IF YOU NEED ADD SOME MORE THINGS ------------- Adjust longs in this   $3A3A3A3A[$1CC-XX-$]

''{***
'-------------------------------------------------------------------------
' run debugger
                mov     lmm_f, #_DEBUG                  '\ run the debugger
                call    #LmmFun         wz,wc           '/
'-------------------------------------------------------------------------
' display returned from debugger message
                mov     lmm_f, #_TXSTRING               '\ set string mode
                mov     lmm_p, strptr_msg1              '| set hub addr of string msg1
                call    #LmmFun         wz,wc           '/
```

```
''{***
'-------------------------------------------------------------------------
' goto ROM MONITOR
Rom_Monitor     mov     lmm_f, #_MONITOR                '\ set rom monitor mode
                call    #LmmFun         wz,wc           '/ never returns!!
'-------------------------------------------------------------------------

''--}}  ' Sapieha's code commented out
''
''#########################################################################################################
''##    Hard coded to   $1E0-8 = $1D8                   ' (fixed location for Sapieha)          ##
                long    $3A3A3A3A[$1D8-$]               ' fill with "::::"
''#########################################################################################################
''              FIT     $1D8                            ' does not work
LMM_Pos
strptr_version  long    @_str_vers                      ' hub address of version string
strptr_code     long    @_str_hcode                     ' hub address of code header string
strptr_long     long    @_str_hlong                     ' hub address of code header string
strptr_msg1     long    @_str_msg1                      ' hub address of message1
''--------------
RunVar_sL       LONG    @RunVar_s
Pin_Var_sL      LONG    @Pin_Var_s
SchVar_sL       LONG    @SchVar_s
spare000        long    0                               '' ;-spare-
'-------------------------------------------------------------------------

''=========================================================================================================
''#########################################################################################################
''##    COG LMM hard coded to   $1F2-18 = $1E0          ' (fixed location for Sapieha)          ##
                long    $3D3D3D3D[$1E0-$]               ' fill with "===="
''#########################################################################################################

''=======[ Cog LMM 'execution unit & parameter(s) ]============================
''
'' Calls save & restore Z & C flags.
'' Currently 16+2=18 longs.

''-------[ LMM 'entry points ]-------------------------
LmmFun          add     lmm_pc, #4                      ' inc PC (skips _LmmRx jump instr)
```

```
LmmRx           add     lmm_pc, #4                      ' inc PC (skips _LmmTx jump instr)
LmmTx                                                   '
''-------[ LMM 'execution loop ]------------------------
LmmLoop         rdlong  lmm_opcode, lmm_pc              ' rdlong        (read LMM hub instr into OPCODE using PC)
                add     lmm_pc, #4                      ' PC++          (inc PC to next LMM hub instr)
lmm_op2         nop                                     ' rdlong delay  (optional 2nd instruction execution)
lmm_opcode      nop                                     ' rdlong result (execute the LMM hub instr)
                jmp     #LmmLoop                        ' loop
''-------[ LMM 'return address ]------------------------
LmmFun_ret
LmmRx_ret
LmmTx_ret       nop                                     ' stores user cog return addr + Z & C flags (11 bits)
''-------[ LMM 'parameters ]----------------------------
lmm_f           long    0                               ' parameter passed to      LMM routine (function options; returns unchanged)
lmm_x           long    0                               ' parameter passed to/from LMM routine (typically a value)
lmm_c           long    0                               ' parameter passed to/from LMM routine (typically a count)
lmm_p           long    0                               ' parameter passed to/from LMM routine (typically a hub/cog ptr/addr)
''-------[ LMM 'workareas ]----------------------------
'  Internal workareas used by the LMM routines.
lmm_pc          long    @_LmmCogReturn + 4              ' LMM PC (program counter)
lmm_sp          long    @_hub_stack                     ' LMM SP (stack pointer)
lmm_w           long    0                               ' workarea for LMM_call
lmm_mode        long    0                               ' saved in LIST
''-------[ LMM 'additional workareas ]--------------------------
'' Additional internal workareas (may not be required in a later revision)
lmm_p2                                  '\\              ' typically a 2nd address (shared with lmm_v)
lmm_v           long    0               '//              ' workarea value
lmm_bittime     long    _bitrate                        ' bit rate for baud
'-------------------------------------------------------------------------------

''#############################################################################################
''# The following is required by Sapieha for his testing. DON'T move, BUT it may be removed    #
''#############################################################################################
''              FIT     $1F2                            '' Does not work!!
'' ------------------------------------------------------
'' |Change        and        to show diferent part of COG |
'' | COG   IStart    List_Count   Var_s_Start   Var_Count |
'' ------------------------------------------------------
''   1F2- 00000000    0000000A    000001E4    00000008
```

```
''-------------------------------------------------------------------------------------------
I_Start         long    $0                              '' IStart
I_countL        long    5                               '' (@I_End - @I_Start+1)    ''IEnd - IStart
Vars_PL         long    LMM_Pos                         '' ??? Have same position as     BYTE    "INDA"
Vars_C          long    8                               '  ??? hub address to LIST (executes as "NOP")
                BYTE    "INDA"                          ''\ Important that 2 BYTE fill position's
                BYTE    "INDB"                          ''/   are in this place
''-------------------------------------------------------------------------------------------
PinRegs         byte    "== $1F8 CODE ==="              ' Place holders for $1F6 (COG I/O registers) Don't remove
DirRegs         byte    "== $1FC CODE ==="              ' Place holders for $1F7 (COG I/O registers) Don't remove
''###########################################################################################

''===========================================================================================
''===========================================================================================

DAT
''###########################################################################################
''##    HUB LMM hard coded to   $01800                                                     ##
''###########################################################################################

''=======[ Hub Resident LMM Routines ]==$01800=================================
                ORG     0                               ' used for label calcs
_HubResident

''-------[ Return to Cog & restores Z & C flags ]-----------------------------
_LmmCogReturn   jmp     LmmTx_ret               wz,wc   ' return to COG user code (indirect & restore Z & C)
'       Upon return, the PC will be left pointing to the following instruction (at _LmmJumpTable).
'       The next user call may modify (add) to PC to skip a number of the following instructions.
'       Therefore, the following group of instructions form a set of "jump tables".
''-------[ First time, or next time thru, code starts here ]------------------
_LmmJumpTable                                           ' these are jump tables:
                add     lmm_pc, #(_LmmTxChar-($+1))*4   '> branch forward (jump to transmit char)
                add     lmm_pc, #(_LmmRxChar-($+1))*4   '> branch forward (jump to receive  char)
                add     lmm_pc, #(_LmmFunction-($+1))*4 '> branch forward (jump to function call)
'-------------------------------------------------------------------------------

''-------[ Tx: Transmit a character in "lmm_x" ]------------------------------
_LmmTxChar
                rdlong  lmm_v, lmm_pc                   ' \ load ptr to _hubtable+
```

```
                long    @__TX                           ' /
                add     lmm_pc, #(_LmmFunct0-($+1))*4   '> br fwd:
'-------------------------------------------------------------------------

''-------[ Rx: Receive a character into "lmm_x"]-------------------------------
_LmmRxChar
                rdlong  lmm_v, lmm_pc                   ' \ load ptr to _hubtable+
                long    @__RX                           ' /
                add     lmm_pc, #(_LmmFunct0-($+1))*4   '> br fwd:
'-------------------------------------------------------------------------

''-------[ Decode the mode and execute the desired function ]-----------------
_LmmFunction

'??? should I reset the SP (just in case my code is wrong)???

                rdlong  lmm_v, lmm_pc                   ' \ load ptr to _hubtable
                long    @__hubtable                     ' /
                mov     lmm_w, lmm_f                    ' copy the mode
                cmp     lmm_w, #_UNKNOWN        wc      ' c if < _UNKNOWN
        if_nc   mov     lmm_w, #_UNKNOWN
                shr     lmm_w, #_SHIFT                  ' extract mode bits
                shl     lmm_w, #2                       '  *4
                add     lmm_v, lmm_w                    '  + offset into table
_LmmFunct0                                              ' here from _LmmTxChar & _LmmRxChar with lmm_v setup
'               'PUSH    lmm_f                          ' \                     < push: function >
                 wrlong  lmm_f, lmm_sp                  ' | PUSH
                 add     lmm_sp, #4                     ' / SP++
'               'FCALL   SP++, @__hubtable+???          ' \                     < call: via __hubtable >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_v                  ' | CALL...
                 nop                                    ' / ...PC = ADDR (from table+offset)

'                ... executes called routine ....

