

```

; **** ***** ****
; **** ****
; **** Code generated by X4th compiler, do not edit ****
; **** ****
; **** Generated on : 2007-06-10 19:58:03 ****
; **** by X4th : 2007.6.8.1 ****
; **** (C) Copyright 1999 .. 2007, Blue Hell. ****
; **** ****
; **** ***** ****

```

```
#define _18f452_
```

```

TITLE      glossary
LIST      F=INHX32
; // Set initial error level 0 == all errors and warnings active
errorlevel 0

; // Set number of lines per page, 0 == all lines on one page
LIST      n=0

; // Set the assembler default radix HEX or DEC
RADIX     DEC

; // Expand macro listings
EXPAND

; // Eject a page now
PAGE

; // Define processor to be used and include the processor specifics

```

```

#ifdef _18c242_
PROCESSOR 18c242
#include  ".\lib\18c242.inc"
#endif
#ifdef _18c252_
PROCESSOR 18c252
#include  ".\lib\18c252.inc"
#endif
#ifdef _18c442_
PROCESSOR 18c442
#include  ".\lib\18c442.inc"
#endif
#ifdef _18c452_
PROCESSOR 18c452
#include  ".\lib\18c452.inc"
#endif
#ifdef _18f242_
PROCESSOR 18f242
#include  ".\lib\18f242.inc"
#endif
#ifdef _18f252_
PROCESSOR 18f252
#include  ".\lib\18f252.inc"
#endif
#ifdef _18f442_
PROCESSOR 18f442
#include  ".\lib\18f442.inc"
#endif
#ifdef _18f452_
PROCESSOR 18f452
#include  ".\lib\18f452.inc"
#endif

```

```

; // //////////////////////////////////////
; // 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 ( - 0x0f)

```

```

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

```

```

; // ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; // :: Start of code area ::::::::::::::::::::::::::::::::::::::::::::
; // Boot loader stuff

; // Fill in start and interrupt vectors, interrupts just jump through to the
; // APPLICATION. Reset must check eeprom and decide whether to jump into the
; // APPLICATION or reprogram the flash from an image present in EEPROM.
; //
; // The EEPROM has been changed to a size of 64 KByte (it was 64 Kbit) so it can
; // hold a new software image. The EEPROM layout has been changed as well the
; // application data starts at an offset 0x400 so that the first KByte can be used
; // for the loader software.
; //
; // EEPROM layout :
; //
; // $0000 .. $00ff : Bootloader memory
; //
; // $0000 : byte : loader_state
; //
; // $00 - Loading flash image from host, not a valid image yet
; // $01 - Valid image in EEPROM, must program FLASH
; // $02 - Programmed new image into flash must contact host ASAP
; // the APPLICATION should check this state and act on it
; // $ff - FLASH programmed, host contacted, normal operation.
; //
; // all other values - run image in flash (normal operation).
; //
; // the APPLICATION should check for these cases and correct the value to $ff
; //
; // $0001 : byte : inverse value of loader_state (1's complement).
; //
; // when this byte is not equal to the 1's complement of
; // loader_state : run image in flash (normal operation).
; //
; // $0002 : word : image_length
; //
; // valid incremental length when loader_state is $00.
; // 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 , CPin_SCL ; // Set SCL high
bcf I2C_TRIS, CPin_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
rrcf CRC_16 + 1, f ; // rrc16
rrcf CRC_16 , f

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

; // Xor OldCrc with shifted polynomial
movlw (0xa001 >> 0) & 0xff ; // xor116
xorwf CRC_16, f
movlw (0xa001 >> 8) & 0xff
xorwf CRC_16 + 1, f

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, CPin_SCL ; // Clock low
bcf I2C_LAT, CPin_SDA ; // Output bit 0 (0)
btfsc tmp1, 7
bsf I2C_LAT, CPin_SDA ; // Output bit 0 (1)

bsf I2C_LAT, CPin_SCL ; // Clock to high

rlncf 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 , CPin_SCL ; // Return SCL to low
bcf      I2C_LAT , CPin_SDA ; // Make SDA low
bsf     I2C_TRIS, CPin_SDA ; // Put SDA line in input state
bsf     I2C_LAT , CPin_SCL ; // Clock high

setf     EE_ACK                ; // Assume ACK
btfsc   I2C_PORT, CPin_SDA ; // Read SDA pin / S/ on ack
clrf    EE_ACK                ; // No ack

bcf      I2C_LAT , CPin_SCL ; // Return SCL to low
bcf      I2C_TRIS, CPin_SDA ; // Put SDA line in output state

return

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

i2cStart
    bsf     I2C_LAT, CPin_SDA
    bsf     I2C_LAT, CPin_SCL                ; // SDA goes low during SCL high

i2cToggle
    nop
    nop
    btg     I2C_LAT, CPin_SDA

return

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

i2cStop
    bcf     I2C_LAT, CPin_SDA
    bsf     I2C_LAT, CPin_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, CPin_SDA ; // Put SDA line in input state

    movlw   8                ; // 8 bits of data
    movwf   tmp2

i2cRxLp
    rlnf    tmp3, f                ; // Shift data to buffer

    ; // Bit in

    bsf     I2C_LAT, CPin_SCL ; // Clock high
    bcf     tmp3, 0                ; // Assume data bit is 0
    btfsc   I2C_PORT, CPin_SDA ; // Read SDA pin
    bsf     tmp3, 0                ; // Data bit is 1 instead

    bcf     I2C_LAT, CPin_SCL ; // Return SCL to low

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

    bsf     I2C_LAT, CPin_SDA ; // Send NAK
    bcf     I2C_TRIS, CPin_SDA ; // Put SDA line in output state
    bsf     I2C_LAT, CPin_SCL ; // Clock high
    movf    tmp3, w                ; // Data into wreg
    bcf     I2C_LAT, CPin_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   0xal                ; // Device address + read

```

```

    rcall    i2cTxByte
    rcall    i2cRxByte           ; // Read byte into
    bra      i2cStop            ; // Issue bus STOP

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

    incf    EE_ADDR, f           ; // incf16
    btfsc   Status, C
    incf    EE_ADDR + 1, f

    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
movlb    0x00              ; // Select the default rambank

rcall    SetupRegs         ; // Initialize some registers

bsf      I2C_LAT , CPin_SDA ; // Set SCL high
bcf      I2C_TRIS, CPin_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    ; // Store byte
movwf    EE_DATA + 1

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

```

```

bcf     Status, c           ; // shr16
rrcf   EE_DATA + 1, f
rrcf   EE_DATA + 0, f

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  ( CAdminStart >> 0) & 0xff ; // Setup EEPROM address
movwf  EE_ADDR + 0
movlw  ( CAdminStart >> 8) & 0xff
movwf  EE_ADDR + 1

BL_loop_erase

; // Erase page

bsf    eecon1, eepgd       ; // point to FLASH program memory
bcf    eecon1, cfgs       ; // access FLASH 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

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

; // Fix CRC

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

movlw   4                     ; // Process 4 bytes
movwf   COUNTER
clr     EE_ADDR               ; // Start at 0

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 !!

HighPriorityInterruptVector
nop           ; // No priority support, just fall through

LowPriorityInterruptVector
; // Save processor status

movwf   WTMP           ; // Copy W to WTMP register !! in ACCESS bank !!
movff   bsr , CBSRTmp ; // Save BSR
movlb   0x00           ; // Select the default rambank

movff   status, CStatusTmp ; // Save Status
movff   fsr0l , CFSRTmpL ; // Save fsr0l
movff   fsr0h , CFSRTmpH ; // Save fsr0h
movff   pclath, CPCHTmp  ; // Save pclath
goto    CLowPriorityInterrupt

HighPriorityInterruptReturn
nop           ; // No priority support, just fall through

LowPriorityInterruptReturn
; // Restore processor status

movff   CPCHTmp , pclath ; // Restore pclath
movff   CFSRTmpH , fsr0h ; // Restore fsr0h
movff   CFSRTmpL , fsr0l ; // Restore fsr0l
movff   CBSRTmp , bsr   ; // Restore bsr
movf    WTMP , w        ; // Restore wreg
movff   CStatusTmp, status ; // Restore status

retfie

; // ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; // ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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

ValueInit           ; // Value initialization code is compiled here

```

```

movlb 0x00 ; // Followed by the rest of ValueBegin ( )
; // Select the default rambank

movlw (SZero >> 0) & 0xff ; // AddressToFsr0
movwf fsr0l
movlw (SZero >> 8) & 0xff
movwf fsr0h

movlw (LZero >> 0) & 0xff ; // AddressToFsr1
movwf fsr1l
movlw (LZero >> 8) & 0xff
movwf fsr1h

goto CMain ; // Jump into the 4th [[Main]] function.

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

; ===== Constant definitions

; Vocabulary 'Forth'

CFlashPageSize equ D'64' ; // constant FlashPageSize
CAdminStart equ D'512' ; // 2constant AdminStart
C0d equ D'0' ; // 2constant 0d
C1d equ D'1' ; // 2constant 1d
C2d equ D'2' ; // 2constant 2d
C4d equ D'4' ; // 2constant 4d
C8d equ D'8' ; // 2constant 8d
C10d equ D'10' ; // 2constant 10d
C16d equ D'16' ; // 2constant 16d
C20d equ D'20' ; // 2constant 20d
C30d equ D'30' ; // 2constant 30d
C32d equ D'32' ; // 2constant 32d
C40d equ D'40' ; // 2constant 40d
C50d equ D'50' ; // 2constant 50d
C64d equ D'64' ; // 2constant 64d
C100d equ D'100' ; // 2constant 100d
C128d equ D'128' ; // 2constant 128d
C256d equ D'256' ; // 2constant 256d
C512d equ D'512' ; // 2constant 512d
C1000d equ D'1000' ; // 2constant 1000d
C1024d equ D'1024' ; // 2constant 1024d
C2048d equ D'2048' ; // 2constant 2048d
C4096d equ D'4096' ; // 2constant 4096d
C8192d equ D'8192' ; // 2constant 8192d
C10000d equ D'10000' ; // 2constant 10000d
C16384d equ D'16384' ; // 2constant 16384d
C32768d equ D'32768' ; // 2constant 32768d
C65535d equ D'65535' ; // 2constant 65535d
C2D1d equ D'65535' ; // 2constant -1d
Cffffd equ D'65535' ; // 2constant ffff
CTrueTrue equ D'65535' ; // 2constant TrueTrue
CFalseTrue equ D'65280' ; // 2constant FalseTrue
CTrueFalse equ D'255' ; // 2constant TrueFalse
CFalseFalse equ D'0' ; // 2constant FalseFalse
C0q equ D'0' ; // 4constant 0q
C2D1q equ -D'1' ; // 4constant -1q

; Vocabulary end.

; ===== Constant definitions end.

; ===== variable definitions

; Vocabulary 'Forth'

CStatusTmp equ 256 * D'0' + D'128' ; // variable StatusTmp ; Bank = 0
CPCHTmp equ 256 * D'0' + D'129' ; // variable PCHTmp ; Bank = 0
CFSRTmpL equ 256 * D'0' + D'130' ; // variable FSRTmpL ; Bank = 0
CFSRtmpH equ 256 * D'0' + D'131' ; // variable FSRtmpH ; Bank = 0
CBSRTmp equ 256 * D'0' + D'132' ; // variable BSRTmp ; Bank = 0
CRamPtr equ 256 * D'0' + D'133' ; // 2variable RamPtr ; Bank = 0
CspSave equ 256 * D'0' + D'135' ; // 2variable spSave ; Bank = 0
CQuotient equ 256 * D'0' + D'137' ; // 4variable Quotient ; Bank = 0
CRemainder equ 256 * D'0' + D'141' ; // 4variable Remainder ; Bank = 0
CDivisor equ 256 * D'0' + D'145' ; // 4variable Divisor ; Bank = 0

; Vocabulary end.

; ===== variable definitions end.

; ===== Code start

; Vocabulary 'Compiler'

; 'Compiler.((fbranch))' =====
;
; ( f -- )
; Size = 0
;
C2828fbranch2929 ; 0:((fbranch))
movwf (tmp1) ; // PopReg ( b -- )
movf postdec0, w

movf tmp1, f ; // To set or clear Z flag
return

; 'Compiler.((of))' =====

