```
' PROGRAM: tuner_code.bs2                    Revision 12/04/03
' written by Jim Garland W8ZR

'     {$stamp BS2sx}                         'Specify Basic Stamp version -- required statement
' ============================================================
'     Program controls automatic EZ-tuner. This version uses a rotary solenoid to turn an 11 position
'     bandswitch and stepper motors to control two variable capacitors. In auto mode, tuner measures TX frequency, looks up corresponding
'     settings in memory, and moves steppers and solenoid to the settings. The variable capacitors and switched inductor can also be
'     manually controlled with optical encoders (caps) or up/down buttons (inductor). The tuner shifts to manual
'     mode whenever the rotary encoders are turned. The tuner powers up by initializing the stepper motors and rotary solenoid,
'     switching to auto mode, and returning to the last-used memory settings. There are 4 dual-function pushbuttons,with
'     functions as follows:

'     UP/DWN Buttons:           In auto mode, step through the memories, in manual mode step through the inductor settings.
'     MODE/STORE Button:        One press toggles automatic/manual mode. If held down, stores current setting in memory.
'     IN/OUT-RESET Button:      One press toggles tuner on-line/off-line. If held down, initializes the steppers and rotary solenoid to
'     home positions.

'     The display shows a 2 sec startup message and then displays the C1,C2, and L settings (top line), and the memory
'     frequency (second line). A message also indicates off-line status.

'               Port Connections & INPUT/OUTPUT Status

'               P0:     IN      Encoder1 A
'               P1:     IN      Encoder1 B
'               P2:     IN      Encoder2 A
'               P3:     IN      Encoder2 B
'               P4:     IN      UP button
'                       OUT     Speaker
'               P5:     IN      DOWN button
'                       IN      Reset limit detect
'               P6:     OUT     Auto Mode LED
'               P7:     IN      IN/OUT-RESET button
'               P8:     IN      MODE/STORE button
'               P9      OUT     IN/OUT relay
'               P10:    OUT     Ext.capacitor relay
'               P11:    OUT     Rotary solenoid step
'               P12:    OUT     Stepper2 step
'               P13:    OUT     Stepper1 step
'               P14:    OUT     Stepper rotation CW=0,CCW=1
'               P15:    IN      TX frequency detect
'               P16:    OUT     LCD serial data

'=============================================================================================
'==================================DEFAULT TUNER SETTINGS=====================================

'Default C1, C2, and L Data in EEPROM (134 bytes). These values match a 50 ohm load. If updating
'program, insert (') before each DATA statement to avoid overwriting existing settings.

' C1 Data (134 values) stored in EEPROM addresses 0-144 ($000-$08F)

    DATA 4,6,8,9,10,12,12,14,15,16,17,19,20,21,23,23,24,25,25,26                    '160meters (0-19)
    DATA 42,43,44,44,45,46,47,48,48,49,50,51,52,53,54,55,56,56,56,57,57,58,58,59,59 '80m(20-44)
    DATA 65,65,65,66,66,66,67,67,68,68,68,69,69,70,70                               '40meters (45-59)
    DATA 52,53,53,54,54                                                             '30meters (60-64)
    DATA 47,47,47,48,48,48,48,49,49,49,50,50,51,51,51,52,52,53                      '20 meters (65-82)
    DATA 11,11,11,11,11,11                                                          '17 meters (83-88)
    DATA 69,69,70,70,71,71,72,72,73                                                 '15meters (89-97)
    DATA 62,62,62                                                                   '12meters (98-100)
                                                                                                        '10
meters (99-134)
    DATA 67,67,67,67,67,68,68,68,68,68,68,69,69,69,69,69,69,69,70,70,70,70,70,70,70,71,71,71,71,71,71,71,71,71

' C2 Data (134 values) stored in EEPROM addresses 145-288 ($090-$11F)

    DATA @$090,49,49,50,51,52,53,54,55,56,57,58,58,59,60,60,61,62,63,64,65          '160meters (0-19)
    DATA 54,55,56,56,57,58,59,60,60,61,62,62,63,64,64,65,65,66,66,67,67,68,68,69,69 '80m(20-44)
    DATA 73,73,73,74,74,74,74,75,75,75,76,76,76,76,76                               '40meters (45-59)
    DATA 82,82,83,83,83                                                             '30meters (60-64)
    DATA 80,80,80,80,80,80,81,81,81,81,81,81,81,82,82,82,82,82                      '20 meters (65-82)
```

```
    DATA 41,41,41,41,41,41                                          '17 meters (83-88)
    DATA 88,88,88,89,89,89,90,90,90                                 '15meters (89-97)
    DATA 64,64,64                                                   '12meters (98-100)
                                                                                        '10
meters (101-134)
    DATA 82,82,82,82,82,82,82,82,81,81,81,81,81,81,81,81,81,80,80,80,80,80,80,80,80,80,79,79,79,79,79,79

' L Data (11 values) stored in EEPROM addresses 289-304 ($120-$12F)
' and startup addr1 ($12C)

    DATA @$120,10,10,9,9,6,5,3,3,2,2,1,0,45,0,0,0
'=============================================================================================

'   SYMBOL TABLE


'   ========================
'   User Supplied Parameters

vernier     con     10          'Parameter specifies the rotary encoder "bandspread"
                                'and must be an even number equal to twice the desired
                                'vernier ratio, e.g., vernier=10 gives a vernier ratio of 5.

