```
PROPELLER 2 MEMORY
------------------

'In the Propeller 2, there are two primary types of memory:

HUB MEMORY

    128K bytes of main memory shared by all cogs

'       - cogs launch from this memory
'       - cogs can access this memory as bytes, words, longs, and quads (4 longs)
'       - $00000..$00E7F is ROM - contains Booter, SHA-256/HMAC, and Monitor
'   - $00E80..$1FFFF is RAM - for application usage


COG MEMORY (8 sets)

    512 longs of register RAM for code and data usage

'       - simultaneous instruction, source, and destination reading, plus writing

    256 longs of push/pop RAM for data and video usage

'       - pushes are 1-clock
'       - pops are 2-clock
'       - video circuit can read data simultaneously and asynchronously



'HUB MEMORY INSTRUCTIONS
'-----------------------

'These instructions read and write hub memory.

'All instructions use D as the data conduit, except WRQUAD/RDQUAD/RDQUADC, which use the four QUAD
'registers. The QUADs can be mapped into cog register space using the SETQUAD instruction or kept
'hidden, in which case they are still useful as data conduit and as a read cache. If mapped, the QUADs
'overlay four contiguous cog registers which can begin at any double-even address (%xxxxxxx00). These
'overlaid registers can be read and written as any other registers, as well as executed.
```

```
'The cached reads RDBYTEC/RDWORDC/RDLONGC/RDQUADC will do a RDQUAD if the current read address is
'outside of the 4-long window of the prior RDQUAD. Otherwise, they will immediately return cached
'data. The CACHEX instruction invalidates the cache, forcing a fresh RDQUAD next time a cached read
'executes.
´
'Hub memory instructions must wait for their cog´s hub cycle, which comes once every 8 clocks.
'The timing relationship between a cog´s instruction stream and its hub cycle is generally indeterminant,
'causing these instructions to take varying numbers of clocks. Timing can be made determinant, though,
'by intentionally spacing these instructions apart so that after the first in a series executes, the
'subsequent hub memory instructions fall on hub cycles, making them take the minimal numbers of
'clocks. The trick is to write useful code to go in between them.

WRBYTE/WRWORD/WRLONG/WRQUAD/RDQUAD 'complete on the hub cycle, making them take 1..8 clocks.

RDBYTE/RDWORD/RDLONG 'complete on the 2nd clock after the hub cycle, making them take 3..10 clocks.

RDBYTEC/RDWORDC/RDLONGC 'take only 1 clock if data is cached, otherwise 3..10 clocks.

RDQUADC 'takes only 1 clock if data is cached, otherwise 1..8 clocks.

'After a RDQUAD, the QUAD registers are accessible via D and S on the 3rd clock and executable on the
'5th clock.


instructions                                                                                    clocks
-------------------------------------------------------------------------------------------------------
000000 000 0 CCCC DDDDDDDDD SSSSSSSSS     WRBYTE  D,S       'write lower byte in D at S              1..8
000000 000 1 CCCC DDDDDDDDD SUPNNNNNN     WRBYTE  D,PTR     'write lower byte in D at PTR            1..8
000000 Z01 0 CCCC DDDDDDDDD SSSSSSSSS     RDBYTE  D,S       'read byte at S into D                  3..10
000000 Z01 1 CCCC DDDDDDDDD SUPNNNNNN     RDBYTE  D,PTR     'read byte at PTR into D                3..10
000000 Z11 0 CCCC DDDDDDDDD SSSSSSSSS     RDBYTEC D,S       'read cached byte at S into D       1, 3..10
000000 Z11 1 CCCC DDDDDDDDD SUPNNNNNN     RDBYTEC D,PTR     'read cached byte at PTR into D     1, 3..10

000001 000 0 CCCC DDDDDDDDD SSSSSSSSS     WRWORD  D,S       'write lower word in D at S              1..8
000001 000 1 CCCC DDDDDDDDD SUPNNNNNN     WRWORD  D,PTR     'write lower word in D at PTR            1..8
000001 Z01 0 CCCC DDDDDDDDD SSSSSSSSS     RDWORD  D,S       'read word at S into D                  3..10
