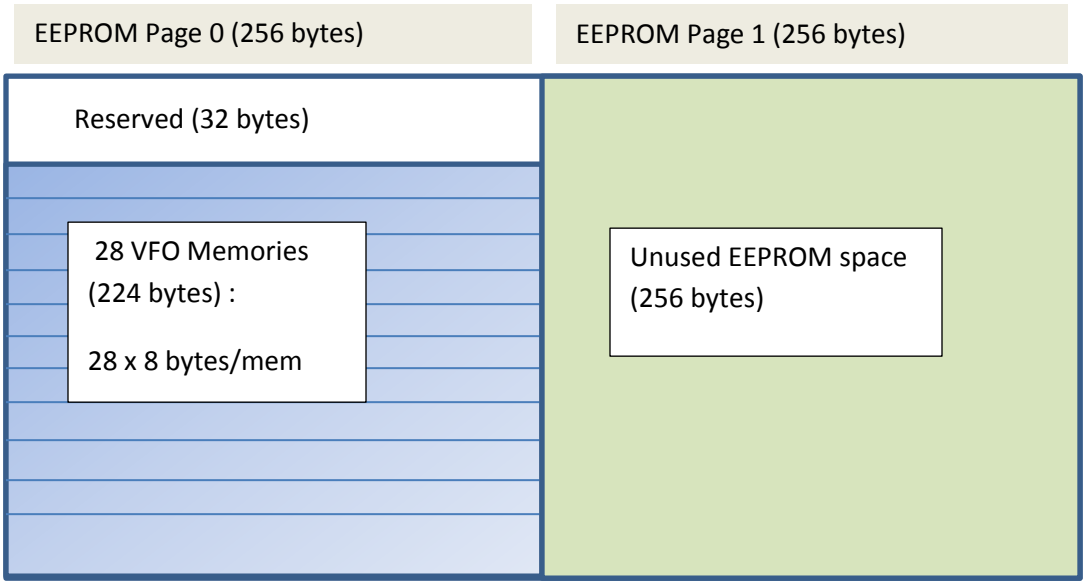
A Programmable Memory Keyer for your DDS Development Kit

By Bruce Hall, W8BH

This article will describe how to add a programmable memory keyer to your DDS Development kit. In the last project (http://w8bh.net/avr/MemoryKeyer.pdf), I created a memory keyer that could send two user-defined messages. This project will build upon that keyer, adding the ability to save messages in EEPROM and edit them without recompiling.

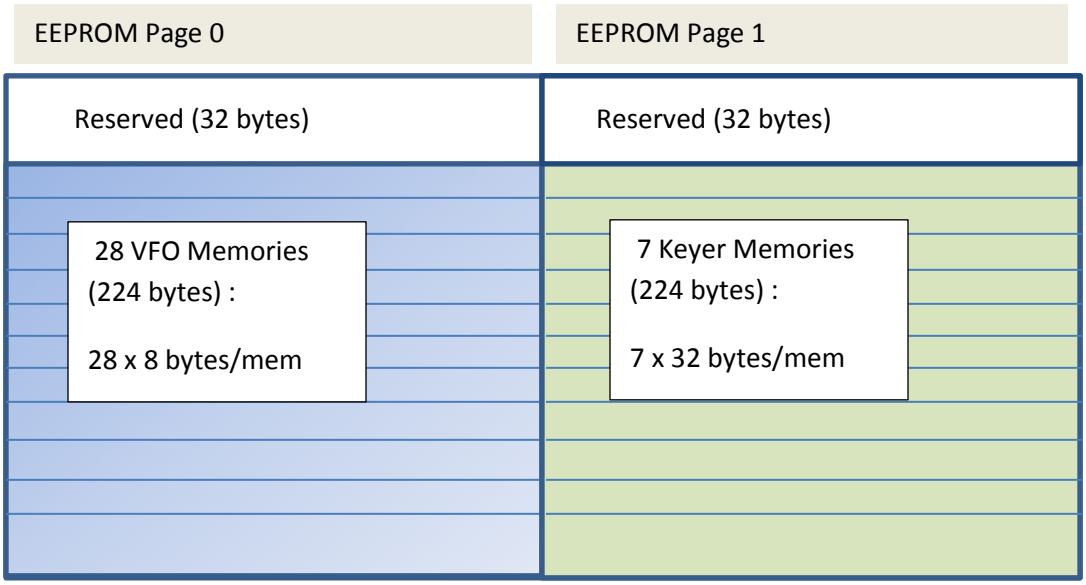
Planning

To get fixed messages from program memory is easy, but to get editable messages from EEPROM will require a bit more work. Where in EEPROM will we store the messages? How many messages can we store? In my EEPROM article, I used almost half of the 512 byte EEPROM for VFO memories, and left the second half for future expansion. The memory map looked as follows:



In planning for keyer memories, the unused 256 bytes could give us a few long messages or a dozen small ones. One of my 3x3 CQ calls takes 28 bytes, so this seems like a reasonable

message size. The LCD displays 32 characters at a time, and 256 divides easily into 32 byte sections, so I choose a message length of 32 bytes. My new memory map looks like this:



I get 7 keyer memories of 32 bytes each, saving the first 32 bytes in the page for future use. We need a way to edit and store these memories. In my VFO article I added a 'mode' to edit the VFO memories, so I decided to do the same here.

I spent some time playing around with the user interface. How should you select the message that you want to edit? I thought that the LCD should show the first message, and that turning the encoder knob should scroll through the messages. You select the message to edit by pressing the encoder button.

The user interface sounds great so far, and gives us a way to select messages. In the next step we can display the message on the screen, and use our encoder to edit it. I got stuck here, though. The encoder already has a defined behavior: to scroll through the messages. How can we ask it to also edit the messages? Do we need to make another mode? And if we made another mode called 'Edit Keyer Message', how would we ensure that we've already selected a message? Does the user have to select the message first, and then change modes to edit it?

The two actions, selecting and editing, interact and are dependent on each other. I thought that creating separate modes for them splits the actions in an unnatural way. So I created two 'submodes' for keyer memories: one for selecting and another for editing. Each submode defines how the user-interface controls behave. Here is a graphic of how the submodes interact:

MODE 3 (KEYER MESSAGES)	**Selection submode**	**Edit submode**
Encoder	Allow the user to scroll through the keyer messages	Allow the user to edit the character at the current cursor position
Button Tap	Select the displayed message, and go to edit submode to edit it.	Advance cursor to next character
Button Hold	Go to next mode	Save the message and go back to selection submode

The three preceeding paragraphs are my feeble attempt at describing how I reasoned out an interface for editing keyer messages. It is a bit confusing. And it is not totally consistent with the way I handled VFO presets. (VFO presets have separate modes for loading and saving. But with presets, saving one does not depend on first selecting it.) I am experimenting! Let me know if you come up with a better way.

How to handle submodes

I use a single bit from my flags variable to mark which submode we're in. Any time that the button is pressed or the encoder is turned, the submode bit is checked and we branch to the appropriate routine. For example, here is the encoder routine:

```
ENCODERMODE3:
       lds    temp1,flags
       sbrs   temp1,2                  ;check for alternate submode
       rjmp   NormEncoder3
       rjmp   AltEncoder3
```

In the code I call the two submodes 'norm' (select the message) and 'alt' (edit the message). We check the bit, and jump to one or the other routine. Simple!

Normal Submode

In this submode we are selecting which keyer message. The encoder scrolls though the keyer messages, and a button tap selects the message.

When the display initializes, the first keyer message is displayed. We need a way to retrieve the message from EEPROM, and put it on the LCD. I decided to load the EEPROM data into a buffer first, and then take whatever I wanted from the buffer and display it on the LCD. After

setting up the source and destination pointers, I transfer 32 bytes (the length of the message) by calling the 8-byte routine four times.

```
LOADEEMSG:
;       loads a keyer message from EEPROM
;       call with message# in temp1
        rcall   SetupEEMsg              ;set source/destination pointers
        rcall   Read8E                  ;read 32 bytes
        rcall   Read8E
        rcall   Read8E
        rcall   Read8E
        ret
```

Now that the message is in SRAM memory, I can manipulate it more easily. There isn't enough room to display the message number and the entire message itself. I decided that showing the message number and first 13 characters of the message, on a single line, would be useful enough to select the message. Here is the code to do it:

```
SHOWMSGPART:
;       displays the first part of the current message on line2
        ldi     temp1,19
        rcall   SetCursor
        ldi     temp2,13                ;do first 13 chars of msg
        ldi     ZH,high(msgbuf)         ;point to msg
        ldi     ZL,low(msgbuf)
sp0:    ld      temp1,Z+                ;get character in msg
        rcall   LCDCHR                  ;put it on LCD
        dec     temp2                   ;all 13 done yet?
        brne    sp0                     ;no, not yet
        ret
```