freqtrim    con     285         'Parameter corrects for clock innacuracy in the BS2sx.
                                'The correction factor is 1/freqtrim, e.g., 1/285=.0035=.35% correction.
                                'Freqtrim varies from device to device and is determined experimentally.

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

old1A       var     bit         'previous A-output, encoder1
old1B       var     bit         'previous B-output  encoder1
old2A       var     bit         'previous A output, encoder2
old2B       var     bit         'previous B output, encoder2
stpcnt1     var     byte        'stepper1 count
stpcnt2     var     byte        'stepper2 count
SWcnt       var     byte        'rotary switch count
memval      var     byte        'C1,C2, or L memory variable
stp         var     byte        'step variable
btnwk1      var     byte        'button workspace variable
btnwk2      var     byte        'button workspace variable
btnwk3      var     byte        'button workspace variable
btnwk4      var     byte        'button workspace variable
x           var     byte        'gen purpose variable
y           var     byte        'gen purpose variable
z           var     byte        'gen purpose variable
freq        var     word        'frequency variable
freq1       var     word        'frequency variable
addr1       var     byte        'C1 EEPROM address variable (0-135)
addr2       var     word        'C2 EEPROM address variable (145-288)
addr3       var     word        'L EEPROM address variable (289-304)
loopcnt1    var     byte        'loop counter
loopcnt2    var     word        'loop counter
N96         con     $40F0       'LCD 9600 baud
I           con     254         'LCD instruction toggle
CLR         con     1           'LCD clear display

DIRS=%0111111001000000          'P0-P5,P7-P8,P15 set to input, others to output
'=============================================================================================
'               PROGRAM BEGINS HERE

'Initialize tuner by zeroing stepper motors and rotary solenoid to home positions. Also
'called when IN/OUT-RESET pushbutton is pushed for 1/2 second.

reset:
    GOSUB longbeep                          '3-tone beep
    OUT9=0                                  'take tuner OFF-LINE
    OUT14=1                                 'set stepper direction to CCW
    PAUSE 1000                              'wait 1sec for LCD to wake up
    SEROUT 16,n96,[I,CLR]                   'clear LCD
    PAUSE 1                                 'wait 1msec for LCD
```

```
    SEROUT 16,n96,[" Please Wait...."]          'display msg on LCD
                                                'and start initialization sequence

'               ==================stepper 1 initialization=========


                reset1: DIR5=0                  'set P5 (limit detect) to input
                        PULSOUT 13,10           'pulse stepper1
                        PAUSE 20                'wait for stepper to finish
                        IF IN5=0 THEN upstep1   'is limit reached?
                       GOTO reset1:             'if not,loop until limit reached

                upstep1:
                        TOGGLE 14               'set stepper direction to CW
                        PULSOUT 13,10           'step one position to free up limit
                        PULSOUT 13,10           'step second position to free up limit
                       stpcnt1=0                'set stpcnt1 to origin
                        TOGGLE 14               'set stepper direction to CCW
                        GOTO reset2             'now reset stepper2

'               ===================stepper 2 initialization===========

                reset2:
                        PULSOUT 12,10           'pulse stepper2
                        PAUSE 20                'wait for stepper to finish
                        IF IN5=0 THEN upstep2   'limit reached?