000001 Z01 1 CCCC DDDDDDDDD SUPNNNNNN     RDWORD  D,PTR     'read word at PTR into D                3..10
000001 Z11 0 CCCC DDDDDDDDD SSSSSSSSS     RDWORDC D,S       'read cached word at S into D       1, 3..10
```

```
000001 Z11 1 CCCC DDDDDDDDD SUPNNNNNN     RDWORDC D,PTR     'read cached word at PTR into D      1, 3..10

000010 000 0 CCCC DDDDDDDDD SSSSSSSSS     WRLONG  D,S       'write D at S                          1..8
000010 000 1 CCCC DDDDDDDDD SUPNNNNNN     WRLONG  D,PTR     'write D at PTR                        1..8
000010 Z01 0 CCCC DDDDDDDDD SSSSSSSSS     RDLONG  D,S       'read long at S into D                3..10
000010 Z01 1 CCCC DDDDDDDDD SUPNNNNNN     RDLONG  D,PTR     'read long at PTR into D              3..10
000010 Z11 0 CCCC DDDDDDDDD SSSSSSSSS     RDLONGC D,S       'read cached long at S into D     1, 3..10
000010 Z11 1 CCCC DDDDDDDDD SUPNNNNNN     RDLONGC D,PTR     'read cached long at PTR into D   1, 3..10

000011 000 0 CCCC DDDDDDDDD 010110000     WRQUAD  D         'write QUADs at D                      1..8
000011 001 1 CCCC SUPNNNNNN 010110000     WRQUAD  PTR       'write QUADs at PTR                    1..8
000011 000 0 CCCC DDDDDDDDD 010110001     RDQUAD  D         'read quad at D into QUADs             1..8
000011 001 1 CCCC SUPNNNNNN 010110001     RDQUAD  PTR       'read quad at PTR into QUADs           1..8
000011 010 0 CCCC DDDDDDDDD 010110001     RDQUADC D         'read cached quad at D into QUADs   1, 1..8
000011 011 1 CCCC SUPNNNNNN 010110001     RDQUADC PTR       'read cached quad at PTR into QUADs 1, 1..8
-------------------------------------------------------------------------------------------------


'PTR expressions:

'    INDEX = -32..+31 for simple offsets, 0..31 for ++´s, or 0..32 for --´s
'    SCALE = 1 for byte, 2 for word, 4 for long, or 16 for quad

    S = 0 for PTRA, 1 for PTRB
    U = 0 to keep PTRx same, 1 to update PTRx
    P = 0 to use PTRx + INDEX*SCALE, 1 to use PTRx (post-modify)
    NNNNNN = INDEX
    nnnnnn = -INDEX


    SUP NNNNNN    PTR expression
    ------------------------------------------------------------------------
    000 000000     PTRA             'use PTRA
    100 000000     PTRB             'use PTRB
    011 000001     PTRA++           'use PTRA,               PTRA += SCALE
    111 000001     PTRB++           'use PTRB,               PTRB += SCALE
    011 111111     PTRA--           'use PTRA,               PTRA -= SCALE
    111 111111     PTRB--           'use PTRB,               PTRB -= SCALE
    010 000001     ++PTRA           'use PTRA + SCALE,       PTRA += SCALE
```

```
    110 000001      ++PTRB              'use PTRB + SCALE,          PTRB += SCALE
    010 111111      --PTRA              'use PTRA - SCALE,          PTRA -= SCALE
    110 111111      --PTRB              'use PTRB - SCALE,          PTRB -= SCALE

    000 NNNNNN      PTRA[INDEX]         'use PTRA + INDEX*SCALE
    100 NNNNNN      PTRB[INDEX]         'use PTRB + INDEX*SCALE
    011 NNNNNN      PTRA++[INDEX]       'use PTRA,                  PTRA += INDEX*SCALE
    111 NNNNNN      PTRB++[INDEX]       'use PTRB,                  PTRB += INDEX*SCALE
    011 nnnnnn      PTRA--[INDEX]       'use PTRA,                  PTRA -= INDEX*SCALE
    111 nnnnnn      PTRB--[INDEX]       'use PTRB,                  PTRB -= INDEX*SCALE
    010 NNNNNN      ++PTRA[INDEX]       'use PTRA + INDEX*SCALE,    PTRA += INDEX*SCALE
    110 NNNNNN      ++PTRB[INDEX]       'use PTRB + INDEX*SCALE,    PTRB += INDEX*SCALE
    010 nnnnnn      --PTRA[INDEX]       'use PTRA - INDEX*SCALE,    PTRA -= INDEX*SCALE
    110 nnnnnn      --PTRB[INDEX]       'use PTRB - INDEX*SCALE,    PTRB -= INDEX*SCALE


'Examples:

000000 Z01 1 CCCC DDDDDDDDD 000000000     RDBYTE  D,PTRA          'read byte at PTRA into D
000001 000 1 CCCC DDDDDDDDD 111000001     WRWORD  D,PTRB++        'write lower word in D at PTRB,      PTRB += 2
