

xVersion" 2007.6.8.1"

module 18xxx2

#document 18xxx2

```
//
// x4th Forth compiler and x4th target libraries
// (C) COPYRIGHT 1999 .. 2007 Blue Hell / Jan Punter
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License version 2 as
// published by the Free Software Foundation.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
//
// For all listed email addresses :
//
// <dot> to be substituted by a dot      '.'
// <at> to be substituted by an at sign '@'
//
// Blue Hell is a trade mark owned by
//
// Jan Punter
// Oogstplein 6
// 7545 HP Enschede
// the Netherlands
// http://www.iaf.nl/Users/BlueHell/
// http://bluehell.electro-music.com/
// j<dot>punter<at>iaf<dot>nl
//
// All rights attributed to Blue Hell are owned by Jan Punter.
//
// -----
// Code templates for Pic4th.
//
// Boot loader support present, unencrypted version [[initialization]]
// Added [[struct]] and [[field]] words
// Added [[except]]ion handling
//
// removed :
//
// utility words          - now in [[utils]].4th
// numerical conversion words - now in [[numconvert]].4th
// string and memory words - now in [[strings]].4th
//
//
// 18xxx2.4th, to be used for the following processors :
//
// 18c242 with #define 18c242
// 18c252 with #define 18c452
// 18c442 with #define 18c442
// 18c452 with #define 18c452
//
// 18f242 with #define 18f242
// 18f252 with #define 18f452
// 18f442 with #define 18f442
// 18f452 with #define 18f452
//
// Stacks :
//
// fsr0 - data stack, TOS in wreg      - saved in interrupts
// fsr1 - loop stack                   - not saved in interrupts
// fsr2 - free to use - in foreground only - not saved in interrupts
//
// -----
// Addressing :
//
// Initially BSR is set to 0 (zero) - this means bank 0 is reachable
// through direct addressing. Further we rely on the PICASM
// assembler to generate correct code to address according to the
// ACCESS BANK mechanism. the following was observed in some
// microchip documents and in generated list files :
//
// when 12 bit addresses are used to identify memory locations :
// (and we _do_ use 12 bits addresses here).
//
// - addresses 0x000 - 0x07f and 0xf80 - 0xffff will result in opcodes
// for which the A bit is low, so they are in the ACCESS bank.
//
// - other addresses result into opcodes where the A bit is set to 1
// so they must be specified (when used for direct addressing) with
// a correct value in the BSR register.
//
// So when BSR is kept 0, direct addressing is possible for the SFR
// registers and for entire bank 0.
//
// So when addressing in banks 1 .. 14 or in the lower half of bank 15
```

```

// is to be performed BSR (and possibly fsr0h etc.) must be set Ok.
//
// - The movff instruction uses full 12 bit addressing
// - Indirect addressing can be used over all ram locations
// without worrying about BSR.
//
// See [[Initialization]]
#endDoc

only Forth definitions

// ===== Rom specs =====
// Only starting address of rom banks is used, but for PIC 18 it's not
// very usable at all ... just for compatibility.
// ForceRomBank has been changed to do nothing for 18fXXX

// @proc@
64 constant FlashPageSize // Size of a processor flash erase page
$200 2constant AdminStart // Flash image start area in EEPROM

#defined 18c242
#defined 18c442 Or
#defined 18f242 Or
#defined 18f442 Or
#if
0 RomBank $0000 $1fff end
#endif

#defined 18c252
#defined 18c452 Or
#defined 18f252 Or
#defined 18f452 Or
#if
0 RomBank $0000 $3fff end
#endif

// ===== Ram specs =====
// These must match the MaxRam related definitions
// @ram@
// @proc@

// bank RamBank First Last
$00 RamBank $080 $0ff end // 128 bytes GPR ram in bank 0
$01 RamBank $000 $0ff end // Normal bank, 256 bytes

#defined 18c252
#defined 18c452 Or
#defined 18f252 Or
#defined 18f452 Or
#if
$02 RamBank $000 $0ff end // @proc@
$03 RamBank $000 $0ff end
$04 RamBank $000 $0ff end
$05 RamBank $000 $0ff end
#endif

#defined 18c242
#defined 18c442 Or
#defined 18f242 Or
#defined 18f442 Or
#if
$01 LocalBank // Use ram bank 1 for local variables
#endif

#defined 18c252
#defined 18c452 Or
#defined 18f252 Or
#defined 18f452 Or
#if
$05 LocalBank // Use ram bank 5 for local variables.
#endif

// Common word constants
0 2constant 0d
1 2constant 1d
2 2constant 2d
4 2constant 4d
8 2constant 8d
10 2constant 10d
16 2constant 16d
20 2constant 20d
30 2constant 30d
32 2constant 32d
40 2constant 40d
50 2constant 50d
64 2constant 64d
100 2constant 100d

```

```

    128 2constant    128d
    256 2constant    256d
    512 2constant    512d
   1000 2constant    1000d
   1024 2constant    1024d
   2048 2constant    2048d
   4096 2constant    4096d
   8192 2constant    8192d
  10000 2constant    10000d
  16384 2constant    16384d
  32768 2constant    32768d
  65535 2constant    65535d
  $ffff 2constant    -1d
  $ffff 2constant    fffffd

  $ffff 2constant    TrueTrue
  $fff0 2constant    FalseTrue
  $00ff 2constant    TrueFalse
  $0000 2constant    FalseFalse

// Common long constants

    0 4constant      0q
   -1 4constant     -1q

$00 requestRamBank

// Claim interrupt swap space for volatile registers

variable StatusTmp  ///< Temporary for processor status register
variable PCHTmp     ///< Temporary for PCLATH
variable FSRTmpL    ///< Temporary for FSR0L
variable FSRTmpH    ///< Temporary for FSR0H
variable BSRTmp     ///< Temporary for BSR

// Utility variables. These may, sometimes must, be used in user programs.
// Being in bank 0 they need no BSR setup usually

2variable RamPtr    ///< Used for Ram operations like Ram@, equivalent to Rom ops.
2variable spSave    ///< Temporary storage for SP checks
4variable Quotient  ///< Division quotient result variable, and General Purpose
4variable Remainder ///< Division remainder result variable, and General Purpose
4variable Divisor   ///< Division divisor input variable, and General Purpose

requestNoRamBank

// //////////////////////////////////////

cr
1 // Assume no proper processor was selected
#ifdef 18c242 // Let assembler know about it
    . ' #define _18c242_'
    . ' '
    message' processor: 18c242' cr // User feedback
    AddDoc' Processor=18c242'
    drop 0 // Signal processor to be OK
#endif
#ifdef 18c252 // Let assembler know about it
    . ' #define _18c252_'
    . ' '
    message' processor: 18c252' cr // User feedback
    AddDoc' Processor=18c252'
    drop 0 // Signal processor to be OK
#endif
#ifdef 18c442 // Let assembler know about it
    . ' #define _18c442_'
    . ' '
    message' processor: 18c442' cr // User feedback
    AddDoc' Processor=18c442'
    drop 0 // Signal processor to be OK
#endif
#ifdef 18c452 // Let assembler know about it
    . ' #define _18c452_'
    . ' '
    message' processor: 18c452' cr // User feedback
    AddDoc' Processor=18c452'
    drop 0 // Signal processor to be OK
#endif
#ifdef 18f242 // Let assembler know about it
    . ' #define _18f242_'
    . ' '
    message' processor: 18f242' cr // User feedback
    AddDoc' Processor=18f242'
    drop 0 // Signal processor to be OK
#endif
#ifdef 18f252 // Let assembler know about it
    . ' #define _18f252_'
    . ' '
    message' processor: 18f252' cr // User feedback
    AddDoc' Processor=18f252'
    drop 0 // Signal processor to be OK
#endif
#endif

```

```

#ifdef 18f442
    . ' #define _18f442_'
    . ' '
    message' processor: 18f442' cr // User feedback
    AddDoc' Processor=18f442'
    drop 0 // Signal processor to be OK
#endif
#ifdef 18f452
    . ' #define _18f452_'
    . ' '
    message' processor: 18f452' cr // User feedback
    AddDoc' Processor=18f452'
    drop 0 // Signal processor to be OK
#endif
// Check processor selection
#if
    AddDoc' Processor=not properly defined'
    error' processor: no proper processor defined' cr
#endif

// ////////////////////////////////////////////////////////////////////
// ////////////////////////////////////////////////////////////////////

// Finalization code, just the End statement wanted by the compiler.
macro Finalization ( ) ( -- )

; // //////////////////////////////////////////////////////////////////// Finalization ////////////////////////////////////////////////////////////////////

    End

endMacro

// ////////////////////////////////////////////////////////////////////
// ////////////////////////////////////////////////////////////////////
// ////////////////////////////////////////////////////////////////////

// Forth memory layout, this should match the ram bank definitions See @ram@
macro Initialization ( ) ( -- )

; // ////////////////////////////////////////////////////////////////////
; // Forth memory layout

False equ 0
True equ -1

MaxRam equ 0x07f // Shared memory over all banks,
// locate it in bank 0.
// using the Access bank mechanism

LDP equ MaxRam - 0 // Base pointer for local variable access
// Local Data Pointer
EFP equ MaxRam - 2 // word : Except Frame Pointer,
// used for exception handling
tmp0 equ MaxRam - 3 // First temporary register
tmp1 equ MaxRam - 4 // First temporary register
tmp2 equ MaxRam - 5
tmp3 equ MaxRam - 6
tmp4 equ MaxRam - 7
tmp5 equ MaxRam - 8 // Last temporary register
WTMP equ MaxRam - 9 // Temporary Wreg save for interrupts

; // Some locations left free (above, after WTMP), can go to -15 ( - 0xf)

LZero equ MaxRam - 0x010 + 1 // Top of Return/Loop Stack + 1
// 32 bytes of loop stack, 8 nested DO loops's
// The loop stack grows downward into the data stack
//
// The data stack grows upwards into the loop stack
SZero equ 0x000 - 1 // Top of data stack + 1, bank 0, 80 bytes stack

// =====
// AsmHelpers ( ) Some helper macro's
//
fBranch macro aLabel
    bz aLabel // B/ zero flag set : fBranch
endm
//
// =====
// // Parameter Stack macro's
// =====
//
PopReg macro Reg
    movwf (Reg) // PopReg ( b -- )
    movf postdec0, w
endm
//
//
// =====
//
Drop macro
    movf postdec0, w // drop ( b -- )
endm

```

```

//
//
// =====
//
SwapDrop macro
    decf    fsr0l, f          ; // SwapDrop ( b1 b2 -- b2 )
endm
//
//
// =====
//
PushLit macro Arg
    movwf  preinc0          ; // Push Arg ( -- b )
    movlw  (Arg)
endm
//
//
// =====
//
PushReg macro Reg
    movwf  preinc0          ; // Push Reg ( -- b )
    movf   (Reg), w
endm
//
//
// =====
//
PushNothing macro
    movwf  preinc0          ; // Push nothing, free wreg
endm
//
//
// =====
//
Dup macro
    movwf  preinc0          ; // Dup ( b1 -- b1 b1 )
endm
//
//
// =====
//
// Return (loop) Stack macro's
// =====
//
LsPushReg macro Reg
    decf   fsr1l            ; // Push Reg [ -- b ]
    movff  (Reg), indf1
endm
//
//
// =====
//
LsPushLit macro Arg
    PushNothing              ; // LsPushLit (Arg)
    movlw  (Arg)             ; // Free wreg to be used
    decf   fsr1l            ; // Arg into wreg
    movwf  indf1            ; // Push Arg [ -- b ] to loop stack
    Drop   Drop              ; // Restore wreg
endm
//
//
// =====
//
LsDrop macro
    incf   fsr1l            ; // drop [ b -- ] from loop stack
endm
//
//
// =====
//
LsFree macro item_count
    PushNothing
    movlw  item_count
    addwf  fsr1l, f          ; // drop [ n1 .. nn -- ] from loop stack
    Drop
endm
//
//
// =====
//
LsPopReg macro Reg
    movff  postinc1, (Reg)   ; // Pop Reg [ b -- ]
endm
//
//
// ; =====
//
AddExceptionFrame macro
    LsPushReg ( EFP + 0)     ; // AddExceptionFrame : link in a new exception frame
    LsPushReg ( EFP + 1)
    movff    fsr1l, EFP + 0 ; // And set up a new Except Frame Pointer
    movff    fsr1h, EFP + 1
endm
//
//
// =====
//
RemoveExceptionFrame macro

```

```

        movff   EFP + 0, fsr1l      ; // RemoveExceptionFrame, unwind one exception level
        movff   EFP + 1, fsr1h      ; // Set loop stack back to EFP

        LSPopReg ( EFP + 1)        ; // Pop back old Exception Frame Pointer
        LSPopReg ( EFP + 0)
    endm
//
// =====
// Ram select macro's
// =====
//
SetBSR macro aRegister
    movlb   (aRegister >> 8) & 0xff ; // SetBsr
endm
//
// =====
//
SetBSRDefault macro
    movlb   0x00                    ; // Select the default rambank
endm
//
// =====
//
AddressToFsr0 macro aRegister
    movlw   (aRegister >> 0) & 0xff ; // AddressToFsr0
    movwf   fsr0l
    movlw   (aRegister >> 8) & 0xff
    movwf   fsr0h
endm
//
// =====
//
AddressToFsr1 macro aRegister
    movlw   (aRegister >> 0) & 0xff ; // AddressToFsr1
    movwf   fsr1l
    movlw   (aRegister >> 8) & 0xff
    movwf   fsr1h
endm
//
// =====
//
AddressToFsr2 macro aRegister
    movlw   (aRegister >> 0) & 0xff ; // AddressToFsr2
    movwf   fsr2l
    movlw   (aRegister >> 8) & 0xff
    movwf   fsr2h
endm
//
// =====
// Utility macro's
// =====
//
PushNextRomAddress macro
    local   BeyondMe

    ; // PushNextRomAddress ( -- wRomAddr )

    PushLit (BeyondMe >> 0) & 0xff ; // Push low byte
    PushLit (BeyondMe >> 8) & 0xff ; // Push high byte
    return
BeyondMe:
endm
//
// =====
//
incf16 macro freg
    incf   freg, f                    ; // incf16
    btfsc Status, C
    incf   freg + 1, f
endm
//
// =====
//
incf32 macro freg
    incf   freg + 0, f                ; // incf32
    btfsc Status, C
    incf   freg + 1, f
    btfsc Status, C
    incf   freg + 2, f
    btfsc Status, C
    incf   freg + 3, f
endm
//
// =====
//
incf24 macro freg
    incf   freg + 0, f                ; // incf24
    btfsc Status, C
    incf   freg + 1, f