                        GOTO reset2:            'if not,loop until limit reached

                upstep2:
                        TOGGLE 14               'set stepper direction to CW
                        PULSOUT 12,10           'step one position to free up limit
                        PULSOUT 12,10           'step second position to free up limit
                        PAUSE 400               'wait 400mS to let stepper time out,to avoid
                                                'loading 24V pwr supply
                        stpcnt2=0               'set stpcnt2 to origin
                        TOGGLE 14               'set stepper direction to CCW
                                                'now reset rotary solenoid

'               ==================rotary solenoid initialization========

                reset3:
                        PULSOUT 11,65000        'pulse rotary solenoid 52 msec
                        PAUSE 100               'wait for solenoid to catch up
                        IF IN5=0 THEN upstep3   'finish if limit reached
                        GOTO reset3             'if not,loop until limit reached

                 upstep3:
                        pulsout 11,65000        'advance solenoid one step
                        SWcnt=1                 'set SWcnt to position 1


'==============================Initialization sequence complete=================================


'======================Initialize LCD and display opening message==============================

 top:
    SEROUT 16,n96,[I,CLR]                       'clear LCD
    PAUSE 1                                     'wait 1msec for LCD
    SEROUT 16,n96,[" W8ZR  EZ-Tuner",I,194,"ver. 1.0"]'disp. startup msg
    PAUSE 2000                                  'wait 2 secs
    SEROUT 16,n96,[I,CLR]                       'clear LCD

'Format LCD for displaying C1, C2, L, and frequency at following locations: L1_C4(131):stpcnt1,
'L1_C14(141):stpcnt2, L2_C6(197):freq, L2_14(205):SWcnt

    SEROUT 16,n96,[I,128,"C1:",I,138,"C2:",I,192,"Freq:",I,203,"L:"]

'================================================================================
'======================Set tuner to ON-LINE and AUTO mode==========================
```

```
    OUT9=1              'Set mode to ON-LINE
    PUT 2,0             'Set mode flag to AUTO
    OUT6=1              'Light LED
    z=vernier/2         'set z at midpoint of vernier

'======================Retrieve settings from last turn-off==================================

'Input memory address stored at last turn-off, compute the frequency segment and move steppers & solenoid.
'This routine used after power-on and when stepping through memories. It bypasses get_freq
'and band lookup routine.


    READ    $12C,addr1                      'Get last-used C1 memory address from EEPROM

comp_freq:                                  'Input addr1 and branch to corresponding band.

    LOOKDOWN addr1,<[20,45,60,65,83,89,98,101,135],x
    BRANCH x,[freq160,freq80,freq40,freq30,freq20,freq17,freq15,freq12,freq10]


'======================Measure transmit frequency============================================

'Measure transmitter frequency at P15. First,poll P15 for 400uS to
'see if signal present. If no signal detected (x=0), skip to rd_buttons.
'Purpose is to avoid slowing down loop unless valid signal present, and to prevent
'errors by ensuring signal is present when freq-measuring gate opens.

'Second part of routine measures frequency twice, with gate of 250*0.4usec=0.1sec,
'and requries both measurements to agree. Reason for two measurements is to prevent errors caused by
'tracking of modulated SSB signals.

get_freq:
            COUNT 15,1,x                        'poll for signal, return 0 if NO,low integer if YES
            IF x=0 THEN rd_buttons              'skip if no signal present
            COUNT   15,250,freq                 'otherwise,measure frequency of signal
            COUNT 15,250,freq1                  'measure it again
            IF NOT freq = freq1 THEN rd_buttons 'skip if measurements not the same
            freq=freq+(freq/freqtrim)           'trim freq to correct for BS2sx clock innacuracy
            old1A=IN0 : old1B=IN1               'update old1A and old1B to fix encoder slippage during count
                                                'now go to freq_tune

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

' Input transmit frequency from get_freq. Confirm if in amateur band, compute memory addresses and frequency
' segment corresponding to the addresses. Then retrieve settings from memory and move steppers/solenoid to settings.
' If frequency not in amateur band, go back to get_freq and poll TX frequency.
' Display shows the lower end of the memory frequency segment, not the actual TX freq.
' Note that addr1 (address of C1) is computed from the get_freq TX frequency. Addr2 tracks addr1
' but is offset by $090 (addr2=addr1+$090).Addr3 is the band index and is obtained from a band lookup table.