000010 Z01 1 CCCC DDDDDDDDD 011111111     RDLONG  D,PTRA--        'read long at PTRA into D,           PTRA -= 4
000011 001 1 CCCC 110000001 010110001     RDQUAD  ++PTRB          'read quad at PTRB+16 into QUADs,    PTRB += 16
000000 000 1 CCCC DDDDDDDDD 010111111     WRBYTE  D,--PTRA        'write lower byte in D at PTRA-1,    PTRA -= 1

000001 000 1 CCCC DDDDDDDDD 100000111     WRWORD  D,PTRB[7]       'write lower word in D to PTRB+7*2
000010 Z11 1 CCCC DDDDDDDDD 011001111     RDLONGC D,PTRA++[15]    'read cached long at PTRA into D,    PTRA += 15*4
000011 001 1 CCCC 111111101 010110000     WRQUAD  PTRB--[3]       'write QUADs at PTRB,                PTRB -= 3*16
000000 000 1 CCCC DDDDDDDDD 010000110     WRBYTE  D,++PTRA[6]     'write lower byte in D to PTRA+6*1,  PTRA += 6*1
000001 Z01 1 CCCC DDDDDDDDD 110110110     RDWORD  D,--PTRB[10]    'read word at PTRB-10*2 into D,      PTRB -= 10*2


Bytes, words, longs, and quads are addressed as follows:

    for WRBYTE/RDBYTE/RDBYTEC, address = %XXXXXXXXXXXXXXXX (bits 16..0 are used)
    for WRWORD/RDWORD/RDWORDC, address = %XXXXXXXXXXXXXXXX- (bits 16..1 are used)
    for WRLONG/RDLONG/RDLONGC, address = %XXXXXXXXXXXXXXXX-- (bits 16..2 are used)
    for WRQUAD/RDQUAD/RDQUADC, address = %XXXXXXXXXXXXXX---- (bits 16..4 are used)

address  byte  word    long        quad
```

```
-----------------------------------------------------------------
00000-   50   *7250   *706F7250   *0C7CCC030C7C200020302E32706F7250
00001-   72    7250    706F7250    0C7CCC030C7C200020302E32706F7250
00002-   6F   *706F    706F7250    0C7CCC030C7C200020302E32706F7250
00003-   70    706F    706F7250    0C7CCC030C7C200020302E32706F7250
00004-   32   *2E32   *20302E32    0C7CCC030C7C200020302E32706F7250
00005-   2E    2E32    20302E32    0C7CCC030C7C200020302E32706F7250
00006-   30   *2030    20302E32    0C7CCC030C7C200020302E32706F7250
00007-   20    2030    20302E32    0C7CCC030C7C200020302E32706F7250
00008-   00   *2000   *0C7C2000    0C7CCC030C7C200020302E32706F7250
00009-   20    2000    0C7C2000    0C7CCC030C7C200020302E32706F7250
0000A-   7C   *0C7C    0C7C2000    0C7CCC030C7C200020302E32706F7250
0000B-   0C    0C7C    0C7C2000    0C7CCC030C7C200020302E32706F7250
0000C-   03   *CC03   *0C7CCC03    0C7CCC030C7C200020302E32706F7250
0000D-   CC    CC03    0C7CCC03    0C7CCC030C7C200020302E32706F7250
0000E-   7C   *0C7C    0C7CCC03    0C7CCC030C7C200020302E32706F7250
0000F-   0C    0C7C    0C7CCC03    0C7CCC030C7C200020302E32706F7250
00010-   45   *FE45   *0DC1FE45   *0D7CC6010C7CC6010CFCB6E30DC1FE45
00011-   FE    FE45    0DC1FE45    0D7CC6010C7CC6010CFCB6E30DC1FE45
00012-   C1   *0DC1    0DC1FE45    0D7CC6010C7CC6010CFCB6E30DC1FE45
00013-   0D    0DC1    0DC1FE45    0D7CC6010C7CC6010CFCB6E30DC1FE45
00014-   E3   *B6E3   *0CFCB6E3    0D7CC6010C7CC6010CFCB6E30DC1FE45
00015-   B6    B6E3    0CFCB6E3    0D7CC6010C7CC6010CFCB6E30DC1FE45
00016-   FC   *0CFC    0CFCB6E3    0D7CC6010C7CC6010CFCB6E30DC1FE45
00017-   0C    0CFC    0CFCB6E3    0D7CC6010C7CC6010CFCB6E30DC1FE45
00018-   01   *C601   *0C7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
00019-   C6    C601    0C7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
0001A-   7C   *0C7C    0C7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
0001B-   0C    0C7C    0C7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
0001C-   01   *C601   *0D7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
0001D-   C6    C601    0D7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
0001E-   7C   *0D7C    0D7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
0001F-   0D    0D7C    0D7CC601    0D7CC6010C7CC6010CFCB6E30DC1FE45
```

* new **word/long/quad**

```
'PTRA/PTRB INSTRUCTIONS
'----------------------

'Each cog has two 17-bit pointers, PTRA and PTRB, which can be read, written, modified,
'and used to access hub memory.