'               'POP     lmm_f                          ' \                     < pop:  function >
                 sub     lmm_sp, #4                     ' | SP--
                 rdlong  lmm_f, lmm_sp                  ' / POP
```

```
'               'FRET    @_LmmCogReturn                  ' \                     < ret: returns to cog program >
                 sub      lmm_pc,#(($+1)-_LmmCogReturn)*4' / br back             (RET to COG)
'-------------------------------------------------------------------------------
__hubtable
' Order must match LMM Call Modes
                long     @_HubTx                         '                       < addr: transmit char(s) >
                long     @_HubAscii                      '                       < addr: display ascii >
                long     @_HubHex                        '                       < addr: display hex >
                long     @_HubList                       '                       < addr: list memory line >
                long     @_HubTxString                   '                       < addr: display string >
                long     @_HubRxString                   '                       < addr: receive string >
                long     @_HubDebug                      '                       < addr: debug/monitor >
                long     @_HubMonitor                    '                       < addr: to rom monitor >
                long     @_HubMove                       '                       < addr: move memory >
                long     @_HubUnknown                    '                       < addr: unknown command >
' And these follow
__TX            long     @_HubTx                         '                       < addr: transmit char(s) >
__RX            long     @_HubRx                         '                       < addr: receive char >
'__NIBBLE       long     @_HubNibble                     '                       < addr: display nibble >
'-------------------------------------------------------------------------------

''=============================================================================================================

''=======[ Hub LMM Routines called from Hub LMM ]==============================

'' These routines call/return using SP (lmm_sp)...
''              'FCALL   SP++, @_HubAscii                '                       < call: display ascii >
''              'FCALL   SP++, @_HubNibble               '                       < call: display nibble >
''              'FCALL   SP++, @_HubTx                   '                       < call: transmit char(s) >
''              'FCALL   SP++, @_HubTxString             '                       < call: display string >
''              'FCALL   SP++, @_HubHex                  '                       < call: display hex >
''              'FCALL   SP++, @_HubRxString             '                       < call: receive string >
''              'FCALL   SP++, @_HubRx                   '                       < call: receive char >
''              'FCALL   SP++, @_HubDebug                '                       < call: debug/monitor >
''              'FCALL   SP++, @_HubList                 '                       < call: list a line >
''              'FCALL   SP++, @_HubUnknown              '                       < call: unknown command >