```

```

    btfsc    Status, C
    incf    freg + 2, f
endm
//
//
// =====
//
decf16 macro    freg
    decf    freg + 0, f        ; // decf16
    btfss   Status, C
    decf    freg + 1, f
endm
//
//
// =====
//
decf16sz macro    freg
    decf    freg + 0, f        ; // decf16sz S if Zero - kills wreg
    btfss   Status, C
    decf    freg + 1, f
    movf    freg + 0, w
    iorwf   freg + 1, w
    btfss   Status, Z        ; // S/ result was zero
endm
//
//
// =====
//
addl16 macro    dst16, lit16
    movlw   (lit16 >> 0) & 0xff    ; // addl16
    addwf   dst16, f
    movlw   (lit16 >> 8) & 0xff
    addwfc  dst16 + 1, f
endm
//
//
// =====
//
shl16 macro    reg16
    bcf     Status, C            ; // shl16
    rlc     reg16 + 0, f
    rlc     reg16 + 1, f
endm
//
//
// =====
//
rrc16 macro    reg16
    rrc     reg16 + 1, f        ; // Same as shrc16
    rrc     reg16 , f          ; // rrc16
endm
//
//
// =====
//
shlc16 macro    reg16
    rlc     reg16 , f           ; // shlc16
    rlc     reg16 + 1, f
endm
//
//
// =====
//
shr16 macro    reg16
    bcf     Status, c          ; // shr16
    rrc     reg16 + 1, f
    rrc     reg16 + 0, f
endm
//
//
// =====
//
shrc16 macro    reg16
    rrc     reg16 + 1, f        ; // Same as rrc16
    rrc     reg16, f           ; // shrc16
endm
//
//
// =====
//
xorl16 macro    dst16, lit16
    movlw   (lit16 >> 0) & 0xff    ; // xorl16
    xorwf   dst16, f
    movlw   (lit16 >> 8) & 0xff
    xorwf   dst16 + 1, f
endm
//
//
// =====
//
shlc32 macro    reg32
    rlc     reg32 , f           ; // shlc32
    rlc     reg32 + 1, f
    rlc     reg32 + 2, f
    rlc     reg32 + 3, f
endm
//
//
// =====
//
xorf16 macro    dst16, src16

```

```

    movf    src16    , w           ; // xorfl6
    xorwf   dst16    , f
    movf    src + 1, w
    xorwf   dst + 1, f
endm
//
//
// =====
//
xorfl6d macro    src16_1, src16_2, dst16
    movf    src16_1 + 0, w       ; // xorfl6d
    xorwf   src16_2 + 0, w
    movwf   dst16    + 0
    movf    src16_1 + 1, w
    xorwf   src16_2 + 1, w
    movwf   dst16    + 1
endm
//
//
// =====
//
clrfl6 macro    reg16
    clrf    reg16                ; // clrfl6
    clrf    reg16 + 1
endm
//
//
// =====
//
clrf32 macro    reg32
    clrf    reg32                ; // clrf32
    clrf    reg32 + 1
    clrf    reg32 + 2
    clrf    reg32 + 3
endm
//
//
// =====
//
cpfslt16 macro    reg16_1, reg16_2

    ; // Skip the next instruction if N1 < N2 (unsigned)
    ; // or skip if N1 - N2 < 0

    local    CheckHiUnEq, N1SmallerN2, N1NotSmallerN2

    movwf   <@" Quotient">      ; // Save wreg

    movf    reg16_1 + 1, w
    subwf   reg16_2 + 1, w
    bnz     CheckHiUnEq        ; // B/ N1 + 1 <> N2 + 1, test hi bytes

    ; // N1 + 1 == N2 + 1, test low bytes

    movf    reg16_2, w          ; // N2 to w
    subwf   reg16_1, w          ; // calc N1 - N2 :: negative -> N1 - N2 < 0 -> N1 < N2
    movff   <@" Quotient">, WREG ; // Restore wreg || NOT affecting flags
    bc     N1NotSmallerN2

    ; // N1 < N2

    bra     N1SmallerN2

CheckHiUnEq

    ; // Hi bytes are not equal, test 'm

    movf    reg16_2 + 1, w      ; // N2 == reg16_2 + 1
    subwf   reg16_1 + 1, w      ; // Sub N1:: negative -> N1 - N2 < 0 -> N1 < N2
    movff   <@" Quotient">, WREG ; // Restore wreg || NOT affecting flags
    bc     N1NotSmallerN2

N1SmallerN2

    bra     N1NotSmallerN2 + 2   ; // Skip next

N1NotSmallerN2

endm
//
//
// =====
//
cpfseq16 macro    reg16_1, reg16_2

    ; // Skip the next instruction if N1 = N2

    local    N1NotEqualN2

    movf    reg16_2 + 1, w
    subwf   reg16_1 + 1, w
    bnz     N1NotEqualN2        ; // Hi bytes not equal

    ; // N1 + 1 equals N2 + 1

    movf    reg16_2, w
    subwf   reg16_1, w
    btfss   Status, Z            ; // S/ N1 = N2 :: so skip next instruction

N1NotEqualN2

```



```

; //      valid total length when loader_state is $01 or $02
; //
; //
; //      $0004 : word : checksum
; //
; //      CRC-16 over preceding bytes, startvalue 0 is used for the CRC
; //      when the CRC is in error : run image in flash (normal operation).
; //
; //
; //      $0006 .. $00ff : not used, can have any value.
; //
; //
; //      $0100 .. $ffff : application storage or flash image
; //
; //      Depends upon loader_state what is present :
; //
; //      $00 - invalid image, partial FLASH image present
; //      $01 - valid FLASH image with length image_length
; //      $02 - valid FLASH image with length image_length
; //      $ff - APPLICATION data
; //      All other values, invalid checksum or invalid loader_state :
; //      APPLICATION data, but possibly invalid.

LoaderVersion equ 1 ; // Loaders version number

; // RAM layout during boot process

EE_ADDR      equ    0x000 ; // Two bytes EEPROM address
EE_DATA      equ    0x002 ; // Two bytes EEPROM data buffer
EE_PTR       equ    0x004 ; // Two bytes pointer in EEPROM
EE_ACK       equ    0x006 ; // One byte ACK / NAK received after write
FLASH_CNT    equ    0x007 ; // Two bytes, nr of bytes to flash
FLASH_PTR    equ    0x009 ; // Two bytes pointer in FLASH
COUNTER      equ    0x00b ; // Two bytes counter
BSTATE      equ    0x00d ; // One byte loader_state
CRC_16       equ    0x00e ; // Two bytes CRC

; // Physical vectors.

org 0x0000

bra BootLoader ; // Jump into the bootloader

; // -----
; // CalcCrc16 : calculate CRC-16 checksum from CRC and value passed in wreg

CalcCrc16

movwf tmp1 ; // Get bData into tmp1

movlw 8 ; // Handle 8 bits
bra Crc16_goon ; // B/ into the 'real thing'

org 0x0008

bra HighPriorityInterruptVector ; // Jump into the application interrupt handler

; // -----
; // SetupRegs : Setup registers to be able to access external eeprom
; //
; // Application dependent : must at least setup some physical IO ok

SetupRegs
movlw B'01001110' ; // Setup most port pins as digital I/O
movwf ADCON1 ; // Initialize i2c port

bsf I2C_LAT, <@" Pin_SCL"> ; // Set SCL high
bcf I2C_TRIS, <@" Pin_SCL"> ; // Put SCL line in output state
return

; // -----
; // LoaderGetVersion : returns the loader version into wreg

LoaderGetVersion

movlw LoaderVersion
return

org 0x0018

bra LowPriorityInterruptVector ; // Jump into the application interrupt handler

; // Helpers for boot loader

; // -----
; // CalcCrc16 : calculate CRC-16 checksum from CRC and value passed in wreg

; // Moved some code into 'vector gaps'

```

```

Crc16_goon
    movwf    tmp2

BLCC16_Loop
    movf    tmp1, w           ; // Data into wreg
    xorwf   CRC_16, w        ; // Xor with old crc -> wreg
    movwf   tmp3             ; // Keep bit 0 in flag for later
    bcf     Status, c        ; // Clear carry, make a right SHIFT
    rrc16   CRC_16           ; // drop a bit of Data
    rrcf    tmp1, f          ; // drop a bit of Data

    rrcf    tmp3, f          ; // rotate bit 0 of tmp3 into carry
    btfss   Status, c        ; // skip next if carry set
    bra     BLCC16_WasZero   ; // B/ carry was not set

    xorl16  CRC_16, 0xa001   ; // Xor OldCrc with shifted polynomial
                                ; // This leaves the carry unchanged

BLCC16_WasZero
    decfsz  tmp2, f          ; // djnz EC_COUNTER, EC_CC_Loop
    bra     BLCC16_Loop

    return

; // -----
; // i2cTxByte : transmit wreg to the i2c device

i2cTxByte
    movwf   tmp1             ; // Store TxBuf

    movlw   8                 ; // Init counter, 8 bits to do
    movwf   tmp2

TXi2cLp
    bcf     I2C_LAT, <@" Pin_SCL"> ; // Clock low
    bcf     I2C_LAT, <@" Pin_SDA"> ; // Output bit 0 (0)
    btfsc   tmp1, 7           ; // Output bit 0 (1)
    bsf     I2C_LAT, <@" Pin_SDA"> ; // Output bit 0 (1)

    bsf     I2C_LAT, <@" Pin_SCL"> ; // Clock to high

    rlnfcf  tmp1, f           ; // Rotate TxBuf left (not through carry)
    decfsz  tmp2, f           ; // 8 bits done ?
    bra     TXi2cLp           ; // No.

    ; // Bit in, reads ACK / NAK

    bcf     I2C_LAT, <@" Pin_SCL"> ; // Return SCL to low
    bcf     I2C_LAT, <@" Pin_SDA"> ; // Make SDA low
    bsf     I2C_TRIS, <@" Pin_SDA"> ; // Put SDA line in input state
    bsf     I2C_LAT, <@" Pin_SCL"> ; // Clock high

    setf    EE_ACK            ; // Assume ACK
    btfsc   I2C_PORT, <@" Pin_SDA"> ; // Read SDA pin / S/ on ack
    clrf    EE_ACK            ; // No ack

    bcf     I2C_LAT, <@" Pin_SCL"> ; // Return SCL to low
    bcf     I2C_TRIS, <@" Pin_SDA"> ; // Put SDA line in output state

    return

; // -----
; // i2cStart : transmit a bus start

i2cStart
    bsf     I2C_LAT, <@" Pin_SDA">
    bsf     I2C_LAT, <@" Pin_SCL"> ; // SDA goes low during SCL high

i2cToggle
    nop
    nop
    btg     I2C_LAT, <@" Pin_SDA">

    return

; // -----
; // i2cStop : transmit a bus stop

i2cStop
    bcf     I2C_LAT, <@" Pin_SDA">
    bsf     I2C_LAT, <@" Pin_SCL"> ; // SDA goes high during SCL high

    bra     i2cToggle

; // -----
; // i2cTxAddress : transmit the 16 bit address contained in EE_ADDR
; // preceded by START and DeviceAddress/Write

i2cTxAddress

```

```

rcall    i2cStart
movlw   0xa0                ; // Device address + write
rcall    i2cTxByte
movf    EE_ADDR + 1, w
rcall    i2cTxByte
movf    EE_ADDR + 0, w
bra     i2cTxByte

; // -----
; // i2cRxByte : read a byte from i2c into wreg

i2cRxByte
    bsf    I2C_TRIS, <@" Pin_SDA"> ; // Put SDA line in input state

    movlw  8                ; // 8 bits of data
    movwf  tmp2

i2cRxLp
    rlncf  tmp3, f          ; // Shift data to buffer
    ; // Bit in

    bsf    I2C_LAT, <@" Pin_SCL"> ; // Clock high
    bcf    tmp3, 0          ; // Assume data bit is 0
    btfsc  I2C_PORT, <@" Pin_SDA"> ; // Read SDA pin
    bsf    tmp3, 0          ; // Data bit is 1 instead

    bcf    I2C_LAT, <@" Pin_SCL"> ; // Return SCL to low

    decfsz tmp2, f          ; // 8 bits done ?
    bra    i2cRxLp

    bsf    I2C_LAT, <@" Pin_SDA"> ; // Send NAK
    bcf    I2C_TRIS, <@" Pin_SDA"> ; // Put SDA line in output state
    bsf    I2C_LAT, <@" Pin_SCL"> ; // Clock high
    movf   tmp3, w          ; // Data into wreg
    bcf    I2C_LAT, <@" Pin_SCL"> ; // Return SCL to low
    return

; // -----
; // PreReadEEByte : read byte from EE_ADDR into wreg

PreReadEEByte
    rcall  i2cTxAddress      ; // Send address prefix stuff

    rcall  i2cStart         ; // re-issue start
    movlw  0x1             ; // Device address + read
    rcall  i2cTxByte
    rcall  i2cRxByte
    bra   i2cStop          ; // Issue bus STOP

; // -----
; // incEeAddr : Increment EE_ADDR
; //           Keeps wreg intact

incEeAddr
    incf16 EE_ADDR
    return

; // -----
; // ReadEEByte : read a byte from eeprom address EE_ADDR into EE_DATA and into wreg
; //           increments EE_ADDR

ReadEEByte
    rcall  PreReadEEByte
    movwf  EE_DATA + 0      ; // Store byte
    bra   incEeAddr         ; // Leaves wreg intact

; // -----
; // WriteEEByte : Write a byte from EE_DATA to eeprom address EE_ADDR
; //           increments EE_ADDR

WriteEEByte
    rcall  i2cTxAddress      ; // Send address prefix stuff
    movf  EE_DATA, w

WriteEEByteAgain
    rcall  i2cTxByte        ; // Write byte to bus
    rcall  i2cStop          ; // Issue bus STOP, starts programming

    ; // Wait for device to respond again after programming

    clrwdt                  ; // Give us some time

WriteEEByteWait
    rcall  i2cStart         ; // Send bus start
    movlw  0xa0             ; // start write op

```

```

rcall    i2cTxByte
; // Check for ACK

btfss   EE_ACK, 0           ; // S/ Ack seen
bra     WriteEEByteWait    ; // B/ no ack seen yet (dog might terminate this)

rcall    i2cStop           ; // Terminate bus activity

bra     incEeAddr          ; // Advance EE device address and return

; // -----
; // WriteEEWord : Write a word from EE_DATA to eeprom address EE_ADDR
; //               increments EE_ADDR by two
WriteEEWord

rcall    WriteEEByte
rcall    i2cTxAddress      ; // Send address prefix stuff
movf    EE_DATA + 1, w
bra     WriteEEByteAgain   ; // Advance EE device address and return

; // -----
; // @@@@ to be removed later for a real 18f452

; // EADR    EQU  H'0FA9'
; // EEDATA  EQU  H'0FA8'
; // EECON2  EQU  H'0FA7'
; // EECON1  EQU  H'0FA6'
; // EEPGD   EQU  H'0007'
; // CFGS    EQU  H'0006'
; // FREE    EQU  H'0004'
; // WRERR   EQU  H'0003'
; // WREN    EQU  H'0002'
; // WR      EQU  H'0001'
; // RD      EQU  H'0000'

; // END @@@@ to be removed later

; // -----
; // StartWrite - starts a write or erase operation
StartWrite

clrwdt
movlw   0x55
movwf   eecon2
movlw   0xaa
movwf   eecon2
bsf     eecon1, wr
nop
return

; // -----
; // BootLoader - does it all
BootLoader

; // Determine what to do based upon eeprom contents
; //
; // Either :
; //
; // - load new software from eeprom into flash
; // - jump into existing software

; // ICE BUG FIX
;
movlw   0xb0           ; // To make table reads work correctly in MPLAB ICE 2000
movwf   0xf9c         ; // Not needed in production code (but it doesn't harm)
;
; // ICE BUG FIX - END

bcf     intcon, gie    ; // Disable all interrupts
SetBSRDefault         ; // Setup BSR register

rcall   SetupRegs     ; // Initialize some registers

bsf     I2C_LAT, <@" Pin_SDA"> ; // Set SCL high
bcf     I2C_TRIS, <@" Pin_SDA"> ; // Put SCL line in output state

; // Check EEPROM

clrf    EE_ADDR + 0    ; // Set address pointer to zero
clrf    EE_ADDR + 1

clrf    CRC_16 + 0    ; // Clear CRC
clrf    CRC_16 + 1

movlw   6              ; // Check six bytes
movwf   COUNTER

BootCrcCheck

rcall   ReadEEByte    ; // Read next byte into EE_DATA and into wreg
rcall   CalcCrc16

decfsz  COUNTER
bra     BootCrcCheck