'On cog startup, the PTRA and PTRB registers are initialized as follows:

    PTRA = %X_XXXXXXXX_XXXXXXXX, data from launching cog, usually a pointer
    PTRB = %X_XXXXXXXX_XXXXXX00, long address in hub where cog code was loaded from


instructions                                                                                clocks
-------------------------------------------------------------------------------------------------
000011 ZCR 1 CCCC DDDDDDDDD 000010010     GETPTRA D          'get PTRA into D, C = PTRA[16]      1
000011 ZCR 1 CCCC DDDDDDDDD 000010011     GETPTRB D          'get PTRB into D, C = PTRB[16]      1

000011 000 1 CCCC DDDDDDDDD 010110010     SETPTRA D          'set PTRA to D                      1
000011 001 1 CCCC nnnnnnnnn 010110010     SETPTRA #n         'set PTRA to 0..511                 1
000011 000 1 CCCC DDDDDDDDD 010110011     SETPTRB D          'set PTRB to D                      1
000011 001 1 CCCC nnnnnnnnn 010110011     SETPTRB #n         'set PTRB to 0..511                 1

000011 000 1 CCCC DDDDDDDDD 010110100     ADDPTRA D          'add D into PTRA                    1
000011 001 1 CCCC nnnnnnnnn 010110100     ADDPTRA #n         'add 0..511 into PTRA               1
000011 000 1 CCCC DDDDDDDDD 010110101     ADDPTRB D          'add D into PTRB                    1
000011 001 1 CCCC nnnnnnnnn 010110101     ADDPTRB #n         'add 0..511 into PTRB               1

000011 000 1 CCCC DDDDDDDDD 010110110     SUBPTRA D          'subtract D from PTRA               1
000011 001 1 CCCC nnnnnnnnn 010110110     SUBPTRA #n         'subtract 0..511 from PTRA          1
000011 000 1 CCCC DDDDDDDDD 010110111     SUBPTRB D          'subtract D from PTRB               1
000011 001 1 CCCC nnnnnnnnn 010110111     SUBPTRB #n         'subtract 0..511 from PTRB          1
-------------------------------------------------------------------------------------------------



QUAD-RELATED INSTRUCTIONS
-------------------------

'Each cog has four QUAD registers which form a 128-bit conduit between the hub memory and the cog.
```

```
'This conduit can transfer four longs every 8 clocks via the WRQUAD/RDQUAD instructions. It can
'also be used as a 4-long/8-word/16-byte read cache, utilized by RDBYTEC/RDWORDC/RDLONGC/RDQUADC.

'Initially hidden, these QUAD registers are mappable into cog register space by using the SETQUAD
'instruction to set a double-even address (%xxxxxxx00) where the base register is to appear, with
'the other three registers following. To hide the QUAD registers, use SETQUAD to set an address
'which is not double-even.