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

```
''-------[ Unknown Command ]-----------------------------------------------------
_HubUnknown
                mov     lmm_x, #"?"                     '                       (displays "?" and returns to cog)
'               'JUMP    @_HubTx                        ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                 long    @_HubTx                         ' / ...PC = ADDR        (does not return here)
'-------------------------------------------------------------------------------


'-------[ Display Ascii ]-------------------------------------------------------
_HubAscii

{{-----
_HubAscii
' On Entry:
        lmm_x = value                                   '   value
        lmm_f = #_ASCII                                 '   mode
' Call Format:
'               'FCALL   SP++, @_HubAscii               ' \                     < call: display ascii>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubAscii                     ' / ...PC = ADDR
' On Return:
        lmm_x = -invalid-                               '   value (now invalid)
        lmm_f = -same-                                  '   mode (unchanged)
-----}}

                and     lmm_x, #$0FF                    ' only 8bit char
                cmp     lmm_x, #" "             wc      ' c if <$20: visible?
        if_nc   cmpr    lmm_x, #"~"             wc      ' c if >$7E: visible?
        if_nc   add     lmm_pc, #(:txchar-($+1))*4      '> br fwd:  (visible so display)
                mov     lmm_v, lmm_x                    ' save char
                mov     lmm_x, #"<"                     ' "<"
'               'FCALL   SP++, @_HubTx                  ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++                (done in called routine)
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
```

```
                shl     lmm_v, #24                      ' prep for displaying 2 hex nibbles
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++                (done in called routine)
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubNibble                    ' / ...PC = ADDR
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++                (done in called routine)
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubNibble                    ' / ...PC = ADDR
                mov     lmm_x, #">"                     ' ">"
:txchar
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++                (done in called routine)
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR

'               'JUMP   @_HubReturn                     ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubReturn                    ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ Display the "nul" 'terminated string ]------------------------------
_HubTxString

{{-----
_HubTxString
' On Entry:
        lmm_p = hub ptr to string                       '   value
        lmm_f = #_TXSTRING                              '   mode
' Call Format:
'               'FCALL  SP++, @_HubTxString             ' \                     < call: display string>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTxString                  ' / ...PC = ADDR
' On Return:
```

```
        lmm_p = -invalid-                               '   value (now invalid)
        lmm_f = -same-                                  '   mode (unchanged)
' Uses:
        lmm_x
-----}}

:loop           rdbyte  lmm_x, lmm_p            wz      ' get char from string: nul?
        if_z    add     lmm_pc, #(:return-($+1))*4      '> br fwd:              (returns to calling program)
                add     lmm_p, #1                       ' advance hub pointer
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++                (done in called routine)
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
                sub     lmm_pc,#(($+1)-:loop)*4         ' / br back
:return
'               'JUMP   @_HubReturn                     ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubReturn                    ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

'-------[ Display Hex ]---------------------------------------------------------
_HubHex                                                 '                       <--- display hex --->

{{-----
_HubHex
' On Entry:
        lmm_x = value                                   '   value
        lmm_f = #_HEX [+_REV] [+_SP] [+n]               '   mode  (n = digits 0..7 where 0=8)
' Call Format:
'               'FCALL  SP++, @_HubHex                  ' \                     < call: display hex>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubHex                       ' / ...PC = ADDR
' On Return:
        lmm_x = -invalid-                               '   value (now invalid)
        lmm_f = -same-                                  '   mode (unchanged)
-----}}
```

```
                test    lmm_f, #_REV            wz      '  reverse mode?
        if_z    add     lmm_pc, #(:Hub2Hex-($+1))*4     '> br fwd: (no)
' reverse bytes...

'               'FCALL   SP++, @_ReverseBytes           ' \                     < call: reverse bytes >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_ReverseBytes                 ' / ...PC = ADDR

'  this can be improved!

:Hub2Hex
                mov     lmm_v, lmm_x                    ' save orig value
                mov     lmm_x, lmm_f                    ' copy the mode
                and     lmm_x, #$07             wz      ' extract the digits count 0..7 where 0=8
        if_z    add     lmm_pc, #(_Hub2Hex8-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #7               wc      ' c if <7
        if_nc   add     lmm_pc, #(_Hub2Hex7-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #6               wc      ' c if <6
        if_nc   add     lmm_pc, #(_Hub2Hex6-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #5               wc      ' c if <5
        if_nc   add     lmm_pc, #(_Hub2Hex5-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #4               wc      ' c if <4
        if_nc   add     lmm_pc, #(_Hub2Hex4-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #3               wc      ' c if <3
        if_nc   add     lmm_pc, #(_Hub2Hex3-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #2               wc      ' c if <2
        if_nc   add     lmm_pc, #(_Hub2Hex2-($+1))*4    '> branch forwards
                rol     lmm_v, #4                       ' next nibble
                cmp     lmm_x, #1               wc      ' c if <1
        if_nc   add     lmm_pc, #(_Hub2Hex1-($+1))*4    '> branch forwards
```

```
                rol     lmm_v, #4                       ' complete circle (0-->8)

_Hub2Hex8
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubNibble                    ' / ...PC = ADDR
_Hub2Hex7
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubNibble                    ' / ...PC = ADDR
                test    lmm_f, #_SP             wz      ' hex space mode?
        if_z    add     lmm_pc, #(_Hub2Hex6-($+1))*4    '> branch (no)
                mov     lmm_x, #" "                     ' " "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
_Hub2Hex6
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubNibble                    ' / ...PC = ADDR
_Hub2Hex5
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubNibble                    ' / ...PC = ADDR
                test    lmm_f, #_SP             wz      ' hex space mode?
        if_z    add     lmm_pc, #(_Hub2Hex4-($+1))*4    '> branch (no)
                mov     lmm_x, #" "                     ' " "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
```

```
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
_Hub2Hex4
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubNibble                     ' / ...PC = ADDR
_Hub2Hex3
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubNibble                     ' / ...PC = ADDR
                test    lmm_f, #_SP             wz      ' hex space mode?
        if_z    add     lmm_pc, #(_Hub2Hex2-($+1))*4    '> branch (no)
                mov     lmm_x, #" "                     ' " "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
_Hub2Hex2
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubNibble                     ' / ...PC = ADDR
_Hub2Hex1
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubNibble                     ' / ...PC = ADDR
                test    lmm_f, #_SP             wz      ' hex space mode?
        if_z    add     lmm_pc, #(_Hub2Hex0-($+1))*4    '> branch (no)
                mov     lmm_x, #" "                     ' " "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
```

```
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
_Hub2Hex0
'               'JUMP   @_HubReturn                     ' \                     <jump>
                rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

'-------[ Display Nibble ]------------------------------------------------------
_HubNibble                                              '                       <--- Display Nibble --->
{{-----
_HubNibble
' On Entry:
        lmm_x = nibble                                  '   value
        lmm_f = -not used-                              '   mode
' Call Format:
'               'FCALL  SP++, @_HubNibble               ' \                     < call: display nibble>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubNibble                     ' / ...PC = ADDR
' On Return:
        lmm_x = -invalid-                               '   value (now invalid)
        lmm_f = -same-                                  '   mode (unchanged)
-----}}

                rol     lmm_v, #4                       ' to next nibble
                mov     lmm_x, lmm_v                    ' copy value
                and     lmm_x, #$0F                     ' extract nibble
                or      lmm_x, #"0"                     ' make numeric
                cmp     lmm_x,#":"              wc      ' c if <$3A
        if_nc   add     lmm_x,#("A"-"9"-1)              ' convert to A-F if reqd
'---------------falls thru--------------

''-------[ Display the char in "lmm_x" ]-----------------------------------------
_HubTx                                                  '                       <--- transmit character(s) --->
{{-----
```

```
_HubTx
' On Entry:
        lmm_x = char(s)                                 '   value (may contain up to 4 chars, lsb first; <nul> terminates
        lmm_f = -not used-                              '   mode
' Call Format:
'               'FCALL   SP++, @_HubTx                  ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
' On Return:
        lmm_x = -invalid-                               '   value
        lmm_f = -same-                                  '   mode (unchanged)
' Uses:
        lmm_w, lmm_x
-----}}

                setp    #_txpin                         ' do it each time to simplify
                getcnt  lmm_w                           ' get initial time              \ setup bit time
                add     lmm_w,lmm_bittime               ' add bit period to time        /   ...
:next           clrp    #_txpin                         '                               \ start bit=0
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b0
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b1
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b2
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b3
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b4
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b5
                setpc   #_txpin                         ' write c to tx pin             |
```

```
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc      ' get next bit into c           \ b6
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                shr     lmm_x,#1                wc,wz   ' get next bit into c           \ b7
                setpc   #_txpin                         ' write c to tx pin             |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                setp    #_txpin                         '                               \ stop bit =1
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
        if_nz   sub     lmm_pc, #(($+1)-:next)*4        '> br back:  (nz = another char in lmm_x)

'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                 long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ Rx: Receive a char in "lmm_x" ]-------------------------------------
_HubRx                                                  '                       <--- receive character --->
{{-----
_HubRx
' On Entry:
        lmm_x = -anything-                              '   value
        lmm_f = -not used-                              '   mode
' Call Format:
'               'FCALL   SP++, @_HubRx                  ' \                     < call: receive char>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubRx                        ' / ...PC = ADDR
' On Return:
        lmm_x = char                                    '   value
        lmm_f = -same-                                  '   mode (unchanged)
' Uses:
        lmm_x, lmm_w
-----}}

:wait1          getp    #_rxpin                 wc      ' ensure stop/idle
