```
 LIST N=0, R=DEC, F=INHX8M
;*****************************************************************************************
;
;       PROGRAM:                PLL.ASM
;
;       ASSEMBLER:              MPASMWIN v05.1
;
;       AUTHOR:                 Steven Jones         EMAIL stevejones@kern.com.au
;
;       REVISION HISTORY:
;
;       VERSION:        DATE            COMMENTS
;
;       1.00            MAR 26 2007     First release.
;
;*****************************************************************************************

;*****************************************************************************************
;
;               Initial Settings.
;
;               You may change these if required.
;               (1)  All frequencies are in KHz.
;               (2)  All the frequencies are checked to make sure they are valid.
;
;*****************************************************************************************

;  Label                        Value KHz.        Comments

_MIN_FREQ               =          2300000      ; 2300 MHz Minimum VCO frequency.
_MAX_FREQ               =          2500000      ; 2500 MHz Minimum VCO frequency.

_CH_1_FREQ              =          2330000      ; 2375 MHz, Channel 1 frequency.
_CH_2_FREQ              =          2335000      ; 2400 MHz, Channel 2 frequency.
_CH_3_FREQ              =          2340000      ; 2425 MHz, Channel 3 frequency.
_CH_4_FREQ              =          2345000      ; 2450 MHz, Channel 4 frequency.

_VCO_OFFSET_FREQ        =                0      ; For RX, -479000 KHz IF offset freq.
                                                ; (VCO FREQ = CH FREQ + VCO OFFSET FREQ)
                                                ; For TX, use VCO OFFSET FREQ of 0 KHz.

_PLL_XTAL_FREQ          =             3200      ; 4 MHz PLL crystal frequency.

                                                ; PLL config info. See SP5055 data sheet.
BYTE4                   EQU     B'10101110'     ; PLL charge pump current = 50uA,
BYTE5                   EQU     B'11111111'     ; PLL P6=ref, P7=F/div. O/P pins active.

;*****************************************************************************************
;
;               End of initial settings.
;
;               Do NOT make any changes beyond here.
;
;*****************************************************************************************

;*****************************************************************************************
;
;       DESCRIPTION:
;
;       programmable synthesizer using a PIC PIC12F629 and SP5055,
;
;       The PIC12F629 uses an internal 4 MHz clock = 1MHz instruction cycle time.
;
;       PLL frequency range of 120 MHz to 2.6 GHz.
;       Frequency steps of 125 KHz with a crystal reference of 4.0 MHz.
;
;       The program can be used for TX or RX.
;       For TX set the _VCO_OFFSET_FREQ label (see above) to 0 KHz.
;       For RX set the _VCO_OFFSET_FREQ label (see above) to -479000 KHz.
;
;       NORMAL MODE.
;
;       1)  This is the mode that will be enabled when power is first applied.
;
;       2)  The PLL frequency is set by switches S1 and S2.
;
;               S2      S1
```

```
;
;              open    open     = channel 1
;              open   closed    = channel 2
;            closed    open     = channel 3
;            closed   closed    = channel 4
;
;       3)  An optional PLL locked LED can be connected to pic pin 7
;           (see PLL_F629.pdf), this LED will light if the PLL is locked.
;
;       4)  An optional PROGRAM press button and PROGRAM LED can be connected to
;           pic pin 2 (see PLL_F629.pdf),
;           this enables you to program the current channel frequency, see below.
;
;       5)  A brief press of the optional press button puts the pic into the
;           ENTER CHANNEL FREQUENCY MODE.
;
;       ENTER CHANNEL FREQUENCY MODE.
;
;       1)  ENTER CHANNEL FREQUENCY MODE is entered from the NORMAL MODE by pressing
;           the optional PROGRAM press button briefly, the PROGRAM LED should flash
;           quickly for 1 second.
;
;       2)  The frequency is entered starting with the GHz digit.
;           Each brief press of the button, adds 1 GHz to the frequency.
;           ie 2 presses, sets the frequency to 2,xxx,xxx KHz.
;           The LED is turned on for 1/4 second as the button is released
;           as an aid to check that the PIC recognized the button press.
;
;       3)  A long press of the button (1 sec) selects the next digit, ie 100 MHz digit.
;           The LED will light after the button has been pressed for 1 second, and
;           once the button is released the LED will go out.
;           Then each brief press of the button, adds 100 MHz to the frequency.
;           ie 4 presses sets the frequency to 2,4xx,xxx KHz.
;           Again the LED is turned on for 1/4 sec as the button is released.
;
;       4)  A long press of the button (1 sec) selects the next digit, ie 10 MHz digit, etc.
;
;       5)  If you enter more than 9 in any digit position, the extra presses are ignored
;           as indicated by the LED not going on as the button is released.
;
;       6)  If a 0 needs to be entered in a digit position, dont briefly press the button,
;           but press the button for 1 second (until the LED lights) to select the
;           next digit.
;
;       7)  You can enter digits down to the 1 KHz position.
;           Any digits entered in the 100 Hz, 10 Hz or 1 Hz position are ignored, as
;           indicated by the LED not going on as the button is released.
;
;       8)  Pressing the button until the LED flashes saves the changes to
;           EEPROM and return you to the NORMAL MODE.
;           ( The PIC add the VCO OFFSET FREQUENCY to the frequency just entered,    )
;           ( then divides the result by 125 KHz to produce the PLL divider number,  )
;           ( and checks that the frequency is within the range of the VCO.          )
;           This new frequency result will be save to the currently selected channel
;           as defined by S1 and S2.
;
;       9)  You can save the frequency at any stage.
;               ie, to enter a frequency of 2,400,000 KHz (2.4 GHz)
;
;                   Briefly press the button 2 times, to enter 2 GHz.
;                   Press the button for 1 second to select the 100 MHz position.
;                   Briefly press the button 4 times, to enter 400 MHz.
;                   Press the button until the LED flashes to save the result
;                   to the currently selected channel as defined by S1 and S2.
;
;      10)  If the button is not pressed for 13 seconds the PROGRAM LED will flash quickly
;           for 1 second and you will be return to the NORMAL MODE, and any changes made
;           to the channel frequency will be ignored.
;
;*******************************************************************************
;       GENERAL COMMENTS
;
;       To minimize disturbance to the PLL, the divider info that sets the PLL, is sent
;       by the PIC to the PLL only if the frequency needs to be changed.
;       (ie after a channel change or after manual frequency changes)
;
;       When the frequency is changed, it is assumed that the PLL is unlocked.
```

```
;       The PIC checks the PLL status continuously until the PLL locks. Once locked,
;       status checks are made every second. if the PLL goes out of lock, continuous
;       checks are started again.
;       (to disable the checks every second, change the 'LOCK_TM' equate to 255)
;
;*******************************************************************************************
;
;       Programming notes.
;
;       1) To enable all of the required functions to be programmed into the limited code
;          space, any code that was used more than once or twice was changed to a
;          subroutine. And wherever a subroutine ended with a CALL and RETURN, eg
;               NOP
;               CALL    XXXXXX
;               RETURN
;
;          The code was changed to a GOTO.
;               NOP
;               GOTO    XXXXXX                  Return via "GOTO".
;
;          I know this is poor programming practice, but it did save a lot of code space,
;          and limited the usage of the 8 level hardware stack.
;          Each of these modifications as identified by the [Return via "GOTO".] comment.
;
;       2) The number in brackets in the comments, after each call statement indicates
;          the number of stack levels used by the call.
;          (I was checking that the stack never came near overflowing)
;
;       3) There is quite a bit of debug code in this asm file, used to drive an I2C
;          LCD display to monitor the PLL divider number.
;          This code is placed between  IF ENDIF statements, and is only assembled
;          if the DEBUG equate is set to TRUE.
;
;*******************************************************************************************

;*******************************************************************************************
;               SP5055 PLL info from data sheet.
;*******************************************************************************************

_PLL_MIN_FREQ           =           120000      ; 120 MHz PLL minimum input frequency.
_PLL_MAX_FREQ           =          2600000      ; 2.6 GHz PLL maximum input frequency.

_PLL_REF_DIV            EQU            512      ; PLL reference divider.
_PLL_PRESCALE           EQU             16      ; PLL RF prescaler divider.

;*******************************************************************************************
;               Convert frequencies to PLL divider numbers.
;*******************************************************************************************

_PLL_STEP_SIZE          = ( _PLL_XTAL_FREQ * _PLL_PRESCALE ) / _PLL_REF_DIV

_MAX_PLL_FREQ           = H'7FFF' * _PLL_STEP_SIZE

_VCO_MIN_FREQ           = _MIN_FREQ
_VCO_MAX_FREQ           = _MAX_FREQ
_CH_1_FREQ              = _CH_1_FREQ + _VCO_OFFSET_FREQ
_CH_2_FREQ              = _CH_2_FREQ + _VCO_OFFSET_FREQ
_CH_3_FREQ              = _CH_3_FREQ + _VCO_OFFSET_FREQ
_CH_4_FREQ              = _CH_4_FREQ + _VCO_OFFSET_FREQ

 IF _VCO_MIN_FREQ > _VCO_MAX_FREQ
 ERROR "VCO MIN Frequency is greater than VCO MAX Frequency"
 ENDIF

 IF _VCO_MAX_FREQ > _MAX_PLL_FREQ
_VCO_MAX_FREQ = _MAX_PLL_FREQ
 ERROR "VCO Frequency to high"
 ENDIF

_PLL_MIN_FREQ           = _PLL_MIN_FREQ / _PLL_STEP_SIZE
_PLL_MAX_FREQ           = _PLL_MAX_FREQ / _PLL_STEP_SIZE
_VCO_MIN_FREQ           = _VCO_MIN_FREQ / _PLL_STEP_SIZE
_VCO_MAX_FREQ           = _VCO_MAX_FREQ / _PLL_STEP_SIZE
_CH_1_FREQ              = _CH_1_FREQ    / _PLL_STEP_SIZE
_CH_2_FREQ              = _CH_2_FREQ    / _PLL_STEP_SIZE
_CH_3_FREQ              = _CH_3_FREQ    / _PLL_STEP_SIZE
_CH_4_FREQ              = _CH_4_FREQ    / _PLL_STEP_SIZE
```

```
;*******************************************************************************************
;               Make sure VCO divider numbers are within the PLL range.
;*******************************************************************************************

 IF _VCO_MIN_FREQ < _PLL_MIN_FREQ
_VCO_MIN_FREQ = _PLL_MIN_FREQ
 ERROR "MIN Frequency to low"
 ENDIF

 IF _VCO_MIN_FREQ > _PLL_MAX_FREQ
_VCO_MIN_FREQ = _PLL_MAX_FREQ
 ERROR "MIN Frequency to high"
 ENDIF

;-------------------------------------------------------------------------------------------

 IF _VCO_MAX_FREQ < _PLL_MIN_FREQ
_VCO_MAX_FREQ = _PLL_MIN_FREQ
 ERROR "MAX Frequency to low"
 ENDIF

 IF _VCO_MAX_FREQ > _PLL_MAX_FREQ
_VCO_MAX_FREQ = _PLL_MAX_FREQ
 ERROR "MAX Frequency to high"
 ENDIF

;*******************************************************************************************
;               Make sure channel frequencies are within the VCO range.