instructions                                                                                clocks
-----------------------------------------------------------------------------------------------
000011 000 1 CCCC 000000000 000001000     CACHEX            'invalidate cache                  1
000011 Z01 1 CCCC DDDDDDDDD 000010001     GETTOPS D         'get top bytes of QUADs into D     1
000011 000 1 CCCC DDDDDDDDD 011100010     SETQUAD D         'set QUAD base address to D        1
000011 001 1 CCCC nnnnnnnnn 011100010     SETQUAD #n        'set QUAD base address to 0..511   1
-----------------------------------------------------------------------------------------------



'HUB INSTRUCTIONS
'----------------

'These instructions are used to control hub circuits and cogs.

'Hub instructions must wait for their cog´s hub cycle, which comes once every 8 clocks. In cases where
'there is no result to wait for (ZCR = %000), these instructions complete on the hub cycle, making
'them take 1..8 clocks, depending on where the hub cycle is in relation to the instruction. In cases
'where a result is anticipated (ZCR <> %000), these instructions complete on the 1st clock after the
'hub cycle, making them take 2..9 clocks.


 COGINIT D,S
-----------

COGINIT 'is used to start cogs. Any cog can be (re)started, whether it is idle or running. A cog
'can even execute a COGINIT to restart itself with a new program.

COGINIT uses D to specify a long address in hub memory 'that is the start of the program that is to be
'loaded into a cog, while S is a 17-bit parameter (usually an address) that will be conveyed to PTRA
```

```
'of the started cog. PTRB of the started cog will be set to the start address of its program that was
'loaded from hub memory.

SETCOG 'must be executed before' COGINIT 'to set the number of the cog to be started (0..7). If SETCOG
'sets a value with bit 3 set (%1xxx), this will cause the next idle cog to be started when COGINIT is
'executed, with the number of the cog started being returned in D, and the C flag returning 0 if okay,
'or 1 if no idle cog was available. Upon cog startup, SETCOG is initialized to %0000.

'When a cog is started, $1F8 contiguous longs are read from hub memory and written to cog registers
'$000..$1F7. The cog will then begin execution at $000. This process takes 1,016 clocks.

Example:

        COGID   COGNUM          'what cog am I?
        SETCOG  COGNUM          'set my cog number
        COGINIT COGPGM,COGPTR   'restart me with the ROM Monitor

COGPGM  LONG    $0070C          'address of the ROM Monitor
COGPTR  LONG    90<<9 + 91      'tx = P90, rx = P91

COGNUM  RES     1


CLKSET  D
---------

CLKSET 'writes the lower 9 bits of D to the hub clock register:

%R_MMMM_XX_SS

R = 1 for hardware reset, 0 for continued operation

MMMM = PLL multiplying factor for XI pin input:
        %0000 for PLL disabled
        %0001..%1111 for 2..16 multiply (XX must be set for XI input or XI/XO crystal oscillator)

XX = XI/XO pin mode:
        00 for XI reads low, XO floats
        01 for XI input, XO floats
```

```
        10 for XI/XO crystal oscillator with 15pF internal loading and 1M-ohm feedback
        11 for XI/XO crystal oscillator with 30pF internal loading and 1M-ohm feedback

SS = Clock selector:
        00 for RCFAST (~20MHz)
        01 for RCSLOW (~20KHz)
        10 for XTAL (10MHz-20MHz)
        11 for PLL

'Because the the clock register is cleared to %0_0000_00_00 on reset, the chip starts up in RCFAST mode
'with both the crystal oscillator and the PLL disabled. Before switching to XTAL or PLL mode from RCFAST
'or RCSLOW, the crystal oscillator must be enabled and given 10ms to stabilize. The PLL stabilizes within
'10us, so it can be enbled at the sime time as the crystal oscillator. Once the crystal is stabilized, you
'can switch between XTAL and RCFAST/RCSLOW without any stability concerns. If the PLL is also enabled, you
'can switch freely among PLL, XTAL, and RCFAST/RCSLOW modes. You can change the PLL multiplier while being
'in PLL mode, but beware that some frequency overshoot and undershoot will occur as the PLL settles to its
'new frequency. This only poses a hardware problem if you are switching upwards and the resulting overshoot
'might exceed the speed limit of the chip.


COGID   D
---------

COGID 'returns the number of the cog (0..7) into D.


COGSTOP D
---------

COGSTOP 'stops the cog specified in D (0..7).


LOCKNEW D
LOCKRET D
LOCKSET D
LOCKCLR D
---------

There are eight semaphore locks available in the chip which can be borrowed with LOCKNEW, returned with
```

LOCKRET, set with **LOCKSET**, **and** cleared with LOCKCLR.