        if_nc   sub     lmm_pc, #(($+1)-:wait1)*4       '> br back:  (not stop/idle)
:wait0          getp    #_rxpin                 wc      ' wait for start                \ start =0 edge
```

```
        if_c    sub     lmm_pc, #(($+1)-:wait0)*4       '> br back:  (stop/idle)        /
                getcnt  lmm_w                           ' get initial time              \ setup 0.5 bit time
                mov     lmm_x, lmm_bittime              ' bit period                    |
                shr     lmm_x, #1                       ' 0.5 bit period                |
                add     lmm_w, lmm_x                    ' add 0.5 bit period to time    |
                waitcnt lmm_w, lmm_bittime              ' center of start bit           /
                getp    #_rxpin                 wc      ' check start bit =0            \ verify start =0
        if_c    sub     lmm_pc, #(($+1)-:wait1)*4       '> br back:  (invalid)          /
                mov     lmm_x, #0                       ' clear char
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b0
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b1
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b2
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b3
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b4
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b5
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b6
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ b7
                rcr     lmm_x, #1                       ' accumulate bit                |
                waitcnt lmm_w,lmm_bittime               ' wait until bit period elapsed /
                getp    #_rxpin                 wc      ' rx pin into c                 \ verify stop bit =1
        if_nc   sub     lmm_pc, #(($+1)-:wait1)*4       '> br back:  (invalid)          /
                shr     lmm_x, #24                      ' shift char into b7..0

'               'JUMP    @_HubReturn                    ' \                     <jump>
```

```
                rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

'-------[ Receive String ]------------------------------------------------------
_HubRxString                                            '                       <--- receive string --->

{{-----
_HubRxString
' On Entry:
        lmm_x = char(s)                                 '   prompt char(s)      (optional)
        lmm_p = stringptr                               '   hub addr of string  (optional)
        lmm_f = #_RXSTRING [+_ECHO]                     '   mode
' Call Format:
'               'FCALL   SP++, @_HubRxString           ' \                     < call: receive string>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubRxString                  ' / ...PC = ADDR
' On Return:
        lmm_p = stringptr                               '   ptr to string
        lmm_c = count                                   '   count of char(s) entered (incl <cr>, excl <nul>)
        lmm_f = -same-                                  '   mode (unchanged)
' Uses:
        lmm_w, lmm_x, lmm_c, lmm_p, lmm_f
-----}}

                test    lmm_f, #_PROMPT         wz      ' prompt ?
        if_z    add     lmm_pc, #(:noprompt-($+1))*4    '> br fwd: (n)
' Display prompt char(s) in lmm_x
'               'FCALL   SP++, @_HubTx                  ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
' setup the hub string address ptr
:noprompt       test    lmm_f, #_ADDR           wz      ' addr supplied option ?
'               'FMOV    lmm_p, #@_hub_buf
        if_z    rdlong  lmm_p, lmm_pc                   '\ move following hub address into PTR
```

```
                long    @_hub_buf                       '/ buf address (executes as nop)
' receive char(s) terminated in <cr>
                mov     lmm_c, #0                       '  set char count=0
:loop
'               'FCALL  SP++, @_HubRx                   ' \                     < call: receive char>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubRx                        ' / ...PC = ADDR
                wrbyte  lmm_x, lmm_p                    '  push to buf (don't inc ptr yet)
                test    lmm_f, #_ECHO           wz      ' echo?
        if_z    add     lmm_pc, #(:noecho-($+1))*4      '> br fwd: (n)
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
                rdbyte  lmm_x, lmm_p                    '  pop fm buf (don't dec ptr) - lmm_x destroyed by Tx
:noecho         cmp     lmm_x, #$0A             wz      ' = <lf> ?
        if_nz   add     lmm_pc, #(:notlf-($+1))*4       '> br fwd: (n)
                test    lmm_f, #_NOLF           wz      ' strip <lf> ?
        if_nz   sub     lmm_pc, #(($+1)-:loop)*4        ' br :loop if strip <lf>
:notlf          cmp     lmm_x, #$08             wz      ' = <bs> ?
        if_nz   add     lmm_pc, #(:notbs-($+1))*4       '> br fwd: (n)
                cmp     lmm_c, #0               wz      ' z if already at start?
        if_z    sub     lmm_pc, #(($+1)-:loop)*4        ' br :loop if ignore <bs>
                sub     lmm_p, #1                       ' n: PTR--
                sub     lmm_c, #1                       ' n: CTR--
                sub     lmm_pc, #(($+1)-:loop)*4        ' br :loop
:notbs          cmp     lmm_c, #_HUBBUFSIZE-2   wc      ' c if < end-of-buf ?
        if_nc   sub     lmm_pc, #(($+1)-:loop)*4        ' br :loop if buf full
                add     lmm_p, #1                       ' PTR++
                add     lmm_c, #1                       ' CTR++
                cmp     lmm_x, #$0D             wz      ' = <cr> ?
        if_nz   sub     lmm_pc, #(($+1)-:loop)*4        ' br :loop if not <cr>

                mov     lmm_x, #0                       '\ load $0 (nul)
                wrbyte  lmm_x, lmm_p                    '/ push to buf
                sub     lmm_p, lmm_c                    ' reset PTR to start of string
```

```
'              'JUMP    @_HubReturn                    ' \                    <jump>
                rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                long    @_HubReturn                    ' / ...PC = ADDR       <returns to calling routine>
'-------------------------------------------------------------------------------

''===============================================================================
               org     0       ' This is necessary before cog overflows - but be careful as branches between sections will fail!!!
''===============================================================================

'-------[ Debug / Monitor ]-----------------------------------------------------
_HubDebug                                              '                      <--- debug/monitor --->

{{-----
_HubDebug
' On Entry:
        lmm_x = -anything-                             '   value
        lmm_f = #_DEBUG                                '   mode
' Call Format:
'              'FCALL   SP++, @_HubDebug              ' \                    < call: debug/monitor >
                wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                add     lmm_sp, #4                     ' | SP++
                rdlong  lmm_pc, lmm_pc                 ' | CALL...
                long    @_HubDebug                     ' / ...PC = ADDR
' On Return:
        lmm_x = -invalid-                              '   value
        lmm_c = -invalid-
        lmm_f = -same-                                 '   mode (unchanged)
        lmm_p = -invalid-
' Uses:
        lmm_w, lmm_x, lmm_c, lmm_p
-----}}

               rdlong  lmm_p, lmm_pc                   ' \ set addr of version string
               long    @_str_vers                      ' /
               mov     lmm_f, #_TXSTRING
'              'FCALL   SP++, @_HubTxString           ' \                    < call: tx string >
                wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                add     lmm_sp, #4                     ' | SP++
```

```
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTxString                   ' / ...PC = ADDR

:start
''              mov     lmm_x, #0                       '  preset addr=0
                mov     lmm_x, I_Start                  '  preset addr=0

:again
' Receive a string with echo
                mov     lmm_f, #_RXSTRING+_ECHO+_PROMPT '  set rx string echo mode
                mov     lmm_x, #"*"                     '  set prompt char(s)
'              'FCALL   SP++, @_HubRxString            ' \                     < call: receive string>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubRxString                   ' / ...PC = ADDR
' returns: lmm_p =str ptr, lmm_c=count of chars
' decode the string...
{*****
we have not saved anything so we cannot do this for now!
                rdbyte  lmm_w, lmm_p                    ' get 1st char
                cmp     lmm_w, #$0D             wz      ' <cr> ?
        if_c    add     lmm_pc, #(:repeat-($+1))*4      '> br fwd: (y)
*****}

                mov     lmm_f, #0                       ' clear lmm_f mode
' get 1st param (hex/addr1)
'              'FCALL   SP++, @_ParseHex               ' \                     < call: parse hex >
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_ParseHex                      ' / ...PC = ADDR
' returns: lmm_x=hex, lmm_c=digitcount, lmm_p=ptrnextchar

' now check if "." follows...
                rdbyte  lmm_w, lmm_p                    ' read next char
                cmp     lmm_w, #"."             wz      ' z if "."
        if_nz   add     lmm_pc, #(:cmd03-($+1))*4       '> br fwd: (n)
                add     lmm_p, #1                       ' PTR++
```

```
'              'PUSH    lmm_x                          ' \                    < push: 'addr' #1 >
                wrlong  lmm_x, lmm_sp                  ' | PUSH
                add     lmm_sp, #4                     ' / SP++
' get param (hex/addr2)
'              'FCALL   SP++, @_ParseHex               ' \                    < call: parse hex >
                wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                add     lmm_sp, #4                     ' | SP++
                rdlong  lmm_pc, lmm_pc                 ' | CALL...
                long    @_ParseHex                     ' / ...PC = ADDR
' returns: lmm_x=hex/addr2, lmm_c=digitcount, lmm_p=ptrnextchar
                mov     lmm_p2, lmm_x                  ' save 'addr2'
                mov     lmm_c, lmm_x                   ' save 'addr2'
''             'POP     lmm_x                          ' \                    < pop:  'addr' #1 >
                sub     lmm_sp, #4                     ' | SP--
                rdlong  lmm_x, lmm_sp                  ' / POP
                mov     lmm_f, #_ADDR2+_HDG            ' set 'addr2' option
'               add     lmm_pc, #(:cmd09-($+1))*4      '> br fwd: (n)

' now check if "," follows...
:cmd03
                rdbyte  lmm_w, lmm_p                   ' read next char
                cmp     lmm_w, #","             wz     ' z if ","
       if_nz    add     lmm_pc, #(:cmd19-($+1))*4      '> br fwd: (n)
                add     lmm_p, #1                      ' PTR++
'              'PUSH    lmm_x                          ' \                    < push: 'addr' #1 >
                wrlong  lmm_x, lmm_sp                  ' | PUSH
                add     lmm_sp, #4                     ' / SP++
' get param (dec/count)
'              'FCALL   SP++, @_ParseDec               ' \                    < call: parse decimal >
                wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                add     lmm_sp, #4                     ' | SP++
                rdlong  lmm_pc, lmm_pc                 ' | CALL...
                long    @_ParseDec                     ' / ...PC = ADDR
' returns: lmm_x=dec/count, lmm_c=digitcount, lmm_p=ptrnextchar
                mov     lmm_c, lmm_x                   ' save 'count'

                mov     I_countL,lmm_c                 ' save 'count'          '---------------- 00000000000000000000000000000000
                        #
```

```
                sub     lmm_sp, #4                      ' | SP--
                 rdlong  lmm_x, lmm_sp                  ' / POP
:cmd19
'               mov     I_countL,lmm_c                  ' save 'count'          ''---------------- 00000000000000000000000000000000
                 #
                mov     lmm_c, I_countL                 '' Restore 'count'      ''---------------- 00000000000000000000000000000000
                          #
                mov     lmm_f, #_COUNT+_HDG             ' set 'count' option

' now check if what command follows...
:cmd09
                rdbyte  lmm_w, lmm_p                    ' read next char
                add     lmm_p, #1                       ' PTR++
' convert lower case a-z to uppercase A-Z
                cmp     lmm_w, #"a"             wc      ' c if <"a"
        if_c    add     lmm_pc, #(:cmd10-($+1))*4       '> br fwd: (y)
                cmp     lmm_w, #"z"+1           wc      ' c if <"z"+1 (ie a-z?)
        if_c    sub     lmm_w, #$20                     ' convert to uppercase
:cmd10
' command ?
                cmp     lmm_w, #"-"             wz      ' "-" ?