;*******************************************************************************************

 IF _CH_1_FREQ < _VCO_MIN_FREQ
_CH_1_FREQ = _VCO_MIN_FREQ
 ERROR "CH 1 Frequency to low"
 ENDIF

 IF _CH_1_FREQ > _VCO_MAX_FREQ
_CH_1_FREQ = _VCO_MAX_FREQ
 ERROR "CH 1 Frequency to high"
 ENDIF

;-------------------------------------------------------------------------------------------

 IF _CH_2_FREQ < _VCO_MIN_FREQ
_CH_2_FREQ = _VCO_MIN_FREQ
 ERROR "CH 2 Frequency to low"
 ENDIF

 IF _CH_2_FREQ > _VCO_MAX_FREQ
_CH_2_FREQ = _VCO_MAX_FREQ
 ERROR "CH 2 Frequency to high"
 ENDIF

;-------------------------------------------------------------------------------------------

 IF _CH_3_FREQ < _VCO_MIN_FREQ
_CH_3_FREQ = _VCO_MIN_FREQ
 ERROR "CH 3 Frequency to low"
 ENDIF

 IF _CH_3_FREQ > _VCO_MAX_FREQ
_CH_3_FREQ = _VCO_MAX_FREQ
 ERROR "CH 3 Frequency to high"
 ENDIF

;-------------------------------------------------------------------------------------------

 IF _CH_4_FREQ < _VCO_MIN_FREQ
_CH_4_FREQ = _VCO_MIN_FREQ
 ERROR "CH 4 Frequency to low"
 ENDIF

 IF _CH_4_FREQ > _VCO_MAX_FREQ
_CH_4_FREQ = _VCO_MAX_FREQ
 ERROR "CH 4 Frequency to high"
 ENDIF
```

```
;*****************************************************************************
;               Processor Definitions
;*****************************************************************************

 PROCESSOR      12F629
 __CONFIG       _WDT_OFF & _INTRC_OSC_NOCLKOUT & _BODEN_ON & _PWRTE_ON & _MCLRE_OFF

_BODEN_ON                       EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FEF'
_WDT_OFF                        EQU     H'3FF7'
_INTRC_OSC_NOCLKOUT             EQU     H'3FFC'
_INTRC_OSC_CLKOUT               EQU     H'3FFD'
_MCLRE_OFF                      EQU     H'3FDF'

 __MAXRAM H'FF'
 __BADRAM H'06'-H'09', H'0D', H'11'-H'14', H'17'-H'18', H'1E'-H'1F', H'60'-H'7F'
 __BADRAM H'86'-H'89', H'8D', H'8F', H'91'-H'94', H'97'-H'98', H'9E'-H'9F', H'E0'-H'FF'

;*****************************************************************************
;               Register Definitions
;*****************************************************************************

W               EQU             H'00'
F               EQU             H'01'
INDF            EQU             H'00'
TMR0            EQU             H'01'
PCL             EQU             H'02'
STATUS          EQU             H'03'
FSR             EQU             H'04'
GPIO            EQU             H'05'
PCLATH          EQU             H'0A'
INTCON          EQU             H'0B'
PIR1            EQU             H'0C'
CMCON           EQU             H'19'

;*****************************************************************************
;               Definitions that eliminate error 302
;*****************************************************************************

OPTION_REG      EQU             H'81' & H'7F'
TRISIO          EQU             H'85' & H'7F'
PIE1            EQU             H'8C' & H'7F'
OSCCAL          EQU             H'90' & H'7F'
WPU             EQU             H'95' & H'7F'
EEDATA          EQU             H'9A' & H'7F'
EEADR           EQU             H'9B' & H'7F'
EECON1          EQU             H'9C' & H'7F'
EECON2          EQU             H'9D' & H'7F'

;*****************************************************************************
;               Definitions of commonly used register bits
;*****************************************************************************

#DEFINE         CARRY           STATUS,0
#DEFINE         ZERO            STATUS,2
#DEFINE         RP0             STATUS,5
#DEFINE         RP1             STATUS,6
#DEFINE         IRP             STATUS,7
#DEFINE         T0IF            INTCON,2
#DEFINE         GIE             INTCON,7
#DEFINE         RD              EECON1,0
#DEFINE         WR              EECON1,1
#DEFINE         WREN            EECON1,2
#DEFINE         WRERR           EECON1,3
#DEFINE         EEIF            PIR1,7

;*****************************************************************************
;               General equates
;*****************************************************************************

TRUE            EQU             H'FF'
FALSE           EQU             H'00'
                                                ; Assembly options.
                                                ; To aid program development.

DEBUG           EQU             FALSE           ; FALSE = Disable I2C LCD display.
                                                ; TRUE = Enable I2C LCD display.
```

```
SIMULATE        EQU             FALSE           ; FALSE if not using the MPLAB simulator.
                                                ; TRUE if using the MPLAB simulator.

I2C_CLK_OC      EQU             FALSE           ; FALSE if I2C clock is O/P only.
                                                ; TRUE if I2C clock is bidirectional I/O.

;-------------------------------------------------------------------------------

RESETVECTOR     EQU              H'00'          ; PIC Reset vector.
INTVECTOR       EQU              H'04'          ; PIC Interrupt vector.
PAGE0           EQU              H'05'          ; Page 0. Constants & lookup tables.

MIN_ON          EQU              2              ; Min valid button pressed time. (2 x 16mS)
MAX_ON          EQU              35             ; Max valid button pressed time.(35 x 16mS)
TIMEOUT_NUM     EQU              3              ; Prog button timeout. 13 sec   (3 x 4.2 S)
LOCK_TM         EQU              61             ; Time between status checks.   (61 x 16mS)
                                                ; 255 = No checks once in lock.
 IF DEBUG
DEBUG_DISP      EQU              B'01001110'    ; I2C LCD display address. (for debugging)
PLL_ADDRESS     EQU              B'01000000'    ; Bus expander address.    (for debugging)
 ELSE
PLL_ADDRESS     EQU              B'11000010'    ; PLL chip address.
 ENDIF

DELAY_CONSTANT  EQU              13             ; I2C delay constant, for I2C clock rate.

CH_BYTES        EQU              16             ; 16 bytes for 4 CH's.

;*******************************************************************************
;               General definitions
;*******************************************************************************

;-------------------------------------------------------------------------------
;               GPIO pin definitions.
;-------------------------------------------------------------------------------

#DEFINE         PLL_LOCKED      GPIO,0          ; Pin 7, I/P. Set high (LED on) if PLL is locked.
#DEFINE         I2C_SDA         GPIO,1          ; Pin 6, I/O. I2C data.
#DEFINE         I2C_SCL         GPIO,2          ; Pin 5, I/O or O/P. I2C clock.
#DEFINE         S1              GPIO,3          ; Pin 4, I/P. Switch 1. Vpp.
#DEFINE         S2              GPIO,4          ; Pin 3, I/P. Switch 2.
#DEFINE         PROG_LED        GPIO,5          ; Pin 2, O/P. Program LED. On if O/P is low.
#DEFINE         PROG_BTN        GPIO,5          ;        I/P. Program button. Low if pressed.

 IF I2C_CLK_OC
GPIO_DIR        EQU             B'00011110'     ; Initial pin directions. 0 = O/P, 1 = I/P.
GPIO_INIT       EQU             B'00100010'     ; Initial pin level.      0 = Low, 1 = High.
 ELSE
GPIO_DIR        EQU             B'00011010'     ; Initial pin directions. 0 = O/P, 1 = I/P.
GPIO_INIT       EQU             B'00100110'     ; Initial pin level.      0 = Low, 1 = High.
 ENDIF

#DEFINE         I2C_SDA_DIR     TRISIO,1        ; 0 = O/P, 1 = I/P.
#DEFINE         I2C_SCL_DIR     TRISIO,2        ;        "
#DEFINE         LED_BTN_DIR     TRISIO,5        ;        "

;*******************************************************************************
;               GENERAL DEFINITIONS
;*******************************************************************************

#DEFINE         BUTTON_EN       DEBUG_PORT,0    ; I2C Button enable.  0 = Enabled
#DEFINE         LCD_RS          DEBUG_PORT,1    ; I2C LCD Reg select. 0 = Inst, 1 = Data
#DEFINE         LCD_E           DEBUG_PORT,2    ; I2C LCD Enable.     1 = Enable
#DEFINE         BK_LIGHT        DEBUG_PORT,3    ; I2C LCD Back light. 0 = On
;                               DEBUG_PORT,4-7  ; I2C LCD 4 bit data bus.

#DEFINE         PRESSED         FLAGS,0         ; Result flag from button pressed call.
#DEFINE         PROG_PRESSED    FLAGS,1         ; Set if the program button is pressed.
#DEFINE         TIMEOUT         FLAGS,2         ; If set, go back to main program.
#DEFINE         LEADING_FLAG    FLAGS,3         ; If set, time to stop leading blanking.
#DEFINE         WRITE_FLAG      FLAGS,4         ; Set if we want to do an EEPROM write.
#DEFINE         I2C_ACK_FLAG    FLAGS,5         ; Set if the I2C chip acknowledged the byte.
#DEFINE         I2C_TX_ACK_FLAG FLAGS,6         ; If set, the byte to the I2C bus will be
                                                ; followed by an acknowledge bit.

#DEFINE         ARG1            ARG1_7          ; Simpler definition for MSD of ARG's.
```

```
#DEFINE         ARG2            ARG2_3          ;               "
#DEFINE         ARG3            ARG3_3          ;               "

;*****************************************************************************
;               Bank 0 RAM variables
;               H'26' to H'53' cleared on power up.
;               H'54' to H'5F' reserved for ICD2.
;*****************************************************************************

        CBLOCK  H'20'                           ; Bank 0 ram 64 bytes. 20-5F hex.

                SAVE_STATUS                     ; Interrupt context is saved here.
                SAVE_W_REG                      ;               "
                                                ;
                COUNT_0                         ; General counter for subroutines.
                COUNT_1                         ;               "
                COUNT_2                         ;               "
                COUNT_3                         ;               "
                                                ;
                TEMP_1                          ; Temporary storage for subroutines.
                TEMP_2                          ;               "
                TEMP_3                          ;               "
                FLAGS                           ; 8 misc flags.
                                                ;
                DEBUG_PORT                      ; Debug ram.
                                                ;
                PROG_BTN_CNT                    ; Button pressed time count. (x 16mS)
                PROG_BTN_OLD                    ; State of old button count.
                                                ;
                LOCK_TIMER                      ; Timer for checking PLL status.
                TIMEOUT_H                       ; 16 Bit display timeout down counter,
                TIMEOUT_L                       ; Return to main display when = 0.
                FLASHER                         ; LED flasher timer.
                                                ;
                PLL_STATUS                      ; Status byte from PLL read.
                PLL_OLD:4                       ; Previous PLL divider number.
                                                ;
                SPARE:4                         ;
                                                ;
                POSITION                        ; Temporary storage for prog routine.
                PRESS_CNT                       ;       "
                                                ;
                ARG1_7                          ; MSD. \ \
                ARG1_6                          ;       | | ARG1, 32 bit maths buffer.
                ARG1_5                          ;       | |
                ARG1_4                          ; LSD.  |/
                ARG1_3                          ;       |
                ARG1_2                          ;       |   ARG1, 64 bit maths buffer.
                ARG1_1                          ;       |   (result from MULTIPLY)
                ARG1_0                          ; LSD. /

                ARG2_3                          ; MSD. \
                ARG2_2                          ;       |   ARG2, 32 bit maths buffer.
                ARG2_1                          ;       |
                ARG2_0                          ; LSD. /

                ARG3_3                          ; MSD. \
                ARG3_2                          ;       |   ARG3, 32 bit maths buffer.
                ARG3_1                          ;       |
                ARG3_0                          ; LSD. /

                CH_1_FREQ:4                     ; Copy of setup data from EEPROM.
                CH_2_FREQ:4                     ;               "
                CH_3_FREQ:4                     ;               "
                CH_4_FREQ:4                     ;               "
                                                ;
                SPARE_B:4                       ;
        ENDC

;*****************************************************************************
;               Lookup tables.
;*****************************************************************************

                ORG     PAGE0

;*****************************************************************************
;               START OF THE CONSTANT TABLES. PLACED IN PAGE 0.
```

```
;*******************************************************************************************

GET_CH_FREQ:    CLRW                            ; Get the channel number in use.
                BTFSS   S1                      ;
                ADDLW   1                       ;
                BTFSS   S2                      ;
                ADDLW   2                       ;
                ADDWF   PCL,F                   ;
                RETLW   CH_1_FREQ               ; Return with the correct CH variable.