freq_tune:

LOOKDOWN freq,<[1800,2001,3500,4001,7000,7351,10100,10201,14000,14351,18068,18169,21000,21451,24890,24991,28000,29701],x
BRANCH
x,[get_freq,band160,get_freq,band80,get_freq,band40,get_freq,band30,get_freq,band20,get_freq,band17,get_freq,band15,get_freq,band12,get
_freq,band10]
            GOTO get_freq                       'if freq>29700, take no action & read the buttons

band160:    addr1=freq/10-180                   'addr1=0-19,addr2=144-163 for 160m band, 10kHz steps
freq160:    LOOKUP addr1/10,[$120,$121],addr3   'addr3=288,289 for 1800-1899kHz and 1900-1999kHz
            freq=10*addr1+1800                  'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers


band80:     addr1=freq/20-155                   'addr1=20-44, addr2=164-188 for 80m band, 20kHz steps
freq80:     LOOKUP addr1/30,[$122,$123],addr3   'addr3=290,291 for 3500-3699kHz and 3700-3999kHz
            freq=20*addr1+3100                  'compute freq corresponding to address
            GOTO mem_step                       'retrieve settings and move steppers
```

```
band40:     addr1=freq/20-305                   'addr1 = 45-59, addr2=189-203 for 40m band, 20kHz steps
freq40:     addr3=$124                          'addr3=292
            freq=20*addr1+6100                  'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers

band30:     addr1=freq/20-445                   'addr1=60-64, addr2=204-208 for 30m band. 20kHz steps
freq30:     addr3=$125                          'addr3=293
            freq=20*addr1+8900                  'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers

band20:     addr1=freq/20-635                   'addr1=65-82, addr2=209-226 for 20m band, 20kHz steps
freq20:     addr3=$126                          'addr3=294
            freq=20*addr1+12700                 'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers

band17:     addr1=freq/20-820                   'addr1=83-88, addr2=227-232 for 17m band, 20kHz steps
freq17:     addr3=$127                          'addr3=295
            freq=20*addr1+16400                 'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers

band15:     addr1=freq/50-331                   'addr1=89-97, addr2=233-241 for 15m band, 50kHz steps
freq15:     addr3=$128                          'addr3=296
            freq=50*addr1+16550                 'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers

band12:     addr1=freq/50-399                   'addr1=98-100, addr2=242-244 for 12m band, 50kHz steps
freq12:     addr3=$129                          'addr3=297
            freq=50*addr1+19950                 'compute freq corresponding to addr1
            GOTO mem_step                       'retrieve settings and move steppers

band10:     addr1=freq/50-459                   'addr1=101-134, addr2=245-278 for 10m band, 50kHz steps
freq10:     addr3=$12A                          'addr3=298
                freq=50*addr1+22950                 'compute freq corresponding to addr1
    GOTO mem_step                           'retrieve settings and move steppers

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

' Input addr1, addr3, current encoder counts, and compute addr2,
' Then retrieve the settings from memory, and move both steppers and rot.solenoid to the memory
' settings. Note that the rotary solenoid only rotates in the CW direction.
' When done, go back and check for new frequency. Also engage ext. cap. if freq is in 160m band.

mem_step:

    addr2=addr1+$090                                    'compute addr2
    IF addr1<20 THEN cap_on                             'turn on ext. cap if inside 160m band
    OUT10=0                                             'turn off ext. cap if outside 160m band

ms1:        READ addr1,memval                           'retrieve memval for addr1
            IF memval > stpcnt1 THEN  CWmemstep1        'test for CW or CCW direction
            IF memval < stpcnt1 THEN CCWmemstep1        'and move stepper
ms2:        READ addr2,memval                           'retrieve memval for addr2
            IF memval > stpcnt2 THEN  CWmemstep2
            IF memval < stpcnt2 THEN CCWmemstep2
ms3:        READ addr3,memval                           'retrieve memval for addr3
            IF memval > SWcnt THEN  pulse3              'step rotary solenoid CW
            IF memval < SWcnt THEN CCWmemstep3          '
ms4:        GOSUB display
            GOTO get_freq                               'if no change, check for new frequency

cap_on:
```

```
                OUT10=1                         'switch on ext. cap
                GOTO ms1                        'and return
'=============================================================================================
' Stepper driver and rotary solenoid routines for CW and CCW rotation. Note
' rotary solenoid turns in CW direction only.