```

```

;
; ( b1 b2 -- b1 | )
; Size = 0
;
C2828of2929          ; 0:((of))
movwf (tmp1)          ; // PopReg ( b -- )
movf postdec0, w

cpfseq tmp1           ; // S// n1 == n2
bra of_uneq          ; // B/ n1 <> n2

; // n1 == n2

movf postdec0, w     ; // drop ( b -- )

bcf Status, Z        ; // Flag match
return

of_uneq

bsf Status, Z        ; // Flag no match
return

; 'Compiler.((loop))' =====
;
; ( -- )
; Size = 0
;
C2828loop2929       ; 0:((loop))
movwf preinc0       ; // Push nothing, free 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
movf postdec0, w    ; // drop ( b -- )

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

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

decf fsr1l, f        ; // restore Loop stack pointer
movf postdec0, w    ; // drop ( b -- )

bsf status, Z        ; // signal index <> limit
return

; 'Compiler.((+loop))' =====
;
; ( b -- )
; Size = 0
;
C28282Bloop2929     ; 0:((+loop))
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))]]

; 'Compiler.((do))' =====
;
; ( bLimit bStart -- )
; Size = 0
;
C2828do2929         ; 0:((do))
movwf (tmp1)         ; // PopReg ( b -- )
movf postdec0, w

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

movwf (tmp2)         ; // PopReg ( b -- )
movf postdec0, w

decf fsr1l           ; // Push tmp2 [ -- b ]
movff (tmp2), indf1

decf fsr1l           ; // Push tmp1 [ -- b ]
movff (tmp1), indf1

bcf status, z        ; // Return with zero flag cleared
return               ; // B/ ----> done (loop taken)

; 'Compiler.((?do))' =====
;
; ( bLimit bStart -- )
; Size = 0
;
C28283Fdo2929       ; 0:((?do))
movwf (tmp1)         ; // PopReg ( b -- )
movf postdec0, w

cpfseq indf1         ; // S/ Start = Limit

```





```

; // 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 |
; // | | +-----+
; // |
; // +-----+
; // <--- + fsrl |
; // +-----+

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

movff postinc1, (( EFP + 1)) ; // Pop ( EFP + 1) [ b -- ]
movff postinc1, (( EFP + 0)) ; // Pop ( EFP + 0) [ b -- ]

movwf preinc0 ; // Push nothing, free wreg

movlw 7
addwf fsrll, f ; // drop [ n1 .. nn -- ] from loop stack
movf postdec0, w ; // drop ( b -- )

```

```

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

```

```
return ; // B/ ----> done
```

```
; 'Compiler.(*)' =====
```

```
;
; ( u1 u2 -- u3 )
; Size = 0
;
```

```
C282A29 ; 0:(*)
```

```
movwf (tmp1) ; // PopReg ( b -- )
movf postdec0, w

mulwf tmp1
movf prodl, w

return
```

```
; 'Compiler.(/)' =====
```

```
;
; ( u1 u2 -- u3 )
; Size = 0
;
```

```
C282F29 ; 0:(/)
```

```
; // 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
decfsz tmp2, f
bra slash_0
```

```

movf    postdec0, w          ; // get result to TOS
return

; 'Compiler.(/Mod)' =====
;
; ( bDividend bDivisor -- bQuotient bRemainder )
; Size = 0
;
C282FMod29                ; 0:(/Mod)

; ///////////////////////////////////////////////////////////////////
; // Function      : /Mod ( bDividend bDivisor -- Quotient Remainder )
; // Input         : bDividend : 8 bits unsigned - high byte on top
; //               : bDivisor  : 8 bits unsigned - high byte on top
; // Output        : Quotient   : 8 bits unsigned - high byte on top
; //               : Remainder  : 8 bits unsigned - high byte on top
; // Description   : 8 / 8 -> 8 bits unsigned divide with remainder
; // Remarks       : Quotient   := bDividend Div bDivisor
; //               : Remainder  := bDividend Mod bDivisor
; ///////////////////////////////////////////////////////////////////
; //
; // From Zilog Z8 Family Design Handbook, June 1988
; //
; // A programmers guide to the Z8 microcomputer,
; // Z8 subroutine library, Application note
; // page 211.
; //
; // Adapted from 16 / 16 -> 16 + 16 algorithm above
; //
; // This is a 8 / 8 -> 8 + 8 udiv routine
; //
; ///////////////////////////////////////////////////////////////////

; // Pop arguments from stack to helper variables

movwf   (CDivisor)          ; // PopReg ( b -- )
movf    postdec0, w

movwf   CQuotient          ; // bDividend into Quotient

clrf    CRemainder

movlw   8                  ; // ld  d32_LEN, #8      !LOOP COUNTER!
movwf   tmp2
bcf     Status, C          ; // rcf                  !carry = 0!

; // Then go for it

dlp_8

rlcf    CQuotient, f       ; // rlc  quot
rlcf    CRemainder, f     ; // rlc  rem
bc      subt_8            ; // B/ C not clear

movf    CDivisor, w       ; // See if we can subtract
subwf   CRemainder, w
bnc     skip_8            ; // B/ carry not set, skip subtraction

subt_8

movf    CDivisor, w       ; // sub  rem, dvsr
subwf   CRemainder, f
bsf     Status, C         ; // scf

skip_8

decfsz  tmp2, f           ; // djnz d16_LEN, dlp_16
bra     dlp_8             ; //                               !no flags affected!

rlcf    CQuotient, f     ; // rlc  quot

; // Now copy results to stack

movf    CQuotient, w      ; // ( bQuotient )
movwf   preinc0          ; // Push CRemainder ( -- b )
movf    (CRemainder), w

return

; 'Compiler.(swap)' =====
;
; ( b1 b2 -- b2 b1 )
; Size = 0
;
C28swap29                ; 0:(swap)
movwf   (tmp2)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp1)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   preinc0          ; // Push tmp2 ( -- b )
movf    (tmp2), w

movwf   preinc0          ; // Push tmp1 ( -- b )
movf    (tmp1), w

```

```

return

; 'Compiler.(=)' =====
;
; ( b1 b2 -- f )
; Size = 0
;
C283D29                ; 0:(=)
    subwf    postdec0, f
    btfsc   status, Z
    retlw   True
    retlw   False

; 'Compiler.<>' =====
;
; ( b1 b2 -- f )
; Size = 0
;
C283C3E29             ; 0:<>
    subwf    postdec0, f
    btfsc   status, Z
    retlw   False
    retlw   True

; 'Compiler.<' =====
;
; ( u1 u2 -- f )
; Size = 0
;
C283C29              ; 0:<
    subwf    postdec0, f
    movlw   True                ; // assume true
    btfss   status, Z           ; // if set then always false
    btfsc   status, C           ; // if not then also analyze carry
    clrfs   wreg                ; // false flag
    return

; 'Compiler.>' =====
;
; ( u1 u2 -- f )
; Size = 0
;
C283E29              ; : 0:>
    call    C28swap29           ; // 0:(swap) (code)
    call    C283C29             ; // 0:< (code)
    return                       ; // B/ ----> (i)

; 'Compiler.<=' =====
;
; ( u1 u2 -- f )
; Size = 0
;
C283C3D29            ; : 0:<=
    call    C283E29             ; // 0:> (colon)
    xorlw   0xff                ; // Not ( b1 -- b2 )
    return                       ; // B/ ----> (i)

; 'Compiler.>=' =====
;
; ( u1 u2 -- f )
; Size = 0
;
C283E3D29            ; : 0:>=
    call    C283C29             ; // 0:< (code)
    xorlw   0xff                ; // Not ( b1 -- b2 )
    return                       ; // B/ ----> (i)

; 'Compiler.(?dup)' =====
;
; ( b -- b b | b -- 0 )
; Size = 0
;
C283Fdup29           ; 0:(?dup)
    iorlw   D'0'
    btfsc   status, Z
    return

    movwf   preinc0             ; // Dup ( b1 -- b1 b1 )

    return

; 'Compiler.(over)' =====
;
; ( b1 b2 -- b1 b2 b1 )
; Size = 0
;
C28over29            ; 0:(over)
    movff   indf0, tmp1
    movwf   preinc0             ; // Push tmp1 ( -- b )
    movf    (tmp1), w

    return

; 'Compiler.(2dup)' =====
;
; ( w -- w w )

```

```

; Size = 0
;
C282dup29          ; : 0:(2dup)
                  ; // 0:(over) (code)
    call    C28over29
                  ; // 0:(over) (code)
    call    C28over29
    return       ; // B/ ----> (i)

; 'Compiler.<<)' =====
;
; ( b1 b2 -- b3 )
; Size = 0
;
C283C3C29          ; 0:<<)
    movwf   (tmp1)          ; // PopReg ( b -- )
    movf    postdec0, w

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

    return

; 'Compiler.>>)' =====
;
; ( b1 b2 -- b3 )
; Size = 0
;
C283E3E29          ; 0:>>)
    movwf   (tmp1)          ; // PopReg ( b -- )
    movf    postdec0, w

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

    return

; 'Compiler.(2Over)' =====
;
; ( w1 w2 -- w1 w2 w1 )
; Size = 0
;
C282Over29         ; 0:(2Over)
    movwf   (tmp4)          ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp3)          ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp2)          ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   tmp1

    movwf   preinc0         ; // Push tmp2 ( -- b )
    movf    (tmp2), w

    movwf   preinc0         ; // Push tmp3 ( -- b )
    movf    (tmp3), w

    movwf   preinc0         ; // Push tmp4 ( -- b )
    movf    (tmp4), w

    movwf   preinc0         ; // Push tmp1 ( -- b )
    movf    (tmp1), w

    movwf   preinc0         ; // Push tmp2 ( -- b )
    movf    (tmp2), w

    return

; 'Compiler.(rot)' =====
;
; ( b1 b2 b3 -- b2 b3 b1 )
; Size = 0
;
C28rot29           ; 0:(rot)
    movwf   (tmp3)          ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp2)          ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp1)          ; // PopReg ( b -- )
    movf    postdec0, w

```

```

movwf preinc0          ; // Push tmp2 ( -- b )
movf (tmp2), w

movwf preinc0          ; // Push tmp3 ( -- b )
movf (tmp3), w

movwf preinc0          ; // Push tmp1 ( -- b )
movf (tmp1), w

return

; 'Compiler.(-rot)' =====
;
; ( b1 b2 b3 -- b3 b1 b2 )
; Size = 0
;
C282Drot29             ; 0:(-rot)
movwf (tmp3)           ; // PopReg ( b -- )
movf postdec0, w

movwf (tmp2)           ; // PopReg ( b -- )
movf postdec0, w

movwf (tmp1)           ; // PopReg ( b -- )
movf postdec0, w

movwf preinc0          ; // Push tmp3 ( -- b )
movf (tmp3), w

movwf preinc0          ; // Push tmp1 ( -- b )
movf (tmp1), w

movwf preinc0          ; // Push tmp2 ( -- b )
movf (tmp2), w

return

; 'Compiler.(pick)' =====
;
; ( u1 -- u2 )
; Size = 0
;
C28pick29              ; 0:(pick)

; // Push u:th entry

movwf preinc0          ; // Push nothing, free wreg

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

; 'Compiler.(put)' =====
;
; ( b u -- )
; Size = 0
;
C28put29               ; 0:(put)

; // u th byte of stack replaced by value b
; // 0th byte = top of stack

sublw D'0'             ; // negate
movwf (tmp1)           ; // PopReg ( b -- )
movf postdec0, w

movwf tmp2             ; // Get value to put
movf tmp1, w           ; // Offset to wreg
movff tmp2, plusw0     ; // Put value
movwf (wreg)           ; // PopReg ( b -- )
movf postdec0, w

return

; 'Compiler.(abs)' =====
;
; ( b -- u )
; Size = 0
;
C28abs29               ; 0:(abs)
btfsc wreg, 7
sublw D'0'
return

; 'Compiler.(Max)' =====
;
; ( u1 u2 -- u3 )
; Size = 0
;
C28Max29               ; 0:(Max)
movwf (tmp1)           ; // PopReg ( b -- )
movf postdec0, w

cpfslt tmp1            ; // S/ tmp1 < wreg , so wreg is maximum
movf tmp1, w           ; // tmp1 >= wreg, so tmp1 is maximum
return

```

```

; 'Compiler.(Min)' =====
;
; ( u1 u2 -- u3 )
; Size = 0
;
C28Min29                ; 0:(Min)
    movwf    (tmp1)          ; // PopReg ( b -- )
    movf     postdec0, w

    cpdfsgt  tmp1           ; // S/ tmp1 > wreg , so wreg is minimum
    movf     tmp1, w        ; //      tmp1 <= wreg, so tmp1 is minimum
    return

; 'Compiler.(@)' =====
;
; ( wRamAddress -- b )
; Size = 0
;
C284029                ; 0:(@)
    movwf    (fsr2h)         ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (fsr2l)         ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    preinc0        ; // Push indf2 ( -- b )
    movf     (indf2), w

    return

; 'Compiler.(!)' =====
;
; ( b wRamAddress -- )
; Size = 0
;
C282129                ; 0:(!)
    movwf    (fsr2h)         ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (fsr2l)         ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (indf2)        ; // PopReg ( b -- )
    movf     postdec0, w

    return

; 'Compiler.(+!)' =====
;
; ( b wRamAddress -- )
; Size = 0
;
C282B2129              ; 0:(+!)
    movwf    (fsr2h)         ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (fsr2l)         ; // PopReg ( b -- )
    movf     postdec0, w

    addwf    indf2           ; // drop ( b -- )
    movf     postdec0, w

    return

; 'Compiler.(-!)' =====
;
; ( b wRamAddress -- )
; Size = 0
;
C282D2129              ; 0:(-!)
    movwf    (fsr2h)         ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (fsr2l)         ; // PopReg ( b -- )
    movf     postdec0, w

    subwf   indf2           ; // drop ( b -- )
    movf     postdec0, w

    return

; 'Compiler.(I)' =====
;
; ( -- u )
; Size = 0
;
C28I2729                ; 0:(I')
    movwf    preinc0        ; // Push D'1' ( -- b )
    movlw    (D'1')

    movf     plusw1, w
    return

; 'Compiler.(J)' =====
;
; ( -- u )
; Size = 0
;
C28J29                  ; 0:(J)
    movwf    preinc0        ; // Push D'2' ( -- b )