Now we need a way to do the scrolling. A counter is incremented or decremented, depending on which way the encoder was turned.

```
ENCODERVALUE:
;       increment/decrement a value, depending on encoder rotation
;       call with temp1 = current value
;                 temp2 = lower limit (0-254)
;                 temp3 = upper limit (0-255)
;       returns with new value in temp1

        tst     encoder                 ;which way did encoder turn?
        brmi    ev1                     ;negative = CCW rotation
        cp      temp1,temp3             ;CW rotation------------
        brge    ev2                     ;hard stop at upper limit
        inc     temp1                   ;go to next higher preset
        rjmp    ev2
ev1:    cp      temp2,temp1             ;CCW rotation------------
        brge    ev2                     ;hard stop at lower limit
        dec     temp1                   ;go to next lower preset
ev2:    clr     encoder                 ;ignore any more requests
        ret



NORMENCODER3:
        ldi     temp2,1                 ;set lower limit
```

```
        ldi     temp3,NumMessages       ;set upper limit
        lds     temp1,msgnum            ;get current message #
        rcall   EncoderValue            ;update message # by encoder
        sts     msgnum,temp1            ;save message #
        rcall   ShowMsgIndex            ;and display it
        ret
```

I split out the encoder-specific code into a separate routine called EncoderValue. I call this routine any time that I use the encoder scroll through a range of values. The only tricky bit is the use of the CP TEMP2,TEMP1 instuction at EV1. Normally, in a compare statement you put the value that you are evaluating (temp1) first. For example, CPI TEMP1,13 compares temp1 with the value 13. A following BRGE instruction will then branch if TEMP1 is greater or equal to 13. But here I have reversed the order, and put temp1 in second place. Why? Because there is no AVR instruction for 'less than or equal to'. Instead, I used the instruction for 'greater or equal to' and reversed the order of the operands (registers). Neat!

Finally, a button press must 'select' the message and take us to the editing code. This is as simple as setting the 'alternate submode' bit, and preparing the display for editing. The routine is called when the button tap is lifted (TU = tap up):

```
NORMTU3:
        rcall   SetAltMode              ;alternate behavior for controls
        rcall   InitAlt3                ;init edit submode display
        ret
```

Alternate Submode

In this submode we are editing the current keyer message. We have already loaded the message from EEPROM into a message buffer. The encoder edits the character at the current cursor position, and a button tap advances the cursor to the next character.

To display a 32 bit message we'll need to use the entire 32-character LCD screen. Sending a character to the LCD will put the character at the current cursor position, and then automatically increment the cursor. Unfortunately, the addresses on a 16x2 display are not consecutive! Here is a diagram of the display, showing the hex address of each character:

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

Actually, that's not exactly correct. I am showing the command byte to move the cursor to each address, but I think you get the idea. There is a big numerical gap between the end of the first line and the start of the second line. If you want to place characters sequentially and move cursors from one line to the next, you need to account for this gap. I wrote a routine called SetCursor to correct for the gap:

```
SETCURSOR:
;       call with cursor position (0-31) in temp1
;       will place LCD cursor at desired position
;       (0 = row 1, column 1;  31 = row 2, column 16)
;       Assumes 2x16 LCD display
        push    temp2                   ;preserve registers
        push    temp1
        andi    temp1,$1F               ;allow only 0-31 input
        cpi     temp1,16                ;is it >=16?
        brge    sc1                     ;yes: on 2nd line
        ldi     temp2,$80               ;no, on first line
        rjmp    sc2
sc1:    subi    temp1,16                ;get 2nd line offset
        ldi     temp2,$C0               ;start of second line
sc2:    add     temp1,temp2             ;add for cursor posn
        rcall   LCDCMD                  ;set the cursor
        pop     temp1                   ;restore registers
        pop     temp2
        ret
```