CWmemstep1:     OUT14=0                         'set stepper1 to CW
                GOTO pulse1

CWmemstep2:     OUT14=0                         'set stepper2 to CW
                GOTO pulse2


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

CCWmemstep1:            OUT14=1                 'set stepper1 to CCW
                        GOTO pulse1


CCWmemstep2:            OUT14=1                 'set stepper2 to CCW
                        GOTO pulse2


CCWmemstep3:            memval=memval+12        'advance memval one revolution (e.g. pos 3 -> pos 15)
                        GOTO pulse3

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

pulse1:         stp = ABS(memval-stpcnt1)       'calculate no. of steps needed
                FOR x = 1 TO stp
                PULSOUT 13,10                   'send stepper to destination
                PAUSE 10                        'wait for stepper to finish
                NEXT
                stpcnt1=memval                  'update stpcnt1
                GOTO ms2


pulse2:         stp = ABS(memval-stpcnt2)       'calculate no. of steps needed
                FOR x = 1 TO stp
                PULSOUT 12,10                   'send stepper to destination
                PAUSE 10                        'wait for stepper to finish
                NEXT
                stpcnt2=memval                  'update stpcnt2
                GOTO ms3


pulse3:         stp = memval-SWcnt              'calculate no. of steps needed
                PAUSE 400                       'wait 400mS for steppers to time out
                                                'to avoid loading 24V pwr supply
                FOR x = 1 TO stp
                PULSOUT 11,65000                'pulse solenoid for 52msec (65K x 0.8usec)
                PAUSE 100                       'wait 100msec for solenoid to finish
                NEXT
                SWcnt=memval//12                'update SWcnt, (mod 12 to keep memval<12)
                GOTO ms4                        'and return

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


' Read the UP, DOWN, MODE/STORE, and IN-OUT/RESET buttons. If pressed, take appropriate action.
' and if not pressed, read the encoders

rd_buttons:

                BUTTON 4,0,254,1,btnwk1,1,upbutton              'UP button pressed?
                BUTTON 5,0,254,1,btnwk2,1,dwnbutton             'DOWN button pressed?
                BUTTON 8,0,254,255,btnwk3,1,mode_store          'toggle mode if 1 press,STORE if held down
iobtn:          BUTTON 7,0,254,255,btnwk4,1,IO_reset            'Toggle IN/OUT if 1 press, RESET if held down
```

```
' The variable loopcnt1 is used to count loops for the MODE/STORE and IO/RESET buttons, which have
' a second function if held down.

                loopcnt1=loopcnt1+1 MAX 255          'count loops for STORE & RESET buttons
                BRANCH out9,[iobtn,rd_encoders]      'if OFF-LINE read I/O button until changed

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

'Mode_store BUTTON command checks status of port 8. Normally,the loop counter = 255
'when the button is first pressed, so the routine toggles the AUTO/MAN mode to AUTO and resets loopcnt to 0. After a delay
'of 254 counts the button status is checked again, and if still pressed the routine branches to STORE and saves
'the current stepper and solenoid positions in EEPROM

mode_store:

                IF loopcnt1=253 THEN store           'store if loop counter=253, otherwise toggle AUTO/MANUAL mode
'================== Short Press - toggle mode MANUAL or AUTO ========================
    mode:       loopcnt1=0                           'reset loop counter
                GET 2,x                              'ck mode status (0=auto, 1=manual)
                BRANCH x,[manmode,automode]          'branch to other mode

    manmode: PUT 2,1                                 'change mode flag to manual
                GOTO chgmode

    automode: PUT 2,0                                'change mode flag to auto
                GOTO chgmode

    chgmode:TOGGLE 6                                 'Toggle LED
                GOSUB beep                           'short beep to confirm press
                GOTO rd_encoders                     'then exit

'================== Long Press - store settings ======================================
    store:WRITE addr1,stpcnt1                        'save settings
                WRITE addr2,stpcnt2
                WRITE addr3,SWcnt
                GOSUB longbeep                       'long beep to confirm store (3 tones)
                PAUSE 500                            'wait 1/2 sec for button to be released
                GOTO rd_encoders                     'and go read the encoders

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

'The IO_reset BUTTON instruction checks status of port 7. Normally,the loop counter = 255
'when the button is first pressed, so the routine toggles the IN/OUT status and resets loopcnt to 0. After a delay
'of 254 loops,the button status is checked again, and if still pressed the routine branches to RESET.