```

```

movlw    (D'2')

movf     plusw1, w
return

; 'Compiler.(J)' =====
;
; ( -- u )
; Size = 0
;
C28J2729          ; 0:(J')
movwf     preinc0          ; // Push D'3' ( -- b )
movlw     (D'3')

movf     plusw1, w
return

; 'Compiler.(K)' =====
;
; ( -- u )
; Size = 0
;
C28K29           ; 0:(K)
movwf     preinc0          ; // Push D'4' ( -- b )
movlw     (D'4')

movf     plusw1, w
return

; 'Compiler.(R>)' =====
;
; ( -- b )
; Size = 0
;
C28R3E29         ; 0:(R>)
movff     postinc1, (tmp1) ; // Pop tmp1 [ b -- ]

movwf     preinc0          ; // Push tmp1 ( -- b )
movf     (tmp1), w

return

; 'Compiler.(>R)' =====
;
; ( b -- )
; Size = 0
;
C283ER29        ; 0:(>R)
movwf     (tmp1)           ; // PopReg ( b -- )
movf     postdec0, w

decf     fsr1l             ; // Push tmp1 [ -- b ]
movff     (tmp1), indf1

return

; Vocabulary end.
; Vocabulary 'Forth'

; 'Forth.d@' =====
;
; ( wRamAddress -- w )
; Size = 0
;
Cd40           ; 0:d@
movwf     (fsr2h)          ; // PopReg ( b -- )
movf     postdec0, w

movwf     (fsr2l)          ; // PopReg ( b -- )
movf     postdec0, w

movwf     preinc0          ; // Push postinc2 ( -- b )
movf     (postinc2), w

movwf     preinc0          ; // Push indf2 ( -- b )
movf     (indf2), w

return

; 'Forth.dabs' =====
;
; ( d -- ud )
; Size = 0
;
Cdabs          ; 0:dabs

; // 16 bits Abs

btfss    wreg, 7
bra      dabs_done

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

dabs_done

```

```

Return

; 'Forth.Negative?' =====
;
; ( b -- f )
; Size = 0
;
CNegative3F                ; 0:Negative?
    btfss    wreg, 7
    retlw   False
    retlw   True

; 'Forth.dNegative?' =====
;
; ( d -- f )
; Size = 0
;
CdNegative3F                ; 0:dNegative?
    decf    fsr01, f          ; // SwapDrop ( b1 b2 -- b2 )

    bra    CNegative3F

; 'Forth.qNegative?' =====
;
; ( q -- f )
; Size = 0
;
CqNegative3F                ; 0:qNegative?
    decf    fsr01, f          ; // SwapDrop ( b1 b2 -- b2 )

    decf    fsr01, f          ; // SwapDrop ( b1 b2 -- b2 )

    bra    CdNegative3F

; 'Forth.dSwap' =====
;
; ( w1 w2 -- w2 w1 )
; Size = 0
;
CdSwap                      ; 0:dSwap
    movwf   (tmp4)            ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp3)            ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp2)            ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (tmp1)            ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   preinc0           ; // Push tmp3 ( -- b )
    movf    (tmp3), w

    movwf   preinc0           ; // Push tmp4 ( -- b )
    movf    (tmp4), w

    movwf   preinc0           ; // Push tmp1 ( -- b )
    movf    (tmp1), w

    movwf   preinc0           ; // Push tmp2 ( -- b )
    movf    (tmp2), w

    return

; 'Forth.a+' =====
;
; ( w1 u -- w2 )
; Size = 0
;
Ca2B                        ; 0:a+
    decf    fsr01, f
    addwf   postinc0, f
    movf    postdec0, w          ; // drop ( b -- )

    return

; 'Forth.3a+' =====
;
; ( w1 -- w2 )
; Size = 0
;
C3a2B                        ; 0:3a+
    incf    indf0, f
    incf    indf0, f
    incf    indf0, f
    return

; 'Forth.4a+' =====
;
; ( w1 -- w2 )
; Size = 0
;
C4a2B                        ; 0:4a+
    incf    indf0, f
    incf    indf0, f
    incf    indf0, f

```

```

    incf    indf0, f
    return

; 'Forth.a-' =====
;
; ( w1 u -- w2 )
; Size = 0
;
Ca2D                                ; 0:a-
    decf    fsr01, f
    subwf   postinc0, f
    movf    postdec0, w                ; // drop ( b -- )

    return

; 'Forth.3a-' =====
;
; ( w1 -- w2 )
; Size = 0
;
C3a2D                                ; 0:3a-
    decf    indf0, f
    decf    indf0, f
    decf    indf0, f
    return

; 'Forth.4a-' =====
;
; ( w1 -- w2 )
; Size = 0
;
C4a2D                                ; 0:4a-
    decf    indf0, f
    decf    indf0, f
    decf    indf0, f
    decf    indf0, f
    return

; 'Forth.@+' =====
;
; ( wAdr1 wAdr2 -- wAdr3 )
; Size = 0
;
C40a2B                                ; : 0:@a+
                                ; // 0:(@) (code)
    call    C284029
                                ; // 0:a+ (code)
    call    Ca2B
                                ; // B/ ----> (i)
    return

; 'Forth.a+@' =====
;
; ( wAdr u -- b )
; Size = 0
;
Ca2B40                                ; : 0:a+@
                                ; // 0:a+ (code)
    call    Ca2B
                                ; // 0:(@) (code)
    call    C284029
                                ; // B/ ----> (i)
    return

; 'Forth.@a+@' =====
;
; ( wAdr1 wAdr2 -- b )
; Size = 0
;
C40a2B40                                ; : 0:@a+@
                                ; // 0:(@) (code)
    call    C284029
                                ; // 0:a+@ (colon)
    call    Ca2B40
                                ; // B/ ----> (i)
    return

; 'Forth.a+!' =====
;
; ( b wAdr u -- )
; Size = 0
;
Ca2B21                                ; : 0:a+!
                                ; // 0:a+ (code)
    call    Ca2B
                                ; // 0:(!) (code)
    call    C282129
                                ; // B/ ----> (i)
    return

; 'Forth.@a+!' =====
;
; ( b wAdr1 wAdd2 -- )
; Size = 0
;
C40a2B21                                ; : 0:@a+!
                                ; // 0:(@) (code)
    call    C284029
                                ; // 0:a+! (colon)
    call    Ca2B21
                                ; // B/ ----> (i)
    return

; 'Forth.adup@' =====
;

```

```

; ( wAdr -- wAdr u )
; Size = 0
;
Cadup40                                ; : 0:adup@
; Aliased aDup Is (2dup)
      call    C282dup29                 ; // 0:(2dup) (colon)
      call    C284029                   ; // 0:(@) (code)
      return   ; // B/ ----> (i)

; 'Forth.3Drop' =====
;
; ( t -- )
; Size = 0
;
C3Drop                                ; : 0:3Drop
      movf    postdec0, w                ; // drop ( b -- )
      movf    postdec0, w                ; // drop ( b -- )
      movf    postdec0, w                ; // drop ( b -- )
      return   ; // B/ ----> (i)

; 'Forth.4Drop' =====
;
; ( q -- )
; Size = 0
;
C4Drop                                ; : 0:4Drop
      movf    postdec0, w                ; // drop ( b -- )
      movf    postdec0, w                ; // drop ( b -- )
      movf    postdec0, w                ; // drop ( b -- )
      movf    postdec0, w                ; // drop ( b -- )
      return   ; // B/ ----> (i)

; 'Forth.4Dup' =====
;
; ( q -- q q )
; Size = 0
;
C4Dup                                ; : 0:4Dup
      call    C2820over29                ; // 0:(2Over) (code)
      call    C2820over29                ; // 0:(2Over) (code)
      return   ; // B/ ----> (i)

; 'Forth.RomPtr!' =====
;
; ( wRomAddress -- )
; Size = 0
;
CRomPtr21                             ; 0:RomPtr!
      movwf   (tblptrh)                  ; // PopReg ( b -- )
      movf    postdec0, w
      movwf   (tblptrl)                  ; // PopReg ( b -- )
      movf    postdec0, w
      return

; 'Forth.RomPtr@' =====
;
; ( -- wRomAddress )
; Size = 0
;
CRomPtr40                             ; 0:RomPtr@
      movwf   preinc0                    ; // Push tblptrl ( -- b )
      movf    (tblptrl), w
      movwf   preinc0                    ; // Push tblptrh ( -- b )
      movf    (tblptrh), w
      return

; 'Forth.Rom@' =====
;
; ( -- b )
; Size = 0
;
CRom40                                ; 0:Rom@
      tblrd*
      movwf   preinc0                    ; // Push tablat ( -- b )
      movf    (tablat), w
      return

; 'Forth.[]Rom@' =====
;
; ( bIndex -- b )
; Size = 0
;

```

```

C5B5DRom40                                ; 0:[Rom@
movff   tblptrl, tmp1                      ; // Save tablepointer
movff   tblptrh, tmp2
addwf   tblptrl, f                          ; // Add bIndex to it
clrf    wreg
addwfc  tblptrh, f
tblrd*  tblrd*                               ; // read bData byte
movf    tablat, w                          ; // return data in TOS
movff   tmp2, tblptrl                      ; // restore table pointer
movff   tmp1, tblptrh
return

; 'Forth.Rom@++' =====
;
; ( -- b )
; Size = 0
;
CRom402B2B                                ; 0:Rom@++
tblrd*+
movwf   preinc0                            ; // Push tablat ( -- b )
movf    (tablat), w

return

; 'Forth.++Rom@' =====
;
; ( -- b )
; Size = 0
;
C2B2BRom40                                ; 0:++Rom@
tblrd*+
movwf   preinc0                            ; // Push tablat ( -- b )
movf    (tablat), w

return

; 'Forth.Rom@--' =====
;
; ( -- )
; Size = 0
;
CRom402D2D                                ; 0:Rom@--
tblrd*-
movwf   preinc0                            ; // Push tablat ( -- b )
movf    (tablat), w

return

; 'Forth.Rom!' =====
;
; ( b -- )
; Size = 0
;
CRom21                                    ; 0:Rom!
movwf   (tablat)                          ; // PopReg ( b -- )
movf    postdec0, w

tblwt*
return

; 'Forth.Rom!++' =====
;
; ( b -- )
; Size = 0
;
CRom212B2B                                ; 0:Rom!++
movwf   (tablat)                          ; // PopReg ( b -- )
movf    postdec0, w

tblwt*+
return

; 'Forth.++Rom!' =====
;
; ( b -- )
; Size = 0
;
C2B2BRom21                                ; 0:++Rom!
movwf   (tablat)                          ; // PopReg ( b -- )
movf    postdec0, w

tblwt*+
return

; 'Forth.Rom!--' =====
;
; ( b -- )
; Size = 0
;
CRom212D2D                                ; 0:Rom!--
movwf   (tablat)                          ; // PopReg ( b -- )
movf    postdec0, w

tblwt*-
return

; 'Forth.RamPtr!' =====
;
; ( wRamAddress -- )
; Size = 0