Now it is much easier to display a message. Just read all 32 characters from the message buffer, one at a time, and place them on the LCD with SetCursor:

```
DISPLAYMSG:
;       call with keyer msg# in temp1
;       will display full 32 byte message on LCD
        rcall   LoadEEmsg               ;get msg from EEPROM
        clr     temp2                   ;top-left cursor
        ldi     ZH,high(msgbuf)         ;point to message
        ldi     ZL,low(msgbuf)
dm1:    mov     temp1,temp2
        rcall   SetCursor               ;set cursor position
        ld      temp1,Z+                ;get next char in msg
        tst     temp1                   ;is it 0=done?
        breq    dm2                     ;yes
        rcall   LCDCHR                  ;no, display it on LCD
        inc     temp2                   ;next cursor position
        cpi     temp2,32                ;are we done?
        brne    dm1                     ;no, get next char
dm2:    ret
```

You don't need to set the cursor for every character, like I did, but it was easier to code. SetCursor turned out to be very useful. You might notice that I use it in some places, but in other places just call LCDCMD with a command byte. I should be more consistent…

With the full 32 bit message on the display, we need a way of editing it. I used the encoder to change the character, and the button to move the cursor. Thanks to the new EncoderValue code, the editing routine is pretty simple:

```
ALTENCODER3:
        ldi     temp2,AsciiMin          ;set character limits
        ldi     temp3,AsciiMax
        lds     temp1,ch                ;get current character
        rcall   EncoderValue            ;update character
        sts     ch,temp1                ;save it
        rcall   LCDCHR                  ;and display it
        mov     temp1,temp5
```

```
        rcall  SetCursor                ;place cursor under char
        ret
```

Every time we send a character to the LCD, the cursor advances automatically. You can either reprogram the LCD to not autoadvance, or just reposition the cursor back under the current character. I had just written SetCursor, so I reused it here. I used temp5 as a temporary holder for the cursor position.

The button's job is a little trickier: we have to save the current character, increment the cursor, and load the next character.

```
ALTTU3:
        lds    temp1,ch                 ;get current character
        st     Z+,temp1                 ;store in message buffer
        inc    temp5                    ;update cursor position
        cpi    temp5,32                 ;did we pass end of msg?
        brlo   ta2                      ;no, continue
        sbiw   Z,32                     ;yes, reset msg pointer
        ldi    temp5,0                  ;reset cursor pointer
ta2:    mov    temp1,temp5
        rcall  SetCursor                ;advance cursor
        ld     temp1,Z                  ;get next char from buffer
        tst    temp1                    ;is it 0?
        brne   ta3                      ;no, continue
        ldi    temp1,$20                ;yes, convert to a space
ta3:    sts    ch,temp1                 ;buffer current character
        rcall  LCDCHR                   ;output new char
        mov    temp1,temp5
        rcall  SetCursor                ;put cursor under char
        ret
```

What should happen when the user advances the cursor to the end of the second line? It can't just keep advancing. Should the editor quit and save the message? I tried that at first, but since there isn't any 'backspace' on this editor, it's more useful to go back to the beginning. So I let the cursor wrap from the end of the message back to the top. The SBIW Z,32 resets the pointer to the beginning the message, and the CLR TEMP5 (or LDI TEMP5,0 – same thing) resets the cursor to the top-left position. Another bit of necessary housekeeping is to get rid of any stray 'End-of-message' characters, for which I used ASCII #0. It doesn't need to go here, but I look for them and change them to spaces whenever they are encountered.

Finally, there needs to be a user-action to save the message. I chose button-hold. My hold-down (HD) routine for the edit submode saves the message to EEPROM, and restores the display back to the scrolling messages:

```
ALTHD3:
        rcall  SaveMessage              ;save message to EEPROM
        rcall  SetNormMode              ;normal control behavior
        ldi    temp1,3
        rcall  ChangeMode               ;init normal mode3 display
        ret
```

Code Speed

When answering a CQ, I try to match the sender's code speed. After playing with this keyer for a while I realized that a 'hard-coded' speed was not good enough. So I added a mode for setting the code speed. The basic user-interface for the encoder and button takes only 20 lines of code, and the speed display function is only an additional 13 instructions. Take a look at the source code to see how easy it is.