        if_z    add     lmm_pc, #(:cmd_store-($+1))*4   '> br fwd: store addr = long
                cmp     lmm_w, #"?"             wz      ' "?" ?
        if_z    add     lmm_pc, #(:cmd_help-($+1))*4    '> br fwd:
                cmp     lmm_w, #"G"             wz      ' z if "G"
        if_z    add     lmm_pc, #(:cmd_G-($+1))*4       '> br fwd:
                cmp     lmm_w, #"L"             wz      ' "L" ?
        if_z    add     lmm_pc, #(:cmd_L-($+1))*4       '> br fwd:
                cmp     lmm_w, #"M"             wz      ' "M" ?
        if_z    add     lmm_pc, #(:cmd_M-($+1))*4       '> br fwd:
                cmp     lmm_w, #"P"             wz      ' "?" ?
        if_z    add     lmm_pc, #(:cmd_P-($+1))*4       '> br fwd:
                cmp     lmm_w, #"Q"             wz      ' z if "Q"
'              'JUMP    @_HubMonitor                    ' \                     <jump>
        if_z     rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubMonitor                   ' / ...PC = ADDR        < goto rom monitor >
' unknown command, so send "?" and loop...
:unknown
                mov     lmm_x, #$0D                     ' send "?"<cr>
```

```
                shl     lmm_x, #8                       '
                or      lmm_x, #"?"                     '
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
                sub     lmm_pc, #(($+1)-:start)*4       '> br back:
'-------------------------------------------------------
' store cog/hub = long (lmm_x=addr)
:cmd_store
' if the next char is <sp> then skip it...
                rdbyte  lmm_w, lmm_p                    ' read the next char following "-"
                cmp     lmm_w, #" "             wz      ' " " ?
       if_z     add     lmm_p, #1                       ' PTR++
:store2
'               'PUSH   lmm_x                           ' \                     < push: 'addr' #1 >
                 wrlong  lmm_x, lmm_sp                  ' | PUSH
                 add     lmm_sp, #4                     ' / SP++

' decode the long (data)
' lmm_p ptr to string
'               'FCALL  SP++, @_ParseHex                ' \                     < call: parse hex >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_ParseHex                     ' / ...PC = ADDR
' returns: lmm_x=hex, lmm_c=digitcount, lmm_p=ptrnextchar

'               'POP    lmm_w                           ' \                     < pop:  'addr' #1 >
                 sub     lmm_sp, #4                     ' | SP--
                 rdlong  lmm_w, lmm_sp                  ' / POP

                cmp     lmm_c, #0               wz      ' z if 0 digits
                cmp     lmm_c, #9               wc      ' c if <9 digits
   if_z_or_nc   sub     lmm_pc, #(($+1)-:unknown)*4     '> br back: (invalid)

' lmm_w = addr, lmm_x = hex-data, lmm_c=digits lmm_f = _HUB/_COG
                cmp     lmm_w, #$1FF            wz,wc   ' z+c if =<$1FF = cog mode?
```

```
        if_be   add     lmm_pc, #(:wrcog-($+1))*4       '> br fwd:  (y: write to cog)

' write byte/word/long in lmm_x to hub addr in lmm_w
                cmp     lmm_c, #3               wc      ' c if 1-2 digits
        if_c    wrbyte  lmm_x, lmm_w                    ' write byte to hub
        if_c    add     lmm_w, #1                       ' W++
        if_c    add     lmm_pc, #(:more-($+1))*4        '> br fwd:
                cmp     lmm_c, #5               wc      ' c if 3-4 digits
        if_c    wrword  lmm_x, lmm_w                    ' write word to hub
        if_c    add     lmm_w, #2                       ' W++
        if_c    add     lmm_pc, #(:more-($+1))*4        '> br fwd:
                wrlong  lmm_x, lmm_w                    ' write a long to hub
                add     lmm_w, #4                       ' W++
                add     lmm_pc, #(:more-($+1))*4        '> br fwd:

' write long in lmm_x to cog addr in lmm_w
:wrcog          rdlong  lmm_op2, lmm_pc                 '\ store the following instr in lmm_op2
                mov     lmm_x, lmm_x                    '| initially has no effect
                movd    lmm_op2, lmm_w                  '| movd lmm_x(srce)into lmm_op2(dest) - "the cog addr"
                ' now lmm_op2 moves the value in lmm_x to the addr pointed to by lmm_w between each instruction...
                mov     lmm_op2, #0                     '/ disable instr at lmm_op2 (make a "nop")
                add     lmm_w, #1                       ' W++
:more
                mov     lmm_x, lmm_w                    ' copy the addr
' now check the command
                rdbyte  lmm_w, lmm_p                    ' read the not-hex char - is it a command?
                add     lmm_p, #1                       ' PTR++
                cmp     lmm_w, #" "             wz      ' " " = continue
        if_z    sub     lmm_pc, #(($+1)-:store2)*4      '> br back:
                sub     lmm_pc, #(($+1)-:again)*4       '> br back:
'-------------------------------------------------------
' LIST memory: lmm_x='addr', lmm_p=input ptr
'               mov     I_countL,lmm_c                  ' save 'count'  ''---------------- 00000000000000000000000000000000 ----- #
:cmd_L
                MOV     I_Start,lmm_x                   ' save 'count'   '---------------- 00000000000000000000000000000000 ----- #

                or      lmm_f, #_LIST+_MON              ' set LIST mode
                rdbyte  lmm_w, lmm_p                    ' get next char after "L" - is it a modifier option
                cmp     lmm_w, #"1"             wz,wc   ' c if <"1", z if ="1"
```

```
''      if_z    add     lmm_p, #1                       ' PTR++
        if_z    or      lmm_f, #_LIST+_SMON
        if_be   add     lmm_pc, #(:cmd_L9-($+1))*4      '> br fwd: (y)
                cmp     lmm_w, #"3"             wz,wc   ' c if <"3" (ie ="2"), z if ="3"
''      if_be   add     lmm_p, #1                       ' PTR++
        if_c    or      lmm_f, #_LIST+_CODE
        if_z    or      lmm_f, #_LIST+_LONG
:cmd_L9
                mov     lmm_c, I_countL                 ' save 'count'  ''---------------- 00000000000000000000000000000000 ----- #
                mov     lmm_p, lmm_x                    ' lmm_p = 'addr' (no need to push as no more input)
'               'FCALL  SP++, @_HubList                 ' \                     < call: LIST a line >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubList                      ' / ...PC = ADDR
:again9         sub     lmm_pc, #(($+1)-:again)*4       '> br back:
'-------------------------------------------------------
' MOVE memory: lmm_x='addr', lmm_p2='addr2', lmm_c='count', lmm_p=input ptr
:cmd_M                                                  ' only moves 'longs'
                mov     lmm_p, lmm_x                    '\ set 'addr'
                mov     lmm_f, #_MOVE                   '/
'               'FCALL  SP++, @_HubMove                 ' \                     < call: MOVE memory >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubMove                      ' / ...PC = ADDR
                sub     lmm_pc, #(($+1)-:again9)*4      '> br back: (chains back)
'-----------------------------------------------------------------------------
:cmd_P
''      P<cr>                           Display Ports status
                rdlong  lmm_p, lmm_pc                   ' \ set addr of pin/dir string
                long    @Pin_Var_s                      ' /
                mov     lmm_f, #_TXSTRING
'               'FCALL  SP++, @_HubTxString             ' \                     < call: tx string >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTxString                  ' / ...PC = ADDR
```

```
                mov     lmm_f, #_LIST+_COUNT+_LONG      '\ set LIST (long) mode
                mov     lmm_c, #2                       '|
                mov     lmm_p, #$1F8                    '/                      $1F8  PINA  PINB  PINC  PIND
                                                        '/                      $1FC  DIRA  DIRB  DIRC  DIRD
'               'FCALL  SP++, @_HubList                 ' \                     < call: LIST a line>
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubList                      ' / ...PC = ADDR
                sub     lmm_pc, #(($+1)-:again9)*4      '> br back: (chains back)
'-------------------------------------------------------------------------------
:cmd_help
''      ?<cr>                           Display help message
                rdlong  lmm_p, lmm_pc                   ' \ set addr of help string
                long    @_str_help                      ' /
                mov     lmm_f, #_TXSTRING
'               'FCALL  SP++, @_HubTxString             ' \                     < call: tx string >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTxString                  ' / ...PC = ADDR
                sub     lmm_pc, #(($+1)-:again9)*4      '> br back: (chains back)
'-----------------------------------------------------
:cmd_G
''      G<cr>           Return to user program
''      xxxG<cr>        Goto cog address xxx
                cmp     lmm_c, #0               wz      ' z  = G<cr>
        if_nz   movs    LmmFun_ret, lmm_x               ' nz = xxxG<cr> replace return address with goto address

'               'JUMP   @_HubReturn                     ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubReturn                    ' / ...PC = ADDR        <returns to calling routine>
'-----------------------------------------------------

'-------[ LIST a line ]-------------------------------------------------------
_HubList                                                '                       <--- LIST a line --->

{{-----
_HubList
```

```
' On Entry:
        lmm_f   = #_LIST [+options]                     '   mode
        lmm_x   = -unused-                              '   -unused-
        lmm_c   = count (otional)                       '   'n' lines if _COUNT specified
        lmm_p   = address                               '   cog/hub ptr
' Call Format:
'               'FCALL   SP++, @_HubList                ' \                     < call: LIST a line >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubList                      ' / ...PC = ADDR
' On Return:
        lmm_f   = -same-                                '   mode (unchanged)
        lmm_x   = -invalid-                             '
        lmm_c   = -invalid-                             '
        lmm_p   = next-address                          '   cog/hub ptr
' Uses:
        lmm_mode, lmm_w, lmm_f, lmm_x, lmm_c, lmm_p
-----}}

                mov     lmm_mode, lmm_f                 ' save the LIST mode
__Count
'               'PUSH    lmm_c                          ' \                     < push: 'count' #0 >
                 wrlong  lmm_c, lmm_sp                  ' | PUSH
                 add     lmm_sp, #4                     ' / SP++
'                ------------------------
' display hdg?
                test    lmm_mode, #_HDG         wz      ' heading?
        if_z    add     lmm_pc, #(:nohdg-($+1))*4       '> br fwd:  (no)
'               'PUSH    lmm_p                          ' \                     < push: 'addr' #1 >
                 wrlong  lmm_p, lmm_sp                  ' | PUSH
                 add     lmm_sp, #4                     ' / SP++
                andn    lmm_mode, #_HDG                 ' disable hdg for subsequent count loops
                mov     lmm_w, lmm_mode
                and     lmm_w, #3               wz      ' extract MON/SMON/CODE/LONG options
        if_z    rdlong  lmm_p, lmm_pc                   ' \ set addr of MON hdg
                long    @_str_hmon                      ' /
        if_z    add     lmm_pc, #(:hdg-($+1))*4         '> br fwd:
                cmp     lmm_w, #2               wz,wc   ' c=1, z=2, else 3
```

```
        if_c    rdlong  lmm_p, lmm_pc                   ' \ set addr of SMON hdg
                long    @_str_hsmon                     ' /
        if_z    rdlong  lmm_p, lmm_pc                   ' \ set addr of CODE hdg
                long    @_str_hcode                     ' /
        if_a    rdlong  lmm_p, lmm_pc                   ' \ set addr of MON hdg
                long    @_str_hlong                     ' /
:hdg            mov     lmm_f, #_TXSTRING
'               'FCALL  SP++, @_HubTxString             ' \                     < call: tx string >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTxString                  ' / ...PC = ADDR
'               'POP    lmm_p                           ' \                     < pop:  'addr' #1 >
                 sub     lmm_sp, #4                     ' | SP--
                 rdlong  lmm_p, lmm_sp                  ' / POP
:nohdg
'               ------------------------
' display addr...
                cmp     lmm_p, #$1FF            wz,wc   ' z|c if =<$1FF = cog mode?
        if_a    mov     lmm_f, #_HEX+5                  ' set hex mode with 5 digits
        if_a    add     lmm_pc, #(:skip-($+1))*4        '> br fwd:  (hub mode)
                mov     lmm_f, #_HEX+3                  ' set hex mode with 3 digits
' display "  "
                mov     lmm_x, #" "                     ' "  "
                shl     lmm_x, #8