```

```

;
CRamPtr21      ; 0:RamPtr!
  movwf      (CRamPtr + 1)      ; // PopReg ( b -- )
  movf      postdec0, w

  movwf      (CRamPtr + 0)      ; // PopReg ( b -- )
  movf      postdec0, w

  return

; 'Forth.RamPtr@' =====
;
; ( -- wRamAddress )
; Size = 0
;
CRamPtr40      ; 0:RamPtr@
  movwf      preinc0      ; // Push CRamPtr + 0 ( -- b )
  movf      (CRamPtr + 0), w

  movwf      preinc0      ; // Push CRamPtr + 1 ( -- b )
  movf      (CRamPtr + 1), w

  return

; 'Forth.Ram@' =====
;
; ( -- b )
; Size = 0
;
CRam40         ; : 0:Ram@
              ; // 0:RamPtr@ (code)
  call      CRamPtr40      ; // 0:(@) (code)
  call      C284029        ; // B/ ----> (i)
  return

; 'Forth.[ ]Ram@' =====
;
; ( bIndex -- b )
; Size = 0
;
C5B5DRam40     ; : 0:[ ]Ram@
              ; // 0:RamPtr@ (code)
  call      CRamPtr40      ; // 0:a+@ (colon)
  call      Ca2B40         ; // B/ ----> (i)
  return

; 'Forth.Ram@++' =====
;
; ( -- b )
; Size = 0
;
CRam402B2B     ; : 0:Ram@++
              ; // 0:Ram@ (colon)
  call      CRam40
  incf      CRamPtr + 0    ; // RamPtr++ ( -- )
  return      ; // B/ ----> (i)

; 'Forth.++Ram@' =====
;
; ( -- b )
; Size = 0
;
C2B2BRam40     ; : 0:++Ram@
              ; // RamPtr++ ( -- )
              ; // 0:Ram@ (colon)
  incf      CRamPtr + 0
  call      CRam40
  return      ; // B/ ----> (i)

; 'Forth.Ram@--' =====
;
; ( -- b )
; Size = 0
;
CRam402D2D     ; : 0:Ram@--
              ; // 0:Ram@ (colon)
  call      CRam40
  decf      CRamPtr + 0    ; // RamPtr-- ( -- )
  return      ; // B/ ----> (i)

; 'Forth.Ram!' =====
;
; ( b -- )
; Size = 0
;
CRam21         ; : 0:Ram!
              ; // 0:RamPtr@ (code)
  call      CRamPtr40      ; // 0:(!) (code)
  call      C282129        ; // B/ ----> (i)
  return

; 'Forth.Ram!++' =====
;
; ( b -- )
; Size = 0
;
CRam212B2B     ; : 0:Ram!++
              ; // 0:Ram! (colon)

```

```

    call    CRam21
    incf   CRamPtr + 0      ; // RamPtr++ ( -- )
    return ; // B/ ----> (i)

; 'Forth.++Ram!' =====
;
; ( b -- )
; Size = 0
;
C2B2BRam21 ; : 0:++Ram!
    incf   CRamPtr + 0      ; // RamPtr++ ( -- )
                                ; // 0:Ram! (colon)
    call   CRam21
    return ; // B/ ----> (i)

; 'Forth.Ram!--' =====
;
; ( b -- )
; Size = 0
;
CRam21D2D ; : 0:Ram!--
            ; // 0:Ram! (colon)
    call   CRam21
    decf   CRamPtr + 0      ; // RamPtr-- ( -- )
    return ; // B/ ----> (i)

; 'Forth.1-!' =====
;
; ( wRamAddress -- )
; Size = 0
;
C12D21 ; : 0:1-!
    movwf (fsr2h)           ; // PopReg ( b -- )
    movf  postdec0, w

    movwf (fsr2l)           ; // PopReg ( b -- )
    movf  postdec0, w

    decf  indf2, f
    return

; 'Forth.1+!' =====
;
; ( wRamAddress -- )
; Size = 0
;
C12B21 ; : 0:1+!
    movwf (fsr2h)           ; // PopReg ( b -- )
    movf  postdec0, w

    movwf (fsr2l)           ; // PopReg ( b -- )
    movf  postdec0, w

    incf  indf2, f
    return

; 'Forth.d!' =====
;
; ( w wRamAddress -- )
; Size = 0
;
Cd21 ; : 0:d!
    movwf (fsr2h)           ; // PopReg ( b -- )
    movf  postdec0, w

    movwf (fsr2l)           ; // PopReg ( b -- )
    movf  postdec0, w

    movwf (tmp1)            ; // PopReg ( b -- )
    movf  postdec0, w

    movwf (postinc2)        ; // PopReg ( b -- )
    movf  postdec0, w

    movff tmp1, indf2
    return

; 'Forth.And!' =====
;
; ( b wRamAddress -- )
; Size = 0
;
CAnd21 ; : 0:And!
    movwf (fsr2h)           ; // PopReg ( b -- )
    movf  postdec0, w

    movwf (fsr2l)           ; // PopReg ( b -- )
    movf  postdec0, w

    andwf indf2
    movf  postdec0, w      ; // drop ( b -- )

    return

; 'Forth.Or!' =====
;
; ( b wRamAddress -- )
; Size = 0
;
COr21 ; : 0:Or!

```

```

movwf (fsr2h) ; // PopReg ( b -- )
movf postdec0, w

movwf (fsr2l) ; // PopReg ( b -- )
movf postdec0, w

iorwf indf2
movf postdec0, w ; // drop ( b -- )

return

; 'Forth.Xor!' =====
;
; ( b wRamAddress -- )
; Size = 0
;
CXor21 ; 0:Xor!
movwf (fsr2h) ; // PopReg ( b -- )
movf postdec0, w

movwf (fsr2l) ; // PopReg ( b -- )
movf postdec0, w

xorwf indf2
movf postdec0, w ; // drop ( b -- )

return

; 'Forth.Dup!' =====
;
; ( b wRamAddress -- b )
; Size = 0
;
CDup21 ; 0:Dup!
movwf (fsr2h) ; // PopReg ( b -- )
movf postdec0, w

movwf (fsr2l) ; // PopReg ( b -- )
movf postdec0, w

movwf indf2
return

; 'Forth.True!' =====
;
; ( wRamAddress -- )
; Size = 0
;
CTrue21 ; 0:True!
movwf (fsr2h) ; // PopReg ( b -- )
movf postdec0, w

movwf fsr2l
movlw True
movwf indf2
movf postdec0, w ; // drop ( b -- )

return

; 'Forth.False!' =====
;
; ( wRamAddress -- )
; Size = 0
;
CFalse21 ; 0:False!
movwf (fsr2h) ; // PopReg ( b -- )
movf postdec0, w

movwf fsr2l
movlw False
movwf indf2
movf postdec0, w ; // drop ( b -- )

return

; 'Forth.^Xor' =====
;
; ( wRamAddress1 wRamAddress2 -- )
; Size = 0
;
C5EXor ; 0:^Xor

; // Indirect 8 bits xor of wRamAddress1^with wRamAddress^2
; // result stored at wRamAddress1

movff fsr1l, tmp1
movff fsr1h, tmp2
movwf (fsr2h) ; // PopReg ( b -- )
movf postdec0, w

movwf (fsr2l) ; // PopReg ( b -- )
movf postdec0, w

movwf (fsr1h) ; // PopReg ( b -- )
movf postdec0, w

movwf fsr1l

movf indf2, w
xorwf indf1, f

```

```

movff    tmp1, fsr1l
movff    tmp2, fsr1h

movf     postdec0, w           ; // drop ( b -- )

return

; 'Forth.^Shl' =====
;
; ( wRamAddress -- )
; Size = 0
;
C5EShl   ; 0:^Shl
movwf    (fsr2h)              ; // PopReg ( b -- )
movf     postdec0, w

movwf    (fsr2l)              ; // PopReg ( b -- )
movf     postdec0, w

rlcf     indf2
return

; 'Forth.^Shr' =====
;
; ( wRamAddress -- )
; Size = 0
;
C5EShr   ; 0:^Shr
movwf    (fsr2h)              ; // PopReg ( b -- )
movf     postdec0, w

movwf    (fsr2l)              ; // PopReg ( b -- )
movf     postdec0, w

rrcf     indf2
return

; 'Forth.^rl' =====
;
; ( wRamAddress -- )
; Size = 0
;
C5Erl    ; 0:^rl
movwf    (fsr2h)              ; // PopReg ( b -- )
movf     postdec0, w

movwf    (fsr2l)              ; // PopReg ( b -- )
movf     postdec0, w

rlncf    indf2
return

; 'Forth.^rr' =====
;
; ( wRamAddress -- )
; Size = 0
;
C5Err    ; 0:^rr
movwf    (fsr2h)              ; // PopReg ( b -- )
movf     postdec0, w

movwf    (fsr2l)              ; // PopReg ( b -- )
movf     postdec0, w

rrncf    indf2
return

; 'Forth.Xor32' =====
;
; ( q q -- q )
; Size = 0
;
CXor32   ; 0:Xor32

movwf    (tmp1)                ; // PopReg ( b -- )
movf     postdec0, w

movwf    (tmp2)                ; // PopReg ( b -- )
movf     postdec0, w

movwf    (tmp3)                ; // PopReg ( b -- )
movf     postdec0, w

movwf    (tmp4)                ; // PopReg ( b -- )
movf     postdec0, w

xorwf    tmp1, f
movf     postdec0, w           ; // drop ( b -- )

xorwf    tmp2, f
movf     postdec0, w           ; // drop ( b -- )

xorwf    tmp3, f
movf     postdec0, w           ; // drop ( b -- )

xorwf    tmp4, w

movwf    preinc0                ; // Push tmp3 ( -- b )

```

```

movf    (tmp3), w

movwf   preinc0           ; // Push tmp2 ( -- b )
movf    (tmp2), w

movwf   preinc0           ; // Push tmp1 ( -- b )
movf    (tmp1), w

return

; 'Forth.<<32' =====
;
; ( q n -- q )
; Size = 0
;
C3C3C32                ; 0:<<32

movwf   (tmp1)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp2)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp3)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp4)           ; // PopReg ( b -- )
movf    postdec0, w

shl32lp

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

decfsz tmp1
bra    shl32lp

movwf   preinc0           ; // Push tmp4 ( -- b )
movf    (tmp4), w

movwf   preinc0           ; // Push tmp3 ( -- b )
movf    (tmp3), w

movwf   preinc0           ; // Push tmp2 ( -- b )
movf    (tmp2), w

return

; 'Forth.>>32' =====
;
; ( q n -- q )
; Size = 0
;
C3E3E32                ; 0:>>32

movwf   (tmp1)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp2)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp3)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp4)           ; // PopReg ( b -- )
movf    postdec0, w

shr32lp

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

decfsz tmp1
bra    shr32lp

movwf   preinc0           ; // Push tmp4 ( -- b )
movf    (tmp4), w

movwf   preinc0           ; // Push tmp3 ( -- b )
movf    (tmp3), w

movwf   preinc0           ; // Push tmp2 ( -- b )
movf    (tmp2), w

return

; 'Forth.qnegate' =====

```

```

;
; ( q -- -q )
; Size = 0
;
Cqnegate                ; 0:qnegate

    movwf    (tmp4)                ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (tmp3)                ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (tmp2)                ; // PopReg ( b -- )
    movf     postdec0, w

    sublw   D'0'
    movwf   tmp1

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

    movf    tmp1, w
    movwf   preinc0                ; // Push tmp2 ( -- b )
    movf    (tmp2), w

    movwf   preinc0                ; // Push tmp3 ( -- b )
    movf    (tmp3), w

    movwf   preinc0                ; // Push tmp4 ( -- b )
    movf    (tmp4), w

    return

; 'Forth.d>r' =====
;
; ( d -- )
; Size = 0
;
Cd3Er                    ; : 0:d>r
                        ; // 0:(swap) (code)
    call    C28swap29
                        ; // 0:(>R) (code)
    call    C283ER29
                        ; // 0:(>R) (code)
    call    C283ER29
                        ; // B/ ----> (i)
    return

; 'Forth.dr>' =====
;
; ( d -- )
; Size = 0
;
Cdr3E                    ; : 0:dr>
                        ; // 0:(R>) (code)
    call    C28R3E29
                        ; // 0:(R>) (code)
    call    C28R3E29
                        ; // 0:(swap) (code)
    call    C28swap29
                        ; // B/ ----> (i)
    return

; 'Forth.q>r' =====
;
; ( q -- )
; Size = 0
;
Cq3Er                    ; : 0:q>r
                        ; // 0:dSwap (code)
    call    CdSwap
                        ; // 0:d>r (colon)
    call    Cd3Er
                        ; // 0:d>r (colon)
    call    Cd3Er
                        ; // B/ ----> (i)
    return

; 'Forth.qr>' =====
;
; ( q -- )
; Size = 0
;
Cqr3E                    ; : 0:qr>
                        ; // 0:dr> (colon)
    call    Cdr3E
                        ; // 0:dr> (colon)
    call    Cdr3E
                        ; // 0:dSwap (code)
    call    CdSwap
                        ; // B/ ----> (i)
    return

; 'Forth.Popq1' =====
;
; ( q -- )
; Size = 0
;
CPopq1                  ; 0:Popq1
                        ; // PopReg ( b -- )
    movwf   (tmp4)
    movf    postdec0, w