IO_reset:       IF loopcnt1=253 THEN reset           'reset if loop counter=254, otherwise toggle IN/OUT
                IF loopcnt1=255 THEN zeroloop        'zero loop counter only if starting count=255
                                                     'otherwise go to inout
    inout:
                TOGGLE 9                             'Toggle port 9 (1=IN-Line, 0=OFF-Line)
                GOSUB beep                           'short beep to confirm mode change
                BRANCH out9,[offline,online]         'read output latch and branch


offline:
                SEROUT 16,n96,[I,192,"   OFF-LINE    "]  'display off-line msg
                GOTO iobtn                           'loop to I/O button until pressed

online:
                SEROUT 16,n96,[I,192,"Freq:",I,197,DEC freq,"  ",I,203,"L:",I,205,DEC SWcnt," "] 'restore 2nd line of display
                GOTO rd_encoders

zeroloop:       loopcnt1=0
                GOTO inout

'======================================== Read encoders and move steppers ====================

rd_encoders:
     enc1:      IF IN0=old1A AND IN1=old1B THEN enc2      'read ports P0,P1, if no change read 2nd encoder
                PUT 2,1                                   'if changed, set mode flag to Manual
```

```
                OUT6=0                                  'turn off LED
                IF IN1=old1A THEN CCWenc1               'test for CCW rotation and go to CCW if bits match
                                                        'if not, go to CWenc
CWenc1:         '==================================
'                       Encoder Vernier Routine - CW

                        z=z-1                           'decrement vernier counter
                        old1A=IN0 : old1B=IN1           'update old1A and old1B
                        if z>0 then  enc1               'loop until count reaches zero
                        z=vernier/2                     'then reset count to vernier midpoint
                                                        'and continue
'               ==================================

                        IF stpcnt1=100 THEN enc1        'loop again if stepper at maximum CW
                        stpcnt1=stpcnt1+1               'increment encoder count,maximum is 102
                        OUT14=0                         'set rotation for CW
                        GOTO pulser1                    'and move stepper1

CCWenc1:        '==================================
'                       Encoder Vernier Routine - CCW

                        z=z+1                           'increment vernier counter
                        old1A=IN0 : old1B=IN1           'update old1A and old1B
                        if z<vernier then enc1          'loop until vernier reached
                        z=vernier/2                     'reset count at vernier midpoint
                '==================================
                        IF stpcnt1=0 THEN enc1          'loop again if stepper at minimum CCW
                        stpcnt1=stpcnt1-1               'decrement encoder count,minimum limit is 2
                        OUT14=1                         'set rotation to CCW
                                                        'when done go to pulser1

    pulser1:
                        PULSOUT 13,10                   '8uS pulse to stepper1
                        SEROUT 16,n96,[I,131, DEC stpcnt1,"  "]  'update C1 on display
                        GOTO enc1                       'return to loop when done

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

enc2:
                        IF IN2=old2A AND IN3=old2B THEN ck_mode     'read ports P2,P3, if no change loop in auto or manual mode
                        PUT 2,1                         'if changed, set mode flag to Manual
                        OUT6=0                          'and turn off LED
                        IF IN3=old2A THEN CCWenc2       'test for CCW rotation, goes to CCW if bits match
                                                        'if not, go to CWenc2
CWenc2:         '==================================
'                       Encoder Vernier Routine - CW

                        z=z-1                           'decrement vernier counter
                        old2A=IN2 : old2B=IN3           'update old2A and old2B
                        if z>0 then  enc2               'loop until count reaches 0
                        z=vernier/2                     'then reset count to vernier midpoint
                                                        'and continue
'               ==================================

                        if stpcnt2=100 then rd_buttons  'loop again if stepper 2 at max cw
                        stpcnt2=stpcnt2+1               'increment encoder count,stop at 102
                        OUT14=0                         'set rotation to CW
                        GOTO pulser2                    'and move stepper2


CCWenc2:        '==================================
'                       Encoder Vernier Routine - CCW

                        z=z+1                           'increment vernier count
                        old2A=IN2 : old2B=IN3           'update old2A and old2B
                        if z<vernier then enc2          'loop until count reaches vernier
                        z=vernier/2                     'then reset count to vernier midpoint
                                                        'and continue
                '==================================
```

```
                        if stpcnt2=0 then rd_buttons        'loop again if stepper2 at minimum CCW
                        stpcnt2=stpcnt2-1                   'decrement encoder count,stop at 2
                        OUT14=1                             'set rotation to CCW
                                                            'when finished go to pulser2

    pulser2:
                        PULSOUT 12,10                       '8uS pulse to stepper2
                        SEROUT 16,n96,[I,141, DEC stpcnt2,"  "]  'update C2 on display
                        GOTO rd_buttons                     'loop in MANUAL mode
'
'=============================================================================================
' Check for AUTO or MANUAL mode, and after 5000 loops of no activity (to avoid excessive
' writing to EEPROM) store updated C1 address (addr1) in EEPROM.