```

```

movf    CRC_16 + 0, w
iorwf   CRC_16 + 1, w
bnz     ResetVector           ; // B/ CRC error, perform no load

; // CRC OK, read state

clrf    EE_ADDR                ; // Read state
rcall   ReadEEByte            ; // Read byte into EE_DATA and into wreg
movwf   BSTATE
rcall   ReadEEByte            ; // Read control byte
xorlw   0xff
cpfseq  BSTATE                ; // S/ control byte OK
bra     ResetVector           ; // B/ error in control byte, normal operation

; // CRC OK, control byte OK, check state
; // State 1 is the only state of interest to us, we must flash a new image then

; // bra     ProgOk           ; // Skip programming, comment out/in as needed

movlw   0x01                   ; // Check 0x01
cpfseq  BSTATE                ; // S/ state = 1
bra     ResetVector           ; // B/ state <> 1, not interested, normal operation

```

#### BootLoaderDoIt

```

; // State = 1, valid FLASH image with length image_length
; //
; // Perform FLASH programming, copy EEPROM to FLASH

movlw   0x02                   ; // Read byte count
movwf   EE_ADDR

rcall   ReadEEByte            ; // was : read eeword
rcall   PreReadEEByte
movwf   EE_DATA + 1           ; // Store byte

movlw   61                     ; // Add 61 to byte count to round up
addwf   EE_DATA + 0, f         ; // should be 63, but use 61 to skip the CRC
clrf    wreg                   ; // The image builder must cooperate by making
addwfc  EE_DATA + 1, f         ; // images a multiple of 64 bytes + 2 for the CRC

movlw   6                       ; // Divide byte count by 64, erase page count
movwf   COUNTER

```

#### BL\_DL

```

shr16   EE_DATA                ; // Shift right one bit
decfsz  COUNTER                ; // S/ done
bra     BL_DL                  ; // B. not done yet

movff   EE_DATA + 0, FLASH_CNT + 0 ; // Store erase page count into FLASH_CNT
movff   EE_DATA + 1, FLASH_CNT + 1

clrf    tblptru                ; // Setup table latch
movlw   ( ResetVector >> 8) & 0xff
movwf   tblptrh
movlw   ( ResetVector >> 0) & 0xff
movwf   tblptrl

movlw   (<@" AdminStart"> >> 0) & 0xff ; // Setup EEPROM address
movwf   EE_ADDR + 0
movlw   (<@" AdminStart"> >> 8) & 0xff
movwf   EE_ADDR + 1

```

#### BL\_loop\_erase

```

; // Erase page

bsf     eecon1, eepgd          ; // point to FLASH program memory
bcf     eecon1, cfgs          ; // access FLAH program memory
bsf     eecon1, wren          ; // enable FLASH writes
bsf     eecon1, free          ; // Enable row erase

rcall   StartWrite            ; // Unlock sequence, start erase, stall CPU

tblrd   *-                    ; // Dummy read

movlw   8                       ; // Setup erase page byte counter, 8 write pages in 1 erase page
movwf   COUNTER + 0

```

#### BL\_loop\_outer

```

movlw   8                       ; // Setup write page byte counter, 8 bytes in a write page
movwf   COUNTER + 1

```

#### BL\_loop\_inner

```

rcall   ReadEEByte            ; // Read data byte into EE_DATA and into wreg
movwf   tablat                ; // into tablat
tblwt   +*                    ; // Short data write to buffer

decfsz  COUNTER + 1
bra     BL_loop_inner

; // Flash 8 bytes

bsf     eecon1, eepgd          ; // Point to FLASH program memory

```

```

bcf     eecon1, cfgs           ; // Access FLASH program memory
bcf     eecon1, free          ; // Disable erase ops
bsf     eecon1, wren          ; // Enable write to memory

rcall   StartWrite            ; // Unlock sequence, start write, stall CPU

; // Write verification

movlw   8                     ; // Set EE_ADDR 8 location back
movwf   COUNTER + 1           ; // Setup read page byte counter, 8 bytes in a page
subwf   EE_ADDR
movlw   0
subwfb  EE_ADDR + 1

movlw   8                     ; // Set TBLPTR 8 location back
subwf   TBLPTRL
movlw   0
subwfb  TBLPTRH

BL_Loop_check

rcall   ReadEEByte            ; // Read next EE byte into EE_DATA and into wreg
tblrd   +*                    ; // Read FLASH byte into TABLAT
cpfseq  tablat                 ; // S/ FLASH contents OK
reset   ; // B/ error detected, just reboot.

decfsz  COUNTER + 1           ; // S/ done verifying 8 bytes
bra     BL_Loop_check         ; // B/ not done

decfsz  COUNTER + 0           ; // S/ all 8 write pages in erase page done
bra     BL_loop_outer         ; // B/ more write pages in this erase page

tblrd   +*                    ; // Dummy read to set erase pointer OK

movlw   1                     ; // Decrement erase page counter
subwf   FLASH_CNT + 0
movlw   0
subwfb  FLASH_CNT + 1
movf    FLASH_CNT + 1, w
iorwf   FLASH_CNT + 0, w
bnz     BL_loop_erase         ; // B/ more erase pages to do

; // All programming OK, proceed

ProgOk                                     ; // Label for debugging.

bcf     eecon1, wren          ; // Disable FLASH writes

clrf    EE_ADDR + 0           ; // Set loader stater OK
clrf    EE_ADDR + 1
movlw   2
movwf   EE_DATA
xorlw   0xff
movwf   EE_DATA + 1
rcall   WriteEEWord

; // Fix CRC

clrf    CRC_16 + 0           ; // Clear CRC
clrf    CRC_16 + 1

movlw   4                     ; // Process 4 bytes
movwf   COUNTER
clrf    EE_ADDR

FixCrcLp

rcall   ReadEEByte            ; // Read next byte into EE_DATA and into wreg
rcall   CalcCrc16

decfsz  COUNTER
bra     FixCrcLp

movff   CRC_16 + 0, EE_DATA + 0
movff   CRC_16 + 1, EE_DATA + 1

; // The following two instructions are not needed, ReadEEByte will do it
; //
; // movlw  0x04
; // movwf  EE_ADDR

rcall   WriteEEWord

Reset                                       ; // Reboot the controller, let APPLICATION handle the rest.

; // End of boot loader

; // ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; // APPLICATION ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

org     0x0200                          ; // RESET

ResetVector
bra     ValueInit                        ; // Go initialize Value defined words.
; // !! a compiler generated entry point !!

```



```

//
// Register setup :
//
// Uses register definitions from somewhere else, these must be
// 4th constant definitions of the form :
//
//     n constant XXXX_IniVal
//
// Where :
//
//     n     : a value
//     XXXX  : a valid PIC18 register name - see processor dependent .INC file.
//
// Use as :
//
//     InitReg XXXX
//
// Generates Code :
//
//     movlw  _XXXX_IniVal
//     movwf  XXXX
//
// //////////////////////////////////////
macro InitReg ( aName ) ( -- )
    movlw  <@" $aName$">_IniVal
    movwf  $aName$
endMacro

// //////////////////////////////////////
#ifdef PageCrossWarnOff // If PageCrossWarnOff is defined don't generate
                        // 'crossing page boundary' messages for 4th
                        // compiled code.
    . ' #define PageWarnOff'
    . ' '
#endif

// //////////////////////////////////////
macro List ( aFormat ) ( -- ) // Control assembly macro listings
    LIST    $aFormat$
endMacro

// //////////////////////////////////////
macro HexFormat ( aFormat ) ( -- ) // Control the default assembler radix
    LIST    F=$aFormat$
endMacro

// //////////////////////////////////////
macro Title ( aTitle ) ( -- ) // Sets a title for the assembler
    TITLE   $aTitle$
    <!
        AddDoc' Title=$aTitle$'
    >
endMacro

// //////////////////////////////////////
macro Page ( ) ( -- ) // Ejects a page in the assmbler output list file
    PAGE
endMacro

// //////////////////////////////////////
macro Config ( aValue ) ( -- ) // Processor configuration word
    CONFIG  $aValue$
    <!
        #defined AddConfigToDocs
        #if
            AddConfigToDocs 0 = not
            #if
                AddDoc' config $aValue$'
            #endif
        #endif
    >
endMacro

// //////////////////////////////////////
macro IdLoc ( aNum aParams ) ( -- ) // Processor ID location byte
    __IDLOCS  _IDLOC$aNum$, $aParams$
endMacro

// //////////////////////////////////////
//
only Compiler definitions also Forth
// Low level abstraction code and maro's

```

```

// The compiler relies on these to be present

// ////////////////////////////////////////////////////////////////////

macro ValueBegin ( ) ( -- )

; // =====
; // ===== Value initialization code =====

ValueInit                ; // Value initialization code is compiled here
                        ; // Followed by the rest of ValueBegin ( )
    SetBSRDefault        ; // Setup BSR
    AddressToFsr0 SZero  ; // Setup data stack
    AddressToFsr1 LZero  ; // Setup loop stack
endMacro

// ////////////////////////////////////////////////////////////////////
// Inbetween the compiler will generate value initialization code
// ////////////////////////////////////////////////////////////////////

macro ValueEnd ( MainBank ) ( -- )

    goto    <@" Main">        ; // Jump into the 4th [[Main]] function.

; // ===== End of value initialization code =====
; // =====
endMacro

// ////////////////////////////////////////////////////////////////////

macro (ForceRomBank) ( Nr Address ) ( -- ) #// CS

; // (ForceRomBank) $Nr$ $Address$
; // a NOOP for 18FXXX, it has no rom banking.
endMacro

// ////////////////////////////////////////////////////////////////////

macro (colon) ( MLabel NLabel Bank ) ( -- ) #// CS

    call    $MLabel$          ; // $Bank$: $NLabel$ (colon)
endMacro

// ////////////////////////////////////////////////////////////////////

macro (code) ( MLabel NLabel Bank ) ( -- ) #// CS

    call    $MLabel$          ; // $Bank$: $NLabel$ (code)
endMacro

// ////////////////////////////////////////////////////////////////////

macro (semi) ( ) ( -- ) #// CS

    return                ; // B/ ----> (;)
endMacro

// ////////////////////////////////////////////////////////////////////

macro (seminoreturn) ( ) ( -- ) #// CS

; // (noreturn;)
endMacro

// ////////////////////////////////////////////////////////////////////

macro (lit) ( Arg Sign ) ( -- b ) #// CS

    PushLit $Sign$D'$Arg$'    ; // literal $Sign$$Arg$
endMacro

// ////////////////////////////////////////////////////////////////////

macro (dlit) ( Arg Sign ) ( -- w ) #// CS

; // dliteral $Sign$$Arg$
    PushLit ($Sign$D'$Arg$' >> 0) & D'255'
    PushLit ($Sign$D'$Arg$' >> 8) & D'255'
endMacro

// ////////////////////////////////////////////////////////////////////

macro (tlit) ( Arg Sign ) ( -- t ) #// CS

; // tliteral $Sign$$Arg$
    PushLit ($Sign$D'$Arg$' >> 0) & D'255'
    PushLit ($Sign$D'$Arg$' >> 8) & D'255'
    PushLit ($Sign$D'$Arg$' >> 16) & D'255'
endMacro

// ////////////////////////////////////////////////////////////////////

macro (qlit) ( Arg Sign ) ( -- q ) #// CS

```

```

; // qliteral $Sign$$Arg$
PushLit ($Sign$D'$Arg$' >> 0) & D'255'
PushLit ($Sign$D'$Arg$' >> 8) & D'255'
PushLit ($Sign$D'$Arg$' >> 16) & D'255'
PushLit ($Sign$D'$Arg$' >> 24) & D'255'
endMacro

```

//

```

macro (str-constdef) ( Name Mangled Size Value ) ( -- ) #// CS
$Mangled$
; // str-constdef $Name$
db D'$Size$', "$Value$"
endMacro

```

//

```

macro (data-constdef) ( Name Mangled Size Value ) ( -- ) #// CS
$Mangled$
; // data-constdef $Name$
db D'$Size$', $Value$
endMacro

```

//

```

macro (str-constant) ( Name Mangled ) ( -- wRomAddr ) #// CS
; // str-constant $Name$
; // ( -- wRomAddr )
PushLit ($Mangled$ >> 0) & D'255'
PushLit ($Mangled$ >> 8) & D'255'
endMacro

```

//

```

macro (constdef) ( Name Mangled Value Sign ) ( -- ) #// CS
$Mangled$ equ $Sign$D'$Value$' ; // constant $Name$
endMacro

```

//

```

macro (constant) ( Name Value Sign ) ( -- b ) #// CS
; // constant $Name$ ( $Sign$$Value$)
PushLit $Sign$D'$Value$'
endMacro

```

//

```

macro (2constdef) ( Name Mangled Value Sign ) ( -- ) #// CS
$Mangled$ equ $Sign$D'$Value$' ; // 2constant $Name$
endMacro

```

//

```

macro (2constant) ( Name Value Sign ) ( -- w ) #// CS
; // 2constant $Name$ ( $Sign$$Value$)
PushLit ($Sign$D'$Value$' >> 0) & D'255'
PushLit ($Sign$D'$Value$' >> 8) & D'255'
endMacro

```

//

```

macro (3constdef) ( Name Mangled Value Sign ) ( -- ) #// CS tripple constant definition
<@" $Name$"> equ $Sign$D'$Value$' ; // 3constant [[$Name$]]
endMacro

```

//

```

macro (3constant) ( Name Value Sign ) ( -- t ) #// CS Tripple constant invocation
; // 3constant [[$Name$]] ( $Sign$$Value$)
PushLit ( <@" $Name$"> >> 0) & D'255'
PushLit ( <@" $Name$"> >> 8) & D'255'
PushLit ( <@" $Name$"> >> 16) & D'255'
endMacro

```

//

```

macro (4constdef) ( Name Mangled Value Sign ) ( -- ) #// CS
$Mangled$ equ $Sign$D'$Value$' ; // 4constant $Name$
endMacro

```

//

```

macro (4constant) ( Name Value Sign ) ( -- q ) #// CS
; // 4constant $Name$ ( $Sign$$Value$)
PushLit ($Sign$D'$Value$' >> 0) & D'255'
PushLit ($Sign$D'$Value$' >> 8) & D'255'
PushLit ($Sign$D'$Value$' >> 16) & D'255'
PushLit ($Sign$D'$Value$' >> 24) & D'255'
endMacro