The most interesting problem is how to convert speed (in WPM) into the correct timing delay for dits and dahs. In the Memory Keyer article, I showed that the duration of each element (dit), in milliseconds, is equal to 1200/WPM. So 20 WPM code requires an element duration of 1200/20 = 60 milliseconds. Now, how do you code for 1200/x? How do you a divide, for that matter? I'll leave that one to the coding experts! I took the easy way, and used a lookup table. First, I made a table in my notebook something like this:

Speed (in WPM)	Timing Delay (in ms)
5	240
6	200
7	171
8	150
9	133
10	120
11	109
12	100
13	92
14	86
15	80
…	
20	60
…	
30	40

Then I copied all of those numbers into a table at the end of the program, like this:

```
ctable:
.db 240, 200, 171, 150, 133, 120, 109, 100
.db  92,  86,  80,  75,  71,  67,  63,  60
.db  57,  55,  52,  50,  48,  46,  44,  43
.db  41,  40
```

Now, if I subtract 5 from the code speed, the resulting number gives me the proper index into the table. For example, to get the delay for 13 WPM, I just go to the 13-5=8th byte after the start of the table, which is 92. Here is the code to do it:

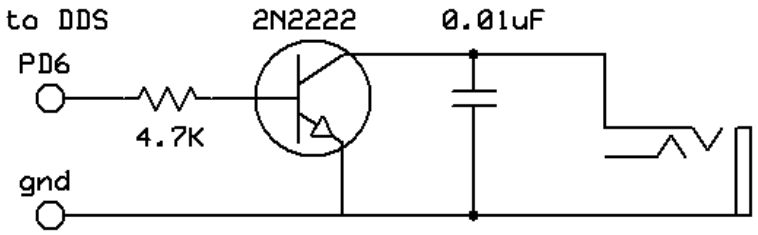
```
GETCWDELAY:
; convert WPM value into CW timing delay
; call with WPM in SPEED variable
; will set timing value in CWDELAY variable
       lds    temp1,speed
       cpi    temp1,MinSpeed            ;is speed >= 5?
       brge   gd1                       ;yes
       ldi    temp1,92                  ;no, use default = 13 WPM
       rjmp   gd2
gd1:   subi   temp1,MinSpeed            ;index into speed table
       ldi    ZH,high(2*ctable)         ;point to table
       ldi    ZL,low(2*ctable)
       add    ZL,temp1                  ;add in index
       clr    temp1
       adc    ZH,temp1
       lpm    temp1,Z                   ;get the value
gd2:   sts    cwdelay,temp1             ;and save it
       ret
```

The ADD ZL,TEMP1 instruction is what adds the (codespeed-5) number to find the correct table entry. This by itself would work in almost all cases, except if your table happened to be located in a spot where ZL was between 231 and 255. For example, if ZL was 250 and our code speed was 30, the ADD would try to add 250+30 = 280. Oops, we can't count past 255 with our 8 bit register! The ADD will give us the incorrect result of 24 instead. To correct the mistake, the next two lines, CLR and ADC, add 1 to the upper register ZH in the event of an overflow (carry).

Does it work?

I watched my little LED blink out code a long time before I bothered hooking it to my rig. I had a microcontroller pin for the output, but how do you hook it up? You QRP guys probably already know how. Just in case, here is a little circuit that works for my rigs:

I used a 2N2222 since I've got a bunch of them. I think a 2N7000 or similar would also work as well, or better. The 2N2222 needs a positive voltage to turn on, and ground to turn off, so I had to modify my KeyUp and KeyDown routines accordingly.