                RETLW   CH_2_FREQ               ;
                RETLW   CH_3_FREQ               ;
                RETLW   CH_4_FREQ               ;

;*******************************************************************************************
;                       CONSTANT MACRO
;*******************************************************************************************

CON             MACRO   NAME, VAL               ; 4 Byte constant definition.
NAME            EQU $ - CONST_START
                DT (VAL>>24) & H'FF'
                DT (VAL>>16) & H'FF'
                DT (VAL>>8) & H'FF'
                DT (VAL) & H'FF'
                ENDM

CONST:          ADDWF   PCL,F
CONST_START:                                    ; Label the base address of the constants.

 CON    ROUND:                  , _PLL_STEP_SIZE/2
 CON    PLL_STEP_SIZE:          , _PLL_STEP_SIZE
 CON    VCO_OFFSET_FREQ:        , _VCO_OFFSET_FREQ
 CON    VCO_MIN_FREQ:           , _VCO_MIN_FREQ
 CON    VCO_MAX_FREQ:           , _VCO_MAX_FREQ

TABLE_END:

;*******************************************************************************************

                ORG     RESETVECTOR     ;

PROGRAM_START:  NOP                     ;       Required by ICD2.
                MOVLW   GPIO_INIT       ;       Set Initial GPIO levels.
                MOVWF   GPIO            ;                 "
                GOTO    CONTINUE        ;       Jump over interrupt routines & text table.

;*******************************************************************************************
;
;               Redirect the Interrupt service routine.
;               Lookup tables and text strings will be placed here. (page 0 and 1)
;
;*******************************************************************************************

                ORG     INTVECTOR       ;
                GOTO    INT_SERVICE     ;

;*******************************************************************************************
;
;               Continue with the program.
;
;*******************************************************************************************

                ORG     TABLE_END       ;
                                        ;
CONTINUE:       BSF     RP0             ;       Select bank 1.
                                        ;
                MOVLW   GPIO_DIR        ;       Set the initial GPIO directions.
                MOVWF   TRISIO          ;                       "
                                        ;
                MOVLW   B'00000101'     ;       Set prescaler to TMR0.
                MOVWF   OPTION_REG      ;       TMR0 rate = 1/64 of instruction clock.
                                        ;
                MOVLW   B'00011111'     ;       Enable pullups.
                MOVWF   WPU             ;
;                                        ;   ** Dont use because the RETLW at 3FF may **
;                CALL    H'3FF'         ;   ** have been erased by the programmer.   **
;                MOVWF   OSCCAL         ;   ** Set calibrated frequency.            **
                                       ;
```

```
                MOVLW   H'00'           ;
                MOVWF   OSCCAL          ;       Set min frequency.
                                        ;
                BCF     RP0             ;       Return to bank 0.
                                        ;
                MOVLW   B'00000111'     ;       Disable analog comparator.
                MOVWF   CMCON           ;
                                        ;
                MOVLW   H'26'           ;       Point to start of RAM we want to clear.
                MOVWF   FSR             ;
                MOVLW   H'2D'           ;       Number of bytes of RAM to clear.
                CALL    CLEAR_BYTES     ; (1)   Clear the RAM.
                                        ;
                CLRF    TMR0            ;       Sets 1st interupt to 8mS.
                MOVLW   B'10100000'     ;       Enable TMR0 overflow interupt.
                MOVWF   INTCON          ;
                                        ;
 IF DEBUG                               ;       Debugging code.
                MOVLW   100             ;
                CALL    MS_WAIT         ; (2)   Wait until the LCD has powered up.
                CALL    LCD_INIT        ; (6)   Setup the LCD display.
 ENDIF                                  ;

;*****************************************************************************************
;
;               Main program loop.
;
;               NORMAL MODE.
;
;               Set the PLL frequency as defined by switches S1 and S2.
;
;                 S2      S1
;
;                open    open     = channel 1
;                open   closed    = channel 2
;               closed   open     = channel 3
;               closed  closed    = channel 4
;
;               If the optional press button is pressed briefly go to the
;               ENTER CHANNEL FREQUENCY MODE.
;
;*****************************************************************************************

MAIN:           CALL    EE_TO_RAM       ; (3)   Get the main setup info from EEPROM.
                                        ;
MAIN_LOOP:      CALL    GET_CH_FREQ     ; (1)
                CALL    COPY_TO_ARG2    ; (1)   Get the channel frequency.
                CALL    UPDATE_PLL      ; (4)   Update the PLL if the freq has changed.
                                        ;       If PLL is locked, light the LED.
 IF DEBUG                               ;
                CALL    DISPLAY_FREQ    ; (6)   Debugging code. Display FREQ on the
 ENDIF                                  ;       I2C LCD display.
                CALL    CHK_PROG_BRIEF  ; (1)
                BTFSS   PRESSED         ;       Has the prog button been pressed briefly ?
                GOTO    MAIN_LOOP       ;       N. Continue normal mode.
                                        ;       Y. Enter channel frequency mode.

;*****************************************************************************************
;
;               ENTER CHANNEL FREQUENCY MODE.
;
;               ENTER CHANNEL FREQUENCY MODE is entered from the NORMAL MODE by pressing
;               the optional PROGRAM press button briefly, the PROGRAM LED will flash
;               quickly for 1 sec.
;
;               Each brief press of the button, adds 1 GHz to the frequency.
;               The LED is turned on for 1/4 second as the button is released
;               as an aid to check that the PIC recognized the button press.
;
;               A long press of the button (1 sec) selects the next digit, ie 100 MHz digit.
;               The LED will light after the button has been pressed for 1 second, and
;               once the button is released the LED will go out.
;               Then each brief press of the button, adds 100 MHz to the frequency.
;
;               A long press of the button (1 sec) selects the next digit, ie 10 MHz digit, etc.
;
;               If you enter more than 9 in any digit position, the extra presses are
```

```
;               ignored as indicated by the LED not going on as the button is released.
;
;               If a 0 needs to be entered in a digit position, dont briefly press the
;               button, but press the button for 1 second (until the LED lights) to
;               select the next digit.
;
;               You can enter digits down to the 1 KHz position.
;               Any digits entered in the 100 Hz, 10 Hz or 1 Hz position are ignored,
;               as indicated by the LED not going on as the button is released.
;
;               Pressing the button until the LED flashes saves the changes to
;               EEPROM and return you to the NORMAL MODE.
;               (The PIC add the VCO OFFSET FREQUENCY to the frequency just entered,  )
;               (then divides the result by 125 KHz to produce the PLL divider number,)
;               (and checks that the frequency is within the range of the VCO.        )
;               This new frequency result will be save to the currently selected channel
;               as defined by S1 and S2.
;
;               If the button is not pressed for 13 seconds the PROGRAM LED will flash
;               quickly for 1 second and you will be return to the NORMAL MODE, and any
;               changes made to the channel frequency will be ignored.
;
;               See the notes at the start of the program for further information.
;
;*******************************************************************************

ENTER_FREQ:     CALL    FLASH_LED       ; (1)   Flash the PROG_LED quickly for 1 sec.
                                        ;
                BSF     PROG_LED        ;       Turn the PROG_LED off.
                CALL    CLEAR_ARG2      ; (1)   ARG2 will hold the frequency in KHz.
                                        ;
                MOVLW   7               ;       Start with digit 7, GHz position.
                MOVWF   POSITION        ;
                                        ;
NEXT_DIGIT:     MOVLW   9               ;       Max 9 presses per digit position.
                MOVWF   PRESS_CNT       ;
                                        ;
                DECF    POSITION,W      ;       Get the current digit position - 1.
                CALL    TEN_TO_POWER_W  ; (4)   Convert it to a power of 10.
                                        ;
                BCF     TIMEOUT         ;       Clear the timeout flag.
ENTER_LOOP:     BTFSC   TIMEOUT         ;       Has the timeout been reached ?
                GOTO    PROG_EXIT       ;       Y. Go back to normal operation.
                                        ;
                MOVF    POSITION,W      ;       Is POSITION = 0 ?
                BTFSC   ZERO            ;       (past the KHz position)
                GOTO    NO_ADD          ;       Y. Ignore brief button presses.
                                        ;
                MOVF    PRESS_CNT,W     ;       If the button is pressed > 9 times,
                BTFSC   ZERO            ;       Ignore brief button presses.
                GOTO    NO_ADD          ;
                                        ;
                CALL    CHK_PROG_BRIEF  ; (1)
                BTFSS   PRESSED         ;       Has the prog button been pressed briefly ?
                GOTO    NO_ADD          ;
                                        ;
                CALL    ADD             ; (3)   Y. Add the current power to ARG2.
                DECF    PRESS_CNT,F     ;          Dec the digit counter.
                                        ;
                BCF     PROG_LED        ;          Turn the PROG_LED on for 256 mS.
                MOVLW   0               ;          To show that the button press
                CALL    MS_WAIT         ; (2)      was recognised.
                BSF     PROG_LED        ;          Turn off the PROG_LED.
NO_ADD:                                 ;
                                        ;
 IF DEBUG                               ;       Debugging code.
                CALL    DISPLAY_ARG2    ; (6)   Display ARG2 as a FREQ on the
 ELSE                                   ;       I2C LCD display.
                MOVLW   50              ;
                CALL    MS_WAIT         ; (2)   Wait a while.
 ENDIF                                  ;
                BTFSS   PROG_BTN_CNT,6  ;       Prog button pressed for 1 sec ?
                GOTO    ENTER_LOOP      ;       N. Continue entering frequency.
                                        ;
                BCF     PROG_LED        ;       Y. Turn on the PROG_LED.
WAIT_RELEASE:   INCF    PROG_BTN_CNT,W  ;          Has prog button been pressed for 4 sec ?
                BTFSC   ZERO            ;
```

```
                GOTO    FINISH          ;           Y. Finished.
                                        ;
                BTFSC   PROG_PRESSED    ;           Wait until the button is released.
                GOTO    WAIT_RELEASE    ;
                BSF     PROG_LED        ;           Turn off the PROG_LED.
                                        ;
                MOVLW   100             ;
                CALL    MS_WAIT         ; (2)       Wait a while.
                                        ;
                CALL    CHK_PROG_BRIEF  ; (1)       Update the Prog button.
                                        ;
                MOVF    POSITION,W      ;           Is POSITION = 0 ?
                BTFSC   ZERO            ;           (past the KHz position)
                GOTO    NO_ADD          ;           Y. Dont select next lower digit position.
                                        ;
                DECF    POSITION,F      ;           Select next lower digit position.
                GOTO    NEXT_DIGIT      ;         N. Continue with next digit of freq.
                                        ;
FINISH:         MOVLW   VCO_OFFSET_FREQ ;         (Convert result to a divider number)
                CALL    CONST_TO_ARG3   ; (1)     Get the VCO offset constant.
                CALL    ADD             ; (3)     Add it to the freq just entered.
                MOVLW   ROUND           ;
                CALL    CONST_TO_ARG3   ; (1)
                CALL    ADD             ; (3)     Add 1/2 PLL step size for rounding.
                BTFSC   ARG2_3,7        ;         Was the result a - number.
                CALL    CLEAR_ARG2      ; (1)     Y. Set it back to 0 KHz
                MOVLW   PLL_STEP_SIZE   ;         Get PLL step size constant.
                CALL    CONST_TO_ARG3   ; (1)     PLL divider nummer =
                CALL    DIVIDE          ; (3)     Freq + VCO offset / step size.
                                        ;
                MOVLW   VCO_MIN_FREQ    ;         Get VCO min freq constant.
                CALL    CONST_TO_ARG3   ; (1)
                CALL    MAXIMUM         ; (3)     Make sure we dont go below the minimum.
                                        ;
                MOVLW   VCO_MAX_FREQ    ;         Get VCO max freq constant.
                CALL    CONST_TO_ARG3   ; (1)
                CALL    MINIMUM         ; (3)     Make sure we dont go above the maximum.
                                        ;
                CALL    GET_CH_FREQ     ; (1)     Save the result to the channel in use.