```

```

// //////////////////////////////////////
macro (vardef) ( Name Mangled Value Bank ) ( -- ) #// CS
$Mangled$ equ 256 * D'$Bank$' + D'$Value$' ; // variable $Name$ :: Bank = $Bank$
endMacro

// //////////////////////////////////////
macro (variable) ( Name Mangled Value Bank ) ( -- wRamAddr ) #// CS
; // variable $Name$ ( $Bank$: $Value$ ) ; $Mangled$
    PushLit D'$Value$'
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////
macro (2vardef) ( Name Mangled Value Bank ) ( -- ) #// CS
$Mangled$ equ 256 * D'$Bank$' + D'$Value$' ; // 2variable $Name$ :: Bank = $Bank$
endMacro

// //////////////////////////////////////
macro (2variable) ( Name Mangled Value Bank ) ( -- wRamAddr ) #// CS
; // 2variable $Name$ ( $Bank$: $Value$ ) ; $Mangled$
    PushLit D'$Value$'
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////
macro (3vardef) ( Name Mangled Value Bank ) ( -- ) #// CS tripple variable definition
<@" $Name$"> equ 256 * D'$Bank$' + D'$Value$' ; // 3variable [[ $Name$ ]] :: Bank = $Bank$
endMacro

// //////////////////////////////////////
macro (3variable) ( Name Mangled Value Bank ) ( -- wRamAddr ) #// CS Tripple variable invocation
; // 3variable [[ $Name$ ]] ( $Bank$: $Value$ ) ; <@" $Name$">
    PushLit D'$Value$'
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////
macro (4vardef) ( Name Mangled Value Bank ) ( -- ) #// CS
$Mangled$ equ 256 * D'$Bank$' + D'$Value$' ; // 4variable $Name$ :: Bank = $Bank$
endMacro

// //////////////////////////////////////
macro (4variable) ( Name Mangled Value Bank ) ( -- wRamAddr ) #// CS
; // 4variable $Name$ ( $Bank$: $Value$ ) ; $Mangled$
    PushLit D'$Value$'
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////
macro (arraydef) ( Name Mangled Value Size Bank ) ( -- ) #// CS
$Mangled$ equ 256 * D'$Bank$' + D'$Value$' ; // $Size$ Array $Name$ :: Bank = $Bank$
endMacro

// //////////////////////////////////////
macro (array) ( Name Mangled Value Size Bank ) ( -- wRamAddr ) #// CS
; // $Size$ Array $Name$ ( $Bank$: $Value$ ) ; $Mangled$
    PushLit D'$Value$'
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////
Macro (fielddef) ( Name Mangled Value Sign ) ( -- ) #// CS Structure field definition
<@" $Name$"> equ $Sign$D'$Value$' ; // Field $Name$
EndMacro

// //////////////////////////////////////
Macro (field) ( Name Value Sign ) ( -- b ) #// CS Structue field invocation
; // Field $Name$ ( $Sign$: $Value$ )
    PushLit <@" $Name$">
EndMacro

// //////////////////////////////////////
Macro (>field) ( Name Value Sign ) ( u1 u2 -- u3 ) #// CS Structure field offset calculation

```

```

; // >Field $Name$ ( $Sign$$Value$)
; // ( bBase -- bBase+bOffset )
PushLit <@" $Name$" >
addwf postdec0, w ; // (+) ( u1 u2 -- u3 )
EndMacro

// //////////////////////////////////////

Macro (s>field) ( Name Value Sign ) ( w -- w ) #// CS Structure field offset calculation
; // s>Field $Name$ ( $Sign$$Value$)
; // ( dBase -- dBase+dOffset )
PushLit ( <@" $Name$" > >> 0 ) & D'255'
PushLit ( <@" $Name$" > >> 8 ) & D'255'
call <@" d+" >
EndMacro

// //////////////////////////////////////

Macro (2fielddef) ( Name Mangled Value Sign ) ( -- ) #// CS Double field definition
<@" $Name$" > equ $Sign$D'$Value$' ; // 2Field $Name$
EndMacro

// //////////////////////////////////////

Macro (2field) ( Name Value Sign ) ( -- w ) #// CS Double field invocation
; // 2Field $Name$ ( $Sign$$Value$)
PushLit ( <@" $Name$" > >> 0 ) & D'255'
PushLit ( <@" $Name$" > >> 8 ) & D'255'
EndMacro

// //////////////////////////////////////

Macro (>2field) ( Name Value Sign ) ( w -- w ) #// CS Structure field offset calculation
; // >2Field $Name$ ( $Sign$$Value$)
; // ( dBase -- dBase+dOffset )
PushLit ( <@" $Name$" > >> 0 ) & D'255'
PushLit ( <@" $Name$" > >> 8 ) & D'255'
call <@" d+" >
EndMacro

// //////////////////////////////////////

Macro (s>2field) ( Name Value Sign ) ( b -- w ) #// CS Structure field offset calculation
; // s>2Field $Name$ ( $Sign$$Value$)
; // ( bBase -- bBase+dOffset )
PushLit 0 ; // perform u>ud
PushLit ( <@" $Name$" > >> 0 ) & D'255'
PushLit ( <@" $Name$" > >> 8 ) & D'255'
call <@" d+" >
EndMacro

// //////////////////////////////////////

Macro (structdef) ( Name Mangled Value Sign ) ( -- ) #// CS Structure definition
<@" $Name$" > equ $Sign$D'$Value$' ; // Struct $Name$
EndMacro

// //////////////////////////////////////

Macro (struct) ( Name Value Sign ) ( -- b ) #// CS Structure invocation
; // Struct $Name$ ( $Sign$$Value$)
PushLit <@" $Name$" >
EndMacro

// //////////////////////////////////////

// ValueBegin and ValueEnd macro's are
// automatically invoked by the pic4th compiler

// //////////////////////////////////////

macro (valuestartup) ( Name Mangled Address Bank Value Sign ) ( -- ) #// CS
; // (valuestartup) - value initialization
; // Value $Name$ ( $Bank$: $Address$) := $Sign$$Value$ ; $Mangled$

movlw $Sign$D'$Value$'
movwf D'$Address$'
endMacro

// //////////////////////////////////////

macro (valuedef) ( Name Mangled Value Bank ) ( -- ) #// CS
$Mangled$ equ 256 * D'$Bank$' + D'$Value$' ; // Value $Name$ :: Bank = $Bank$
endMacro

// //////////////////////////////////////

macro (value) ( Name Mangled Address Bank Fetch ) ( -- b ) #// CS
; // (value) Value $Name$ ( $Bank$: $Address$) ; $Mangled$

PushLit D'$Address$'
PushLit D'$Bank$'

```

```

    call    $Fetch$
endMacro

// //////////////////////////////////////

macro (to) ( Name Mangled ) ( b -- ) ##// CS
    ; // =: $Name$ / To $Name$ ( b -- ) ; $Mangled$
    PopReg $Mangled$
endMacro

// //////////////////////////////////////

macro (InitLocalSpace) ( aLocalStart ) ( -- ) ##// CS
    ; // Initialize the local data pointer (LDP)
    movlw  D'$aLocalStart$'
    movwf  LDP
endMacro

// //////////////////////////////////////

macro (GetLocalSpace) ( anAmount ) ( -- ) ##// CS
    ; // Reserve space ($anAmount$ bytes) for local variables.
    ; // Save TOS
    movwf  tmp1
    movf   LDP, w
    ; // Get LDP
    ; // Add $anAmount$ to it
    addlw  D'$anAmount$'
    movwf  LDP
    movf   tmp1, w
    ; // Restore TOS
endMacro

// //////////////////////////////////////

macro (FreeLocalSpace) ( anAmount ) ( -- ) ##// CS
    ; // Free space ($anAmount$ bytes) for local variables.
    ; // Save TOS
    movwf  tmp1
    movf   LDP, w
    ; // Get LDP
    ; // Subtract $anAmount$ from it
    sublw  D'$anAmount$'
    movwf  LDP
    movf   tmp1, w
    ; // Restore TOS
endMacro

// //////////////////////////////////////

macro (locvar) ( Name Mangled Value Bank Sign ) ( -- wRamAddr ) ##// CS
    ; // LocVar $Name$ ( $Bank$: $Sign$ $Value$ ) ; $Mangled$
    PushLit $$Sign$D'$Value$'
    addwf  LDP, w
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////

macro (loc2var) ( Name Mangled Value Bank Sign ) ( -- wRamAddr ) ##// CS
    ; // Loc2Var $Name$ ( $Bank$: $Sign$ $Value$ ) ; $Mangled$
    PushLit $$Sign$D'$Value$'
    addwf  LDP, w
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////

macro (loc3var) ( Name Mangled Value Bank Sign ) ( -- wRamAddr ) ##// CS Local tripple variable invocation
    ; // [[Loc3Var]] [[ $Name$ ]] ( $Bank$: $Sign$ $Value$ ) ; <@" $Name$">
    PushLit $$Sign$D'$Value$'
    addwf  LDP, w
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////

macro (loc4var) ( Name Mangled Value Bank Sign ) ( -- wRamAddr ) ##// CS
    ; // Loc4Var $Name$ ( $Bank$: $Sign$ $Value$ ) ; $Mangled$
    PushLit $$Sign$D'$Value$'
    addwf  LDP, w
    PushLit D'$Bank$'
endMacro

// //////////////////////////////////////

macro (label) ( Label ) ( -- ) ##// CS
    $Label$
endMacro

// //////////////////////////////////////

macro (goto) ( Label ) ( -- ) ##// CS
    ; // (goto) $Label$
    bra    $Label$
endMacro

```

```

// ////////////////////////////////////////////////////////////////////
macro (jumpto) ( Name Mangled ) ( -- ) #// CS
    bra    $Mangled$          ; // (jumpto) $Name$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (call) ( Label ) ( -- ) #// CS
    call  $Label$            ; // (call) $Label$
endMacro

// ////////////////////////////////////////////////////////////////////
code ((fbranch)) ( f -- ) #// CS
    PopReg  tmp1              ; // Pop Flag into tmp1
    movf   tmp1, f            ; // To set or clear Z flag
    return
endcode

// ////////////////////////////////////////////////////////////////////
macro (begin) ( Label ) ( -- ) #// CS
    $Label$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (until) ( Label ) ( -- ) #// CS
    fBranch $Label$          ; // (until) - branch if Z flag set
endMacro

// ////////////////////////////////////////////////////////////////////
macro (again) ( Label ) ( -- ) #// CS
    bra    $Label$           ; // (again) - unconditional branch
endMacro

// ////////////////////////////////////////////////////////////////////
macro (while) ( Label ) ( -- ) #// CS
    fBranch $Label$          ; // (while) - branch if Z flag set
endMacro

// ////////////////////////////////////////////////////////////////////
macro (repeat) ( Label1 label2 ) ( -- ) #// CS
    bra    $Label1$          ; // (repeat) - unconditional branch
    $Label2$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (if) ( Label ) ( -- ) #// CS
    fBranch $Label$          ; // (if) - branch if Z flag set
endMacro

// ////////////////////////////////////////////////////////////////////
macro (else) ( Label1 Label2 ) ( -- ) #// CS
    bra    $Label2$          ; // (else) - unconditional branch
    $Label1$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (then) ( Label ) ( -- ) #// CS
    $Label$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (case) ( ) ( -- ) #// CS
    ; // (case)- place holder only
endMacro

```

```

// ////////////////////////////////////////////////////////////////////
macro (endcase) ( Label ) ( -- ) #// CS
                                ; // (endcase) - a label
$Label$
endMacro

// ////////////////////////////////////////////////////////////////////
code ((of)) ( b1 b2 -- b1 | ) #// CS
  PopReg  tmp1                ; // Get & drop n2 into tmp1
  cpfseq  tmp1                ; // S// n1 == n2
  bra     of_uneq             ; // B/ n1 <> n2

  ; // n1 == n2

  Drop    Status, Z           ; // Drop n1
  bcf     Status, Z           ; // Flag match
  return

of_uneq

  bsf     Status, Z           ; // Flag no match
  return
endCode

// ////////////////////////////////////////////////////////////////////
macro (of) ( Label ) ( -- ) #// CS
                                ; // (of) - branch if Z flag set
  fBranch $Label$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (endof) ( Label1 Label2 ) ( -- ) #// CS
                                ; // (endof) - unconditional branch
  bra     $Label1$
$Label2$
endMacro

// ////////////////////////////////////////////////////////////////////
macro (table) ( ) ( -- ) #// CS
                                ; // (table)
  bcf     wreg, 7
  rlnsf   wreg
  addwf   pcl, f
endMacro
  deprecated

// ////////////////////////////////////////////////////////////////////
macro (TableElt) ( Arg Sign ) ( -- ) #// CS
                                ; // (TableElt)
  retlw   $Sign$D'$Arg$'
endMacro

  deprecated

// ////////////////////////////////////////////////////////////////////
macro ((table)) ( Arg Bank ) ( -- ) #// CS
                                ; // ((table)) $Bank$:$Arg$
  movwf   tmp1
  movlw   ($Arg$ >> 0) & 0xff
  addwf   tmp1, w
  movlw   ($Arg$ >> 8) & 0xff
  btfsc   status, C
  addlw   D'1'
  movwf   pclath
  movf    tmp1, w

  call    $Arg$
endMacro

  deprecated

// ////////////////////////////////////////////////////////////////////
code ((loop)) ( -- ) #// CS
  PushNothing                ; // Save wreg

  incf    indf1                ; // Advance index
  movf    postinc1, w          ; // get index into wreg
  xorwf   indf1, w            ; // Compare with limit
  bnz     LoopMore            ; // B/ Index <> Limit

LoopNoMore                    ; // Also used by [[(+loop)]]

  incf    fsr1l, f            ; // Index == Limit
  Drop    ; // Restore wreg