While any cog can set or clear any lock without using LOCKNEW or LOCKRET, LOCKNEW and LOCKRET are provided so that cog programs have a dynamic and simple means of acquiring and relinquishing the locks at run-time.

When a lock is set with LOCKSET, its state is set to 1 and its prior state is returned in C. LOCKCLR works the same way, but clears the lock's state to 0. By having the hub perform the atomic operation of setting/clearing and reporting the prior state, cogs can utilize locks to insure that only one cog has permission to do something at once. If a lock starts out cleared and multiple cogs vie for the lock by doing a 'LOCKSET locknum wc', the cog to get C=0 back 'wins' and he can have exclusive access to some shared resource while the other cogs get C=1 back. When the winning cog is done, he can do a 'LOCKCLR locknum' to clear the lock and give another cog the opportunity to get C=0 back.

LOCKNEW returns the next available **lock into** D, with **C=1 if** no **lock** was free.

LOCKRET frees the **lock in** D so that it can be checked **out** again by LOCKNEW.

LOCKSET sets the **lock in** D **and** returns its prior state **in** C.

LOCKCLR clears the **lock in** D **and** returns its prior state **in** C.

```
instructions                                                                                clocks
-------------------------------------------------------------------------------------------------
000011 ZCR 0 CCCC DDDDDDDDD SSSSSSSSS     COGINIT D,S       'launch cog at D, cog PTRA = S   1..9
000011 000 1 CCCC DDDDDDDDD 000000000     CLKSET  D         'set clock to D                  1..8
000011 001 1 CCCC DDDDDDDDD 000000001     COGID   D         'get cog number into D           2..9
000011 000 1 CCCC DDDDDDDDD 000000011     COGSTOP D         'stop cog in D                   1..8
000011 ZC1 1 CCCC DDDDDDDDD 000000100     LOCKNEW D         'get new lock into D, C = busy   2..9
000011 000 1 CCCC DDDDDDDDD 000000101     LOCKRET D         'return lock in D                1..8
000011 0C0 1 CCCC DDDDDDDDD 000000110     LOCKSET D         'set lock in D, C = prev state   1..9
000011 0C0 1 CCCC DDDDDDDDD 000000111     LOCKCLR D         'clear lock in D, C = prev state 1..9
```

'INDIRECT REGISTERS
'------------------

'Each cog has two indirect registers: INDA and INDB. They are located at $1F6 and $1F7, respectively.

```
INDA and INDB 'each have three hidden 9-bit registers associated with them: the current pointer, the bottom
'limit, and the top limit. The top and bottom limits are inclusive values which set automatic wrapping for
'the current pointer. This way, limited circular buffers can be established within cog RAM.

SETINDA/SETINDB/SETINDS 'is used to set or adjust the current pointer value(s) while forcing the associated
'bottom limit(s) to $000 and the top limit(s) to $1FF.

FIXINDA/FIXINDB/FIXINDS 'sets the current pointer(s) to an inital value, while setting the bottom limit(s)
'to the lower of the initial and terminal values and the top limit(s) to the higher.

By using INDA or INDB for D or S, 'the register pointed at by INDA´s or INDB´s current pointer is addressed.

'Because indirect addressing occurs very early in the pipeline and indirect pointers are affected earlier than
'the last stage, where the conditional bit field (CCCC) normally comes into use, the CCCC field is repurposed
'for indirect operations. The top two bits of CCCC are used for indirect D and the bottom two bits are used
'for indirect S. All instructions which use indirect registers will execute unconditionally.