                CALL    COPY_FROM_ARG2  ; (1)
                                        ;
                CALL    RAM_TO_EE       ; (3)     Update the EEPROM.
                                        ;
PROG_EXIT:      CALL    FLASH_LED       ; (1)     Flash the PROG_LED quickly for 1 sec.
                BSF     PROG_LED        ;         Turn off the PROG_LED.
                                        ;
                BTFSC   PROG_PRESSED    ;         Wait until the button is released.
                GOTO    $-1             ;
                                        ;
                MOVLW   100             ;
                CALL    MS_WAIT         ; (2)     Wait a while.
                                        ;
                CALL    CHK_PROG_BRIEF  ; (1)     Update the Prog button.
                                        ;
                GOTO    MAIN            ;         Go back to normal operation.

;*****************************************************************************
;
;     NAMES:    FLASH_LED             Flash the PROG_LED quickly for 1 sec.
;
; VARIABLES:    None.
;
; STACK USE:    0
;
;*****************************************************************************

FLASH_LED:      CLRF    FLASHER         ;         Clear the LED flasher counter.
WAIT_FLASH:     BTFSC   FLASHER,2       ;         Flash the PROG_LED quickly.
                BSF     PROG_LED        ;
                BTFSS   FLASHER,2       ;
                BCF     PROG_LED        ;
                BTFSS   FLASHER,6       ;         Been flashing for 1 sec ?
                GOTO    WAIT_FLASH      ;         N. Wait for 1 second.
                RETURN                  ;

;*****************************************************************************
;
```

```
;     NAMES:    ADD                     ARG2 = ARG2 + ARG3.     (all routines are 32 bit)
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    2
;
;*****************************************************************************************

ADD:            CALL    ARG2_TO_ARG1    ; (1)   ARG1 = ARG2.
                CALL    ADD_ARG1_ARG3   ; (2)   ARG1 = ARG2 + ARG3.
                GOTO    ARG1_TO_ARG2    ; (0)   ARG2 = ARG2 + ARG3. Return via "GOTO".

;*****************************************************************************************
;
;     NAMES:    SUB                     ARG2 = ARG2 - ARG3.     (all routines are 32 bit)
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    1
;
;*****************************************************************************************

SUB:            CALL    ARG2_TO_ARG1    ; (1)   ARG1 = ARG2.
                CALL    SUB_ARG1_ARG3   ; (1)   ARG1 = ARG2 - ARG3.
                GOTO    ARG1_TO_ARG2    ; (0)   ARG2 = ARG2 - ARG3. Return via "GOTO".

;*****************************************************************************************
;
;     NAMES:    ADD_ARG1_ARG3           ARG1 = ARG1 + ARG3.     (all routines are 32 bit)
;
; VARIABLES:    FSR
;
; STACK USE:    1
;
;*****************************************************************************************

ADD_ARG1_ARG3:  CALL    NEG_ARG3        ; (1)   ARG3 = -ARG3.
                CALL    SUB_ARG1_ARG3   ; (1)   ARG1 = ARG2 + ARG3.
                GOTO    NEG_ARG3        ; (0)   ARG3 = ARG3. Return via "GOTO".

;*****************************************************************************************
;
;     NAMES:    SUB_ARG1_ARG3           ARG1 = ARG1 - ARG3.     (all routines are 32 bit)
;
; VARIABLES:    FSR
;
; STACK USE:    0
;
;*****************************************************************************************

SUB_ARG1_ARG3:  MOVF    ARG3_0,W        ;
                SUBWF   ARG1_4,F        ;       Sub the 1st (LSD) bytes.
                MOVF    ARG3_1,W        ;
                BTFSS   CARRY           ;
                INCFSZ  ARG3_1,W        ;       If there was a carry, inc the next byte.
                                        ;
                SUBWF   ARG1_5,F        ;       Sub the 2nd bytes.
                MOVF    ARG3_2,W        ;
                BTFSS   CARRY           ;
                INCFSZ  ARG3_2,W        ;       If there was a carry, inc the next byte.
                                        ;
                SUBWF   ARG1_6,F        ;       Sub the 3rd bytes.
                MOVF    ARG3_3,W        ;
                BTFSS   CARRY           ;
                INCFSZ  ARG3_3,W        ;       If there was a carry, inc the next byte.
                                        ;
                SUBWF   ARG1_7,F        ;       Sub the 4th bytes.
                RETURN                  ;

;*****************************************************************************************
;
;     NAMES:    NEG_ARG2                ARG2 = -ARG2.           (all routines are 32 bit)
;               NEG_ARG3                ARG3 = -ARG3.
;
; VARIABLES:    FSR.
;
; STACK USE:    0
```

```
;
;*****************************************************************************************

NEG_ARG3:       MOVLW   ARG3            ;
                GOTO    NEG_X           ;
NEG_ARG2:       MOVLW   ARG2            ;
NEG_X:          MOVWF   FSR             ;       Place address of ARG in FSR
                COMF    INDF,F          ;       Compliment the arg.
                INCF    FSR,F           ;
                COMF    INDF,F          ;
                INCF    FSR,F           ;
                COMF    INDF,F          ;
                INCF    FSR,F           ;
                COMF    INDF,F          ;
                                        ;
                INCFSZ  INDF,F          ;       Increment the arg.
                RETURN                  ;
                DECF    FSR,F           ;
                INCFSZ  INDF,F          ;
                RETURN                  ;
                DECF    FSR,F           ;
                INCFSZ  INDF,F          ;
                RETURN                  ;
                DECF    FSR,F           ;
                INCF    INDF,F          ;
                RETURN                  ;       ARGx = -ARGx.

;*****************************************************************************************
;
;     NAMES:    CLEAR_ARG1              ARG1 = 0.
;               CLEAR_ARG2              ARG2 = 0.
;               CLEAR_ARG3              ARG3 = 0.
;
; VARIABLES:    FSR, COUNT_1.
;
; STACK USE:    0
;
;*****************************************************************************************

CLEAR_ARG1:     MOVLW   ARG1            ;       Clear all 8 bytes.
                MOVWF   FSR             ;
                MOVLW   8               ;
                GOTO    CLEAR_BYTES     ;

CLEAR_ARG3:     MOVLW   ARG3            ;
                GOTO    CLEAR           ;
CLEAR_ARG2:     MOVLW   ARG2            ;
CLEAR:          MOVWF   FSR             ;
                MOVLW   4               ;
CLEAR_BYTES:    MOVWF   COUNT_1         ;       Save the count.
CLEAR_LOOP:     CLRF    INDF            ;       Clear the RAM byte.
                INCF    FSR,F           ;       Inc the RAM address pointer.
                DECFSZ  COUNT_1,F       ;       Dec the count.
                GOTO    CLEAR_LOOP      ;       Loop until all bytes cleared.
                RETURN                  ;

;*****************************************************************************************
;
;      NAME:    COMPARE
;
;   PURPOSE:    32 bit compare.  ARG2 - ARG3
;
;     INPUT:    32 bit argment ARG2.
;               32 bit argment ARG3.
;
;    OUTPUT:    None.
;               Carry flag clear (borrow) if result was negative.
;               Zero flag set if equal.
;               (ARG2 & ARG3 unchanged)
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    1
;
;*****************************************************************************************

COMPARE:        CALL    OFFSET_ARGS     ; (1)   Add a large offset to ARG2 & ARG3.
```

```
                                        ;       (makes sure we are dealing with)
                CALL    ARG2_TO_ARG1    ; (1)   (positive numbers only         )
                CALL    SUB_ARG1_ARG3   ; (1)   Do the compare. Result in ARG1.
OFFSET_ARGS:    MOVLW   B'10000000'     ;       Remove the offset from ARG2 and ARG3.
                XORWF   ARG2_3,F        ;
                XORWF   ARG3_3,F        ;
                MOVF    ARG1_7,W        ;       Get the result for zero check.
                IORWF   ARG1_6,W        ;       (sets zero flag if result was 0)
                IORWF   ARG1_5,W        ;
                IORWF   ARG1_4,W        ;
                RETURN                  ;

;*****************************************************************************************
;
;      NAME:    MINIMUM
;
;   PURPOSE:    32 bit minimum. Return the smallest of two argments.
;
;       INPUT:  32 bit argment in ARG2.
;               32 bit argment in ARG3.
;
;       OUTPUT: The smallest argment is returned in ARG2.
;               (ARG3 unchanged)
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    2
;
;*****************************************************************************************

MINIMUM:        CALL    COMPARE         ; (2)   ARG2 - ARG3
                BTFSC   CARRY           ;
                CALL    ARG3_TO_ARG2    ; (1)   Y. Set ARG2 the same as ARG3.
                RETURN                  ;       N. No change required.

;*****************************************************************************************
;
;      NAME:    MAXIMUM
;
;   PURPOSE:    32 bit maximum. Return the largest of two argments.
;
;       INPUT:  32 bit argment in ARG2.
;               32 bit argment in ARG3.
;
;       OUTPUT: The largest argment is returned in ARG2.
;               (ARG3 unchanged)
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    2
;
;*****************************************************************************************

MAXIMUM:        CALL    COMPARE         ; (2)   ARG2 - ARG3
                BTFSS   CARRY           ;       Is ARG2 > ARG3 ?
                CALL    ARG3_TO_ARG2    ; (1)   N. Set ARG2 the same as ARG3.
                RETURN                  ;       Y. No change required.

;*****************************************************************************************
;
;      NAME:    DIVIDE
;
;   PURPOSE:    32 bit / 32 bit unsigned divide.    (ARG2 = ARG2 / ARG3)
;
;     INPUT:    ARG2 (+ numbers only )
;               ARG3 (+ numbers only )
;
;    OUTPUT:    ARG2 = result
;               (ARG3 unchanged)
;
; VARIABLES:    FSR, COUNT_1, COUNT_3, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    2
;
;*****************************************************************************************

DIVIDE:         CALL    CLEAR_ARG1      ; (1)
```

```
                MOVLW   32              ;       32 bits to divide.
                MOVWF   COUNT_3         ;
                                        ;
DLOOP:          BCF     CARRY           ;
                RLF     ARG2_0,F        ;       Rotate dividend left 1 bit position.
                RLF     ARG2_1,F        ;
                RLF     ARG2_2,F        ;
                RLF     ARG2_3,F        ;
                RLF     ARG1_4,F        ;       Rotate remainder left 1 bit position.
                RLF     ARG1_5,F        ;
                RLF     ARG1_6,F        ;
                RLF     ARG1_7,F        ;
                                        ;
                BTFSC   CARRY           ;       Invert carry and exclusive or with the
                GOTO    CLR_LSB         ;       MSB of the divisor then move this bit
                BTFSS   ARG3_3,7        ;       into the LSB of the dividend.
                INCF    ARG2_0,F        ;
                GOTO    CONT            ;
CLR_LSB:        BTFSC   ARG3_3,7        ;
                INCF    ARG2_0,F        ;
                                        ;
CONT:           BTFSS   ARG2_0,0        ;       Is the LSB of the dividend =0 ?
                CALL    ADD_ARG1_ARG3   ; (2)   Y. ARG1_4..0 = ARG1_4..0 + ARG3_4..0
                BTFSC   ARG2_0,0        ;
                CALL    SUB_ARG1_ARG3   ; (1)   N. ARG1_4..0 = ARG1_4..0 - ARG3_4..0
                                        ;
                DECFSZ  COUNT_3,F       ;       Do 32 times.
                GOTO    DLOOP           ;
                                        ;
                BCF     CARRY           ;
                RLF     ARG2_0,F        ;       Shift lower 32 bits of dividend 1 bit
                RLF     ARG2_1,F        ;       Position left
                RLF     ARG2_2,F        ;
                RLF     ARG2_3,F        ;
                BTFSC   ARG1_7,7        ;       Exclusive or the inverse of the MSB of
                GOTO    CHK_1           ;       the dividend with the MSB of the divisor
                BTFSS   ARG3_3,7        ;       store in the LSB of the dividend.