```

```

    bcf     status, Z           ; // signal index = limit
    return

LoopMore                               ; // Also used by [(+loop)]

    decf   fsr11, f           ; // restore Loop stack pointer
    Drop                                     ; // Restore wreg
    bsf    status, Z         ; // signal index <> limit
    return
endCode

// ////////////////////////////////////////

code ((+loop)) ( b -- ) #// CS
    addwf  indf1              ; // Index + inc -> Index
    movf   postinc1, w        ; // Get index LSP++
    subwf  indf1, W           ; // Limit - Index
    bc     LoopMore           ; // B/ Carry , Not done, see [((loop))]

    bra    LoopNoMore        ; // B/ No carry, done , see [((loop))]
endCode
resources
((loop))
endResources

// ////////////////////////////////////////

code ((do)) ( bLimit bStart -- ) #// CS
    PopReg  tmp1              ; // Start into tmp1

dodo                                     ; // entry point for [((?do))] if loop taken

    PopReg  tmp2              ; // Limit into tmp2
    LsPushReg tmp2            ; // Limit on loop stack
    LsPushReg tmp1            ; // Start on loop stack (as Index)
    bcf     status, z         ; // Return with zero flag cleared
    return                                     ; // B/ ----> done (loop taken)
endCode

// ////////////////////////////////////////

code ((?do)) ( bLimit bStart -- ) #// CS
    PopReg  tmp1              ; // bStart into tmp1
    cpfseq  indf1              ; // S/ Start = Limit
    bra     dodo               ; // B/ start <> Limit, perform [((do))]

    Drop                                     ; // start = limit : drop bLimit
    bsf     status, z         ; // Return with zero flag set (loop skipped)
    return                                     ; // B/ ----> done
endCode
resources
((do))
endResources

// ////////////////////////////////////////

macro (do) ( SkipLabel LoopLabel ) ( bLimit bStart -- ) #// CS
    call    <@" ((do))">
    ; // (do)
$LoopLabel$
endMacro

// ////////////////////////////////////////

macro (?do) ( SkipLabel LoopLabel ) ( bLimit bStart -- ) #// CS
    call    <@" ((?do))">
    fBranch $SkipLabel$
    ; // (?do) : check if limit is count, if so skip loop
$LoopLabel$
endMacro

// ////////////////////////////////////////

macro (loop) ( LoopLabel SkipLabel ) ( -- ) #// CS
    fBranch $LoopLabel$
    ; // (loop) branch if Z flag set
$SkipLabel$
endMacro

// ////////////////////////////////////////

code (Leave) ( -- ) #// CS
    PushNothing                ; // Save wreg
    incf   fsr11               ; // Point to limit
    decf   postdec1, w         ; // Get limit, dec limit, result into wreg
    movwf  indf1               ; // Store limit - 1 as index
    Drop
    return
endCode

// ////////////////////////////////////////

```



```

movwf    tosl
enable   ; // Interrupts back on

PushReg tmp1 ; // Repush bException on restored data stack

; // Exception frame as on loop stack
; //
; //      | prev_frame      | <--- | EFP | fsrl |
; //      +-----+-----+ +-----+-----+
; //      |                 |
; //      |                 |
; //      |                 |
; //      +-----+-----+
; //      | addr_except low |
; //      ~                 ~

return   ; // B/ ----> Returns to (except) or (finally) handler
EndCode

// //////////////////////////////////////

Code ((no_except)) ( -- ) #// CS Helper for (except) and [[(finally)]]

; // Discard one (except) frame

; // Exception frame as on loop stack
; //
; //      +--> | prev_frame |
; //      +-----+-----+
; //      |                 |
; //      |                 |
; //      |                 |
; //      +-----+-----+
; //      | addr_except low | --- ROM address of ((except)) to invoke
; //      | addr_except high| -+
; //      | addr_except upper| -+
; //      | old_stkptr      | ---- Old return stack pointer
; //      | old_wreg        | ---- Value that was on top of old stack
; //      | old_data_sp low | --- old data stack including wreg
; //      | old_data_sp high| -+
; //      +--<-| old_EFP    low | +-----+
; //      +--<-| old_EFP    high| <--- + EFP |
; //      +-----+-----+ +-----+
; //      |                 |
; //      |                 |
; //      |                 | <--- + fsrl|
; //      +-----+-----+ +-----+

RemoveExceptionFrame ; // Unwind one exception level
LsFree 7 ; // Drop old_data_sp, old_wreg and addr_except

; // Exception frame as on loop stack
; //
; //      | prev_frame      | <--- | EFP | fsrl |
; //      +-----+-----+ +-----+-----+
; //      |                 |
; //      |                 |
; //      |                 |
; //      +-----+-----+
; //      | addr_except low |
; //      ~                 ~

return   ; // B/ ----> done
EndCode

// //////////////////////////////////////

Macro (try) ( aTryLabel ) ( -- ) #// CS Compiled [[try]]

; // (try) ( -- )

PushLit (( $aTryLabel$ ) >> 16) & 0xff ; // Setup a fake return address for (raise)
PushLit (( $aTryLabel$ ) >> 8) & 0xff ; // long jump to (except) handler
PushLit (( $aTryLabel$ ) >> 0) & 0xff ; // using REVERSED push

call    <@" ((try))"> ; // ((try)) Create (except) frame on return stack
; // ( tRomAddr -- ) [ -- tRomAddr dDSP ]

; // end (try) ( -- ) [ -- tRomAddr dDSP ]

EndMacro

// //////////////////////////////////////

Macro (except) ( aTryLabel anEndLabel ) ( b -- b ) #// CS Compiled [[except]]

; // (except) ( bException | -- bException | )

; // OK :: We fell trough, no exception, nothing on stack
; // ( )
call    <@" ((no_except))"> ; // ((no_except)) remove (except) frame

bra    $anEndLabel$ ; // B/ skip to (endtry)

$aTryLabel$
; // EXCEPTION :: We came in ththrough a Raise action, exception on stack
; // ( bException )

```

```

; // end ( except)
EndMacro

// ////////////////////////////////////////
Macro (finally) ( aTryLabel aDummyForCompatiltyReasons ) ( b -- ) #// CS Compiled [[finally]]
; // (finally) ( bException | -- bException | )
; // OK :: We fell trough, no exception, nothing on stack
; // ( )
call <@" ((no_except))"> ; // ((no_except)) remove(except) frame
PushLit 0 ; // Signal 'no exception'
; // (finally) Fall through to $aTryLabel$
$aTryLabel$
; // EXCEPTION :: We came in through a Raise action, exception on stack
; // or we fell having no exception, zero on stack
; // ( bException )
Drop ; // But just forget about it for now.
; // (This should be re-raised at EndTry)
; // end ( finally)
EndMacro

// ////////////////////////////////////////
Macro (endtry) ( anEndLabel ) ( -- ) #// CS Compiled [[endTry]]
; // (endtry), place holder
$aEndLabel$
EndMacro

// ////////////////////////////////////////
Macro (raise) ( ) ( b -- ) #// CS Compiled [[Raise]]
; // (raise) ( bExceptionOnCurrentDataStack -- bExceptionOnOldDataStack ) [ tRomAddr dDSP -- ]
call <@" ((raise))"> ; // ((raise)) jump to (except) or (finally) handler
; // through (except) frame on return stack
EndMacro

// ////////////////////////////////////////
macro (+) ( ) ( u1 u2 -- u3 ) #// CS
addwf postdec0, w ; // (+) ( u1 u2 -- u3 )
endMacro

// ////////////////////////////////////////
macro (-) ( ) ( u1 u2 -- u3 ) #// CS
subwf postdec0, w ; // (-) ( u1 u2 -- u3 )
endMacro

// ////////////////////////////////////////
code (*) ( u1 u2 -- u3 ) #// CS
PopReg tmp1
mulwf tmp1
movf prodl, w

return
endcode

// ////////////////////////////////////////
code (/) ( u1 u2 -- u3 ) #// CS
; // also calculates remainder,
; // but we don't care about that!
; // remainder
; // u2
; // loop counter
; // to index
clrf tmp1
movwf tmp3
movlw D'8'
movwf tmp2

slash_0
rlcf indf0, w ; // rotate u1
rlcf tmp1, f ; // into remainder
movf tmp3, w ; // get u2
subwf tmp1, f ; // Try to subtract
bc slash_1 ; // B/ subtract did work, carry is set

addwf tmp1, f ; // subtract did not work, so add it again
bcf status, C ; // and clear carry

slash_1
rlcf indf0, f ; // result in indf0

```



```

// ////////////////////////////////////////////////////////////////////
macro (drop) ( ) ( b -- ) #// CS           ; // drop ( b -- )
    Drop
endMacro

// ////////////////////////////////////////////////////////////////////
macro (dup) ( ) ( b -- b b ) #// CS       ; // (dup) ( b -- b b )
    Dup
endMacro

// ////////////////////////////////////////////////////////////////////
code (swap) ( b1 b2 -- b2 b1 ) #// CS
    PopReg tmp2
    PopReg tmp1
    PushReg tmp2
    PushReg tmp1
    return
endcode

// ////////////////////////////////////////////////////////////////////
macro (True) ( ) ( -- f ) #// CS          ; // (True) ( -- f )
    PushLit True
endMacro

// ////////////////////////////////////////////////////////////////////
macro (False) ( ) ( -- f ) #// CS         ; // (False) ( -- f )
    PushLit False
endMacro

// ////////////////////////////////////////////////////////////////////
macro (Space) ( ) ( -- b ) #// CS         ; // (space) ( -- b )
    PushLit 0x20
endMacro

// ////////////////////////////////////////////////////////////////////
macro (negate) ( ) ( b -- b ) #// CS      ; // negate ( b1 -- b2 )
    sublw D'0'
endMacro

// ////////////////////////////////////////////////////////////////////
macro (not) ( ) ( b -- b ) #// CS         ; // Not ( b1 -- b2 )
    xorlw 0xff
endMacro

// ////////////////////////////////////////////////////////////////////
code (=) ( b1 b2 -- f ) #// CS
    subwf postdec0, f
    btfsc status, Z
    retlw True
    retlw False
endcode

// ////////////////////////////////////////////////////////////////////
code (<>) ( b1 b2 -- f ) #// CS
    subwf postdec0, f
    btfsc status, Z
    retlw False
    retlw True
endcode

// ////////////////////////////////////////////////////////////////////
code (<) ( u1 u2 -- f ) #// CS
    subwf postdec0, f
    movlw True ; // assume true
    btfss status, Z ; // if set then always false
    btfsc status, C ; // if not then also analyze carry
    clrf wreg ; // false flag
    return
endcode

// ////////////////////////////////////////////////////////////////////
: (>) ( u1 u2 -- f ) #// CS
    Swap <

```

```

;

// ////////////////////////////////////////

: (<=) ( u1 u2 -- f ) #// CS
> Not
;

// ////////////////////////////////////////

: (>=) ( u1 u2 -- f ) #// CS
< Not
;

// ////////////////////////////////////////

code (?dup) ( b -- b b | b -- 0 ) #// CS
iorlw D'0'
btfsc status, Z
return

Dup
return
endcode

// ////////////////////////////////////////

macro (2drop) ( ) ( w -- ) #// CS
Drop
Drop
endmacro

// ////////////////////////////////////////

code (over) ( b1 b2 -- b1 b2 b1 ) #// CS
movf indf0, tmp1
PushReg tmp1
return
endcode

// ////////////////////////////////////////

: (2dup) ( w -- w w ) #// CS
Over
Over
;

// ////////////////////////////////////////

macro (lshift) ( ) ( b -- b ) #// CS
rlncf wreg
bcf wreg, 0
endmacro

// ////////////////////////////////////////

macro (rshift) ( ) ( b -- b ) #// CS
rrncf wreg
bcf wreg, 7
endmacro

// ////////////////////////////////////////

code (<<) ( b1 b2 -- b3 ) #// CS

PopReg tmp1
; // Shift b1 left over b2 positions

shllp

bcf Status, C
rlcf wreg, f
decfsz tmp1
bra shllp

return
endcode

// ////////////////////////////////////////

code (>>) ( b1 b2 -- b3 ) #// CS

PopReg tmp1
; // Shift b1 right over b2 positions

shrlp

bcf Status, C
rrcf wreg, f
decfsz tmp1
bra shrlp

```

```
return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (2Over) ( w1 w2 -- w1 w2 w1 ) #// CS
PopReg tmp4
PopReg tmp3
PopReg tmp2
movwf tmp1

PushReg tmp2
PushReg tmp3
PushReg tmp4
PushReg tmp1
PushReg tmp2

return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (rot) ( b1 b2 b3 -- b2 b3 b1 ) #// CS
PopReg tmp3
PopReg tmp2
PopReg tmp1

PushReg tmp2
PushReg tmp3
PushReg tmp1

return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
alias (rot) (+rot) #// CS (rot) also known as (+rot)
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (-rot) ( b1 b2 b3 -- b3 b1 b2 ) #// CS
PopReg tmp3
PopReg tmp2
PopReg tmp1

PushReg tmp3
PushReg tmp1
PushReg tmp2

return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (pick) ( u1 -- u2 ) #// CS
; // Push u:th entry

PushNothing
sublw D'0' ; // negate
movf plusw0, w ; // w = offset of reqd. byte,
return ; // 0th byte references TOS
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (put) ( b u -- ) #// CS
; // u th byte of stack replaced by value b
; // 0th byte = top of stack

sublw D'0' ; // negate
PopReg tmp1 ; // Get offset
movwf tmp2 ; // Get value to put
movf tmp1, w ; // Offset to wreg
movff tmp2, plusw0 ; // Put value
PopReg wreg ; // Pop top item of stack into wreg
return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (abs) ( b -- u ) #// CS
btfs wreg, 7
sublw D'0'
return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code (Max) ( u1 u2 -- u3 ) #// CS
PopReg tmp1
```



```

code (J') ( -- u ) #// CS
    PushLit D'3'                ; // Get 2nd loop limit
    movf    plusw1, w
    return
endcode

// ////////////////////////////////////////////////////////////////////

code (K) ( -- u ) #// CS
    PushLit D'4'                ; // Get 3d loop index
    movf    plusw1, w
    return
endcode

// ////////////////////////////////////////////////////////////////////

code (R>) ( -- b ) #// CS [ b -- ]
    LsPopReg    tmp1
    PushReg     tmp1
    return
endcode

// ////////////////////////////////////////////////////////////////////

code (>R) ( b -- ) #// CS [ -- b ]
    PopReg      tmp1
    LsPushReg   tmp1
    return
endcode

// ////////////////////////////////////////////////////////////////////
// ////////////////////////////////////////////////////////////////////

Only Forth Definitions also Compiler

// Normal user words - the library words

// ////////////////////////////////////////////////////////////////////

macro BreakPoint ( aMsg ) ( -- )
    nop          ; BREAKPOINT $aMsg$
endmacro

// ////////////////////////////////////////////////////////////////////

macro Nop ( ) ( -- )
    nop          ; // Nop ( -- )
endmacro

// ////////////////////////////////////////////////////////////////////

macro ClrWDT ( ) ( -- )
    clrwdt       ; // ClrWDT ( -- )
endmacro

// ////////////////////////////////////////////////////////////////////

macro Sleep ( ) ( -- )
    sleep        ; // Sleep ( -- )
endmacro

// ////////////////////////////////////////////////////////////////////

macro Reset ( ) ( -- )
    Reset        ; // Reset ( -- )
endmacro

// ////////////////////////////////////////////////////////////////////

macro Disable ( ) ( -- )
    Disable      ; // Disable ( -- ), disable interrupts
endmacro

// ////////////////////////////////////////////////////////////////////

macro Enable ( ) ( -- )
    Enable       ; // Enable ( -- ) // enable interrupts
endmacro

// ////////////////////////////////////////////////////////////////////

code d@ ( wRamAddr -- w ) [DFetch]
    PopReg    fsr2h                ; // Get bank selector into fsr2h
    PopReg    fsr2l
    PushReg   postinc2
    PushReg   indf2
    return

```

endcode

//

code dabs ( d -- ud )