Keyer Interface Circuit

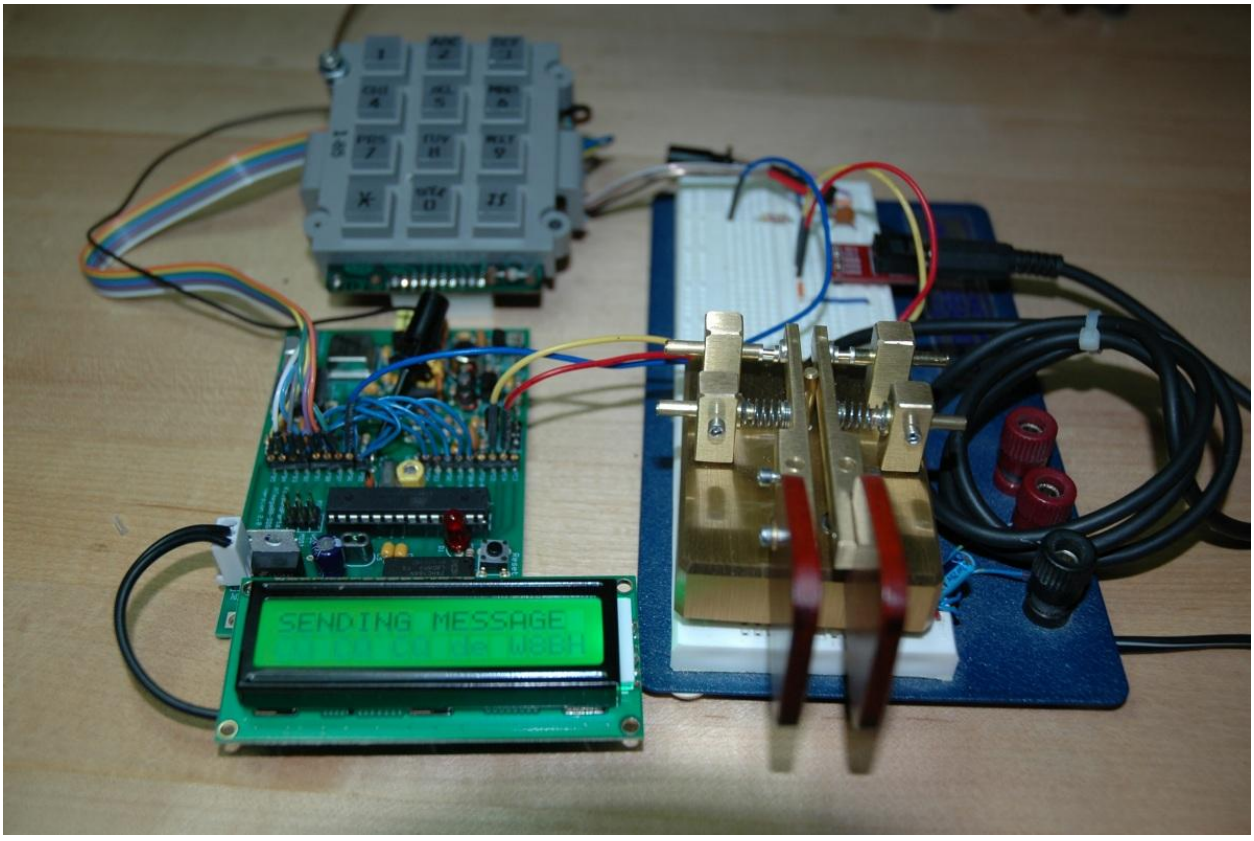
```
KEYDOWN:
       sbi    PortD,KeyOut              ;turn on output line
       cbi    PortC,LED                 ;turn on LED
       ret
```

KeyDown now takes the output pin high with the SBI (set bit I/O) instruction. The LED is active low, and turned on with CBI by taking its pin low.

Initially, my rig would key a fraction of a second every time the DDS was reset. Not good! It took me a while to find the cause. In the original source code, there is a line that enables pullup resistors on all of the PortD outputs. Diz labeled it 'Not sure if we need this'. I was really tempted to remove it! Instead, I just modified it so that the PD6 line is not pulled high:

```
ldi     temp1,$BF                   ;!! W8BH - removed pullup from PD6
out     PORTD,temp1
```

$BF is 10111111 in binary. The zero in bit 6 prevents a pullup resistor from being placed on the key out line. Now the rig doesn't key whenever the DDS resets.



Sending keyer messages

In the earlier Memory keyer article I used the paddle to send messages: while holding the encoder button down, hit the left paddle for message 1 and right paddle for message 2. Now there are 7 messages to choose from. I still wanted to use the paddles somehow and, if I could, keep any new method compatible with the original one. My solution was to hit the paddles repeatedly (or hold them down) until the desired message was selected. For example, a single hit on the left paddle brings up message #1. By making the right paddle advance by 2, a single hit on the right will bring up the second message. For messages 5, 6, and 7, you can actually send the number in morse: dah-dah-dit-dit-dit = 2+2+1+1+1 = 7. The code required that each time the paddle went down during a button press, we'd need to advance our message number and queue up the message for sending:

```
QUEUEMSG1:
;       puts the next available cw message into message buffer
;       displays the pending message on line2
        clr     hold                    ;dont trigger hold while keying in #
        lds     temp1,msgnum            ;get message number
        inc     temp1                   ;go to next number
        cpi     temp1,NumMessages       ;have we passed the last msg?
        brlo    ss1                     ;no, continue
        clr     temp1                   ;yes, so allow escape
        rcall   ShowTuning              ;and erase queued msg display
ss1:    sts     msgnum,temp1            ;get queued value
        tst     temp1                   ;retrieve message number
        breq    ss2                     ;zero = nothing was queued
        rcall   ShowMsgIndex            ;show queued message on LCD
ss2:    rcall   DitWait                 ;keyer debouncer
        rcall   DitWait
        ret
```