```

```

movwf (tmp3) ; // PopReg ( b -- )
movf postdec0, w

movwf (tmp2) ; // PopReg ( b -- )
movf postdec0, w

movwf (tmp1) ; // PopReg ( b -- )
movf postdec0, w

return

; 'Forth.Pushq1' =====
;
; ( -- q )
; Size = 0
;
CPushq1 ; 0:Pushq1
movwf preinc0 ; // Push tmp1 ( -- b )
movf (tmp1), w

movwf preinc0 ; // Push tmp2 ( -- b )
movf (tmp2), w

movwf preinc0 ; // Push tmp3 ( -- b )
movf (tmp3), w

movwf preinc0 ; // Push tmp4 ( -- b )
movf (tmp4), w

return

; 'Forth.Popq2' =====
;
; ( q -- )
; Size = 0
;
CPopq2 ; 0:Popq2

movwf (CQuotient + 3) ; // PopReg ( b -- )
movf postdec0, w

movwf (CQuotient + 2) ; // PopReg ( b -- )
movf postdec0, w

movwf (CQuotient + 1) ; // PopReg ( b -- )
movf postdec0, w

movwf (CQuotient + 0) ; // PopReg ( b -- )
movf postdec0, w

return

; 'Forth.Pushq2' =====
;
; ( -- q )
; Size = 0
;
CPushq2 ; 0:Pushq2

movwf preinc0 ; // Push CQuotient + 0 ( -- b )
movf (CQuotient + 0), w

movwf preinc0 ; // Push CQuotient + 1 ( -- b )
movf (CQuotient + 1), w

movwf preinc0 ; // Push CQuotient + 2 ( -- b )
movf (CQuotient + 2), w

movwf preinc0 ; // Push CQuotient + 3 ( -- b )
movf (CQuotient + 3), w

return

; 'Forth.Popq3' =====
;
; ( q -- )
; Size = 0
;
CPopq3 ; 0:Popq3

movwf (CRemainder + 3) ; // PopReg ( b -- )
movf postdec0, w

movwf (CRemainder + 2) ; // PopReg ( b -- )
movf postdec0, w

movwf (CRemainder + 1) ; // PopReg ( b -- )
movf postdec0, w

movwf (CRemainder + 0) ; // PopReg ( b -- )
movf postdec0, w

return

```

```

; 'Forth.Pushq3' =====
;
; ( -- q )
; Size = 0
;
CPushq3                ; 0:Pushq3

    movwf    preinc0          ; // Push CRemainder + 0 ( -- b )
    movf     (CRemainder + 0), w

    movwf    preinc0          ; // Push CRemainder + 1 ( -- b )
    movf     (CRemainder + 1), w

    movwf    preinc0          ; // Push CRemainder + 2 ( -- b )
    movf     (CRemainder + 2), w

    movwf    preinc0          ; // Push CRemainder + 3 ( -- b )
    movf     (CRemainder + 3), w

    return

; 'Forth.4rot' =====
;
; ( q1 q2 q3 -- q2 q3 q1 )
; Size = 0
;
C4rot                  ; : 0:4rot
                    ; // 0:Popq3 (code)
    call     CPopq3
                    ; // 0:Popq2 (code)
    call     CPopq2
                    ; // 0:Popq1 (code)
    call     CPopq1
                    ; // 0:Pushq2 (code)
    call     CPushq2
                    ; // 0:Pushq3 (code)
    call     CPushq3
                    ; // 0:Pushq1 (code)
    call     CPushq1
                    ; // B/ ----> ( ; )
    return

; 'Forth.4over' =====
;
; ( q1 q2 -- q1 q2 q1 )
; Size = 0
;
C4over                 ; : 0:4over
                    ; // 0:Popq2 (code)
    call     CPopq2
                    ; // 0:Popq1 (code)
    call     CPopq1
                    ; // 0:Pushq1 (code)
    call     CPushq1
                    ; // 0:Pushq2 (code)
    call     CPushq2
                    ; // 0:Pushq1 (code)
    call     CPushq1
                    ; // B/ ----> ( ; )
    return

; 'Forth.4swap' =====
;
; ( q1 q2 -- q2 q1 )
; Size = 0
;
C4swap                 ; : 0:4swap
                    ; // 0:Popq2 (code)
    call     CPopq2
                    ; // 0:Popq1 (code)
    call     CPopq1
                    ; // 0:Pushq2 (code)
    call     CPushq2
                    ; // 0:Pushq1 (code)
    call     CPushq1
                    ; // B/ ----> ( ; )
    return

; 'Forth.^>>32' =====
;
; ( wRamAddress n -- )
; Size = 0
;
C5E3E3E32             ; 0:^>>32

; // Indirect 32 bits right shift over n positions

    movwf    (tmp1)          ; // PopReg ( b -- )
    movf     postdec0, w

    movwf    (fsr2h)         ; // PopReg ( b -- )
    movf     postdec0, w

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

movf    postdec0, w          ; // drop ( b -- )

return

; 'Forth.^<<32' =====
;
; ( wRamAddress n -- )
; Size = 0
;
C5E3C3C32          ; 0:^<<32

; // Indirect 32 bits left shift over n positions

movwf   (tmp1)          ; // PopReg ( b -- )
movf    postdec0, w

movwf   (fsr2h)        ; // PopReg ( b -- )
movf    postdec0, w

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

movf    postdec0, w      ; // drop ( b -- )

return

; 'Forth.d+' =====
;
; ( wu1 wu2 -- wu3 )
; Size = 0
;
Cd2B          ; 0:d+

movwf   (tmp1)          ; // PopReg ( b -- )
movf    postdec0, w

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

; 'Forth.d-' =====
;
; ( wu1 wu2 -- wu3 )
; Size = 0
;
Cd2D          ; 0:d-

movwf   (tmp1)          ; // PopReg ( b -- )
movf    postdec0, w

decf    fsr0l          ; // 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

; 'Forth.d=' =====
;
; ( w1 w2 -- f )
; Size = 0
;
Cd3D          ; 0:d=

movwf   (tmp1)          ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp2)          ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp3)          ; // PopReg ( b -- )
movf    postdec0, w

cpfseq  tmp2
retlw   False

```

```

movf    tmp3, w
cpfseq  tmp1
retlw   False
retlw   True

; 'Forth.d>' =====
;
; ( wul wu2 -- f )
; Size = 0
;
Cd3E                                ; 0:d>
                                ; // Return True if N1 (u1,u2) > N2 (u3,u4)   (or N2 < N1)
movwf   (tmp3)                  ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp4)                  ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp1)                  ; // PopReg ( b -- )
movf    postdec0, w

movwf   tmp2

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

movwf   CQuotient               ; // Save wreg

movf    tmp4 + 1, w
subwf   tmp2 + 1, w
bnz     cpfslt16_CheckHiUnEq_1   ; // B/ N1 + 1 <> N2 + 1, test hi bytes

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

movf    tmp2, w                 ; // N2 to w
subwf   tmp4, w                 ; // calc N1 - N2 :: negative -> N1 - N2 < 0 -> N1 < N2
movff   CQuotient, WREG         ; // Restore wreg || NOT affecting flags
bc      cpfslt16_N1NotSmallerN2_1

; // N1 < N2

bra     cpfslt16_N1SmallerN2_1

cpfslt16_CheckHiUnEq_1

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

movf    tmp2 + 1, w             ; // N2 == tmp2 + 1
subwf   tmp4 + 1, w             ; // Sub N1:: negative -> N1 - N2 < 0 -> N1 < N2
movff   CQuotient, WREG         ; // Restore wreg || NOT affecting flags
bc      cpfslt16_N1NotSmallerN2_1

cpfslt16_N1SmallerN2_1

bra     cpfslt16_N1NotSmallerN2_1 + 2   ; // Skip next

cpfslt16_N1NotSmallerN2_1

retlw   False                   ; // Assumptionwrong, return false
retlw   True

; 'Forth.d<' =====
;
; ( wul wu2 -- f )
; Size = 0
;
Cd3C                                ; 0:d<
                                ; // Return True if N1 (u1,u2) < N2 (u3,u4)   (or N2 > N1)
movwf   (tmp3)                  ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp4)                  ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp1)                  ; // PopReg ( b -- )
movf    postdec0, w

movwf   tmp2

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

movwf   CQuotient               ; // Save wreg

movf    tmp2 + 1, w
subwf   tmp4 + 1, w
bnz     cpfslt16_CheckHiUnEq_2   ; // B/ N1 + 1 <> N2 + 1, test hi bytes

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

movf    tmp4, w                 ; // N2 to w
subwf   tmp2, w                 ; // calc N1 - N2 :: negative -> N1 - N2 < 0 -> N1 < N2
movff   CQuotient, WREG         ; // Restore wreg || NOT affecting flags
bc      cpfslt16_N1NotSmallerN2_2

```

```

; // N1 < N2
bra    cpfslt16_N1SmallerN2_2

cpfslt16_CheckHiUnEq_2

; // Hi bytes are not equal, test 'm
movf   tmp4 + 1, w           ; // N2 == tmp4 + 1
subwf  tmp2 + 1, w           ; // Sub N1:: negative -> N1 - N2 < 0 -> N1 < N2
movff  CQuotient, WREG       ; // Restore wreg || NOT affecting flags
bc     cpfslt16_N1NotSmallerN2_2

cpfslt16_N1SmallerN2_2

bra    cpfslt16_N1NotSmallerN2_2 + 2    ; // Skip next

cpfslt16_N1NotSmallerN2_2

retlw  False
retlw  True

; 'Forth.sd<' =====
;
; ( sd sd -- f )
; Size = 0
;
Csd3C          ; : 0:sd<
               ; // 0:d- (code)
call   Cd2D           ; // 0:dNegative? (code)
call   CdNegative3F   ; // B/ ----> (i)
return

; 'Forth.sd>' =====
;
; ( sd sd -- f )
; Size = 0
;
Csd3E          ; : 0:sd>
               ; // 0:dSwap (code)
call   CdSwap        ; // 0:sd< (colon)
call   Csd3C         ; // B/ ----> (i)
return

; 'Forth.sd>=' =====
;
; ( sd sd -- f )
; Size = 0
;
Csd3E3D        ; : 0:sd>=
               ; // 0:sd< (colon)
call   Csd3C         ; // Not ( b1 -- b2 )
xorlw  0xff          ; // B/ ----> (i)
return

; 'Forth.sd<=' =====
;
; ( sd sd -- f )
; Size = 0
;
Csd3C3D        ; : 0:sd<=
               ; // 0:sd> (colon)
call   Csd3E         ; // Not ( b1 -- b2 )
xorlw  0xff          ; // B/ ----> (i)
return

; 'Forth.0=' =====
;
; ( b -- f )
; Size = 0
;
C03D          ; 0:0=
iorlw  D'0'
btfsc  status, Z
retlw  True
retlw  False

; 'Forth.1=' =====
;
; ( b -- f )
; Size = 0
;
C13D          ; 0:1=
decfsz wreg
retlw  False
retlw  True

; 'Forth.0<>' =====
;
; ( b -- f )
; Size = 0
;
C03C3E        ; 0:0<>
iorlw  D'0'
btfsc  status, Z
retlw  False
retlw  True

```

```

; 'Forth.0<' =====
;
; ( b -- f )
; Size = 0
;
C03C                                ; : 0:0<
                                     ; // literal 128
movwf  preinc0                       ; // Push D'128' ( -- b )
movlw  (D'128')

andwf  postdec0, w                   ; // (and) ( b1 b2 -- b3 )
                                     ; // 0:0<> (code)

call   C03C3E                        ; // B/ ----> ( ; )
return

; 'Forth.s>d' =====
;
; ( b -- w )
; Size = 0
;
Cs3Ed                                ; 0:s>d
movwf  preinc0                       ; // Push nothing, free wreg

btfsc  indf0, 7
retlw  D'255'
retlw  D'0'

; 'Forth.d>q' =====
;
; ( w -- q )
; Size = 0
;
Cd3Eq                                ; 0:d>q
movwf  preinc0                       ; // Push D'0' ( -- b )
movlw  (D'0')

btfsc  indf0, 7
movlw  D'255'
movwf  preinc0                       ; // Push D'0' ( -- b )
movlw  (D'0')

btfsc  indf0, 7
movlw  D'255'
return

; 'Forth.d*' =====
;
; ( u1 u2 -- w )
; Size = 0
;
Cd2A                                ; 0:d*
movwf  (tmp1)                        ; // PopReg ( b -- )
movf   postdec0, w

mulwf  tmp1
movf   prodl, w
movwf  preinc0                       ; // Push prodh ( -- b )
movf   (prodh), w

return

; 'Forth.q*' =====
;
; ( w1 w2 -- q )
; Size = 0
;
Cq2A                                ; 0:q*
movwf  (tmp4)                        ; // PopReg ( b -- )
movf   postdec0, w

movwf  (tmp3)                        ; // PopReg ( b -- )
movf   postdec0, w

movwf  (tmp2)                        ; // PopReg ( b -- )
movf   postdec0, w

movwf  tmp1

movlw  17
movwf  tmp5

clrf   CDivisor
clrf   CDivisor + 1

bcf    Status, C

qstar_1
rrcf   CDivisor + 1, f
rrcf   CDivisor + 0, f
rrcf   tmp2, f
rrcf   tmp1, f
bnc    qstar_2                       ; // B/ Carry clear, skip add

; Carry set, add