                INCF    ARG2_0,F        ;
                GOTO    DIV_END         ;
CHK_1:          BTFSC   ARG3_3,7        ;
                INCF    ARG2_0,F        ;
DIV_END:        RETURN                  ;

;*************************************************************************************
;
;      NAME:    ARG3_TIMES_W
;
;   PURPOSE:    Calculate ARG3 x W (0 to 255 times)
;
;     INPUT:    W = Multiplier.
;
;    OUTPUT:    ARG3 = ARG3 x W.
;
;  RAM USED:    FSR, COUNT_0, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    2
;
;*************************************************************************************

ARG3_TIMES_W:   ADDLW   1               ;       W = multiplier.
                MOVWF   COUNT_0         ;       Save the multiplier + 1.
                CALL    CLEAR_ARG1      ; (1)   ARG1 = 0.
                GOTO    X_LOOP_IP       ;
X_LOOP:         CALL    ADD_ARG1_ARG3   ; (2)   Add ARG3 to ARG1, W times.
X_LOOP_IP:      DECFSZ  COUNT_0,F       ;
                GOTO    X_LOOP          ;
                MOVLW   ARG1            ;       Place the result into ARG3.
                GOTO    COPY_TO_ARG3    ; (0)   Return via "GOTO".

;*************************************************************************************
;
;      NAME:    TEN_TO_POWER_W
;                         (W)
;   PURPOSE:    Calculate 10^
;
;     INPUT:    W = Power.
;                       (W)
```

```
;    OUTPUT:    ARG3 = 10^
;
; VARIABLES:    FSR, COUNT_0, COUNT_1. COUNT_3, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    3
;
;*****************************************************************************************

TEN_TO_POWER_W: ADDLW   1               ;
                MOVWF   COUNT_3         ;       Save the power + 1.
                CALL    CLEAR_ARG3      ; (1)
                INCF    ARG3_0,F        ;       Set ARG3 to 1.
                GOTO    END_TEST        ;
POWER_LOOP:     MOVLW   10              ;
                CALL    ARG3_TIMES_W    ; (3)
END_TEST:       DECFSZ  COUNT_3,F       ;       Keep multiplying by ten until done.
                GOTO    POWER_LOOP      ;
                RETURN                  ;

;*****************************************************************************************
;
;      NAME:    CON2_TO_ARG3
;
;   PURPOSE:    Place the 4 byte constant pointed to by W, into ARG3.
;
;     INPUT:    W = Pointer
;
;    OUTPUT:    ARG3 = Requested constant
;
; VARIABLES:    TEMP_1, COUNT_1.
;
; STACK USE:    1
;
;*****************************************************************************************

CONST_TO_ARG3:  MOVWF   TEMP_1          ;       Save the pointer to the required const.
                MOVLW   ARG3_3          ;       Set destination pointer to ARG3.
                MOVWF   FSR             ;
                MOVLW   4               ;
                MOVWF   COUNT_1         ;       4 Bytes to get.
                                        ;
CON_LOOP:       MOVF    TEMP_1,W        ;       Get the pointer.
                CALL    CONST           ; (1)   Get the byte from the constant table.
                MOVWF   INDF            ;       Place the byte at destination address.
                INCF    FSR,F           ;
                INCF    TEMP_1,F        ;       Inc dest address.
                DECFSZ  COUNT_1,F       ;
                GOTO    CON_LOOP        ;       Loop until all bytes copied.
                RETURN                  ;

;*****************************************************************************************
;
;     NAMES:    COPY_TO_ARG2            Copy 4 bytes pointed to by W to ARG2.
;               COPY_TO_ARG3            Copy 4 bytes pointed to by W to ARG3.
;
;               COPY_FROM_ARG2          Copy ARG2 to the address pointed to by W.
;               COPY_FROM_ARG3          Copy ARG3 to the address pointed to by W.
;
;               ARG2_TO_ARG1            Copy ARG2 to ARG1.
;               ARG2_TO_ARG3            Copy ARG2 to ARG3.
;               ARG1_TO_ARG3            Copy ARG1 to ARG3.
;               ARG1_TO_ARG2            Copy ARG1 to ARG2.
;               ARG3_TO_ARG2            Copy ARG3 to ARG2.
;
;   PURPOSE:    Move 4 bytes of RAM from source address to dest address.
;               (starting at source and dest, then source+1 dest+1, etc.)
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
;               TEMP_1 = destination address.
;               TEMP_2 = source address.
;               COUNT_1 = bytes to move.
;
; STACK USE:    0
;
;*****************************************************************************************
```

```
ARG2_TO_ARG1:    MOVLW   ARG1            ;       \
COPY_FROM_ARG2:  MOVWF   TEMP_1          ;        \
                 MOVLW   ARG2            ;         \
                 MOVWF   TEMP_2          ;          |
                 GOTO    COPY_4          ;          |
                                         ;          |
ARG3_TO_ARG2:    MOVLW   ARG3            ;          |
                 GOTO    COPY_TO_ARG2    ;          |
ARG1_TO_ARG2:    MOVLW   ARG1            ;          |
COPY_TO_ARG2:    MOVWF   TEMP_2          ;           \  set the various source and
                 MOVLW   ARG2            ;           /  destination addresses.
                 GOTO    COPY            ;          |
                                         ;          |
COPY_FROM_ARG3:  MOVWF   TEMP_1          ;          |
                 MOVLW   ARG3            ;          |
                 MOVWF   TEMP_2          ;          |
                 GOTO    COPY_4          ;          |
                                         ;         /
COPY_TO_ARG3:    MOVWF   TEMP_2          ;        /
                 MOVLW   ARG3            ;       /
                                         ;
COPY:            MOVWF   TEMP_1          ;
COPY_4:          MOVLW   4               ;       4 bytes to move.
                 MOVWF   COUNT_1         ;
COPY_LOOP:       MOVF    TEMP_2,W        ;       Get source address.
                 MOVWF   FSR             ;              "
                 MOVF    INDF,W          ;       Get the byte from the source address.
                 MOVWF   TEMP_3          ;                     "
                 MOVF    TEMP_1,W        ;       Get destination address.
                 MOVWF   FSR             ;               "
                 MOVF    TEMP_3,W        ;       Get the source byte.
                 MOVWF   INDF            ;       Place the byte at destination address.
                 INCF    TEMP_1,F        ;       Inc dest address.
                 INCF    TEMP_2,F        ;       Inc source address.
                 DECFSZ  COUNT_1,F       ;
                 GOTO    COPY_LOOP       ;       Loop until all bytes copied.
                 RETURN                  ;       ARG2 and destination.

;*****************************************************************************************
;
;     NAMES:    EE_TO_RAM               Copy 16 bytes of setup EEPROM to RAM.
;               RAM_TO_EE               Copy 16 bytes of setup RAM to EEPROM.
;
;     INPUT:    none.
;
; VARIABLES:    FSR, COUNT_1, TEMP_1, TEMP_2, TEMP_3.
;
;               TEMP_1 = destination address.
;               TEMP_2 = source address.
;               COUNT_1 = bytes to move.
;
; STACK USE:    2
;
;*****************************************************************************************

RAM_TO_EE:       BSF     WRITE_FLAG      ;       Indicate we want to write to EEPROM.
EE_TO_RAM:       MOVLW   CH_BYTES        ;
                 MOVWF   COUNT_1         ;       16 bytes to move.
                 MOVLW   CH_1_FREQ       ;       Set start address in RAM.
                 MOVWF   FSR             ;
                 MOVLW   EE_CH_1         ;       Set start address in EEPROM.
                 GOTO    EEPROM_MOVE     ; (2)   Do the move. Return via "GOTO".

;*****************************************************************************************
;
;      NAME:    EEPROM_MOVE
;
;   PURPOSE:    Move several bytes, between RAM and EEPROM, or EEPROM and RAM.
;               If the RAM and EEPROM data are the same, the EEPROM is not reprogrammed.
;               Assumes the number of bytes to move COUNT_1 has been set,
;               and the RAM source/destination address FSR has been set,
;               and the EEPROM source/destination address is in W,
;               Setting the WRITE_FLAG before calling EEPROM_MOVE indicates that the
;               direction will be from RAM to EEPROM, otherwise it will be EEPROM to RAM.
;
;     INPUT:    WRITE_FLAG = direction. (Set = RAM to EEPROM)
;                  COUNT_1 = bytes to move.
```

```
;                       FSR = RAM address.
;                         W = EEPROM address.
;
;    OUTPUT:    None.
;
; VARIABLES:    FSR,  COUNT_1.
;
; STACK USE:    2
;
;*****************************************************************************************

EEPROM_MOVE:    BSF     RP0             ;       Select bank 1 for EEPROM reg access.
                MOVWF   EEADR           ;       Save the EEPROM address.
                BCF     RP0             ;       Return to bank 0.
EEPROM_LOOP:    BTFSC   WRITE_FLAG      ;       Do we want to write ? (WRITE_FLAG set)
                CALL    WRITE_EEPROM    ; (2)   Y. Write a byte from RAM to EEPROM.
                CALL    READ_EEPROM     ; (1)   Get a byte from EEPROM.
                MOVWF   INDF            ;       Place it in RAM.
                BSF     RP0             ;       Select bank 1 for EEPROM reg access.
                INCF    EEADR,F         ;       Inc the EEPROM address.
                BCF     RP0             ;       Return to bank 0.
                INCF    FSR,F           ;       Inc the RAM address.
                DECFSZ  COUNT_1,F       ;
                GOTO    EEPROM_LOOP     ;       Loop until all bytes moved.
                BCF     WRITE_FLAG      ;
                RETURN                  ;

;*****************************************************************************************
;
;      NAME:    WRITE_EEPROM
;
;   PURPOSE:    Write the byte of RAM pointed to by FSR,
;               to the EEPROM address pointed to by EEADR.
;               If the RAM and EEPROM data are the same, the EEPROM is not reprogrammed.
;               Assumes the source address FSR has been set,
;               and the destination address EEADR has been set.
;
;     INPUT:    FSR, EEADR.
;
;    OUTPUT:    None.
;
; VARIABLES:    FSR.
;
; STACK USE:    1
;
;*****************************************************************************************

WRITE_EEPROM:   CALL    READ_EEPROM     ; (1)   Get the current byte from EEPROM.
                SUBWF   INDF,W          ;
                BTFSC   ZERO            ;       Is it the same as it is in RAM ?
                RETURN                  ;       Y. All done.
                MOVF    INDF,W          ;       N. Get the data byte from RAM.
                BCF     EEIF            ;          Clear the EEPROM write done flag.
                BSF     RP0             ;          Select bank 1 for EEPROM reg access.
                MOVWF   EEDATA          ;          Put it in the buffer.
                BSF     WREN            ;          Enable EEPROM write.
                BCF     GIE             ;          Disable interrupts.
                MOVLW   H'55'           ;          Perform required safety steps.
                MOVWF   EECON2          ;
                MOVLW   H'AA'           ;
                MOVWF   EECON2          ;
                BSF     WR              ;          Begin the write.
                BSF     GIE             ;          Enable interrupts.
                BCF     RP0             ;          Return to bank 0 for port access.
WRITE_LOOP:     BTFSS   EEIF            ;          Wait until the write is complete.
                GOTO    WRITE_LOOP      ;
                RETURN                  ;

;*****************************************************************************************
;
;      NAME:    READ_EEPROM
;
;   PURPOSE:    Read the byte of EEPROM data pointed to by EEADR.
;               Assumes the read address EEADR has been set.
;
;     INPUT:    EEADR.
;
```

```
;    OUTPUT:    W = The byte read from EEPROM.
;
; VARIABLES:    None.
;
; STACK USE:    0
;
;*****************************************************************************

READ_EEPROM:    BSF     RP0             ;       Select bank 1 for EEPROM reg access.
                BSF     RD              ;       Perform an EEPROM read.
                MOVF    EEDATA,W        ;       Get the read byte.