                or      lmm_x, #" "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s) >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
' display address
:skip           mov     lmm_x, lmm_p                    ' get cog/hub address (for displaying)
'               'FCALL  SP++, @_HubHex                  ' \                     < call: display hex >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubHex                       ' / ...PC = ADDR
' display "- "
```

```
                mov     lmm_x, #" "                     ' "- " (loadup in reverse)
                shl     lmm_x, #8
                or      lmm_x, #"-"
'               'FCALL   SP++, @_HubTx                  ' \                     < call: transmit char(s) >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR
'               ------------------------
                cmp     lmm_p, #$1FF            wz,wc   ' z|c if =<$1FF = cog mode?
' get the long from the hub into lmm_w
       if_a     rdlong  lmm_w, lmm_p                    ' read a long (hub)
' get the long from the cog into lmm_w..
       if_be    rdlong  lmm_op2, lmm_pc                 '\ store the following instr in lmm_op2
                mov     lmm_w, lmm_w                    '| initially has no effect
       if_be    movs    lmm_op2, lmm_p                  '| movs lmm_ptr(srce)into lmm_op2(srce) - "the cog addr"
                ' now lmm_op2 moves the value pointed to by lmm_p to the register lmm_w between each instruction...
                mov     lmm_op2, #0                     '/ disable instr at lmm_op2 (make a "nop")

'               'PUSH    lmm_w                          ' \                     < push: 'long' #1 >
                 wrlong  lmm_w, lmm_sp                  ' | PUSH
                 add     lmm_sp, #4                     ' / SP++

'               ------------------------
' determine data display mode...
                test    lmm_mode, #2    wz              ' z if MON /SMON else CODE/LONG
                test    lmm_mode, #1    wc              ' c if SMON/LONG else MON /CODE
 if_nz_and_nc   add     lmm_pc, #(:code-($+1))*4        '> br fwd:  (code)
'               ------------------------
' display 4 longs (MON/SMON/LONG): lmm_p = ptr to 1st long
 if_nz_and_c    mov     lmm_f, #_HEX+0                  '   long:     set hex with 8(=0) digits
 if_z           mov     lmm_f, #_HEX+_REV+_SP+0         '   mon/smon: set hex reversed space mode with 8(=0) digits
                mov     lmm_c, #4                       '   set 4 longs
:xmon1
'               'FCALL   SP++, @_RdLongCogHub           ' \                     < call: read cog/hub long >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_RdLongCogHub                 ' / ...PC = ADDR
```

```
'               'FCALL   SP++, @_HubHex                 ' \                     < call: display hex>
                 wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                 add     lmm_sp, #4                      ' | SP++
                 rdlong  lmm_pc, lmm_pc                  ' | CALL...
                 long    @_HubHex                        ' / ...PC = ADDR
                test     lmm_mode, #2    wz              ' z if MON /SMON else CODE/LONG
        if_z     add     lmm_pc, #(:nospaces-($+1))*4    '> br fwd:  (mon/smon)
' display "    "
                 mov     lmm_x, #" "                     ' "    " (loadup in reverse)
                 shl     lmm_x, #8
                 or      lmm_x, #" "
                 shl     lmm_x, #8
                 or      lmm_x, #" "
                 shl     lmm_x, #8
                 or      lmm_x, #" "
'               'FCALL   SP++, @_HubTx                  ' \                     < call: transmit char(s) >
                 wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                 add     lmm_sp, #4                      ' | SP++
                 rdlong  lmm_pc, lmm_pc                  ' | CALL...
                 long    @_HubTx                         ' / ...PC = ADDR
:nospaces
                 sub     lmm_c, #1               wz      '\
        if_nz    sub     lmm_pc, #(($+1)-:xmon1)*4       '/ djnz lmm_c,#:xmon1

                 test    lmm_mode, #1            wz      ' SMON/LONG mode?
        if_nz    add     lmm_pc, #(:noascii-($+1))*4     '> br fwd:  (y)

                 cmp     lmm_p, #$1FF            wz,wc   '  z|c if =<$1FF = cog mode?
        if_be    sub     lmm_p, #4                       '\ reset ptr to 1st long
        if_a     sub     lmm_p, #16                      '/
                 mov     lmm_c, #4                       '  set count = 4 longs
                 add     lmm_pc, #(:doascii-($+1))*4     '> br fwd:  (n)
'                ------------------------
' display code (in lmm_w)
:code            mov     lmm_c, #14                      ' 14 binary bits
:binary          shl     lmm_w, #1               wc
        if_c     mov     lmm_x, #"1"
        if_nc    mov     lmm_x, #"0"
'               'PUSH    lmm_w                           ' \                     < push: 'binary #2 >
```

```
                wrlong  lmm_w, lmm_sp                  ' | PUSH
                add     lmm_sp, #4                    ' / SP++
'               'FCALL  SP++, @_HubTx                 ' \                     < call: transmit char(s) >
                wrlong  lmm_pc, lmm_sp                ' | PUSH PC
                add     lmm_sp, #4                    ' | SP++
                rdlong  lmm_pc, lmm_pc                ' | CALL...
                long    @_HubTx                       ' / ...PC = ADDR
                cmp     lmm_c, #9               wz
        if_nz   cmp     lmm_c, #6               wz
        if_nz   cmp     lmm_c, #5               wz
        if_nz   cmp     lmm_c, #1               wz
        if_nz   add     lmm_pc, #(:nospace-($+1))*4   '> br fwd:
                mov     lmm_x, #" "                   ' <space>
'               'FCALL  SP++, @_HubTx                 ' \                     < call: transmit char(s) >
                wrlong  lmm_pc, lmm_sp                ' | PUSH PC
                add     lmm_sp, #4                    ' | SP++
                rdlong  lmm_pc, lmm_pc                ' | CALL...
                long    @_HubTx                       ' / ...PC = ADDR
:nospace
'               'POP    lmm_w                         ' \                     < pop:  'binary' #2 >
                sub     lmm_sp, #4                    ' | SP--
                rdlong  lmm_w, lmm_sp                 ' / POP
                sub     lmm_c, #1               wz    '\
        if_nz   sub     lmm_pc, #(($+1)-:binary)*4    '/ djnz lmm_c,#:binary

' display D & S as 3 hex digits
                mov     lmm_f, #_HEX+3                ' set hex mode with 3 digits
                mov     lmm_x, lmm_w
                shr     lmm_x, #23                    ' D operand
'               'PUSH   lmm_w                         ' \                     < push: 'binary' #2 >
                wrlong  lmm_w, lmm_sp                 ' | PUSH
                add     lmm_sp, #4                    ' / SP++
'               'FCALL  SP++, @_HubHex                ' \                     < call: display hex >
                wrlong  lmm_pc, lmm_sp                ' | PUSH PC
                add     lmm_sp, #4                    ' | SP++
                rdlong  lmm_pc, lmm_pc                ' | CALL...
                long    @_HubHex                      ' / ...PC = ADDR
                mov     lmm_x, #" "                   ' <space>
'               'FCALL  SP++, @_HubTx                 ' \                     < call: transmit char(s) >
```

```
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
'             'POP      lmm_x                           ' \                     < pop:  'binary' #2 >
                sub     lmm_sp, #4                      ' | SP--
                rdlong  lmm_w, lmm_sp                   ' / POP
                shr     lmm_x, #14                      ' S operand
                and     lmm_x, #$1FF
'             'FCALL    SP++, @_HubHex                  ' \                     < call: display hex >
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubHex                        ' / ...PC = ADDR
' display " - "
                mov     lmm_x, #" "                     ' " - " (loadup in reverse)
                shl     lmm_x, #8
                or      lmm_x, #"-"
                shl     lmm_x, #8
                or      lmm_x, #" "
'             'FCALL    SP++, @_HubTx                   ' \                     < call: transmit char(s) >
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR

'             'POP      lmm_x                           ' \                     < pop:  'long' #1 >
                sub     lmm_sp, #4                      ' | SP--
                rdlong  lmm_x, lmm_sp                   ' / POP
'             'PUSH     lmm_x                           ' \                     < push: 'long' #1 >
'''notreqd      wrlong  lmm_x, lmm_sp                   ' | PUSH
                add     lmm_sp, #4                      ' / SP++

' display hex
                mov     lmm_f, #_HEX
'             'FCALL    SP++, @_HubHex                  ' \                     < call: display hex >
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
```

```
                long    @_HubHex                        ' / ...PC = ADDR

' display " - "
                mov     lmm_x, #" "                     ' " - " (loadup in reverse)
                shl     lmm_x, #8
                or      lmm_x, #"-"
                shl     lmm_x, #8
                or      lmm_x, #" "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s) >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTx                        ' / ...PC = ADDR

'               'POP    lmm_x                           ' \                     < pop:  'long' #1 >
                 sub     lmm_sp, #4                     ' | SP--
                 rdlong  lmm_x, lmm_sp                  ' / POP
'               'PUSH   lmm_x                           ' \                     < push: 'long' #1 >
'''notreqd       wrlong  lmm_x, lmm_sp                  ' | PUSH
                 add     lmm_sp, #4                     ' / SP++

' display hex reversed
                mov     lmm_f, #_HEX+_REV+_SP+0
'               'FCALL  SP++, @_HubHex                  ' \                     < call: display hex >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubHex                       ' / ...PC = ADDR

'               ------------------------
' display ascii 'xxxx' or 'xxxxxxxxxxxxxxxx' (MON/CODE): lmm_p = ptr to 1st long
                mov     lmm_c, #1                       ' count = 1 long (MON)
:doascii                                                ' count = 4 longs (CODE) (preset above)
' display " '"
                mov     lmm_x, #"'"                     ' " '" (loadup in reverse)
                shl     lmm_x, #8
                or      lmm_x, #" "
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s) >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
```

```
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
:ascii0
' read a long from cog/hub into lmm_x pointed to by lmm_p and inc lmm_p
'               'FCALL  SP++, @_RdLongCogHub            ' \                     < call: read cog/hub long >
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_RdLongCogHub                  ' / ...PC = ADDR
' convert 4 bytes to visible
                mov     lmm_f, #4                       ' CTR=4 (lmm_f as temp counter)
:convert        mov     lmm_w, lmm_x                    ' duplicate
                andn    lmm_x, #$FF                     ' clear   lower byte
                and     lmm_w, #$FF                     ' extract lower byte
                cmp     lmm_w, #" "             wc      ' c if <$20: invisible?
        if_c    mov     lmm_w, #"."                     ' y: replace
                cmp     lmm_w, #$7F             wc      ' c if <$7F: visible?
        if_nc   mov     lmm_w, #"."                     ' n: replace
                or      lmm_x, lmm_w                    ' replace lower byte
                ror     lmm_x, #8                       ' next byte
                sub     lmm_f, #1               wz      '\ CTR--
        if_nz   sub     lmm_pc, #(($+1)-:convert)*4     '/ djnz lmm_c,#:convert
' display 4 ascii bytes
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
                sub     lmm_c, #1               wz      '\
        if_nz   sub     lmm_pc, #(($+1)-:ascii0)*4      '/ djnz lmm_c,#:ascii0

' display "'"
                mov     lmm_x, #"'"                     ' "'"
'               'FCALL  SP++, @_HubTx                   ' \                     < call: transmit char(s)>
                wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                add     lmm_sp, #4                      ' | SP++
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_HubTx                         ' / ...PC = ADDR
```

```
'               ------------------------
:noascii
' now remove the long from the stack
'               'POP      lmm_x                       ' \                     < pop:  'long' #1 >
                 sub      lmm_sp, #4                  ' | SP--
'''notreqd       rdlong   lmm_x, lmm_sp               ' / POP