```

```

movf    tmp3, w
addwf   CDivisor + 0, f
movf    tmp4, w
addwfc  CDivisor + 1, f

qstar_2

decfsz  tmp5, f
bra     qstar_1

movf    tmp1, w
movwf   preinc0           ; // Push tmp2 ( -- b )
movf    (tmp2), w

movwf   preinc0           ; // Push CDivisor + 0 ( -- b )
movf    (CDivisor + 0), w

movwf   preinc0           ; // Push CDivisor + 1 ( -- b )
movf    (CDivisor + 1), w

return

; 'Forth.*_8_32' =====
;
; ( q1 u -- q2 )
; Size = 0
;
C2A_8_32           ; 0:*_8_32

movwf   (tmp1)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp5)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp4)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   (tmp3)           ; // PopReg ( b -- )
movf    postdec0, w

movwf   tmp2

movlw   9                 ; // 9 Shifts
movwf   CQuotient

clrf    CDivisor         ; // clrf32
clrf    CDivisor + 1
clrf    CDivisor + 2
clrf    CDivisor + 3

bcf     Status, C

tstar_1

rrcf    CDivisor + 3, f           ; // Ignore byte 5
rrcf    CDivisor + 2, f
rrcf    CDivisor + 1, f
rrcf    CDivisor + 0, f
rrcf    tmp1, f
bnc     tstar_2           ; // B/ Carry clear, skip add

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

movf    tmp2, w
addwf   CDivisor + 0, f
movf    tmp3, w
addwfc  CDivisor + 1, f
movf    tmp4, w
addwfc  CDivisor + 2, f
movf    tmp5, w
addwfc  CDivisor + 3, f

tstar_2

decfsz  CQuotient, f
bra     tstar_1

movf    tmp1, w
movwf   preinc0           ; // Push CDivisor + 0 ( -- b )
movf    (CDivisor + 0), w

movwf   preinc0           ; // Push CDivisor + 1 ( -- b )
movf    (CDivisor + 1), w

movwf   preinc0           ; // Push CDivisor + 2 ( -- b )
movf    (CDivisor + 2), w

Return

; 'Forth./mod16' =====
;
; ( wDividend wDivisor -- wQuotient wRemainder )
; Size = 0
;

```

C2Fmod16

; 0:/mod16

```
; ////////////////////////////////////////////////////////////////////  
; // 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

```
movwf (CDivisor + 1) ; // PopReg ( b -- )  
movf postdec0, w
```

```
movwf (CDivisor + 0) ; // PopReg ( b -- )  
movf postdec0, w
```

```
movwf (CQuotient + 1) ; // PopReg ( b -- )  
movf postdec0, w
```

```
movwf CQuotient + 0
```

```
clrf CRemainder ; // clrfl6  
clrf CRemainder + 1
```

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

d1p\_16

```
rlcf CQuotient, f ; // shlc16  
rlcf CQuotient + 1, f
```

```
rlcf CRemainder, f ; // shlc16  
rlcf CRemainder + 1, f
```

```
bc subt_16 ; // jr c, subt_16
```

```
movf CRemainder + 1, w ; // cp dvsr_hi, rem_hi  
cpfsgt CDivisor + 1 ; // jr ugt, skp_16  
bra udiv_16_0 ; // B/ f <= W -> dvsr_hi <= rem_hi  
bra skp_16 ; // B/ f > W -> dvsr_hi > rem_hi
```

udiv\_16\_0 ; // dvsr\_hi <= rem\_hi

```
cpfseq CDivisor + 1 ; // jr ult, subt_16  
bra subt_16 ; // B/ dvsr_hi < rem_h
```

; // dvsr\_hi = rem\_hi

```
movf CRemainder, w ; // cp dvsr_lo, rem_lo  
cpfsgt CDivisor ; // jr ugt, skp_16  
bra subt_16 ; // B/ f <= W -> dvsr_lo <= rem_lo  
bra skp_16 ; // B/ f > W -> dvsr_lo > rem_lo
```

subt\_16

```
movf CDivisor, w ; // sub rem_lo, dvsr_lo
```

```
subwf CRemainder, f
```

```
movf CDivisor + 1, w ; // sub rem_hi, dvsr_hi
```

```
subwfb CRemainder + 1, f
```

```
bsf Status, C ; // scf
```

skp\_16

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

```
rlcf CQuotient, f ; // shlc16  
rlcf CQuotient + 1, f
```

; // Copy results to stack

```
movf CQuotient + 0, w  
movwf preinc0 ; // Push CQuotient + 1 ( -- b )  
movf (CQuotient + 1), w
```

```
movwf preinc0 ; // Push CRemainder + 0 ( -- b )  
movf (CRemainder + 0), w
```

```
movwf preinc0 ; // Push CRemainder + 1 ( -- b )  
movf (CRemainder + 1), w
```

```

return

; 'Forth./16' =====
;
; ( wDividend wDivisor -- wQuotient )
; Size = 0
;
C2F16                ; : 0:/16
                    ; // 0:/mod16 (code)
    call    C2Fmod16
    movf    postdec0, w      ; // drop ( b -- )

    movf    postdec0, w      ; // drop ( b -- )

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

; 'Forth./mod32' =====
;
; ( qDividend qDivisor -- qQuotient qRemainder )
; Size = 0
;
C2Fmod32             ; 0:/mod32

; ////////////////////////////////////////////////////////////////////
; // 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

movwf    (CDivisor + 3)      ; // PopReg ( b -- )
movf     postdec0, w

movwf    (CDivisor + 2)      ; // PopReg ( b -- )
movf     postdec0, w

movwf    (CDivisor + 1)      ; // PopReg ( b -- )
movf     postdec0, w

movwf    (CDivisor + 0)      ; // PopReg ( b -- )
movf     postdec0, w

movwf    (CQuotient + 3)     ; // PopReg ( b -- )
movf     postdec0, w

movwf    (CQuotient + 2)     ; // PopReg ( b -- )
movf     postdec0, w

movwf    (CQuotient + 1)     ; // PopReg ( b -- )
movf     postdec0, w

movwf    CQuotient + 0

clrf     CRemainder          ; // clrf32
clrf     CRemainder + 1
clrf     CRemainder + 2
clrf     CRemainder + 3

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

dlp_32

rlcf     CQuotient , f       ; // shlc32
rlcf     CQuotient + 1, f
rlcf     CQuotient + 2, f
rlcf     CQuotient + 3, f

rlcf     CRemainder , f      ; // shlc32
rlcf     CRemainder + 1, f
rlcf     CRemainder + 2, f
rlcf     CRemainder + 3, f

bc       subt_32            ; // jr  c, subt_32

movf     CRemainder + 3, w   ; // cp  dvsr_3, rem_3
cpfsgt  CDivisor + 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  CDivisor + 3       ; // jr  ult, subt_32
bra     subt_32            ; // B/ dvsr_3 < rem_3

; // dvsr_3 = rem_3

```

```

movf    CRemainder + 2, w ; // cp  dvsr_2, rem_2
cpfsgt  CDivisor    + 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  CDivisor + 2 ; // jr  ult,  subt_32
bra     subt_32      ; // B/ dvsr_2 < rem_2

; // dvsr_2 = rem_2

movf    CRemainder + 1, w ; // cp  dvsr_1, rem_1
cpfsgt  CDivisor    + 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  CDivisor + 1 ; // jr  ult,  subt_32
bra     subt_32      ; // B/ dvsr_2 < rem_2

; // dvsr_1 = rem_1

movf    CRemainder, w ; // cp  dvsr_lo, rem_lo
cpfsgt  CDivisor    ; // jr  ugt,  skp_32
bra     subt_32      ; // B/ f <= W -> dvsr_lo <= rem_lo
bra     skp_32       ; // B/ f > W -> dvsr_lo > rem_lo

subt_32

movf    CDivisor      , w ; // sub  rem_0, dvsr_0
subwfb  CRemainder   , f
movf    CDivisor    + 1, w ; // sub  rem_1, dvsr_1
subwfb  CRemainder   + 1, f
movf    CDivisor    + 2, w ; // sub  rem_2, dvsr_2
subwfb  CRemainder   + 2, f
movf    CDivisor    + 3, w ; // sub  rem_3, dvsr_3
subwfb  CRemainder   + 3, f
bsf     Status, C ; // scf

skp_32

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

rlcf    CQuotient   , f ; // shlc32
rlcf    CQuotient  + 1, f
rlcf    CQuotient  + 2, f
rlcf    CQuotient  + 3, f

; // Copy results to stack

movf    CQuotient  + 0, w
movwf   preinc0 ; // Push CQuotient + 1 ( -- b )
movf    (CQuotient + 1), w

movwf   preinc0 ; // Push CQuotient + 2 ( -- b )
movf    (CQuotient + 2), w

movwf   preinc0 ; // Push CQuotient + 3 ( -- b )
movf    (CQuotient + 3), w

movwf   preinc0 ; // Push CRemainder + 0 ( -- b )
movf    (CRemainder + 0), w

movwf   preinc0 ; // Push CRemainder + 1 ( -- b )
movf    (CRemainder + 1), w

movwf   preinc0 ; // Push CRemainder + 2 ( -- b )
movf    (CRemainder + 2), w

movwf   preinc0 ; // Push CRemainder + 3 ( -- b )
movf    (CRemainder + 3), w

return

; 'Forth./32' =====
;
; ( qDividend qDivisor -- qQuotient )
; Size = 0
;
C2F32 ; : 0:/32
; // 0:/mod32 (code)
call   C2Fmod32 ; // 0:4Drop (colon)
call   C4Drop ; // B/ ----> (i)
return ; // B/ ----> (i)

; 'Forth.d0' =====
;
; ( -- w )
; Size = 0
;
Cd0 ; : 0:d0
; // dliteral 0

```

```

movwf preinc0          ; // Push (D'0' >> 0) & D'255' ( -- b )
movlw ((D'0' >> 0) & D'255')

movwf preinc0          ; // Push (D'0' >> 8) & D'255' ( -- b )
movlw ((D'0' >> 8) & D'255')

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

; 'Forth.d0>' =====
;
; ( w -- f )
; Size = 0
;
Cd03E                  ; : 0:d0>
                        ; // 0:d0 (colon)
    call    Cd0          ; // 0:d> (code)
    call    Cd3E         ; // 0:d> (code)
    return   ; // B/ ----> ( ; )

; 'Forth.d0=' =====
;
; ( w -- f )
; Size = 0
;
Cd03D                  ; : 0:d0=
iorwf postdec0, w      ; // (or) ( b1 b2 -- b3 )
                        ; // 0:0= (code)
    call    C03D         ; // 0:0= (code)
    return   ; // B/ ----> ( ; )

; 'Forth.t@' =====
;
; ( wRamAddress -- w b )
; Size = 0
;
Ct40                   ; : 0:t@
; Aliased aDup Is (2dup)
                        ; // 0:(2dup) (colon)
    call    C282dup29    ; // 0:d@ (code)
    call    Cd40         ; // 0:dSwap (code)
    call    CdSwap       ; // 2a+ ( w1 -- w2 )
    incf   indf0, f      ; // 0:(@) (code)
    incf   indf0, f
    call    C284029     ; // B/ ----> ( ; )
    return

; 'Forth.t!' =====
;
; ( w b wRamAddress -- )
; Size = 0
;
Ct21                   ; : 0:t!
; Aliased (+rot) Is (rot)
                        ; // 0:(rot) (code)
    call    C28rot29     ; // literal 2
    movwf  preinc0      ; // Push D'2' ( -- b )
    movlw  (D'2')
                        ; // 0:(pick) (code)
    call    C28pick29    ; // literal 2
                        ; // Push D'2' ( -- b )
    movwf  preinc0      ; // 0:(pick) (code)
    movlw  (D'2')
                        ; // 2a+ ( w1 -- w2 )
    call    C28pick29    ; // 0:(!) (code)
    incf   indf0, f      ; // 0:d! (code)
    incf   indf0, f
    call    C282129     ; // 0:d! (code)
    call    Cd21
    return   ; // B/ ----> ( ; )

; 'Forth.q@' =====
;
; ( wRamAddress -- q )
; Size = 0
;
Cq40                   ; : 0:q@
; Aliased aDup Is (2dup)
                        ; // 0:(2dup) (colon)
    call    C282dup29    ; // 0:d@ (code)
    call    Cd40         ; // 0:dSwap (code)
    call    CdSwap       ; // 2a+ ( w1 -- w2 )
    incf   indf0, f
    incf   indf0, f

```

```

; // 0:d@ (code)
call Cd40
return ; // B/ ----> (i)