; // 16 bits Abs

```
btfs    wreg, 7
bra     dabs_done

comf    indf0, f
incf    indf0, f
btfsc   Status, z
decf    wreg, f
comf    wreg, f
```

dabs\_done

Return

endcode

//

code Negative? ( b -- f )

```
btfs    wreg, 7
retlw   False
retlw   True
```

endcode

//

code dNegative? ( d -- f )

```
SwapDrop
bra     <@" Negative?">
```

endcode

//

code qNegative? ( q -- f )

```
SwapDrop
SwapDrop
bra     <@" dNegative?">
```

endcode

//

code dSwap ( w1 w2 -- w2 w1 )

```
PopReg  tmp4
PopReg  tmp3
PopReg  tmp2
PopReg  tmp1
```

```
PushReg tmp3
PushReg tmp4
PushReg tmp1
PushReg tmp2
```

return

endcode

//

code 2Pick ( b0 b1 b3 -- b0 b1 b2 b0 ) // Copy 3rd stack entry to top

; // Push 3rd entry

```
PushNothing          ; // Make room on stack
movlw  -D'2'          ; // negated offset
movf   plusw0, w      ; // w = offset of reqd. byte,
return               ; // 0th byte references TOS
```

endcode

//

code 3Pick ( b0 b1 b2 b4 -- b0 b1 b2 b4 b0 ) // Copy 4th stack entry to top

; // Push 4th entry

```
PushNothing          ; // Make room on stack
movlw  -D'3'          ; // negated offset
movf   plusw0, w      ; // w = offset of reqd. byte,
return               ; // 0th byte references TOS
```

endcode

//

code 4Pick ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 b4 b0 ) // Copy 5th stack entry to top

; // Push 5th entry

```
PushNothing          ; // Make room on stack
movlw  -D'4'          ; // negated offset
movf   plusw0, w      ; // w = offset of reqd. byte,
return               ; // 0th byte references TOS
```

endcode

```
// ////////////////////////////////////////////////////////////////////
code a+ ( w1 u -- w2 )
  decf   fsr01, f
  addwf  postinc0, f
  Drop
  return
endcode

// ////////////////////////////////////////////////////////////////////
macro 1a+ ( ) ( wRamAddr -- wRamAddr )
  incf   indf0, f
endmacro
; // 1a+ ( w1 -- w2 )

// ////////////////////////////////////////////////////////////////////
macro 2a+ ( ) ( wRamAddr -- wRamAddr )
  incf   indf0, f
  incf   indf0, f
endmacro
; // 2a+ ( w1 -- w2 )

// ////////////////////////////////////////////////////////////////////
code 3a+ ( w1 -- w2 )
  incf   indf0, f
  incf   indf0, f
  incf   indf0, f
  return
endcode

// ////////////////////////////////////////////////////////////////////
code 4a+ ( w1 -- w2 )
  incf   indf0, f
  incf   indf0, f
  incf   indf0, f
  incf   indf0, f
  return
endcode

// ////////////////////////////////////////////////////////////////////
code a- ( w1 u -- w2 )
  decf   fsr01, f
  subwf  postinc0, f
  Drop
  return
endcode

// ////////////////////////////////////////////////////////////////////
macro 1a- ( ) ( wRamAddr -- wRamAddr )
  decf   indf0, f
endmacro
; // 1a- ( w1 -- w2 )

// ////////////////////////////////////////////////////////////////////
macro 2a- ( ) ( wRamAddr -- wRamAddr )
  decf   indf0, f
  decf   indf0, f
endmacro
; // 2a- ( w1 -- w2 )

// ////////////////////////////////////////////////////////////////////
code 3a- ( w1 -- w2 )
  decf   indf0, f
  decf   indf0, f
  decf   indf0, f
  return
endcode

// ////////////////////////////////////////////////////////////////////
code 4a- ( w1 -- w2 )
  decf   indf0, f
  decf   indf0, f
  decf   indf0, f
  decf   indf0, f
  return
endcode

// ////////////////////////////////////////////////////////////////////
//
// Fetch Add contents of var at wAdr2 to wAdr1
//
```

```

: @a+ ( wAdr1 wAdr2 -- wAdr3 )
@ a+
;

// ////////////////////////////////////////
//
// Fetch b from wAdr + u
//

: a+@ ( wAdr u -- b )
a+ @
;

// ////////////////////////////////////////
//
// Fetch b from wAdr1 + wAdr2 @
//

: @a+@ ( wAdr1 wAdr2 -- b )
@a+@
;

// ////////////////////////////////////////
//
// Store b at wAdr + u
//

: a+! ( b wAdr u -- )
a+ !
;

// ////////////////////////////////////////
//
// Store n at wAdr1 + wAdr2 @
//

: @a+! ( b wAdr1 wAdd2 -- )
@a+!
;

// ////////////////////////////////////////
//
alias (2Drop) aDrop      ( wAddr -- )
alias (2Dup)  aDup       ( wAddr -- wAddr wAddr )
alias (2Over) aOver      ( wAddr1 wAddr2 -- wAddr1 wAddr2 wAddr1 )
alias dSwap  aSwap      ( wAddr1 wAddr2 -- wAddr2 wAddr1 )
alias dSwap  (2Swap)    ( wAddr1 wAddr2 -- wAddr2 wAddr1 )
alias (space) bl        ( -- b )

// ////////////////////////////////////////
//
: adup@ ( wAdr -- wAdr u )
// Dup wAdr and fetch a value from there.

aDup @
;

// ////////////////////////////////////////
//
: 3Drop ( t -- )
Drop
2Drop
;

// ////////////////////////////////////////
//
: 4Drop ( q -- )
2Drop
2Drop
;

// ////////////////////////////////////////
//
: 3Dup ( t -- t t )
2pick
2pick
2pick
;

// ////////////////////////////////////////
//
: 4Dup ( q -- q q )
2Over
2Over
;

// ////////////////////////////////////////

```

```
code RomPtr! ( wRomAddr -- )
  PopReg tblptrh
  Popreg tblptrl
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code RomPtr@ ( -- wRomAddr )
  Pushreg tblptrl
  PushReg tblptrh
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
macro RomPtr++ ( -- ) ( -- )
  tblrd*+ ; // RomPtr++ ( -- )
endmacro
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Rom@ ( -- b )
  tblrd*
  PushReg tablat
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code []Rom@ ( bIndex -- b )
  movff tblptrl, tmp1 ; // Save tablepointer
  movff tblptrh, tmp2
  addwf tblptrl, f ; // Add bIndex to it
  clrf wreg
  addwfc tblptrh, f
  tblrd* ; // read bData byte
  movf tablat, w ; // return data in TOS
  movff tmp2, tblptrl ; // restore table pointer
  movff tmp1, tblptrh
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Rom@++ ( -- b )
  tblrd*+
  PushReg tablat
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code ++Rom@ ( -- b )
  tblrd*+
  PushReg tablat
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Rom@-- ( -- )
  tblrd*-
  PushReg tablat
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Rom! ( b -- )
  PopReg tablat
  tblwt*
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Rom!++ ( b -- )
  PopReg tablat
  tblwt*+
  return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code ++Rom! ( b -- )
  PopReg tablat
  tblwt*+
  return
```

endcode

//

```
code Rom!-- ( b -- )
  PopReg  tablat
  tblwt*-
  return
endcode
```

//

```
code RamPtr! ( wRamAddr -- )
  PopReg  <@" RamPtr"> + 1
  Popreg  <@" RamPtr"> + 0
  return
endcode
```

//

```
code RamPtr@ ( -- wRamAddr )
  Pushreg <@" RamPtr"> + 0
  PushReg <@" RamPtr"> + 1
  return
endcode
```

//

```
macro RamPtr++ ( ) ( -- )
  incf    <@" RamPtr"> + 0      ; // RamPtr++ ( -- )
endMacro
```

//

```
macro RamPtr-- ( ) ( -- )
  decf    <@" RamPtr"> + 0      ; // RamPtr-- ( -- )
endMacro
```

//

```
: Ram@ ( -- b )
  RamPtr@ @
;
```

//

```
: []Ram@ ( bIndex -- b )
  RamPtr@ a+@
;
```

//

```
: Ram@++ ( -- b )
  Ram@ RamPtr++
;
```

//

```
: ++Ram@ ( -- b )
  RamPtr++ Ram@
;
```

//

```
: Ram@-- ( -- b )
  Ram@ RamPtr--
;
```

//

```
: Ram! ( b -- )
  RamPtr@ !
;
```

//

```
: Ram!++ ( b -- )
  Ram! RamPtr++
;
```

//

```
: ++Ram! ( b -- )
  RamPtr++ Ram!
;
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
: Ram!-- ( b -- )  
Ram! RamPtr--  
;
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 1-! ( wRamAddr -- )  
PopReg fsr2h  
PopReg fsr2l  
decf indf2, f  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 1+! ( wRamAddr -- )  
PopReg fsr2h  
PopReg fsr2l  
incf indf2, f  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code d! ( w wRamAddr -- )  
PopReg fsr2h  
PopReg fsr2l  
PopReg tmp1  
PopReg postinc2  
movff tmp1, indf2  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code And! ( b wRamAddr -- )  
PopReg fsr2h  
PopReg fsr2l  
andwf indf2  
Drop  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Or! ( b wRamAddr -- )  
PopReg fsr2h  
PopReg fsr2l  
iorwf indf2  
Drop  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Xor! ( b wRamAddr -- )  
PopReg fsr2h  
PopReg fsr2l  
xorwf indf2  
Drop  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code Dup! ( b wRamAddr -- b )  
PopReg fsr2h  
PopReg fsr2l  
movwf indf2  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code True! ( wRamAddr -- )  
PopReg fsr2h  
movwf fsr2l  
movlw True  
movwf indf2  
Drop  
return  
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code False! ( wRamAddr -- )  
PopReg fsr2h  
movwf fsr2l  
movlw False
```

```
movwf indf2
Drop
return
endcode
```

```
////////////////////////////////////////////////////////////////////
alias False! 0! ( wRamAddr -- ) // False! also known as 0! from now on
alias True! ff! ( wRamAddr -- ) // True! also known as ff! from now on
```

```
////////////////////////////////////////////////////////////////////
code ^Xor ( wRamAddr1 wRamAddr2 -- )
```

```
; // Indirect 8 bits xor of wRamAddr1^with wRamAddr^2
; // result stored at wRamAddr1
```

```
movff fsr1l, tmp1
movff fsr1h, tmp2
PopReg fsr2h
PopReg fsr2l
PopReg fsr1h
movwf fsr1l
```

```
movf indf2, w
xorwf indf1, f
```

```
movff tmp1, fsr1l
movff tmp2, fsr1h
```

```
Drop
return
```

```
endcode
```

```
////////////////////////////////////////////////////////////////////
code ^Shl ( wRamAddr -- ) // Indirect left shift
```

```
PopReg fsr2h
PopReg fsr2l
rlcf indf2
return
```

```
endcode
```

```
////////////////////////////////////////////////////////////////////
code ^Shr ( wRamAddr -- ) // Indirect right shift
```

```
PopReg fsr2h
PopReg fsr2l
rrcf indf2
return
```

```
endcode
```

```
////////////////////////////////////////////////////////////////////
code ^rl ( wRamAddr -- ) // Indirect left rotate
```

```
PopReg fsr2h
PopReg fsr2l
rlncf indf2
return
```

```
endcode
```

```
////////////////////////////////////////////////////////////////////
code ^rr ( wRamAddr -- ) // Indirect right rotate
```

```
PopReg fsr2h
PopReg fsr2l
rrncf indf2
return
```

```
endcode
```

```
////////////////////////////////////////////////////////////////////
code Xor32 ( q q -- q )
```

```
PopReg tmp1
PopReg tmp2
PopReg tmp3
PopReg tmp4
```

```
xorwf tmp1, f
Drop
xorwf tmp2, f
Drop
xorwf tmp3, f
Drop
xorwf tmp4, w
```

```
PushReg tmp3
PushReg tmp2
PushReg tmp1
```

```
return
```

```
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code <<32 ( q n -- q )
```

```
PopReg tmp1
PopReg tmp2
PopReg tmp3
PopReg tmp4
```

```
shl32lp
```

```
bcf Status, C
rlcf wreg
rlcf tmp4
rlcf tmp3
rlcf tmp2
```

```
decfsz tmp1
bra shl32lp
```

```
PushReg tmp4
PushReg tmp3
PushReg tmp2
```

```
return
```

```
endCode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code >>32 ( q n -- q )
```

```
PopReg tmp1
PopReg tmp2
PopReg tmp3
PopReg tmp4
```

```
shr32lp
```

```
bcf Status, C
rrcf tmp2
rrcf tmp3
rrcf tmp4
rrcf wreg
```

```
decfsz tmp1
bra shr32lp
```

```
PushReg tmp4
PushReg tmp3
PushReg tmp2
```

```
return
```

```
endCode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code qnegate ( q -- -q )
```

```
PopReg tmp4
PopReg tmp3
PopReg tmp2
sublw D'0'
movwf tmp1
```

```
movlw 0
subwfb tmp2, f
subwfb tmp3, f
subwfb tmp4, f
```

```
movf tmp1, w
PushReg tmp2
PushReg tmp3
PushReg tmp4
```

```
return
```

```
endCode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
: d>r ( d -- )
```

```
swap
>r >r
```

```
;
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
: dr> ( d -- )
```

```
r> r>
swap
```

```
;
```

```

// ////////////////////////////////////////

: q>r ( q -- )
  dSwap
  d>r d>r
;

// ////////////////////////////////////////

: qr> ( q -- )
  dr> dr>
  dSwap
;

// ////////////////////////////////////////

code Popq1 ( q -- )
  PopReg tmp4           ; // Pop q
  PopReg tmp3
  PopReg tmp2
  PopReg tmp1

  return
endcode

// ////////////////////////////////////////

code Pushq1 ( -- q )
  PushReg tmp1
  PushReg tmp2
  PushReg tmp3
  PushReg tmp4

  return
endcode

// ////////////////////////////////////////

code Popq2 ( q -- )

  PopReg <@" Quotient"> + 3   ; // Pop q
  PopReg <@" Quotient"> + 2
  PopReg <@" Quotient"> + 1
  PopReg <@" Quotient"> + 0

  return
endcode

// ////////////////////////////////////////

code Pushq2 ( -- q )

  PushReg <@" Quotient"> + 0
  PushReg <@" Quotient"> + 1
  PushReg <@" Quotient"> + 2
  PushReg <@" Quotient"> + 3

  return
endcode

// ////////////////////////////////////////

code Popq3 ( q -- )

  PopReg <@" Remainder"> + 3   ; // Pop q
  PopReg <@" Remainder"> + 2
  PopReg <@" Remainder"> + 1
  PopReg <@" Remainder"> + 0

  return
endcode

// ////////////////////////////////////////

code Pushq3 ( -- q )

  PushReg <@" Remainder"> + 0
  PushReg <@" Remainder"> + 1
  PushReg <@" Remainder"> + 2
  PushReg <@" Remainder"> + 3

  return
endcode

// ////////////////////////////////////////

: 4rot ( q1 q2 q3 -- q2 q3 q1 )

  Popq3
  Popq2
  Popq1