ck_mode:

                        loopcnt2=loopcnt2 + 1               'increment loop counter
                        IF loopcnt2=5000 THEN store_last    'store C1 address after 5000 loops of no activity
                ck_md1: GET 2,x                             'retrieve mode (0=auto,1=manual)
                        BRANCH x,[get_freq,rd_buttons]      'and branch accordingly.

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

'Store current C1 address (addr1) in EEPROM at $12C after 5000 loops and if
'addr1 has changed.

store_last:
                        READ $12C,memval                    'get last stored addr1
                        IF memval=addr1 THEN resetlc2       'skip if no change
                        WRITE $12C,addr1                    'if changed, store new addr1

                resetlc2: loopcnt2=0                        'reset loopcnt2 after store
                        GOTO ck_md1                         'go back to ck_mode


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

'Increment or decrement indices and update stepper settings if UP or DOWN button pressed.
'In AUTO mode, step through memories, in MANUAL mode, step inductor.

upbutton:
                GET 2,x                                     'check if auto (0) or manual (1) mode
                BRANCH x,[autoup,manup]

dwnbutton:
                GET 2,x                                     'check if auto (0) or manual (1) mode
                BRANCH x,[autodwn,mandwn]

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

autoup:                 addr1=addr1+1 MAX 134               'increments addr1 and limits to highest value
                        GOTO comp_freq                      'output new address to stepper

autodwn:                IF addr1=0 THEN comp_freq           'exit if addr1 is zero
                        addr1=addr1-1                       'decrement address if greater than zero
                        GOTO comp_freq                      'output new address to stepper (comp_freq)


'           ================================================
'Advance rotary solenoid 1 step for manup routine, and 11 steps for mandwn (switch has 11
'positions plus 0. For manup routine, switch counter rolls over to 1 after it reaches 11. For mandwn
'routine, switch counter stops at 1, to minimize needless switch rotations.

manup:                  SWcnt=(SWcnt+1)//12                 'increment switch count, rollover if equals 12.
                        IF SWcnt>0 THEN manup1              'skip to manup1 if SWcnt not 0
                        PULSOUT 11, 65000                   'advance to pos. 1 if SWcnt = 0
                        PAUSE 200
                        SWcnt=1                             'and set SWcnt=1

    manup1:   PULSOUT 11, 65000                             'pulse solenoid 52 msec
```

```
                PAUSE 100                       'wait for solenoid to catch up
                GOSUB display                   'update display
                GOTO rd_buttons

mandwn:         IF SWcnt=1 THEN skp_mandwn      'skip if count=1
                SWcnt=SWcnt-1                   'decrement rotary switch count if not zero
                FOR z = 1 TO 11                 'rotate solenoid 11 positions
                PULSOUT 11, 65000               'pulse solenoid 52msec
                PAUSE 100                       'wait for stepper to catch up
                NEXT
                GOSUB display                   'update display
 skp_mandwn:    GOTO rd_buttons

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

'Display stepper counts and frequency segments at following locations: L1_C4(131):stpcnt1,
'L1_C14(141):stpcnt2, L2_C6(197):freq, L2_14(205):SWcnt.


display:

    SEROUT 16,n96,[I,131,DEC stpcnt1," ",I,141,DEC stpcnt2," ",I,197,DEC freq," ",I,205,DEC SWcnt,"  "]
    old1A=IN0 : old1B=IN1                       'refresh old1A and old1B to avoid encoder slippage
    old2A=IN2 : old2B=IN3                       'refresh old2A and old2B to avoid encoder slippage
    RETURN

'=============================================================================================
beep:                                           'short beep
    FREQOUT 4,250,1000
    RETURN

'=============================================================================================
longbeep:                                       'long (3-tone) beep

    FREQOUT 4,300,800
    FREQOUT 4,300,1000
    FREQOUT 4,500,1200
    RETURN
```