; 'Forth.q!' =====
;
; ( q wRamAddress -- )
; Size = 0
;
Cq21 ; : 0:q!
; // 0:dSwap (code)
call CdSwap

; Aliased aOver Is (2Over)
; // 0:(2Over) (code)
call C282Over29
; // 2a+ ( w1 -- w2 )
incf indf0, f
incf indf0, f
; // 0:d! (code)
call Cd21
; // 0:d! (code)
call Cd21
return ; // B/ ----> (i)

; 'Forth.2q@' =====
;
; ( wRamAddress -- q q )
; Size = 0
;
C2q40 ; : 0:2q@
; // (dup) ( b -- b b )
; // Dup ( b1 -- b1 b1 )
movwf preinc0
; // 0:RamPtr! (code)
call CRamPtr21
; // 0:q@ (colon)
call Cq40
; // 0:RamPtr@ (code)
call CRamPtr40
; // 0:4a+ (code)
call C4a2B
; // 0:q@ (colon)
call Cq40
return ; // B/ ----> (i)

; 'Forth.@Compare!' =====
;
; ( fn wRamAddress -- fc )
; Size = 0
;
C40Compare21 ; : 0:@Compare!
; // 0:adup@ (colon)
call Cadup40
; // literal 3
movwf preinc0
movlw (D'3')
; // 0:(pick) (code)
call C28pick29
xorwf postdec0, w
; // (xor) ( b1 b2 -- b3 )
; // 0:(>R) (code)
call C283ER29
; // 0:(!) (code)
call C282129
; // 0:(R>) (code)
call C28R3E29
return ; // B/ ----> (i)

; 'Forth.@0=' =====
;
; ( wRamAddress -- f )
; Size = 0
;
C4003D ; : 0:@0=
; // 0:(@) (code)
call C284029
; // 0:0= (code)
call C03D
return ; // B/ ----> (i)

; 'Forth.0>' =====
;
; ( u -- f )
; Size = 0
;
C03E ; : 0:0>
; // literal 0
; // Push D'0' ( -- b )
movwf preinc0
movlw (D'0')
; // 0:(>) (colon)
call C283E29
return ; // B/ ----> (i)

; 'Forth.@0>' =====
;
; ( wRamAddress -- f )
; Size = 0
;

```

```

C4003E                ; : 0:@0>
                    ; // 0:(@) (code)
    call    C284029    ; // 0:0> (colon)
    call    C03E      ; // B/ ----> (;)
    return

; 'Forth.@0!' =====
;
; ( wRamAddress -- b )
; Size = 0
;
C40021                ; : 0:@0!

; Aliased aDup Is (2dup)
                    ; // 0:(2dup) (colon)
    call    C282dup29 ; // Disable ( -- ), disable interrupts
    Disable
                    ; // 0:(@) (code)
    call    C284029    ; // 0:(-rot) (code)
    call    C282Drot29

; Aliased 0! Is False!
                    ; // 0:False! (code)
    call    CFalse21   ; // Enable ( -- ) // enable interrupts
    Enable
    return            ; // B/ ----> (;)

; 'Forth.@ff!' =====
;
; ( wRamAddress -- b )
; Size = 0
;
C40ff21               ; : 0:@ff!

; Aliased aDup Is (2dup)
                    ; // 0:(2dup) (colon)
    call    C282dup29 ; // Disable ( -- ), disable interrupts
    Disable
                    ; // 0:(@) (code)
    call    C284029    ; // 0:(-rot) (code)
    call    C282Drot29

; Aliased ff! Is True!
                    ; // 0:True! (code)
    call    CTrue21    ; // Enable ( -- ) // enable interrupts
    Enable
    return            ; // B/ ----> (;)

; 'Forth.4/' =====
;
; ( ul -- u2 )
; Size = 0
;
C42F                  ; 0:4/
    rrcf    wreg, f
    rrcf    wreg, f
    andlw   0x3f

    return

; 'Forth.8/' =====
;
; ( ul -- u2 )
; Size = 0
;
C82F                  ; 0:8/
    rrcf    wreg, f
    rrcf    wreg, f
    rrcf    wreg, f
    andlw   0x1f

    return

; 'Forth.32/' =====
;
; ( ul -- u2 )
; Size = 0
;
C322F                 ; 0:32/
    swapf   wreg, f
    rrcf    wreg, f
    andlw   0x07

    return

; 'Forth.64/' =====
;
; ( ul -- u2 )
; Size = 0
;
C642F                 ; 0:64/
    swapf   wreg, f
    rrcf    wreg, f

```

```

rrcf    wreg, f
andlw   0x03

return

; 'Forth.128/' =====
;
; ( u1 -- u2 )
; Size = 0
;
C1282F          ; 0:128/
    swapf    wreg, f
    rrcf     wreg, f
    rrcf     wreg, f
    rrcf     wreg, f
    andlw    0x01

    return

; 'Forth.3*' =====
;
; ( u1 -- u2 )
; Size = 0
;
C32A          ; 0:3*
    movwf    tmp1
    rlcfc    wreg, f
    andlw    0xfe
    addwf    tmp1, w
    return

; 'Forth.4*' =====
;
; ( u1 -- u2 )
; Size = 0
;
C42A          ; 0:4*
    rlcfc    wreg, f
    rlcfc    wreg, f
    andlw    0xfc

    return

; 'Forth.8*' =====
;
; ( u1 -- u2 )
; Size = 0
;
C82A          ; 0:8*
    rlcfc    wreg, f
    rlcfc    wreg, f
    rlcfc    wreg, f
    andlw    0xf8

    return

; 'Forth.32*' =====
;
; ( u1 -- u2 )
; Size = 0
;
C322A          ; 0:32*
    swapf    wreg, f
    rlcfc    wreg, f
    andlw    0xe0

    return

; 'Forth.64*' =====
;
; ( u1 -- u2 )
; Size = 0
;
C642A          ; 0:64*
    swapf    wreg, f
    rlcfc    wreg, f
    rlcfc    wreg, f
    andlw    0xc0

    return

; 'Forth.128*' =====
;
; ( u1 -- u2 )
; Size = 0
;
C1282A          ; 0:128*
    swapf    wreg, f
    rlcfc    wreg, f
    rlcfc    wreg, f
    rlcfc    wreg, f
    andlw    0x80

    return

; 'Forth.sp@' =====
;
; ( -- w )
; Size = 0
;

```

```

Csp40                                ; 0:sp@
movwf preinc0                        ; // Push fsr0l ( -- b )
movf (fsr0l), w

movwf preinc0                        ; // Push fsr0h ( -- b )
movf (fsr0h), w

return

; 'Forth.sp0' =====
;
; ( -- w )
; Size = 0
;
Csp0                                ; 0:sp0
movwf preinc0                        ; // Push ( SZero >> 0 ) & 0xff ( -- b )
movlw (( SZero >> 0 ) & 0xff)

movwf preinc0                        ; // Push ( SZero >> 8 ) & 0xff ( -- b )
movlw (( SZero >> 8 ) & 0xff)

return

; 'Forth.sp0?' =====
;
; ( -- f )
; Size = 0
;
Csp03F                              ; 0:sp0?
movwf preinc0                        ; // Push tmp1 ( -- b )
movf (tmp1), w

movlw (( SZero + 1 ) >> 0 ) & 0xff
cpfseq fsr0l                        ; // S/ stack not in error
retlw True                          ; // ----> stack error -> True

movlw (( SZero + 1 ) >> 8 ) & 0xff
cpfseq fsr0h                        ; // S/ stack Ok
retlw True                          ; // ----> stack error -> True
retlw False                         ; // ----> stack ok -> False

; 'Forth.@spSave' =====
;
; ( -- wRamAddress )
; Size = 0
;
C40spSave                            ; : 0:@spSave
movwf preinc0                        ; // 2variable spSave ( 0:135 ) ; CspSave
movlw (D'135')                      ; // Push D'135' ( -- b )

movwf preinc0                        ; // Push D'0' ( -- b )
movlw (D'0')

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

; 'Forth.spSave@' =====
;
; ( -- w )
; Size = 0
;
CspSave40                            ; : 0:spSave@
call C40spSave                      ; // 0:@spSave (colon)
call Cd40                          ; // 0:d@ (code)
return                               ; // B/ ----> ( ; )

; 'Forth.spSave!' =====
;
; ( w -- )
; Size = 0
;
CspSave21                            ; : 0:spSave!
call C40spSave                      ; // 0:@spSave (colon)
call Cd21                          ; // 0:d! (code)
return                               ; // B/ ----> ( ; )

; 'Forth.sp!' =====
;
; ( -- )
; Size = 0
;
Csp21                                ; : 0:sp!
call Csp40                          ; // 0:sp@ (code)
call CspSave21                      ; // 0:spSave! (colon)
return                               ; // B/ ----> ( ; )

; 'Forth.sp+!' =====
;
; ( b -- )
; Size = 0
;

```

```

Csp2B21                                ; : 0:sp+!
movwf  preinc0                          ; // Push D'0' ( -- b )
movlw  (D'0')

                                ; // 0:spSave@ (colon)
call   CspSave40

                                ; // 0:d+ (code)
call   Cd2B

                                ; // 0:spSave! (colon)
call   CspSave21
return                                ; // B/ ----> (i)

; 'Forth.sp-!' =====
;
; ( b -- )
; Size = 0
;
Csp2D21                                ; : 0:sp-!
                                ; // 0:(>R) (code)
call   C283ER29

                                ; // 0:spSave@ (colon)
call   CspSave40

                                ; // 0:(R>) (code)
call   C28R3E29
movwf  preinc0                          ; // Push D'0' ( -- b )
movlw  (D'0')

                                ; // 0:d- (code)
call   Cd2D

                                ; // 0:spSave! (colon)
call   CspSave21
return                                ; // B/ ----> (i)

; 'Forth.sp?' =====
;
; ( -- f )
; Size = 0
;
Csp3F                                  ; : 0:sp?
                                ; // 0:sp@ (code)
call   Csp40

                                ; // 0:spSave@ (colon)
call   CspSave40

                                ; // 0:d= (code)
call   Cd3D
return                                ; // B/ ----> (i)

; 'Forth.ld+' =====
;
; ( w1 -- w2 )
; Size = 0
;
C1d2B                                  ; : 0:ld+
                                ; // dliteral 1
movwf  preinc0                          ; // Push (D'1' >> 0) & D'255' ( -- b )
movlw  ((D'1' >> 0) & D'255')

                                ; // Push (D'1' >> 8) & D'255' ( -- b )
movwf  preinc0                          ; // Push (D'1' >> 8) & D'255' ( -- b )
movlw  ((D'1' >> 8) & D'255')

                                ; // 0:d+ (code)
call   Cd2B
return                                ; // B/ ----> (i)

; 'Forth.ld-' =====
;
; ( w1 -- w2 )
; Size = 0
;
C1d2D                                  ; : 0:ld-
                                ; // dliteral 1
movwf  preinc0                          ; // Push (D'1' >> 0) & D'255' ( -- b )
movlw  ((D'1' >> 0) & D'255')

                                ; // Push (D'1' >> 8) & D'255' ( -- b )
movwf  preinc0                          ; // Push (D'1' >> 8) & D'255' ( -- b )
movlw  ((D'1' >> 8) & D'255')

                                ; // 0:d- (code)
call   Cd2D
return                                ; // B/ ----> (i)

; 'Forth.-ld=' =====
;
; ( w -- f )
; Size = 0
;
C2D1d3D                                ; : 0:-ld=
                                ; // 2constant -ld ( 65535)
movwf  preinc0                          ; // Push (D'65535' >> 0) & D'255' ( -- b )
movlw  ((D'65535' >> 0) & D'255')

                                ; // Push (D'65535' >> 8) & D'255' ( -- b )
movwf  preinc0                          ; // Push (D'65535' >> 8) & D'255' ( -- b )
movlw  ((D'65535' >> 8) & D'255')

                                ; // 0:d= (code)
call   Cd3D
return                                ; // B/ ----> (i)

; 'Forth.d>=' =====
;

```

```

; ( w1 w2 -- f )
; Size = 0
;
Cd3E3D                ; : 0:d>=
                    ; // 0:d< (code)
    call    Cd3C
    xorlw   0xff
    return   ; // Not ( b1 -- b2 )
            ; // B/ ----> (i)

; 'Forth.d<=' =====
;
; ( w1 w2 -- f )
; Size = 0
;
Cd3C3D                ; : 0:d<=
                    ; // 0:d> (code)
    call    Cd3E
    xorlw   0xff
    return   ; // Not ( b1 -- b2 )
            ; // B/ ----> (i)

; 'Forth.d0<' =====
;
; ( d -- f )
; Size = 0
;
Cd03C                ; : 0:d0<
    decf    fsr01, f          ; // SwapDrop ( b1 b2 -- b2 )