                BCF     RP0             ;       Return to ram bank 0.
                RETURN                  ;

;*****************************************************************************
;
;     NAMES:    CHK_PROG_BRIEF  Return a true flag in PRESSED, if the
;                               button was pressed briefly.
;
;     INPUT:    None.
;
;    OUTPUT:    None. (PRESSSED flag set if button pressed)
;
; VARIABLES:    TEMP_1.
;
; STACK USE:    0
;
;*****************************************************************************

CHK_PROG_BRIEF: MOVF    PROG_BTN_OLD,W  ;       Get old button count.
                MOVWF   TEMP_1          ;       Save the old button state.
                SUBWF   PROG_BTN_CNT,W  ;       Has button been released ? (current-old)
                MOVF    PROG_BTN_CNT,W  ;
                MOVWF   PROG_BTN_OLD    ;          Copy current state to old state.
                BTFSC   CARRY           ;
                GOTO    FALSE_FLAG      ;       N. Return a false flag.
                MOVF    TEMP_1,W        ;       Y. Was the button on less than min ?
                SUBLW   MIN_ON          ;          (min on period - old)
                BTFSC   CARRY           ;
                GOTO    FALSE_FLAG      ;          Y. Return a false flag.
                MOVF    TEMP_1,W        ;          N. Was the button on less than max ?
                SUBLW   MAX_ON          ;             (max on period - old)
                BSF     PRESSED         ;             Y. Return true flag.
                BTFSS   CARRY           ;
FALSE_FLAG:     BCF     PRESSED         ;             N. Return false flag.
                RETURN                  ;

        IF DEBUG

;*****************************************************************************
;
;      NAME:    DISPLAY_FREQ
;
;   PURPOSE:    Display the frequency in ARG2 on the LCD.
;               Including decimal point and leading zero blanking.
;
;     INPUT:    ARG2 = frequency.
;               (ARG2 unchanged)
;
;    OUTPUT:    None.
;
; STACK USE:    5
;
;*****************************************************************************

DISPLAY_ARG2:   MOVLW   CH_1_FREQ       ;       Save ARG2 in CH 1 frequency.
                CALL    COPY_FROM_ARG2  ; (1)
                MOVLW   CH_2_FREQ       ;       Save ARG3 in CH 2 frequency.
                CALL    COPY_FROM_ARG3  ; (1)
                GOTO    DISP_ARG        ;

DISPLAY_FREQ:   MOVLW   CH_1_FREQ       ;       Save ARG2 in CH 1 frequency.
                CALL    COPY_FROM_ARG2  ; (1)
                MOVLW   CH_2_FREQ       ;       Save ARG3 in CH 2 frequency.
                CALL    COPY_FROM_ARG3  ; (1)
                                        ;
                MOVLW   PLL_STEP_SIZE   ;
```

```
                CALL    CONST_TO_ARG3   ; (1)
                CALL    MULTIPLY        ; (3)   Multiply by the PLL step size.
                MOVLW   ARG1_3          ;
                CALL    COPY_TO_ARG2    ; (1)   Put result back in ARG2.
                MOVLW   VCO_OFFSET_FREQ ;       Get the VCO offset constant.
                CALL    CONST_TO_ARG3   ; (1)
                CALL    SUB             ; (3)   Sub from the freq.
                                        ;
DISP_ARG:       CALL    LCD_HOME        ; (4)
                CALL    BIN_TO_DEC      ; (2)   Convert the number to ASCII decimal.
                CALL    SET_ARG1        ; (1)   Set FSR = ARG1_7, COUNT_1 = 8.
DISP_LOOP:      MOVF    INDF,W          ;       Get char pointed to by FSR.
                CALL    LCD_CHR         ; (5)   Display the char.
                MOVLW   7               ;
                SUBWF   COUNT_1,W       ;       Is it time to insert a decimal point?
                MOVLW   H'A5'           ;
                BTFSC   ZERO            ;
                CALL    LCD_CHR         ; (5)   Y. Display a decimal point.
                INCF    FSR,F           ;       Inc the pointer.
                DECFSZ  COUNT_1,F       ;       All done ?
                GOTO    DISP_LOOP       ;       N. Continue.
                MOVLW   ' '             ;
                CALL    LCD_CHR         ; (5)
                MOVLW   'G'             ;
                CALL    LCD_CHR         ; (5)
                MOVLW   'H'             ;
                CALL    LCD_CHR         ; (5)
                MOVLW   'z'             ;
                CALL    LCD_CHR         ; (5)
                MOVLW   CH_1_FREQ       ;       Recover ARG2.
                CALL    COPY_TO_ARG2    ; (1)
                MOVLW   CH_2_FREQ       ;       Recover ARG3.
                CALL    COPY_TO_ARG3    ; (1)
                GOTO    EE_TO_RAM       ; (2)   Get the CH frequencies from EEPROM.
                                        ;       Return via "GOTO".

;*****************************************************************************************
;
;      NAME:    BIN_TO_DEC
;
;   PURPOSE:    Convert a 32 bit binary number to a 8 digit ASCII decimal number.
;               First the 32 bit binary number is converted to 8 digit decimal.
;               It is then converted to ASCII, including decimal point, leading zero
;               blanking, ready for display on the LCD.
;
;     INPUT:    ARG2 = frequency. (+ numbers only )
;
;    OUTPUT:    8 digit ASCII decimal number in ARG1_7..0. (MSD..LSD)
;               (ARG2 unchanged)
;
; VARIABLES:    FSR, COUNT_1, COUNT_2, TEMP_1.
;
; STACK USE:    1
;
;*****************************************************************************************

BIN_TO_DEC:     CALL    CLEAR_ARG1      ; (1)
                MOVLW   32              ;
                MOVWF   COUNT_2         ;       32 bits to process.
                GOTO    SHIFT_TO_DEC    ;
                                        ;
LOOP32:         CALL    SET_ARG1        ; (1)   Set FSR = ARG1_7, COUNT_1 = 8.
DEC_LOOP:       MOVLW   3               ;
                ADDWF   INDF,W          ;       If num + 3 > 7 then num = num + 3.
                MOVWF   TEMP_1          ;       Add 3 to num. (adds 6, after next shift)
                BTFSS   TEMP_1,3        ;       Is result > 7 ? (bit 3 set)
                GOTO    NO_ADJ          ;
                ADDLW   B'01111000'     ;       Y. Move bit 3 to bit 7.
                MOVWF   INDF            ;          Put the number back in the buffer.
NO_ADJ:         INCF    FSR,F           ;       Point to next byte in buffer.
                DECFSZ  COUNT_1,F       ;
                GOTO    DEC_LOOP        ;       Loop until all 8 bytes done.
                                        ;
SHIFT_TO_DEC:   RLF     ARG2_3,W        ;       Shift 32 bit number left by one bit
                RLF     ARG2_0,F        ;       into dec buffer.
                RLF     ARG2_1,F        ;
                RLF     ARG2_2,F        ;       After 32 shifts ARG2 unchanged.
```

```
                RLF     ARG2_3,F        ;       MSD of ARG2
                RLF     ARG1_0,F        ;       LSD of ARG1
                RLF     ARG1_1,F        ;
                RLF     ARG1_2,F        ;
                RLF     ARG1_3,F        ;
                RLF     ARG1_4,F        ;
                RLF     ARG1_5,F        ;
                RLF     ARG1_6,F        ;
                RLF     ARG1_7,F        ;       MSD of ARG1.
                DECFSZ  COUNT_2,F       ;
                GOTO    LOOP32          ;       Loop until all 32 bits processed.
                                        ;
                BCF     LEADING_FLAG    ;       Blank leading zero's until flag is set.
                                        ;
                CALL    SET_ARG1        ; (1)   Set FSR = ARG1_7, COUNT_1 = 8.
ASCII_LOOP:     MOVF    INDF,W          ;       Get the number.
                BTFSC   ZERO            ;       Is it = 0 ?
                GOTO    BLANK           ;
NO_BLANK        ADDLW   H'30'           ;       N. Convert it to ASCII.
                MOVWF   INDF            ;
                BSF     LEADING_FLAG    ;          Set the leading blank flag,
                GOTO    NUM_DONE        ;          following zero's will not be blanked.
BLANK:          BTFSC   LEADING_FLAG    ;       Y. Is the leading blank flag set.
                GOTO    NO_BLANK        ;          Y. Convert it to ASCII.
                MOVF    FSR,W           ;          N. Save the location so that a - sign
                MOVWF   COUNT_2         ;             can be inserted.
BLANK_IT:       MOVLW   ' '             ;
                MOVWF   INDF            ;             Blank the byte.
NUM_DONE:       INCF    FSR,F           ;       Point to the next byte.
                MOVLW   7               ;
                SUBWF   COUNT_1,W       ;       Time to cancel leading blanking?
                BTFSC   ZERO            ;
                BSF     LEADING_FLAG    ;       Y. Always display char before DP.
                DECFSZ  COUNT_1,F       ;
                GOTO    ASCII_LOOP      ;
                RETURN                  ;       N. Just return.

;-------------------------------------------------------------------------------------

SET_ARG1:       MOVLW   ARG1_7          ;       Set FSR = ARG1_7, COUNT_1 = 8.
                MOVWF   FSR             ;
                MOVLW   8               ;
                MOVWF   COUNT_1         ;
                RETURN                  ;

;*************************************************************************************
;
;      NAME:    MULTIPLY
;
;   PURPOSE:    32 bit * 32 bit unsigned multiply.    (ARG1 = ARG2 x ARG3)
;
;     INPUT:    ARG2  (+ or - numbers )
;               ARG3  (+ numbers only )
;
;    OUTPUT:    ARG1_7..0  (MSD..LSD)
;               (ARG3 unchanged)
;
; VARIABLES:    FSR, COUNT_1, COUNT_3, TEMP_1, TEMP_2, TEMP_3.
;
; STACK USE:    2
;
;*************************************************************************************

MULTIPLY:       CALL    CLEAR_ARG1      ; (1)
                MOVLW   32              ;       32 bits to multiply.
                MOVWF   COUNT_3         ;
MLOOP:          RRF     ARG2_0,W        ;       Rotate the multiplier right 1 bit
                RRF     ARG2_3,F        ;       into the carry bit.
                RRF     ARG2_2,F        ;       ( after 32 rotates ARG2 will be )
                RRF     ARG2_1,F        ;       ( returned to its original value )
                RRF     ARG2_0,F        ;
                                        ;
                BTFSC   CARRY           ;       If carry bit is set, add ARG3 to partial
                CALL    ADD_ARG1_ARG3   ; (2)   product. ARG1_7..4 = ARG1_7..4 + ARG3_3..0
                                        ;
                RRF     ARG1_7,F        ;       Right shift the partial product.
                RRF     ARG1_6,F        ;
```

```
                RRF     ARG1_5,F        ;
                RRF     ARG1_4,F        ;
                RRF     ARG1_3,F        ;
                RRF     ARG1_2,F        ;
                RRF     ARG1_1,F        ;
                RRF     ARG1_0,F        ;
                                        ;
                DECFSZ  COUNT_3,F       ;       All 40 bits done ?
                GOTO    MLOOP           ;       N. Keep going.
                RETURN                  ; (0)      All done.

;*************************************************************************************
;
;     NAMES:    LCD_CLEAR               Clear the display and home the cursor.
;
; VARIABLES:    TEMP_3, COUNT_0.
;
; STACK USE:    4
;
;*************************************************************************************

LCD_CLEAR:      MOVLW   H'01'           ;       Clear the display. then wait 8mS.
                CALL    LCD_CMD         ; (4)
                GOTO    WAIT_8MS        ; (1)   Wait 8mS. Return via "GOTO".

;*************************************************************************************
;
;     NAMES:    LCD_HOME        Home the cursor.
;               LCD_CMD         Send the command byte in W to the LCD.
;               LCD_CHR         Send the data byte in W to the LCD.
;
; VARIABLES:    COUNT_2, TEMP_1, TEMP_3.