```

```

Pushq2
Pushq3
Pushq1
;

// ////////////////////////////////////////

: 4over ( q1 q2 -- q1 q2 q1 )

Popq2
Popq1

Pushq1
Pushq2
Pushq1
;

// ////////////////////////////////////////

: 4swap ( q1 q2 -- q2 q1 )

Popq2
Popq1
Pushq2
Pushq1
;

// ////////////////////////////////////////

code ^>>32 ( wRamAddr n -- )

; // Indirect 32 bits right shift over n positions

PopReg tmp1 ; // Shift wRamAddr^ right over n positions
PopReg fsr2h ; // let fsr2 point to end of data (highest byte)

shri32lp

addlw 3
movwf fsr2l

bcf Status, C ; // Clear carry

rrcf postdec2, f
rrcf postdec2, f
rrcf postdec2, f
rrcf indf2, f

movf fsr2l, w

decfsz tmp1
bra shri32lp

Drop
return
endcode

// ////////////////////////////////////////

code ^<<32 ( wRamAddr n -- )

; // Indirect 32 bits left shift over n positions

PopReg tmp1 ; // Shift wRamAddr^ left over n positions
PopReg fsr2h ; // let fsr2 point to first (lowest) byte

shli32lp

movwf fsr2l
bcf Status, C ; // Clear carry

rlcf postinc2, f
rlcf postinc2, f
rlcf postinc2, f
rlcf indf2, f

movf fsr2l, w
addlw 3

decfsz tmp1
bra shli32lp

Drop
return
endcode

// ////////////////////////////////////////

code d+ ( wu1 wu2 -- wu3 )

PopReg tmp1 ; // Pop hi2 into tmp1, leaving lo2 in wreg
decf fsr0l ; // point to lo1
addwf postinc0 ; // Add lo bytes (lo3), point to hi1
movf tmp1, w ; // get hi2
addwfc postdec0, w ; // Add hi bytes into w, point to lo3
return
endcode

```

```

// ////////////////////////////////////////
code d- ( wu1 wu2 -- wu3 )

    PopReg    tmp1                ; // Pop hi2 into tmp1, leaving lo2 in wreg
    decf     fsr01                ; // point to lo1
    subwf    postinc0            ; // Sub lo bytes (lo3), point to hi1
    movf     tmp1, w             ; // get hi2
    subwfb   postdec0, w         ; // Sub hi bytes into w, point to lo3
    return
endCode

// ////////////////////////////////////////
// ////////////////////////////////////////

macro 1+ ( ) ( b -- b )
    addlw    D'1'                ; // 1+ ( u1 -- u3 )
endMacro

// ////////////////////////////////////////

macro 2+ ( ) ( b -- b )
    addlw    D'2'                ; // 2+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 3+ ( ) ( b -- b )
    addlw    D'3'                ; // 3+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 4+ ( ) ( b -- b )
    addlw    D'4'                ; // 4+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 5+ ( ) ( b -- b )
    addlw    D'5'                ; // 5+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 6+ ( ) ( b -- b )
    addlw    D'6'                ; // 6+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 7+ ( ) ( b -- b )
    addlw    D'7'                ; // 7+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 8+ ( ) ( b -- b )
    addlw    D'8'                ; // 8+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 9+ ( ) ( b -- b )
    addlw    D'9'                ; // 9+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 10+ ( ) ( b -- b )
    addlw    D'10'               ; // 10+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro '0'+ ( ) ( b -- b )
    addlw    0x30                ; // '0'+ ( u1 -- u2 )
endMacro

// ////////////////////////////////////////

macro 1- ( ) ( b -- b )
    addlw    D'255'              ; // 1- ( u1 -- u2 )
endMacro

```

```

// //////////////////////////////////////
macro 2- ( ) ( b -- b )
    addlw D'254' ; // 2- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 3- ( ) ( b -- b )
    addlw D'253' ; // 3- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 4- ( ) ( b -- b )
    addlw D'252' ; // 4- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 5- ( ) ( b -- b )
    addlw D'251' ; // 5- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 6- ( ) ( b -- b )
    addlw D'250' ; // 6- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 7- ( ) ( b -- b )
    addlw D'249' ; // 7- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 8- ( ) ( b -- b )
    addlw D'248' ; // 8- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 9- ( ) ( b -- b )
    addlw D'247' ; // 9- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro 10- ( ) ( b -- b )
    addlw D'246' ; // 10- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
macro '0'- ( ) ( b -- b )
    addlw 0xD0 ; // '0'- ( u1 -- u2 )
endMacro

// //////////////////////////////////////
code d= ( w1 w2 -- f )
    PopReg tmp1 ; // u4
    PopReg tmp2 ; // u3
    PopReg tmp3 ; // u2

    cpfseq tmp2
    retlw False

    movf tmp3, w
    cpfseq tmp1
    retlw False
    retlw True
endCode

// //////////////////////////////////////
code d> ( wu1 wu2 -- f )
    ; // Return True if N1 (u1,u2) > N2 (u3,u4) (or N2 < N1)
    PopReg tmp3 ; // Pop N2
    PopReg tmp4
    PopReg tmp1 ; // Pop N1
    movwf tmp2

    cpfls16 tmp4, tmp2 ; // S/ N2 < N1
    retlw False ; // Assumption wrong, return false
    retlw True

```



```
alias 0d ud>uq ( wu -- qu ) // unsigned
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code d>q ( w -- q ) // with sign extension
```

```
PushLit D'0'  
btfsc   indf0, 7  
movlw   D'255'  
PushLit D'0'  
btfsc   indf0, 7  
movlw   D'255'  
return
```

```
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code d* ( u1 u2 -- w ) // 8*8->16
```

```
PopReg  tmp1  
mulwf   tmp1  
movf    prodl, w  
PushReg prodh
```

```
return
```

```
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code q* ( w1 w2 -- q ) // 16*16->32
```

```
PopReg  tmp4                ; // b  
PopReg  tmp3                ; // a  
PopReg  tmp2  
movwf   tmp1
```

```
movlw   17  
movwf   tmp5
```

```
clrfl6  <@" Divisor">
```

```
bcf     Status, C
```

```
qstar_1
```

```
rrcf    <@" Divisor"> + 1, f  
rrcf    <@" Divisor"> + 0, f  
rrcf    tmp2, f  
rrcf    tmp1, f  
bnc     qstar_2                ; // B/ Carry clear, skip add
```

```
; Carry set, add
```

```
movf    tmp3, w  
addwf   <@" Divisor"> + 0, f  
movf    tmp4, w  
addwfc  <@" Divisor"> + 1, f
```

```
qstar_2
```

```
decfsz  tmp5, f  
bra     qstar_1
```

```
movf    tmp1, w  
PushReg tmp2  
PushReg <@" Divisor"> + 0  
PushReg <@" Divisor"> + 1
```

```
return
```

```
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code *_8_32 ( q1 u -- q2 ) // 8*32->32
```

```
PopReg  tmp1                ; // Get byte arg  
PopReg  tmp5                ; // Get long arg  
PopReg  tmp4  
PopReg  tmp3  
movwf   tmp2
```

```
movlw   9                    ; // 9 Shifts  
movwf   <@" Quotient">
```

```
clrfl32 <@" Divisor">
```

```
bcf     Status, C
```

```
tstar_1
```

```
; // Ignore byte 5  
rrcf    <@" Divisor"> + 3, f  
rrcf    <@" Divisor"> + 2, f  
rrcf    <@" Divisor"> + 1, f  
rrcf    <@" Divisor"> + 0, f  
rrcf    tmp1, f  
bnc     tstar_2                ; // B/ Carry clear, skip add
```

```

; // Carry set, add, ignoring 5th byte

movf    tmp2, w
addwf   <@" Divisor"> + 0, f
movf    tmp3, w
addwfc  <@" Divisor"> + 1, f
movf    tmp4, w
addwfc  <@" Divisor"> + 2, f
movf    tmp5, w
addwfc  <@" Divisor"> + 3, f

tstar_2

decfsz  <@" Quotient">, f
bra     tstar_1

movf    tmp1, w
PushReg <@" Divisor"> + 0
PushReg <@" Divisor"> + 1
PushReg <@" Divisor"> + 2

Return
endcode

// ////////////////////////////////////////
code /mod16 ( wDividend wDivisor -- wQuotient wRemainder )

; // ////////////////////////////////////////
; // Function      : udiv16( wDividend, wDivisor)
; // Input         : wDividend : 16 bits unsigned
; //              : wDivisor  : 16 bits unsigned
; // Output        : Quotient, Remainder
; // Description   : 16 / 16 -> 16 bits unsigned divide with remainder
; // Remarks      : Quotient := wDividend Div wDivisor
; //              : Remainder := wDividend Mod wDivisor
; // ////////////////////////////////////////
; //
; // From Zilog Z8 Family Design Handbook, June 1988
; //
; // A programmers guide to the Z8 microcomputer,
; // Z8 subroutine library, Application note
; // page 211.
; //
; // This is a 16 / 16 -> 16 + 16 udiv routine
; //
; // ////////////////////////////////////////

; // Pop arguments from stack to helper variables

PopReg  <@" Divisor"> + 1 ; // pop ( ul -- )
PopReg  <@" Divisor"> + 0
PopReg  <@" Quotient"> + 1
movwf   <@" Quotient"> + 0

clrfl6  <@" Remainder">

movlw   16 ; // ld  d16_LEN, #16      !LOOP COUNTER!
movwf   tmp2
bcf     Status, C ; // rcf              !carry = 0!

d1p_16

shlcl6  <@" Quotient"> ; // rlc  quot_lo/hi
shlcl6  <@" Remainder"> ; // rlc  rem_lo/hi
bc      subt_16 ; // jr   c, subt_16

movf    <@" Remainder"> + 1, w ; // cp  dvsr_hi, rem_hi
cpfsgt  <@" Divisor"> + 1 ; // jr  ugt, skip_16
bra     udiv_16_0 ; // B/ f <= W -> dvsr_hi <= rem_hi
bra     skip_16 ; // B/ f > W -> dvsr_hi > rem_hi

udiv_16_0 ; // dvsr_hi <= rem_hi

cpfseq  <@" Divisor"> + 1 ; // jr  ult, subt_16
bra     subt_16 ; // B/ dvsr_hi < rem_h

; // dvsr_hi = rem_hi

movf    <@" Remainder">, w ; // cp  dvsr_lo, rem_lo
cpfsgt  <@" Divisor"> ; // jr  ugt, skip_16
bra     subt_16 ; // B/ f <= W -> dvsr_lo <= rem_lo
bra     skip_16 ; // B/ f > W -> dvsr_lo > rem_lo

subt_16

movf    <@" Divisor">, w ; // sub  rem_lo, dvsr_lo
subwfb  <@" Remainder">, f ; // sub  rem_hi, dvsr_hi
movf    <@" Divisor"> + 1, w ; // sub  rem_hi, dvsr_hi
subwfb  <@" Remainder"> + 1, f ; // sub  rem_hi, dvsr_hi
bsf     Status, C ; // scf

skip_16

decfsz  tmp2, f ; // djnz d16_LEN, d1p_16
bra     d1p_16 ; //
; // !no flags affected!

shlcl6  <@" Quotient"> ; // rlc  quot_lo/hi

```

```

; // Copy results to stack
movf   <@" Quotient"> + 0, w
PushReg <@" Quotient"> + 1

PushReg <@" Remainder"> + 0
PushReg <@" Remainder"> + 1

return
endcode

// ////////////////////////////////////////
: /16 ( wDividend wDivisor -- wQuotient )
/mod16 2Drop
;

// ////////////////////////////////////////
code /mod32 ( qDividend qDivisor -- qQuotient qRemainder )

; // ////////////////////////////////////////
; // Function      : udiv32( qDividend, wDivisor)
; // Input         : qDividend : 32 bits unsigned
; //               : wDivisor  : 16 bits unsigned
; // Output        : Quotient, Remainder
; // Description   : 32 / 32 -> 32 + 32 bits unsigned divide with remainder
; // Remarks       : Quoatient := qDividend Div qDivisor
; //               : Remainder := qDividend Mod qDivisor
; // ////////////////////////////////////////
; //
; // Adapted from 16 / 16 -> 16 + 16 algorithm above
; //
; // This is a 32 / 32 -> 32 + 32 udiv routine
; //
; // ////////////////////////////////////////

; // Pop arguments from stack to helper variables

PopReg <@" Divisor"> + 3 ; // pop Divisor
PopReg <@" Divisor"> + 2
PopReg <@" Divisor"> + 1
PopReg <@" Divisor"> + 0
PopReg <@" Quotient"> + 3 ; // Pop Dividend
PopReg <@" Quotient"> + 2
PopReg <@" Quotient"> + 1
movwf  <@" Quotient"> + 0

clrf32 <@" Remainder">

movlw  32 ; // ld  d16_LEN, #32 !LOOP COUNTER!
movwf tmp2
bcf   Status, C ; // rcf !carry = 0!

dlp_32

shlc32 <@" Quotient"> ; // rlc  qout_0..3
shlc32 <@" Remainder"> ; // rlc  rem_0..3
bc     subt_32 ; // jr   c, subt_32

movf   <@" Remainder"> + 3, w ; // cp  dvsr_3, rem_3
cpfsgt <@" Divisor"> + 3 ; // jr  ugt, skp_32
bra    udiv_32_3 ; // B/ f <= W -> dvsr_3 <= rem_3
bra    skp_32 ; // B/ f > W -> dvsr_3 > rem_3

udiv_32_3 ; // dvsr_3 <= rem_3

cpfseq <@" Divisor"> + 3 ; // jr  ult, subt_32
bra    subt_32 ; // B/ dvsr_3 < rem_3

; // dvsr_3 = rem_3

movf   <@" Remainder"> + 2, w ; // cp  dvsr_2, rem_2
cpfsgt <@" Divisor"> + 2 ; // jr  ugt, skp_32
bra    udiv_32_2 ; // B/ f <= W -> dvsr_2 <= rem_2
bra    skp_32 ; // B/ f > W -> dvsr_2 > rem_2

udiv_32_2 ; // dvsr_2 <= rem_2

cpfseq <@" Divisor"> + 2 ; // jr  ult, subt_32
bra    subt_32 ; // B/ dvsr_2 < rem_2

; // dvsr_2 = rem_2

movf   <@" Remainder"> + 1, w ; // cp  dvsr_1, rem_1
cpfsgt <@" Divisor"> + 1 ; // jr  ugt, skp_32
bra    udiv_32_1 ; // B/ f <= W -> dvsr_1 <= rem_1
bra    skp_32 ; // B/ f > W -> dvsr_1 > rem_1

udiv_32_1 ; // dvsr_1 <= rem_1

cpfseq <@" Divisor"> + 1 ; // jr  ult, subt_32
bra    subt_32 ; // B/ dvsr_2 < rem_2

; // dvsr_1 = rem_1

movf   <@" Remainder">, w ; // cp  dvsr_lo, rem_lo
cpfsgt <@" Divisor"> ; // jr  ugt, skp_32