Here is the INDA/INDB 'usage scheme which repurposes the CCCC field:

OOOOOO ZCR I CCCC DDDDDDDDD SSSSSSSSS
-------------------------------------
xxxxxx xxx x 00xx 111110110 xxxxxxxxx       D = INDA        'use INDA
xxxxxx xxx x 00xx 111110111 xxxxxxxxx       D = INDB        'use INDB
xxxxxx xxx x 01xx 111110110 xxxxxxxxx       D = INDA++      'use INDA,      INDA += 1
xxxxxx xxx x 01xx 111110111 xxxxxxxxx       D = INDB++      'use INDB,      INDB += 1
xxxxxx xxx x 10xx 111110110 xxxxxxxxx       D = INDA--      'use INDA,      INDA -= 1
xxxxxx xxx x 10xx 111110111 xxxxxxxxx       D = INDB--      'use INDB       INDB -= 1
xxxxxx xxx x 11xx 111110110 xxxxxxxxx       D = ++INDA      'use INDA+1,    INDA += 1
xxxxxx xxx x 11xx 111110111 xxxxxxxxx       D = ++INDB      'use INDB+1,    INDB += 1

xxxxxx xxx 0 xx00 xxxxxxxxx 111110110       S = INDA        'use INDA
xxxxxx xxx 0 xx00 xxxxxxxxx 111110111       S = INDB        'use INDB
xxxxxx xxx 0 xx01 xxxxxxxxx 111110110       S = INDA++      'use INDA,      INDA += 1
xxxxxx xxx 0 xx01 xxxxxxxxx 111110111       S = INDB++      'use INDB,      INDB += 1
xxxxxx xxx 0 xx10 xxxxxxxxx 111110110       S = INDA--      'use INDA,      INDA -= 1
xxxxxx xxx 0 xx10 xxxxxxxxx 111110111       S = INDB--      'use INDB       INDB -= 1
xxxxxx xxx 0 xx11 xxxxxxxxx 111110110       S = ++INDA      'use INDA+1,    INDA += 1
xxxxxx xxx 0 xx11 xxxxxxxxx 111110111       S = ++INDB      'use INDB+1,    INDB += 1
```

```
'If both D and S are the same indirect register, the two 2-bit fields in CCCC are OR´d together to get the
'post-modifier effect:

101000 001 0 0011 111110110 111110110        MOV INDA,++INDA     'Move @INDA+1 into @INDA,   INDA += 1
100000 001 0 1100 111110111 111110111        ADD ++INDB,INDB     'Add @INDB into @INDB+1,    INDB += 1

'Note that only '´++INDx,INDx' and 'INDx,++INDx' combinations provide different registers from the same INDx.


'Here are the instructions which are used to establish the current pointer, top limit, and bottom limit
values for INDA and INDB:

instructions *                                                                                  clocks
-------------------------------------------------------------------------------------------------------
111000 000 0 0001 000000000 AAAAAAAAA        SETINDA currA                                        1
111000 000 0 0011 000000000 AAAAAAAAA        SETINDA deltA                                        1

111000 000 0 0100 BBBBBBBBB 000000000        SETINDB currB                                        1
111000 000 0 1100 BBBBBBBBB 000000000        SETINDB deltB                                        1

111000 000 0 0101 BBBBBBBBB AAAAAAAAA        SETINDS currB,currA                                  1
111000 000 0 0111 BBBBBBBBB AAAAAAAAA        SETINDS currB,deltA                                  1
111000 000 0 1101 BBBBBBBBB AAAAAAAAA        SETINDS deltB,currA                                  1
111000 000 0 1111 BBBBBBBBB AAAAAAAAA        SETINDS deltB,deltA                                  1

111001 000 0 0001 TTTTTTTTT IIIIIIIII        FIXINDA terminal,initial                             1
111001 000 0 0100 TTTTTTTTT IIIIIIIII        FIXINDB terminal,initial                             1
111001 000 0 0101 TTTTTTTTT IIIIIIIII        FIXINDS terminal,initial                             1
-------------------------------------------------------------------------------------------------------
* currA/currB/terminal/initial = register (0..$1FF), deltA/deltB = signed value (-$100..$FF)

Examples:

111000 000 0 0001 000000000 000000101        SETINDA 5         'INDA = 5, bottom = 0, top = $1FF
111000 000 0 0011 000000000 000000011        SETINDA +3        'INDA += 3, bottom = 0, top = $1FF
111000 000 0 1100 111111100 000000000        SETINDB -4        'INDB -= 4, bottom = 0, top = $1FF
111000 000 0 0111 000000111 000001000        SETINDS 7,+8      'INDB = 7, INDA += 8, bottoms = 0, tops = $1FF
```

```
111001 000 0 0001 000001111 000001000        FIXINDB 15,8      'INDA = 8, bottom = 8, top = 15
111001 000 0 0100 000010000 000011111        FIXINDB 16,31     'INDB = 31, bottom = 16, top = 31
111001 000 0 0101 001100011 000110010        FIXINDS 99,50     'INDA/INDB = 50, bottoms = 50, tops = 99
```