;
; STACK USE:    3
;
;*************************************************************************************

LCD_HOME:       MOVLW   H'80'           ;       Position the cursor, line 1 pos 1.
LCD_CMD:        BCF     LCD_RS          ;       Select the instruction register.
                GOTO    LCD_BYTE        ; (3)   Send the byte to the LCD.

;-------------------------------------------------------------------------------------

LCD_CHR:        BSF     LCD_RS          ;       Select display data register.
LCD_BYTE:       MOVWF   TEMP_3          ;       Save the byte to send to the LCD.
                                        ;
                ANDLW   B'11110000'     ;
                IORWF   DEBUG_PORT,F    ;       Put the nibble into the TX buffer.
                CALL    I2C_START       ; (2)
                MOVLW   DEBUG_DISP      ;       Send the display address.
                CALL    I2C_TX_BYTE     ; (3)
                BSF     BUTTON_EN       ;
                BSF     LCD_E           ;
                MOVF    DEBUG_PORT,W    ;       Send the nibble.
                CALL    I2C_TX_BYTE     ; (3)   (LCD E high, buttons off)
                BCF     LCD_E           ;       Clock the LCD. (set LCD E low)
                MOVF    DEBUG_PORT,W    ;
                CALL    I2C_TX_BYTE     ; (3)
                MOVLW   B'00001111'     ;       Clear the nibble from the TX buffer.
                ANDWF   DEBUG_PORT,F    ;
                                        ;
                SWAPF   TEMP_3,W        ;       Send the low nibble to the LCD.
                                        ;
                ANDLW   B'11110000'     ;
                IORWF   DEBUG_PORT,F    ;       Put the nibble into the TX buffer.
                BSF     LCD_E           ;
                MOVF    DEBUG_PORT,W    ;       Send the nibble.
                CALL    I2C_TX_BYTE     ; (3)   (LCD E high, buttons off)
                BCF     LCD_E           ;       Clock the LCD. (set LCD E low)
                MOVF    DEBUG_PORT,W    ;
                CALL    I2C_TX_BYTE     ; (3)
                BCF     BUTTON_EN       ;       Turn buttons on.
                MOVF    DEBUG_PORT,W    ;
                IORLW   B'11110000'     ;       Set LCD data bus high.
                CALL    I2C_TX_BYTE     ; (3)
                CALL    I2C_STOP        ; (2)   Send the stop command.
                MOVLW   B'00001111'     ;       Clear the nibble from the TX buffer.
```

```
                ANDWF   DEBUG_PORT,F    ;
                GOTO    WAIT_200US      ; (0)   Return via "GOTO".

;*****************************************************************************
;
;      NAME:    INIT_LCD
;
;   PURPOSE:    Initialise the 16 x 2 liquid crystal display.
;               The LCD controller chip must be equivalent to the HITACHI 44780.
;
;     INPUT:    None.
;
;    OUTPUT:    None.
;
; VARIABLES:    COUNT_0.
;
; STACK USE:    5
;
;*****************************************************************************

LCD_INIT:       CALL    LCD_RESET       ; (5)   Reset the LCD.
                CALL    LCD_RESET       ; (5)   Reset the LCD.
                CALL    LCD_RESET       ; (5)   Reset the LCD.
                                        ;
                MOVLW   B'00100000'     ;       LCD 4 bit mode command.
                CALL    SEND_NIBBLE     ; (4)
                CALL    WAIT_200US      ; (1)
                MOVLW   B'00101000'     ;       Set LCD to 2 line, 5x7 dot.
                CALL    LCD_CMD         ; (4)
                MOVLW   B'00001000'     ;       Turn the display off.
                CALL    LCD_CMD         ; (4)
                MOVLW   B'00000001'     ;       Clear the display.
                CALL    LCD_CMD         ; (4)
                CALL    WAIT_8MS        ; (2)   Then wait 8mS.
                MOVLW   B'00000110'     ;       Cursor increments, no display shift.
                CALL    LCD_CMD         ; (4)
                MOVLW   B'00001100'     ;       Turn the display on.
                GOTO    LCD_CMD         ; (3)   Return via "GOTO".

;-----------------------------------------------------------------------------

LCD_RESET:      MOVLW   B'00110000'     ;       LCD reset command.
                CALL    SEND_NIBBLE     ; (4)
                GOTO    WAIT_8MS        ; (1)   Wait a while. Return via "GOTO".

;-----------------------------------------------------------------------------

SEND_NIBBLE:    ANDLW   B'11110000'     ;
                IORWF   DEBUG_PORT,F    ;       Put the nibble into the TX buffer.
                CALL    I2C_START       ; (2)
                MOVLW   DEBUG_DISP      ;       Send the display address.
                CALL    I2C_TX_BYTE     ; (3)
                BSF     BUTTON_EN       ;
                BSF     LCD_E           ;
                MOVF    DEBUG_PORT,W    ;       Send the nibble.
                CALL    I2C_TX_BYTE     ; (3)   (LCD E high, buttons off)
                BCF     LCD_E           ;       Clock the LCD. (set LCD E low)
                MOVF    DEBUG_PORT,W    ;
                CALL    I2C_TX_BYTE     ; (3)
                BCF     BUTTON_EN       ;       Turn buttons on.
                MOVF    DEBUG_PORT,W    ;
                IORLW   B'11110000'     ;       Set LCD data bus high.
                CALL    I2C_TX_BYTE     ; (3)
                CALL    I2C_STOP        ; (2)   Send the stop command.
                MOVLW   B'00001111'     ;       Clear the nibble from the TX buffer.
                ANDWF   DEBUG_PORT,F    ;
                RETURN                  ;
 ENDIF

;*****************************************************************************
;
;      NAME:    WAIT_200US              Do nothing for 200 uS.
;
; VARIABLES:    None.
;
; STACK USE:    0
;
```

```
;*****************************************************************************

WAIT_200US:     MOVLW   256-50          ;       Set the wait loop to 200uS.
WAIT_LOOP:      ADDLW   1               ;       1 clk cycles \
                BTFSS   ZERO            ;       1     "       > 4 clk's x 1uS
                GOTO    WAIT_LOOP       ;       2     "      /
                RETURN                  ;

;*****************************************************************************
;
;       NAME:   WAIT_8MS                Do nothing for 8 mS.
;               MS_WAIT                 Wait W = mS. (0 = 256 mS)
;
; VARIABLES:    COUNT_0.
;
; STACK USE:    1
;
;*****************************************************************************

WAIT_8MS:       MOVLW   8               ;       Set the outer loop to 8mS.
MS_WAIT:        MOVWF   COUNT_0         ;       Save the number of mS delay.
OUTER_LOOP:     CLRW                    ;       Set the wait loop to 256.
                CALL    WAIT_LOOP       ; (1)   256 loops x 4 x 1uS = 1.024mS.
                DECFSZ  COUNT_0,F       ;       Dec the mS counter.
                GOTO    OUTER_LOOP      ;
                RETURN                  ;

;*****************************************************************************
;
;       NAME:   UPDATE_PLL              Sends the divider num in ARG2 to the PLL.
;                                       If the frequency has changed,
;                                       get the status byte from the PLL,
;                                       and set the PLL locked LED.
; VARIABLES:    COUNT_1, TEMP_1.
;
; STACK USE:    3
;
;*****************************************************************************

UPDATE_PLL:     MOVLW   PLL_OLD         ;       Get the old PLL divider number.
                CALL    COPY_TO_ARG3    ; (1)
                                        ;
                CALL    COMPARE         ; (2)   Same as the new divider number ?
                BTFSC   ZERO            ;
                GOTO    GET_PLL_STATUS  ;
                                        ;
                MOVLW   PLL_OLD         ;       N. Save the new divider number,
                CALL    COPY_FROM_ARG2  ; (1)      to PLL_OLD.
                                        ;
                CALL    I2C_START       ; (2)
                MOVLW   PLL_ADDRESS     ;          Send the PLL address.
                CALL    I2C_TX_BYTE     ; (3)
                MOVF    ARG2_1,W        ;          Send MSD byte of the PLL divider.
                CALL    I2C_TX_BYTE     ; (3)
                MOVF    ARG2_0,W        ;          Send LSB byte of the PLL divider.
                CALL    I2C_TX_BYTE     ; (3)
                MOVLW   BYTE4           ;
                CALL    I2C_TX_BYTE     ; (3)      Send the 4th data byte to the PLL,
                MOVLW   BYTE5           ;
                CALL    I2C_TX_BYTE     ; (3)      Send the 5th data byte to the PLL,
                CALL    I2C_STOP        ; (2)      Send the stop command.
                                        ;
                BCF     PLL_STATUS,6    ;          Assume the PLL is not locked.
                                        ;
GET_PLL_STATUS: BTFSS   PLL_STATUS,6    ;       Was PLL locked when last checked ?
                GOTO    GET_STATUS      ;
                MOVF    LOCK_TIMER,W    ;       Y. Get the current PLL locked timer.
                BTFSS   ZERO            ;          Has the counter reached 0 ?
                RETURN                  ;          N. All done.
                                        ;
GET_STATUS:     CALL    I2C_START       ; (2)   Send an I2C start signal.
                MOVLW   PLL_ADDRESS + 1 ;       I2C address & read bit.
                CALL    I2C_TX_BYTE     ; (3)   Send the address byte to the PLL.
                                        ;
                BCF     I2C_TX_ACK_FLAG ;       RX byte will be followed by a NACK.
                CALL    I2C_RX_BYTE     ; (3)
                CALL    I2C_STOP        ; (2)   Send the stop command.
```

```
                                    ;
                BTFSS   PLL_STATUS,6    ;       Is PLL locked now ?
                GOTO    NOT_LOCKED      ;
                                        ;
                MOVLW   LOCK_TM         ;       Y. Reset the locked timer.
                MOVWF   LOCK_TIMER      ;
                BSF     PLL_LOCKED      ;          Light the PLL locked LED.
                RETURN                  ;
                                        ;
NOT_LOCKED:     CLRF    LOCK_TIMER      ;       N. Clear the locked timer.
                BCF     PLL_LOCKED      ;          Turn off the PLL locked LED.
                RETURN                  ;

;*************************************************************************************
;
;       NAME:   I2C_TX_BYTE             Send the byte in W to the I2C bus.
;
; VARIABLES:    COUNT_2, TEMP_1.
;
; STACK USE:    2
;
;*************************************************************************************

I2C_TX_BYTE:    MOVWF   TEMP_1          ;       Save the byte to TX.
                MOVLW   8               ;       Set the number of bits to transmit.
                MOVWF   COUNT_2         ;
TX_BIT_LOOP:    RLF     TEMP_1,F        ;       Move the bit (MSB first) into the carry.
                BTFSC   CARRY           ;       Carry set ?
                CALL    LET_SDA_HIGH    ; (1)   Y. Let SDA float high. (Set SDA as I/P)
                BTFSS   CARRY           ;
                CALL    SET_SDA_LOW     ; (1)   N. Set SDA low. (Set SDA as O/P)
                CALL    I2C_CLOCK       ; (2)   Latch the data into the slave.
                DECFSZ  COUNT_2,F       ;
                GOTO    TX_BIT_LOOP     ;       Loop until 8 bits are sent.
                                        ;
                CALL    LET_SDA_HIGH    ; (1)   Let SDA float high. (Set SDA as I/P)
                CALL    I2C_DELAY       ; (1)
                CALL    LET_SCL_HIGH    ; (1)   Set SCL high.
                CALL    I2C_DELAY       ; (1)
                                        ;
                BTFSS   I2C_SDA         ;
                BCF     I2C_ACK_FLAG    ;       Save the slaves acknowledge bit.
                BTFSC   I2C_SDA         ;
                BSF     I2C_ACK_FLAG    ;              "
                                        ;
                CALL    SET_SCL_LOW     ; (1)   Set SCL low.
                GOTO    I2C_DELAY       ; (0)   Return via "GOTO".

;*************************************************************************************
;
;       NAME:   I2C_RX_BYTE             Read a byte into PLL_STATUS from the I2C bus.