```

```

bra      sub_32          ; // B/ f <= W -> dvsr_lo <= rem_lo
bra      skp_32         ; // B/ f > W -> dvsr_lo > rem_lo

subt_32

movf     <@" Divisor">   , w ; // sub rem_0, dvsr_0
subwf   <@" Remainder"> , f
movf     <@" Divisor">   + 1, w ; // sub rem_1, dvsr_1
subwfb  <@" Remainder"> + 1, f
movf     <@" Divisor">   + 2, w ; // sub rem_2, dvsr_2
subwfb  <@" Remainder"> + 2, f
movf     <@" Divisor">   + 3, w ; // sub rem_3, dvsr_3
subwfb  <@" Remainder"> + 3, f
bsf     Status, C       ; // scf

skp_32

decfsz  tmp2, f        ; // djnz d32_LEN, dlp_32
bra      dlp_32        ; // !no flags affected!

shlc32  <@" Quotient"> ; // rlc quot_0..3

; // Copy results to stack

movf     <@" Quotient"> + 0, w
PushReg <@" Quotient"> + 1
PushReg <@" Quotient"> + 2
PushReg <@" Quotient"> + 3

PushReg <@" Remainder"> + 0
PushReg <@" Remainder"> + 1
PushReg <@" Remainder"> + 2
PushReg <@" Remainder"> + 3

return
endcode

```

```

// ////////////////////////////////////////

: /32 ( qDividend qDivisor -- qQuotient )
/mod32 4Drop
;

// ////////////////////////////////////////

: d0 ( -- w )
0.
;

// ////////////////////////////////////////

: d0> ( w -- f )
d0 d>
;

// ////////////////////////////////////////

: d0= ( w -- f )
or 0=
;

// ////////////////////////////////////////

: t@ ( wRamAddr -- w b )
aDup      // ( wAdr wAdr )
d@        // ( wAdr wDataLo )
dSwap     // ( wDataLo wAdr )
2a+      // ( wDataLo wAdr+2 )
@         // ( wDataLo bDataHi )
;

// ////////////////////////////////////////

: t! ( w b wRamAddr -- )
+rot      // ( wValLo wAdr bValHi )
2Pick 2Pick // ( wValLo wAdr bValHi wAdr )
2a+      // ( wValLo wAdr bValHi wAdr+2 )
!        // ( wValLo wAdr )
d!
;

// ////////////////////////////////////////

: q@ ( wRamAddr -- q )
aDup      // ( wAdr wAdr )
d@        // ( wAdr wDataLo )
dSwap     // ( wDataLo wAdr )
2a+      // ( wDataLo wAdr+2 )
d@        // ( wDataLo wDataHi )
;

// ////////////////////////////////////////

```

```

: q! ( q wRamAddr -- )
dSwap // ( wValLo wAdr wValHi )
aOver // ( wValLo wAdr wValHi wAdr )
2a+
d! // ( wValLo wAdr )
d!
;

// //////////////////////////////////////

: 2q@ ( wRamAddr -- q q )
Dup RamPtr!
q@
RamPtr@ 4a+
q@
;

// //////////////////////////////////////

: @Compare! ( fn wRamAddr -- fc )

// gets a flag from wRamAddr, if it's different from the fn parameter
// the fc parameter true is returned, otherwise false is returned for fc.
// As a side effect : fn is stored into wRamAddr.

// Checked
adup@ // ( fn wAdr fo )
3 pick // ( fn wAdr fo fn )
xor // ( fn wAdr fc )
>r // ( fn wAdr )
! // ( )
r> // ( fc )
;

// //////////////////////////////////////

: @0= ( wRamAddr -- f )

// fetch from wRamAddr, if it's 0 return true

@ 0=
;

// //////////////////////////////////////

: 0> ( u -- f )
0 >
;

// //////////////////////////////////////

: @0> ( wRamAddr -- f )

// fetch from wRamAddr, if it's > 0 return true

@ 0>
;

// //////////////////////////////////////

macro DropTrue ( ) ( b -- f )
movlw True ; // Drop True ( b -- f )
endMacro

// //////////////////////////////////////

macro DropFalse ( ) ( b -- f )
movlw False ; // Drop False ( b -- f )
endMacro

// //////////////////////////////////////

: @0! ( wRamAddr -- b )

// Fetch b from wRamAddr, set wRamAddr^ to 0 - interrupt safe.

adup // ( al ah al ah )
Disable
@ // ( al ah b )
-rot // ( b al ah )
0! // ( b )
Enable
;

// //////////////////////////////////////

alias @0! @False!

// //////////////////////////////////////

```

```

: @ff! ( wRamAddr -- b )

// Fetch b from wRamAddr, set wRamAddr^ to $ff - interrupt safe.

adup                // ( al ah al ah )
Disable
@                   // ( al ah b )
-rot                // ( b al ah )
ff!                 // ( b )
Enable
;

// //////////////////////////////////////

alias @ff! @True!

// //////////////////////////////////////

macro 2/ ( ) ( b -- b )
; // 2/ ( u1 -- u2 )

    rrcf    wreg, f
    andlw   0x7f
endMacro

// //////////////////////////////////////

code 4/ ( u1 -- u2 )
    rrcf    wreg, f
    rrcf    wreg, f
    andlw   0x3f

    return
endCode

// //////////////////////////////////////

code 8/ ( u1 -- u2 )
    rrcf    wreg, f
    rrcf    wreg, f
    rrcf    wreg, f
    andlw   0x1f

    return
endCode

// //////////////////////////////////////

macro 16/ ( ) ( b -- b )
; // 16/ ( u1 -- u2 )

    swapf   wreg, f
    andlw   0x0f
endMacro

// //////////////////////////////////////

code 32/ ( u1 -- u2 )
    swapf   wreg, f
    rrcf    wreg, f
    andlw   0x07

    return
endCode

// //////////////////////////////////////

code 64/ ( u1 -- u2 )
    swapf   wreg, f
    rrcf    wreg, f
    rrcf    wreg, f
    andlw   0x03

    return
endCode

// //////////////////////////////////////

code 128/ ( u1 -- u2 )
    swapf   wreg, f
    rrcf    wreg, f
    rrcf    wreg, f
    rrcf    wreg, f
    andlw   0x01

    return
endCode

// //////////////////////////////////////

macro 2* ( ) ( b -- b )
; // 2* ( u1 -- u2 )

```

```
    rlcw    wreg, f
    andlw  0xfe
endMacro
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 3* ( u1 -- u2 )
    movwf  tmp1
    rlcw   wreg, f
    andlw  0xfe
    addwf  tmp1, w
    return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 4* ( u1 -- u2 )
    rlcw   wreg, f
    rlcw   wreg, f
    andlw  0xfc

    return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 8* ( u1 -- u2 )
    rlcw   wreg, f
    rlcw   wreg, f
    rlcw   wreg, f
    andlw  0xf8

    return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
macro 16* ( ) ( b -- b ) ; // 16* ( u1 -- u2 )
    swapf  wreg, f
    andlw  0xf0
endMacro
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 32* ( u1 -- u2 )
    swapf  wreg, f
    rlcw   wreg, f
    andlw  0xe0

    return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 64* ( u1 -- u2 )
    swapf  wreg, f
    rlcw   wreg, f
    rlcw   wreg, f
    andlw  0xc0

    return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code 128* ( u1 -- u2 )
    swapf  wreg, f
    rlcw   wreg, f
    rlcw   wreg, f
    rlcw   wreg, f
    andlw  0x80

    return
endcode
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
macro SwapDrop ( ) ( b b -- b ) ; // SwapDrop ( b1 b2 -- b2 )
    SwapDrop
endMacro
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
code sp@ ( -- w )
    PushReg fsr0l
    PushReg fsr0h
    return
endcode
```



```

// ////////////////////////////////////////
: d>= ( w1 w2 -- f )
d< Not
;

// ////////////////////////////////////////
: d<= ( w1 w2 -- f )
d> Not
;

// ////////////////////////////////////////
: d0< ( d -- f ) // True when d < 0
SwapDrop $80 And 0<>
;

// ////////////////////////////////////////
code q+ ( uq1 uq2 -- uq3 )

PopReg <@" Quotient"> + 3
PopReg <@" Quotient"> + 2
PopReg <@" Quotient"> + 1
PopReg <@" Quotient"> + 0

PopReg <@" Remainder"> + 3
PopReg <@" Remainder"> + 2
PopReg <@" Remainder"> + 1
movwf <@" Remainder"> + 0

; // Now calculate Remainder + Quotient -> Remainder

movf <@" Quotient"> + 0, w
addwf <@" Remainder"> + 0, f
movf <@" Quotient"> + 1, w
addwfc <@" Remainder"> + 1, f
movf <@" Quotient"> + 2, w
addwfc <@" Remainder"> + 2, f
movf <@" Quotient"> + 3, w
addwfc <@" Remainder"> + 3, f

; // Done

movf <@" Remainder"> + 0, w
PushReg <@" Remainder"> + 1
PushReg <@" Remainder"> + 2
PushReg <@" Remainder"> + 3

return
endcode

// ////////////////////////////////////////
code q- ( uq1 uq2 -- uq3 )

PopReg <@" Quotient"> + 3
PopReg <@" Quotient"> + 2
PopReg <@" Quotient"> + 1
PopReg <@" Quotient"> + 0

PopReg <@" Remainder"> + 3
PopReg <@" Remainder"> + 2
PopReg <@" Remainder"> + 1
movwf <@" Remainder"> + 0

; // Now calculate Remainder - Quotient -> Remainder

DoQMin

movf <@" Quotient"> + 0, w
subwf <@" Remainder"> + 0, f
movf <@" Quotient"> + 1, w
subwfb <@" Remainder"> + 1, f
movf <@" Quotient"> + 2, w
subwfb <@" Remainder"> + 2, f
movf <@" Quotient"> + 3, w
subwfb <@" Remainder"> + 3, f

; // Done

movf <@" Remainder"> + 0, w
PushReg <@" Remainder"> + 1
PushReg <@" Remainder"> + 2
PushReg <@" Remainder"> + 3

return
endcode

// ////////////////////////////////////////
code revq- ( uq1 uq2 -- uq3 ) // uq2 - uq1 -> uq2, reversed q-

```

```

PopReg <@" Remainder"> + 3
PopReg <@" Remainder"> + 2
PopReg <@" Remainder"> + 1
PopReg <@" Remainder"> + 0

PopReg <@" Quotient"> + 3
PopReg <@" Quotient"> + 2
PopReg <@" Quotient"> + 1
movwf <@" Quotient"> + 0

bra DoQMin
endcode

// //////////////////////////////////////

: q-$80And>r3Dropr> ( q1 q2 -- b ) #// Helper for q comparison words
q- $80 And // q1 - q2 :: b <> 0 when subtraction result negative (bit 31 set)
>r // b to loop stack
3Drop // Drop three least significant subtraction result bytes
r> // Get b from loop stack
;

// //////////////////////////////////////

: q< ( q1 q2 -- f ) #// True when q1 < q2
q-$80And>r3Dropr> // bit 31 set -> q1 - q2 < 0 -> q1 < q2
0<> // true when bit 31 set
;

// //////////////////////////////////////

: q>= ( q1 q2 -- f ) #// True when q1 >= q2
q-$80And>r3Dropr> // bit 31 cleared -> q1 - q2 >= 0 -> q1 >= q2
0= // True when bit 31 cleared
;

// //////////////////////////////////////

: q> ( q1 q2 -- f ) #// True when q1 > q2
4Swap q<
;

// //////////////////////////////////////

: q<= ( q1 q2 -- f ) #// True when q1 <= q2
4Swap q>=
;

// //////////////////////////////////////

: q0= ( q -- f ) #// True when q = 0
or or or 0=
;

// //////////////////////////////////////

: q= ( q1 q2 -- f ) #// True when q1 = q2
q- q0=
;

// //////////////////////////////////////

: q0<> ( q -- f ) #// True when q <> 0
or or or 0<>
;

// //////////////////////////////////////

: 4pickAnd>r ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 | -- b ) #// Helper for quad operations
4Pick And >r
;

// //////////////////////////////////////

: 4pickOr>r ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 | -- b ) #// Helper for quad operations
4Pick Or >r
;

// //////////////////////////////////////

: 4pickXor>r ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 | -- b ) #// Helper for quad operations
4Pick Xor >r
;

// //////////////////////////////////////

: qAnd ( qul qu2 -- qu3 ) #// Quad value bitwise and
4PickAnd>r
4PickAnd>r
4PickAnd>r
4PickAnd>r
4Drop

```

```

qr>
;

// ////////////////////////////////////////

: qOr ( qu1 qu2 -- qu3 ) #// Quad value bitwise or
4PickOr>r
4PickOr>r
4PickOr>r
4PickOr>r
4Drop
qr>
;

// ////////////////////////////////////////

: qXor ( qu1 qu2 -- qu3 ) #// Quad value bitwise xor
4PickXor>r
4PickXor>r
4PickXor>r
4PickXor>r
4Drop
qr>
;

// ////////////////////////////////////////

: qNot ( qu1 -- qu2 ) #// Quad value bitwise invert
-1q
qXor
;

// ////////////////////////////////////////

: qNotAnd ( qu1 qu2 -- qu3 ) #// bitwise: ! qu2 and qu1
qNot
qAnd
;

// ////////////////////////////////////////

: qNotAnd0<> ( qu1 qu2 -- f ) // true when bitwise ! qu1 and qu2 <> 0
qNotAnd
q0<>
;

// ////////////////////////////////////////

macro FillUpWithRandom ( ) ( -- )

; // Fill out program space with random numbers until
; // a modulo <@" FlashPageSize"> boundary is hit.

    variable Seed = 12345

Random macro
Seed = ( Seed * 1103515245) + 12345
    dw    ( Seed >> 5) & 0xffff
endm

while $ % <@" FlashPageSize"> != 0
    Random
endw

endMacro

// ////////////////////////////////////////
// ////////////////////////////////////////

only forth definitions

{

    // Some meta utility functions

: Build ( -- )
    Initialization           // Expand initialization macro
    GenerateCode             // Generate target code
    FillupWithRandom         // Required by boot loader software
    Finalization           // Expand the finalization macro
;

: .info ( <name> <value> -- )    #// print name = value to console
+ message
;

: .infoTime ( <name> <value> -- ) #// print name = value as time to console
unix>str + message
;

: .CompileTime ( -- )           #// Type CompileTime = value to console
" CompileTime : "
CompileTime

```

```

.infoTime
;
: PushIdLoc ( c n -- )          /// Push back " idloc n 'c' " into the input stream

>$ " " +                        /// ( c " n " )
swap " ' "                      /// ( " n " c " ' " )
swap + " ' "                    /// ( " n " " 'c' " ' " )
+ +                              /// ( " n 'c' " )
" idloc "                        /// ( " n 'c' " " idloc " )
swap +                            /// ( " idloc n 'c' " )
space untoken                    /// ( ) // pushes back " idloc n 'c' " into input stream
;

: SetIdLocs ( <string-token> -- )

/// Use first eight chars of string to program the processor ID locations
/// the string must be at least 8 characters long
/// After execution the following will have been pushed into the input stream :
///
/// idloc 1 's[0]' idloc 2 's[1]' idloc 3 's[2]' idloc 4 's[3]' idloc 5 's[4]' idloc 6 's[5]' idloc 7 's[8]'

base@ >r Decimal                /// Work decimal remeber old base
8 0                              /// Use first eight characters of input string
do
  dup                            /// ( $ $ )
  7 i - $[]                      /// ( $ c ) Get (7 - i)th character, working back to front
  8 i -                            /// ( $ c n ) Push the index of the id location to use
  PushIdLoc                       /// ( $ ) Generate an IdLoc entry in the input stream
loop
drop                              /// ( )
r> base!                          /// restore old base
;
}

////////////////////////////////////

endModule

```