                 mov      lmm_x, #$0D                 ' <cr>
'               'FCALL    SP++, @_HubTx               ' \                     < call: transmit char(s)>
                 wrlong   lmm_pc, lmm_sp              ' | PUSH PC
                 add      lmm_sp, #4                  ' | SP++
                 rdlong   lmm_pc, lmm_pc              ' | CALL...
                 long     @_HubTx                     ' / ...PC = ADDR

' do we have a count loop or to-addr2 ?
                 mov      lmm_f, lmm_mode             ' restore the LIST mode
'               'POP      lmm_c                       ' \                     < pop:  'count' #0 >
                 sub      lmm_sp, #4                  ' | SP--
                 rdlong   lmm_c, lmm_sp               ' / POP
' count?
                 test     lmm_f, #_COUNT          wz  ' count loop?
        if_z     add      lmm_pc, #(:addr2-($+1))*4   '> br fwd: (n)
                 sub      lmm_c, #1               wz  '\ if_nz djnz lmm_c,#:count
        if_nz    add      lmm_pc, #(:count-($+1))*4   '/
                 add      lmm_pc, #(:return-($+1))*4  '> br fwd:

' addr2?
:addr2           test     lmm_f, #_ADDR2          wz  ' addr2 loop?
        if_z     add      lmm_pc, #(:return-($+1))*4  '> br fwd:
                 cmp      lmm_p, lmm_c            wc  ' \ c if ptr < addr2
        if_nc    add      lmm_pc, #(:return-($+1))*4  '> br fwd:
:count
'               'JMP      @__Count                    ' \                     <jump>
        if_nz     rdlong   lmm_pc, lmm_pc              ' | JUMP...
                  long     @__Count                    ' / ...PC = ADDR

' return to cog calling routine
:return
```

```
'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubReturn                    ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ MOVE memory ]-------------------------------------------------------
_HubMove                                                '                       <--- Move memory --->

{{-----
_HubMove
' On Entry:
        lmm_f   = #_MOVE                                '   mode
        lmm_x   = -unused-                              '   -unused-
        lmm_c   = count                                 '   'n' longs
        lmm_p   = address  (from)                       '   cog/hub ptr
        lmm_p2  = address2 (to)                         '   cog/hub ptr
' Call Format:
'               'FCALL   SP++, @_HubMove                ' \                     < call: MOVE memory >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubMove                      ' / ...PC = ADDR
' On Return:
        lmm_f   = -same-                                '   mode (unchanged)
        lmm_x   = -invalid-                             '
        lmm_c   = -invalid-                             '
        lmm_p   = -invalid-                             '
        lmm_p2  = -invalid-                             '
' Uses:
        lmm_x, lmm_c, lmm_p, lmm_p2
-----}}

:count
' read the long from cog/hub
'               'FCALL   SP++, @_RdLongCogHub           ' \                     < call: read cog/hub long >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_RdLongCogHub                 ' / ...PC = ADDR
```

```
' write the long to cog/hub
'               'FCALL   SP++, @_WrLongCogHub            ' \                     < call: write cog/hub long >
                 wrlong  lmm_pc, lmm_sp                   ' | PUSH PC
                 add     lmm_sp, #4                       ' | SP++
                 rdlong  lmm_pc, lmm_pc                   ' | CALL...
                 long    @_WrLongCogHub                   ' / ...PC = ADDR
' count--
                 sub     lmm_c, #1               wz       '\  djnz lmm_c,#:count
        if_nz    sub     lmm_pc, #(($+1)-:count)*4        '/

'               'JUMP    @_HubReturn                      ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                   ' | JUMP...
                 long    @_HubReturn                      ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ Read Cog/Hub Long ]-------------------------------------------------
_RdLongCogHub

{{-----
_RdLongCogHub
' On Entry:
        lmm_p    = 'addr'                                '   ptr to cog/hub addr
' Call Format:
'               'FCALL   SP++, @_RdLongCogHub            ' \                     < call: read cog/hub long >
                 wrlong  lmm_pc, lmm_sp                   ' | PUSH PC
                 add     lmm_sp, #4                       ' | SP++
                 rdlong  lmm_pc, lmm_pc                   ' | CALL...
                 long    @_RdLongCogHub                   ' / ...PC = ADDR
' On Return:
        lmm_x    = 'long'                                '    value
        lmm_p    = 'addr++'                              '    ptr to next cog/hub addr
' Uses:
        lmm_x, lmm_p
-----}}

                 cmp     lmm_p, #$1FF            wz,wc    ' z|c if =<$1FF = cog mode?
' read the 'long' into lmm_x from hub 'addr' in lmm_p
        if_a     rdlong  lmm_x, lmm_p                     ' read a long (hub)
        if_a     add     lmm_p, #4                        ' PTR++
```

```
' read the 'long' into lmm_x from cog 'addr' in lmm_p..
        if_be   rdlong  lmm_op2, lmm_pc                 '\ store the following instr in lmm_op2
                mov     lmm_x, lmm_x                    '| initially has no effect
        if_be   movs    lmm_op2, lmm_p                  '| movs lmm_ptr(srce)into lmm_op2(srce) - "the cog addr"
                ' now lmm_op2 moves the 'long' into lmm_x  from cog 'addr' in lmm_p (between each instruction)...
                mov     lmm_op2, #0                     '| disable instr at lmm_op2 (make a "nop")
        if_be   add     lmm_p, #1                       '/ PTR++

'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                 long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ Write Cog/Hub Long ]-------------------------------------------------
_WrLongCogHub

{{-----
_WrLongCogHub
' On Entry:
        lmm_x   = 'long'                                '   value
        lmm_p2  = 'addr2'                               '   ptr to cog/hub addr
' Call Format:
'               'FCALL   SP++, @_WrLongCogHub           ' \                     < call: write cog/hub long >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_WrLongCogHub                 ' / ...PC = ADDR
' On Return:
        lmm_p2  = 'addr2++'                             '   ptr to next cog/hub addr
' Uses:
        lmm_x, lmm_p2
-----}}

                cmp     lmm_p2, #$1FF           wz,wc   ' z|c if =<$1FF = cog mode?
' write the long in lmm_x to hub 'addr' in lmm_p2
        if_a    wrlong  lmm_x, lmm_p2                   ' write a long (hub)
        if_a    add     lmm_p2, #4                      ' PTR++
' write the long in lmm_x to cog 'addr' in lmm_p2..
        if_be   rdlong  lmm_op2, lmm_pc                 '\ store the following instr in lmm_op2
```

```
                mov     lmm_x, lmm_x                    '| initially has no effect
        if_be   movd    lmm_op2, lmm_p2                 '| movd lmm_p2(srce)into lmm_op2(dest) - "the cog addr"
                ' now lmm_op2 moves the 'long' into cog 'addr' in lmm_p2 from lmm_x (between each instruction)...
                mov     lmm_op2, #0                     '| disable instr at lmm_op2 (make a "nop")
        if_be   add     lmm_p2, #1                      '/ PTR++

'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                 long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ Reverse Bytes ]-----------------------------------------------------
_ReverseBytes

{{-----
_ReverseBytes
' On Entry:
        lmm_x   = long                                  '   value
' Call Format:
'               'FCALL   SP++, @_ReverseBytes           ' \                     < call: reverse bytes >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_ReverseBytes                 ' / ...PC = ADDR
' On Return:
        lmm_x   = long (byte order reversed)
' Uses:
        lmm_w
-----}}
                                                        ' x: b0  b1  b2  b3
                movs    lmm_w, lmm_x                    '    x   x   x   b3
                rol     lmm_x, #8                       ' x: b1  b2  b3  b0
                ror     lmm_w, #8                       '    b3  x   x   x
                movs    lmm_w, lmm_x                    '    b3  x   x   b0
                rol     lmm_x, #8                       ' x: b2  b3  b0  b1
                ror     lmm_w, #8                       '    b0  b3  x   x
                movs    lmm_w, lmm_x                    '    b0  b3  x   b1
                rol     lmm_x, #8                       ' x: b3  b0  b1  b2
                and     lmm_x, #$FF                     ' x: 0   0   0   b2
```

```
                ror     lmm_w, #8                       '    b1  b0  b3  x
                andn    lmm_w, #$FF                     '    b1  b0  b3  0
                or      lmm_x, lmm_w                    ' x: b1  b0  b3  b2
                ror     lmm_x, #16                      ' x: b3  b2  b1  b0

'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                 long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

'-------[ Parse hex input ]-----------------------------------------------------
_ParseHex

{{-----
_ParseHex
' On Entry:
        lmm_p   = ptr to string (hub)                   '   value
' Call Format:
'               'FCALL   SP++, @_ParseHex               ' \                     < call: parse hex >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_ParseHex                     ' / ...PC = ADDR
' On Return:
        lmm_x   = hex value
        lmm_c   = count of hex digits
        lmm_p   = ptr to next char
' Uses:
        lmm_w
-----}}

' first skip any <sp>
:space          rdbyte  lmm_w, lmm_p                    ' read a char from string
                cmp     lmm_w, #" "             wz      ' " " ?
        if_z    add     lmm_p, #1                       ' (y) PTR++
        if_z    sub     lmm_pc, #(($+1)-:space)*4       '> br back:

                mov     lmm_x, #0                       ' preset hex=0
                mov     lmm_c, #0                       ' preset count=0
```

```
:loop
                rdbyte  lmm_w, lmm_p                    ' read a char from string
                cmp     lmm_w, #"0"             wc      ' c if <"0"
        if_c    add     lmm_pc, #(:nothex-($+1))*4      '> br fwd: not hex
                cmp     lmm_w, #":"             wc      ' c if "0"-"9"
        if_c    add     lmm_pc, #(:hex-($+1))*4         '> br fwd: hex
                or      lmm_w, #$20                     ' force lower case a-z
                cmp     lmm_w, #"a"             wc      ' c if <"a"
        if_c    add     lmm_pc, #(:nothex-($+1))*4      '> br fwd: not hex
                cmp     lmm_w, #"f"+1           wc      ' nc if >="g"
        if_nc   add     lmm_pc, #(:nothex-($+1))*4      '> br fwd: not hex
                sub     lmm_w, #("A"-"9"-1)             ' convert from A-F/a-f
:hex            and     lmm_w, #$0F                     ' extract valid nibble
                shl     lmm_x, #4                       ' shift nibbles
                add     lmm_x, lmm_w                    '   and add nibble
                add     lmm_p, #1                       ' PTR++
                add     lmm_c, #1                       ' CTR++
                sub     lmm_pc, #(($+1)-:loop)*4        '> br back:
:nothex
                cmp     lmm_c, #0               wz      ' z if = $0
''--------------
        if_z    MOV     lmm_x, I_Start                  '' 0000000000 --- I_Start --- Variable that I have after Yours COG_LMM
:nothex2
'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                  ' | JUMP...
                 long    @_HubReturn                     ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

'-------[ Parse decimal input ]-----------------------------------------------
_ParseDec

{{-----
_ParseDec
' On Entry:
        lmm_p   = ptr to string (hub)                   '   value
' Call Format:
'               'FCALL   SP++, @_ParseDec               ' \                     < call: parse decimal >
                 wrlong  lmm_pc, lmm_sp                  ' | PUSH PC
                 add     lmm_sp, #4                      ' | SP++
```

```
                rdlong  lmm_pc, lmm_pc                  ' | CALL...
                long    @_ParseDec                      ' / ...PC = ADDR
' On Return:
        lmm_x   = decimal value
        lmm_c   = count of dec digits
        lmm_p   = ptr to next char
' Uses:
        lmm_w
-----}}

' first skip any <sp>
:space          rdbyte  lmm_w, lmm_p                    ' read a char from string
                cmp     lmm_w, #" "             wz      ' " " ?
        if_z    add     lmm_p, #1                       ' (y) PTR++
        if_z    sub     lmm_pc, #(($+1)-:space)*4       '> br back:

                mov     lmm_x, #0                       ' preset hex=0
                mov     lmm_c, #0                       ' preset count=0
:loop
                rdbyte  lmm_w, lmm_p                    ' read a char from string