;
; VARIABLES:    COUNT_2, TEMP_1.
;
; STACK USE:    2
;
;*************************************************************************************
;
;       NAME:   ACK                     Send an I2C acknowledge to the I2C bus.
;                                       (SDA low while generating a clock signal)
;               NACK                    Send an I2C negative acknowledge to the I2C bus.
;                                       (SDA high while generating a clock signal)
; VARIABLES:    None.
;
; STACK USE:    2
;
;*************************************************************************************

I2C_RX_BYTE:    MOVLW   8               ;
                MOVWF   COUNT_2         ;       Set the number of bits to receive.
RX_BIT_LOOP:    CALL    I2C_DELAY       ; (1)
                CALL    LET_SCL_HIGH    ; (1)   Set SCL high.
                CALL    I2C_DELAY       ; (1)
                                        ;
                RLF     PLL_STATUS,F    ;       Make room for the received bit.
                BTFSC   I2C_SDA         ;
                BSF     PLL_STATUS,0    ;       Save the received bit. (1)
```

```
                BTFSS   I2C_SDA         ;
                BCF     PLL_STATUS,0    ;       Save the received bit. (0)
                                        ;
                CALL    SET_SCL_LOW     ; (1)   Set SCL low.
                DECFSZ  COUNT_2,F       ;
                GOTO    RX_BIT_LOOP     ;       Loop until 8 bits received.
                BTFSC   I2C_TX_ACK_FLAG ;
                GOTO    ACK             ; (2)   Send an acknowledge, Return via "GOTO".
                                        ;       Or send a negative ack.
NACK:           CALL    LET_SDA_HIGH    ; (1)   Let SDA float high. (Set SDA as I/P)
                CALL    I2C_CLOCK       ; (2)   (Send an I2C negative acknowledge)
 IF SIMULATE                            ;
I2C_DELAY       RETURN                  ;       Short delay.
 ELSE                                   ;
I2C_DELAY       MOVLW   DELAY_CONSTANT  ;          Load W with count for the delay.
I2C_DELAY_LOOP  ADDLW   -1              ;          1 CLK cycles \
                BTFSS   ZERO            ;          1     "       > 4 CLK's x 1uS
                GOTO    I2C_DELAY_LOOP  ;          2     "      /
                RETURN                  ;
 ENDIF                                  ;

;-------------------------------------------------------------------------------------

ACK:            CALL    SET_SDA_LOW     ; (1)   Set SDA low. (Set SDA as O/P)
                CALL    I2C_CLOCK       ; (2)   (Send an I2C acknowledge)
                CALL    I2C_DELAY       ; (1)
LET_SDA_HIGH:   BSF     RP0             ;       Select bank 1.
                BSF     I2C_SDA_DIR     ;       Let SDA float high. (Set SDA as I/P)
                BCF     RP0             ;       Return to bank 0.
                RETURN                  ;

;*************************************************************************************
;
;       NAME:   I2C_STOP                Send an I2C stop signal on the bus.
;               I2C_START               Send an I2C start signal on the bus.
;
; VARIABLES:    None.
;
; STACK USE:    1
;
;*************************************************************************************

I2C_STOP:       CALL    SET_SDA_LOW     ; (1)   Set SDA low. (Set SDA as O/P)
                CALL    I2C_DELAY       ; (1)
                CALL    LET_SCL_HIGH    ; (1)   Set SCL high.
                CALL    I2C_DELAY       ; (1)
                CALL    LET_SDA_HIGH    ; (1)   Let SDA float high. (Set SDA as I/P)
                CALL    I2C_DELAY       ; (1)
                BSF     GIE             ;       Enable interrupts.
                RETURN                  ;

;-------------------------------------------------------------------------------------

I2C_START:      BCF     GIE             ;       Disable interrupts.
                CALL    LET_SCL_HIGH    ; (1)   Set SCL high.
                CALL    LET_SDA_HIGH    ; (1)   Let SDA float high. (Set SDA as I/P)
                CALL    I2C_DELAY       ; (1)
                CALL    SET_SDA_LOW     ; (1)   Set SDA low. (Set SDA as O/P)
                GOTO    I2C_CLOCK2      ; (1)   Return via "GOTO".

;-------------------------------------------------------------------------------------

I2C_CLOCK:      CALL    I2C_DELAY       ; (1)
                CALL    LET_SCL_HIGH    ; (1)   Let SCL float high. (Set SCL as I/P)
I2C_CLOCK2:     CALL    I2C_DELAY       ; (1)
                GOTO    SET_SCL_LOW     ; (0)   Set SCL low. Return via "GOTO".

;-------------------------------------------------------------------------------------

SET_SDA_LOW:    BCF     I2C_SDA         ;       SDA will be low when set as O/P.
                BSF     RP0             ;       Select bank 1.
                BCF     I2C_SDA_DIR     ;       Set SDA low. (Set SDA as O/P)
RET_BANK_0:     BCF     RP0             ;       Return to bank 0.
                RETURN                  ;

;-------------------------------------------------------------------------------------
```

```
LET_SCL_HIGH:                           ;
 IF I2C_CLK_OC                          ;
 IF SIMULATE                            ;
                RETURN                  ;
 ELSE                                   ;
                BSF     RP0             ;       Select bank 1.
                BSF     I2C_SCL_DIR     ;       Let SCL float high. (Set SCL as I/P)
                BCF     RP0             ;       Return to bank 0.
I2C_CLOCK_WAIT: BTFSS   I2C_SCL         ;
                GOTO    I2C_CLOCK_WAIT  ;       Wait until SCL goes high.
                RETURN                  ;       (Slave may stretch the clock)
 ENDIF                                  ;
 ELSE                                   ;
                BSF     I2C_SCL         ;       Set SCL high.
                RETURN                  ;
 ENDIF                                  ;

;-------------------------------------------------------------------------------

SET_SCL_LOW:                            ;
 IF I2C_CLK_OC                          ;
                BCF     I2C_SCL         ;       SCL will be low when set as O/P.
                BSF     RP0             ;       Select bank 1.
                BCF     I2C_SCL_DIR     ;       Set SCL low. (Set SCL as O/P)
                BCF     RP0             ;       Return to bank 0.
                RETURN                  ;
 ELSE                                   ;
                BCF     I2C_SCL         ;       Set SCL low.
                RETURN                  ;
 ENDIF                                  ;

;*******************************************************************************
;
;               Interrupt service routine
;
;               Service the TMR0 interupt (every 16mS)
;
;               Debounce the PROG button.
;               Increment the button pressed counter,
;               if the button is pressed, reset the timeout counter.
;               if the button is not pressed, reset button pressed counter.
;
;               Decrement the 16 bit down counter. when it reaches 0, set the
;               timeout flag. (if the button has not been pressed for a while.)
;
;               Increment the LCD flasher counter.
;
;               Dec the 8 bit lock timer, down counter. wont dec past 0.
;               When equal to 0 its time to check if the PLL is locked.
;
; STACK USE:    0
;
;*******************************************************************************


;-------------------------------------------------------------------------------
;               Save STATUS, and W registers.
;-------------------------------------------------------------------------------

INT_SERVICE:    MOVWF   SAVE_W_REG      ;       Save W reg.
                SWAPF   STATUS,W        ;
                BCF     RP0             ;       Make sure we are addressing bank 0.
                MOVWF   SAVE_STATUS     ;       Save STATUS reg.

;-------------------------------------------------------------------------------
;               Debounce the PROG button.
;-------------------------------------------------------------------------------

                BCF     PROG_PRESSED    ;
                                        ;
                INCFSZ  PROG_BTN_CNT,W  ;       Inc the PROG button count.
                MOVWF   PROG_BTN_CNT    ;       But don't inc past 255.
                                        ;
                BSF     RP0             ;       Select bank 1.
                BSF     LED_BTN_DIR     ;       Set PROG_BTN as I/P.
                BCF     RP0             ;       Return to bank 0.
                NOP                     ;
```

```
                NOP                     ;
                BTFSS   PROG_BTN        ;       Is PROG button pressed ?   (low if pressed)
                BSF     PROG_PRESSED    ;       Y. Set the PROG button pressed flag.
                BSF     RP0             ;       Select bank 1.
                BCF     LED_BTN_DIR     ;       Return PROG_BTN to an 0/P.
                BCF     RP0             ;       Return to bank 0.
                                        ;
                BTFSS   PROG_PRESSED    ;       Is PROG pressed flag set ? (high if pressed)
                CLRF    PROG_BTN_CNT    ;       N. Clear the button count.
                BTFSS   PROG_PRESSED    ;
                GOTO    BUTTON_END      ;
                MOVLW   TIMEOUT_NUM     ;       Y. Reset the display timeout counter.
                MOVWF   TIMEOUT_H       ;
                CLRF    TIMEOUT_L       ;          Load timeout with 13 sec.
BUTTON_END:                             ;

;-------------------------------------------------------------------------------
;               Increment the LCD flasher counter.
;-------------------------------------------------------------------------------

                INCF    FLASHER,F       ;       Inc the PROG_LED flasher counter.

;-------------------------------------------------------------------------------
;               Decrement the Display timeout counter.
;-------------------------------------------------------------------------------

                DECF    TIMEOUT_L,F     ;       Dec the timeout counter.
                BTFSC   ZERO            ;
                DECF    TIMEOUT_H,F     ;       Is it = 0 ?
                BTFSC   ZERO            ;
                BSF     TIMEOUT         ;       Y. Set the timeout flag.

;-------------------------------------------------------------------------------
;               Decrement the PLL locked counter.
;-------------------------------------------------------------------------------

                INCF    LOCK_TIMER,W    ;       Get the lock time counter.
                BTFSC   ZERO            ;       Is it = 255?
                GOTO    NO_LOCK_TM_DEC  ;
                MOVF    LOCK_TIMER,W    ;       N. Get the lock time counter.
                BTFSS   ZERO            ;          Is it = 0 ?
                DECF    LOCK_TIMER,F    ;          N. Dec the counter.
NO_LOCK_TM_DEC:                         ;       Y. Dont dec the counter.

;-------------------------------------------------------------------------------
;               Interrupt exit. Restore STATUS, and W registers.
;-------------------------------------------------------------------------------

                SWAPF   SAVE_STATUS,W   ;
                MOVWF   STATUS          ;       Restore STATUS reg.
                SWAPF   SAVE_W_REG,F    ;
                SWAPF   SAVE_W_REG,W    ;       Restore W reg without changing STATUS.
                BCF     T0IF            ;       Clear TMR0 interupt flag.
                RETFIE                  ;

;*******************************************************************************
;               End of interrupt service routine
;*******************************************************************************

;*******************************************************************************
;               EEPROM VARIABLE MACROs
;*******************************************************************************

VAR_1           MACRO   NAME, VAL       ;       1 Byte variable definition.
NAME            EQU $ - EE_VAR_START    ;
                DE (VAL) & H'FF'        ;
                ENDM                    ;

VAR_4           MACRO   NAME, VAL       ;       4 Byte variable definition.
NAME            EQU $ - EE_VAR_START    ;
                DE (VAL>>24) & H'FF'    ;
                DE (VAL>>16) & H'FF'    ;
                DE (VAL>>8) & H'FF'     ;
                DE (VAL) & H'FF'        ;
                ENDM                    ;

;*******************************************************************************
```

```
;               EEPROM variable definitions.
;*******************************************************************************************

 ORG     H'2100'

EE_VAR_START

; size           name                            value           comment

 VAR_4          EE_CH_1                        ,_CH_1_FREQ     ; CH 1 Frequency.
 VAR_4          EE_CH_2                        ,_CH_2_FREQ     ; CH 2 Frequency.
 VAR_4          EE_CH_3                        ,_CH_3_FREQ     ; CH 3 Frequency.
 VAR_4          EE_CH_4                        ,_CH_4_FREQ     ; CH 4 Frequency.

;*******************************************************************************************

                END
```