                                ; // literal 128
    movwf   preinc0          ; // Push D'128' ( -- b )
    movlw   (D'128')

    andwf   postdec0, w      ; // (and) ( b1 b2 -- b3 )
                                ; // 0:0<> (code)

    call    C03C3E
    return   ; // B/ ----> (i)

; 'Forth.q+' =====
;
; ( uq1 uq2 -- uq3 )
; Size = 0
;
Cq2B                ; : 0:q+
    movwf   (CQuotient + 3)   ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (CQuotient + 2)   ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (CQuotient + 1)   ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (CQuotient + 0)   ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (CRemainder + 3)  ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (CRemainder + 2)  ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   (CRemainder + 1)  ; // PopReg ( b -- )
    movf    postdec0, w

    movwf   CRemainder + 0

    ; // Now calculate Remainder + Quotient -> Remainder

    movf    CQuotient + 0, w
    addwf   CRemainder + 0, f
    movf    CQuotient + 1, w
    addwfc  CRemainder + 1, f
    movf    CQuotient + 2, w
    addwfc  CRemainder + 2, f
    movf    CQuotient + 3, w
    addwfc  CRemainder + 3, f

    ; // Done

    movf    CRemainder + 0, w
    movwf   preinc0          ; // Push CRemainder + 1 ( -- b )
    movf    (CRemainder + 1), w

    movwf   preinc0          ; // Push CRemainder + 2 ( -- b )
    movf    (CRemainder + 2), w

    movwf   preinc0          ; // Push CRemainder + 3 ( -- b )
    movf    (CRemainder + 3), w

    return

; 'Forth.q-' =====
;
; ( uq1 uq2 -- uq3 )
; Size = 0
;
Cq2D                ; : 0:q-

```

```

movwf (CQuotient + 3)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CQuotient + 2)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CQuotient + 1)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CQuotient + 0)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CRemainder + 3)          ; // PopReg ( b -- )
movf  postdec0, w

movwf (CRemainder + 2)          ; // PopReg ( b -- )
movf  postdec0, w

movwf (CRemainder + 1)          ; // PopReg ( b -- )
movf  postdec0, w

movwf CRemainder + 0

; // Now calculate Remainder - Quotient -> Remainder

DoQMin

movf  CQuotient + 0, w
subwf CRemainder + 0, f
movf  CQuotient + 1, w
subwfb CRemainder + 1, f
movf  CQuotient + 2, w
subwfb CRemainder + 2, f
movf  CQuotient + 3, w
subwfb CRemainder + 3, f

; // Done

movf  CRemainder + 0, w
movwf preinc0                   ; // Push CRemainder + 1 ( -- b )
movf  (CRemainder + 1), w

movwf preinc0                   ; // Push CRemainder + 2 ( -- b )
movf  (CRemainder + 2), w

movwf preinc0                   ; // Push CRemainder + 3 ( -- b )
movf  (CRemainder + 3), w

return

; 'Forth.revq-' =====
;
; ( uq1 uq2 -- uq3 )
; Size = 0
;
Crevq2D                          ; 0:revq-
movwf (CRemainder + 3)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CRemainder + 2)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CRemainder + 1)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CRemainder + 0)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CQuotient + 3)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CQuotient + 2)           ; // PopReg ( b -- )
movf  postdec0, w

movwf (CQuotient + 1)           ; // PopReg ( b -- )
movf  postdec0, w

movwf CQuotient + 0

bra   DoQMin

; 'Forth.q-$80And>r3Dropr>' =====
;
; ( q1 q2 -- b )
; Size = 0
;
Cq2D2480And3Er3Dropr3E          ; : 0:q-$80And>r3Dropr>
; // 0:q- (code)
call  Cq2D                       ; // literal 128
movwf preinc0                   ; // Push D'128' ( -- b )
movlw (D'128')

andwf postdec0, w               ; // (and) ( b1 b2 -- b3 )
; // 0:(>R) (code)
call  C283ER29

```

```

        call    C3Drop                ; // 0:3Drop (colon)
        call    C28R3E29              ; // 0:(R>) (code)
        return  ; // B/ ----> (i)

; 'Forth.q<' =====
;
; ( q1 q2 -- f )
; Size = 0
;
Cq3C                ; : 0:q<
                   ; // 0:q-$80And>r3Dropr> (colon)
        call    Cq2D2480And3Er3Dropr3E ; // 0:0<> (code)
        call    C03C3E                ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.q>=' =====
;
; ( q1 q2 -- f )
; Size = 0
;
Cq3E3D              ; : 0:q>=
                   ; // 0:q-$80And>r3Dropr> (colon)
        call    Cq2D2480And3Er3Dropr3E ; // 0:0= (code)
        call    C03D                  ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.q>' =====
;
; ( q1 q2 -- f )
; Size = 0
;
Cq3E                ; : 0:q>
                   ; // 0:4swap (colon)
        call    C4swap                ; // 0:q< (colon)
        call    Cq3C                  ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.q<=' =====
;
; ( q1 q2 -- f )
; Size = 0
;
Cq3C3D              ; : 0:q<=
                   ; // 0:4swap (colon)
        call    C4swap                ; // 0:q>= (colon)
        call    Cq3E3D                ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.q0=' =====
;
; ( q -- f )
; Size = 0
;
Cq03D                ; : 0:q0=
                   ; // (or) ( b1 b2 -- b3 )
        iorwf  postdec0, w            ; // (or) ( b1 b2 -- b3 )
        iorwf  postdec0, w            ; // (or) ( b1 b2 -- b3 )
        iorwf  postdec0, w            ; // 0:0= (code)
        call    C03D                  ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.q=' =====
;
; ( q1 q2 -- f )
; Size = 0
;
Cq3D                ; : 0:q=
                   ; // 0:q- (code)
        call    Cq2D                  ; // 0:q0= (colon)
        call    Cq03D                 ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.q0<>' =====
;
; ( q -- f )
; Size = 0
;
Cq03C3E              ; : 0:q0<>
                   ; // (or) ( b1 b2 -- b3 )
        iorwf  postdec0, w            ; // (or) ( b1 b2 -- b3 )
        iorwf  postdec0, w            ; // (or) ( b1 b2 -- b3 )
        iorwf  postdec0, w            ; // 0:0<> (code)
        call    C03C3E                ; // B/ ----> (i)
        return  ; // B/ ----> (i)

; 'Forth.2Pick' =====
;
; ( b0 b1 b3 -- b0 b1 b2 b0 )
; Size = 0
;
C2Pick                ; 0:2Pick

; // Push 3rd entry

```

```

movwf  preinc0          ; // Push nothing, free wreg
movlw  -D'2'            ; // negated offset
movf   plusw0, w       ; // w = offset of reqd. byte,
return                ; //      0th byte references TOS

; 'Forth.3Pick' =====
;
; ( b0 b1 b2 b4 -- b0 b1 b2 b4 b0 )
; Size = 0
;
C3Pick                ; 0:3Pick

; // Push 4th entry

movwf  preinc0          ; // Push nothing, free wreg
movlw  -D'3'            ; // negated offset
movf   plusw0, w       ; // w = offset of reqd. byte,
return                ; //      0th byte references TOS

; 'Forth.4Pick' =====
;
; ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 b4 b0 )
; Size = 0
;
C4Pick                ; 0:4Pick

; // Push 5th entry

movwf  preinc0          ; // Push nothing, free wreg
movlw  -D'4'            ; // negated offset
movf   plusw0, w       ; // w = offset of reqd. byte,
return                ; //      0th byte references TOS

; 'Forth.4pickAnd>r' =====
;
; ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 | -- b )
; Size = 0
;
C4pickAnd3Er         ; : 0:4pickAnd>r
                    ; // 0:4Pick (code)
    call    C4Pick
    andwf   postdec0, w ; // (and) ( b1 b2 -- b3 )
                    ; // 0:(>R) (code)
    call    C283ER29
    return  ; // B/ ----> ( ; )

; 'Forth.4pickOr>r' =====
;
; ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 | -- b )
; Size = 0
;
C4pickOr3Er          ; : 0:4pickOr>r
                    ; // 0:4Pick (code)
    call    C4Pick
    iorwf   postdec0, w ; // (or) ( b1 b2 -- b3 )
                    ; // 0:(>R) (code)
    call    C283ER29
    return  ; // B/ ----> ( ; )

; 'Forth.4pickXor>r' =====
;
; ( b0 b1 b2 b3 b4 -- b0 b1 b2 b3 | -- b )
; Size = 0
;
C4pickXor3Er         ; : 0:4pickXor>r
                    ; // 0:4Pick (code)
    call    C4Pick
    xorwf   postdec0, w ; // (xor) ( b1 b2 -- b3 )
                    ; // 0:(>R) (code)
    call    C283ER29
    return  ; // B/ ----> ( ; )

; 'Forth.qAnd' =====
;
; ( qu1 qu2 -- qu3 )
; Size = 0
;
CqAnd                ; : 0:qAnd
                    ; // 0:4pickAnd>r (colon)
    call    C4pickAnd3Er
                    ; // 0:4pickAnd>r (colon)
    call    C4pickAnd3Er
                    ; // 0:4pickAnd>r (colon)
    call    C4pickAnd3Er
                    ; // 0:4pickAnd>r (colon)
    call    C4pickAnd3Er
                    ; // 0:4Drop (colon)
    call    C4Drop
                    ; // 0:qr> (colon)
    call    Cqr3E
    return  ; // B/ ----> ( ; )

; 'Forth.qOr' =====
;
; ( qu1 qu2 -- qu3 )
; Size = 0

```

```

;
CqOr                                ; : 0:qOr
; // 0:4pickOr>r (colon)
    call    C4pickOr3Er              ; // 0:4pickOr>r (colon)
    call    C4pickOr3Er              ; // 0:4pickOr>r (colon)
    call    C4pickOr3Er              ; // 0:4pickOr>r (colon)
    call    C4pickOr3Er              ; // 0:4pickOr>r (colon)
    call    C4Drop                    ; // 0:4Drop (colon)
    call    C4Drop                    ; // 0:qr> (colon)
    call    Cqr3E                      ; // B/ ----> (i)
return

; 'Forth.qXor' =====
;
; ( qu1 qu2 -- qu3 )
; Size = 0
;
CqXor                                ; : 0:qXor
; // 0:4pickXor>r (colon)
    call    C4pickXor3Er              ; // 0:4pickXor>r (colon)
    call    C4pickXor3Er              ; // 0:4pickXor>r (colon)
    call    C4pickXor3Er              ; // 0:4pickXor>r (colon)
    call    C4pickXor3Er              ; // 0:4pickXor>r (colon)
    call    C4Drop                    ; // 0:4Drop (colon)
    call    C4Drop                    ; // 0:qr> (colon)
    call    Cqr3E                      ; // B/ ----> (i)
return

; 'Forth.qNot' =====
;
; ( qu1 -- qu2 )
; Size = 0
;
CqNot                                ; : 0:qNot
; // 4constant -1q ( -1)
; // Push (-D'1' >> 0 ) & D'255' ( -- b )
    movwf   preinc0                    ; // Push (-D'1' >> 8 ) & D'255' ( -- b )
    movlw   ((-D'1' >> 0 ) & D'255')
    movwf   preinc0                    ; // Push (-D'1' >> 16) & D'255' ( -- b )
    movlw   ((-D'1' >> 8 ) & D'255')
    movwf   preinc0                    ; // Push (-D'1' >> 24) & D'255' ( -- b )
    movlw   ((-D'1' >> 16) & D'255')
    movwf   preinc0                    ; // Push (-D'1' >> 24) & D'255' ( -- b )
    movlw   ((-D'1' >> 24) & D'255')
    call    CqXor                      ; // 0:qXor (colon)
    return                               ; // B/ ----> (i)

; 'Forth.qNotAnd' =====
;
; ( qu1 qu2 -- qu3 )
; Size = 0
;
CqNotAnd                             ; : 0:qNotAnd
; // 0:qNot (colon)
    call    CqNot                      ; // 0:qAnd (colon)
    call    CqAnd                      ; // 0:qAnd (colon)
    return                               ; // B/ ----> (i)

; 'Forth.qNotAnd0<>' =====
;
; ( qu1 qu2 -- f )
; Size = 0
;
CqNotAnd03C3E                        ; : 0:qNotAnd0<>
; // 0:qNotAnd (colon)
    call    CqNotAnd                  ; // 0:q0<> (colon)
    call    Cq03C3E                  ; // 0:q0<> (colon)
    return                               ; // B/ ----> (i)

; 'Forth.main' =====
;
; ( -- )
; Size = 0
;
Cmain                                 ; : 0:main
    return                               ; // B/ ----> (i)

; Vocabulary end.
; ===== Code end.

; // Fill out program space with random numbers until
; // a modulo CFlashPageSize boundary is hit.
variable Seed = 12345

```

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

while $ % CFlashPageSize != 0
  Random
endw

; // :::::::::::::: Finalization ::::::::::::::::::::
End
```