                cmp     lmm_w, #"0"             wc      ' c if <"0"
        if_c    add     lmm_pc, #(:notdec-($+1))*4      '> br fwd: not dec
                cmp     lmm_w, #"9"+1           wc      ' c if "0"-"9"
        if_nc   add     lmm_pc, #(:notdec-($+1))*4      '> br fwd: not dec
                and     lmm_w, #$0F                     ' extract valid digit
' multiply lmm_x by 10 and add lmm_w
                shl     lmm_x, #1                       ' *2
                add     lmm_w, lmm_x                    '
                shl     lmm_x, #2                       ' *8
                add     lmm_w, lmm_x
                mov     lmm_x, lmm_w                    ' and recover total
                add     lmm_p, #1                       ' PTR++
                add     lmm_c, #1                       ' CTR++
                sub     lmm_pc, #(($+1)-:loop)*4        '> br back:
:notdec
'               cmp     lmm_x, #0               wz      ' z if = $0
''--------------
'       if_z    MOV     lmm_x, I_Start                  '' 0000000000 --- I_Start --- Variable that I have after Yours COG_LMM
```

```
'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubReturn                    ' / ...PC = ADDR        <returns to calling routine>
'-------------------------------------------------------------------------------

''-------[ Common 'Return to calling routine ]---------------------------------
_HubReturn                                              '                       <return to calling routine>
{---
' Format:
'               'JUMP    @_HubReturn                    ' \                     <jump>
                 rdlong  lmm_pc, lmm_pc                 ' | JUMP...
                 long    @_HubReturn                    ' / ...PC = ADDR        <returns to calling routine>
---}

'               'FRET    SP--                           ' \
                 sub     lmm_sp, #4                     ' | SP--
                 rdlong  lmm_w, lmm_sp                  ' | POP PC...
                 add     lmm_w, #8                      ' | ...PC++2...         (reposn retn adr -skips rdlong/long)
                 mov     lmm_pc, lmm_w                  ' / ...RET
'-------------------------------------------------------------------------------

''-------[ Goto Rom 'Monitor ]-------------------------------------------------
_HubMonitor                                             '                       <--- goto rom monitor --->

{{-----
_HubMonitor
' On Entry:
        lmm_f = #_MONITOR                               '   mode
' Call Format:
'               'FCALL   SP++, @_HubMonitor             ' \                     < call: goto rom monitor >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubMonitor                   ' / ...PC = ADDR
' On Return: Does not return !!
-----}}

                 rdlong  lmm_p, lmm_pc                  ' \ set addr of monitor string
                 long    @_str_mon                      ' /
```

```
                mov     lmm_f, #_TXSTRING
'               'FCALL  SP++, @_HubTxString             ' \                     < call: tx string >
                 wrlong  lmm_pc, lmm_sp                 ' | PUSH PC
                 add     lmm_sp, #4                     ' | SP++
                 rdlong  lmm_pc, lmm_pc                 ' | CALL...
                 long    @_HubTxString                  ' / ...PC = ADDR

                cogid   lmm_x                           ' get cogid
                setcog  lmm_x                           '   and setcog
                rdlong  lmm_p, lmm_pc                   ' \ setup monitor addr & params
                long    $70C                            ' | monitor pgm addr
                rdlong  lmm_x, lmm_pc                   ' |
                long    90<<9 + 91                      ' / monitor params (si/so pins)
                coginit lmm_p, lmm_x                    '                       < coginit monitor_pgm, monitor_ptr >
'-------------------------------------------------------------------------------

_str_vers       byte    $0D
                byte    "=== Cluso's P2 Debugger v0.67 ===",$0D
                byte    "== To List Commands type ?<cr> ==",$0D,0
_str_mon        byte    "To Rom Monitor - hit <space>",$0D,0
_str_hsmon                                              ' same as _str_hmon
_str_hmon       byte    $0D
                Byte    " addr-  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F",$0D
                byte    " ------------------------------------------------",$0D,0
_str_hcode      byte    $0D
                Byte    " addr- instr  zcr i cccc dst src -  3 2 1 0 -  0  1  2  3",$0D
                byte    " ------------------------------------------------------",$0D,0
_str_hlong      byte    $0D," addr- - +$0 --    - +$4 --    - +$8 --    - +$C --",$0D,0         ''For listing from HUB
_str_hlong2     byte    $0D," addr- - + 1 --    - + 2 --    - + 3 --    - + 4 --",$0D,0         ''For listing from COG
_str_msg1       byte    "Successfully returned from debugger",$0D,0

_str_help       byte    $0D
                byte    " ------------ Help ------------------------------------------------------",$0D
                Byte    "    ?<cr>              Show Help - this text ",$0D,$0D
                Byte    "    G<cr>              Return to user program ",$0D
                Byte    "   0G<cr>              Goto cog address $0 (restart without initialisation) ",$0D
                Byte    " xxxG<cr>              Goto cog address $xxx ",$0D
                Byte    "    P<cr>              Show Port's status ",$0D
                Byte    "    Q<cr>              Passes control to the Rom Monitor.",$0D,$0D
```

```
'               byte    "                                                       ",$0D
                Byte    " xxxxxL[n]<cr>           LIST -from cog/hub -- (Change Position ",$0D                 ''-- I_start
                Byte    " xxxxx [,cc]L[n]<cr>          -from cog/hub -- (Change Position, Counter) ",$0D       ''-- I_start, I_countL
                Byte    "       [,cc]L[n]<cr>          -from cog/hub -- (Change Counter) ",$0D                 ''-- I_countL
                Byte    "      L[n]                    -Repeat       -- last address and counter",$0D
                Byte    " xxxxx[.yyyyy]L[n]<cr>        -From_To      -- address of cog/hub --",$0D
                Byte    "                         ---------' ",$0D
                Byte    "                         xxxxx is 'from addr' ",$0D
                Byte    "                         yyyyy is optional 'to addr'",$0D
                Byte    "                         cc    is optional 'New Lines counter'",$0D
                Byte    "                         n     is optional n=1/2/3 for smon/code/long else mon",$0D
                Byte    "                         ---------' ",$0D
                Byte    " xxxxx.yyyyy,ccM<cr>     MOVE (longs) -from cog/hub xxxxx to cog/hub yyyyy",$0D
                Byte    "                           cc is count in longs",$0D,$0D
                Byte    " xxxxx-hh hh hh hh ... hh<cr>          stores byte(s) at hub ",$0D
                Byte    "                                       =OR= store 4 bytes as 1 long at cog",$0D
                Byte    " xxxxx-hhhh hhhh ... hhhh<cr>          stores word(s) at hub",$0D
                Byte    " xxxxx-hhhhhhhh ... hhhhhhhh<cr>       stores long(s) at cog/hub",$0D,$0D
'               byte    "                                                       ",$0D
                byte    "   where xxxxx = cog/hub address, COG <$1FF else HUB.",$0D
                byte    "   where G, Q, L may be upper or lower case.",$0D,$0D
                byte    " == Cluso's P2 Debugger v0.66 ==",$0D
                Byte    " Free user area in Debugger mode ----- $000 to $1CC",$0D
                byte    " ------------------------------------------------------------------------",$0D,0
''
-------------------------------------------------------------------------------------------------------------------------------
--

                long    0                               ' force long alignment
''=============================================================================================
''=============================================================================================
''
Pin_Var_s       byte    "-------------------------------------------------------",$0D
                byte    "|You Change value by enter- addr + offset_ 1Fx-XXXXXXXX|",$0D'
                byte    "|-------------------------------------------------------|",$0D
                byte    "|$1F8  PINA        PINB        PINC        PIND        |",$0D
                byte    "|$1FC  DIRA        DIRB        DIRC        DIRD        |",$0D
                byte    "|-------------------------------------------------------|",$0D
                byte    "|$1Fx- - +$0 --    - +$9 --    - +$A --    - +$B --    |",$0D
```

```
                byte    "|$1Fx- - +$C --    - +$D --    - +$E --    - +$F --    |",$0D
                byte    "-----------------------------------------------------",$0D,0
''=============================================================================================
''=============================================================================================
RunVar_s        byte    $0D
                byte    "-----------------------------------------------------",$0D
                byte    "|              - Debugger variables -                 |",$0D
                byte    "|-----------------------------------------------------|",$0D
                byte    "|$1E8  lmm_f       lmm_x       lmm_c       lmm_p      |",$0D
                byte    "|$1EC  lmm_pc      lmm_sp      lmm_w       lmm_mode   |",$0D
                byte    "|$1F0  lmm_p2      lmm_bittime                        |",$0D
                byte    "-----------------------------------------------------",$0D,0
''
'               byte    "|$1F4  Unused      Unused      Res_INDA    Res_INDB    |",$0D
''
SchVar_s
                byte    "-----------------------------------------------------",$0D
                byte    "|Change        and        to show diferent part of COG |",$0D
                byte    "| COG   IStart    List_Count   Var_s_Start   Var_Count |",$0D
                byte    "-----------------------------------------------------",$0D,0
'               byte    "| COG  I_Start     countL      Prog_End    Variables   |",$0D
'RunVar_sL      LONG    @RunVar_s
'Pin_Var_sL     LONG    @Pin_Var_s
'SchVar_sL      LONG    @SchVar_s
''
''#############################################################################################
''##                                                                                         ##
''##    ------  Used for  CUT  and  PASTE  in Debuger window ------ After <Ctrl-L twice in PNut ##
''##                                                                                         ##
''##            p2load -b 115200 -v -s LSD_067.obj -h -T                                     ##
''##                                                                                         ##
''############## the following is example code and may be deleted ###################################

                byte    "=== HUB STACK =="              ' 16 byte identifier    (maybe removed to save space)

''=======[ Hub LMM Buffers ]=================================================
' These must be long aligned!!
_hub_buf        byte    0[_HUBBUFSIZE]                  ' hub buffer            (maybe reduced to save space)
_hub_stack      long    0[64]                           ' hub stack             (maybe reduced to save space)
```

```
              byte    "=== HUB  END ==="             ' 16 byte identifier    (maybe removed to save space)

''=============================================================================================

{{
'+-------------------------------------------------------------------------------------------------------------+
'|                                                TERMS OF USE: MIT License                                    |
|
'+-------------------------------------------------------------------------------------------------------------+
'|Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation    |
'|files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy,    |
'|modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software|
'|is furnished to do so, subject to the following conditions:                                                   |
'|                                                                                                              |
'|The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.|
'|                                                                                                              |
'|THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE          |
'|WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR         |
'|COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,   |
'|ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.                       |
'+-------------------------------------------------------------------------------------------------------------+
}}
```