Whenever the button was released, we'd see if a message was queued and then send it. This part requires a check in the button tap-up event of mode0:

```
TAPUP0:
        lds     temp1,msgnum            ;get message counter
        tst     temp1                   ;was a message number entered?
        breq    t98                     ;no
        rcall   SendMorseMsg            ;send the message
        clr     temp1                   ;done, so remove msg#
        sts     msgnum,temp1            ;and save it.
t98:    ret
```

Other Stuff

After experimenting with editing and saving the keyer messages, the messages got a bit messed up. I realized that there should be a way to get the messages back to their original state. We need a 'factory reset' button! We don't have any extra buttons, and the one button

we've got already is doing a LOT of stuff. How can we possibly get it to do more? Some consumer gear lets you reset it if you hold down the reset button when turning it on. We can do the same thing with our DDS by checking the encoder button at startup. If it's depressed, then let's do a little factory reset routine, loading our EEPROM with all of the default VFO presets, keyer memories, and whatever else we want to save. Here is how to check the button at, at the end of our startup code:

```
sbis   PinD,Button              ;is button down on startup?
rcall  ProgramEE                ;yes, so initialize EEPROM
```

That was easy! When the button is in its usual state, external pull-up resistors keep the input pin high. The SBIS instruction skips the next line, and the reset is not done. If the button is down, however, the input pin is pulled to ground, SBIS doesn't skip, and the reset is called.

One other thing I wanted to add was a continuation character. The 32-character messages are nice, but you might want a few longer ones rather than a bunch of smaller ones. I made the plus sign '+' a continuation character. It indicates that the next message should also be sent, without pausing between them. You could put all 7 messages together, giving you a maximum message length of (6*31)+32 = 218 characters. This useful addition only took six extra instructions in our sending routine, highlighted below:

```
SENDMORSEMSG:
;       sends the current message in morse code
        ldi    temp1,10                 ;Show 'Sending Message'
        rcall  DisplayLine1             ;on LCD line 1
        lds    temp1,msgnum             ;get current msg#
mr0:    rcall  LoadEEmsg                ;load message from EEPROM
        rcall  MarkEnd                  ;remove trailing spaces
        ldi    ZH,high(msgbuf)          ;point to msg
        ldi    ZL,low(msgbuf)
        rcall  MorseMsg                 ;send out the morse code
        cpi    temp1,ContChar           ;was last char a continuation?
        brne   mr1                      ;no, so done
        lds    temp1,msgnum             ;grab msg#
        inc    temp1                    ;go to next msg#
        sts    msgnum,temp1             ;and remember it
        rjmp   mr0                      ;send next message
mr1:    ldi    temp1,0                  ;restore LCD
        rcall  ChangeMode               ;to tuning mode
```

Other DDS articles

- Keypad tutorial: http://w8bh.net/avr/AddKeypadFull.pdf
- VFO Memory Project: http://w8bh.net/avr/AddMemories.pdf
- Extending Encoder Button Functionality: http://w8bh.net/avr/ButtonEvents.pdf
- How to use the EEPROM: http://w8bh.net/avr/EEPROM.pdf
- A Simple Iambic Keyer: http://w8bh.net/avr/IambicKeyer.pdf
- A Memory Keyer: http://w8bh.net/avr/MemoryKeyer.pdf

Full Source Code

```
;ATmega88.asm
;version 1.00 Feb 23, 2010
;author: W8DIZ Dieter (Diz) Gentzow
;email: w8diz@tampabay.rr.com

;Additions and Modifications
;by Bruce Hall - W8BH
;email: bhall66@gmail.com
;last edited 25 Aug 2010



.include "m328def.inc"                ;use this for ATmega328 chip
;.include "m88def.inc"                ;use this for ATmega88 chip

;***********************************************************
;*      PROGRAM SETTINGS
;***********************************************************

.equ DefTuneRate  =  3                ;default tuning rate: 3= 1KHz step
.equ NumPresets   = 20                ;number of VFO presets (27 max)
.equ NumMessages  =  7                ;number of keyer msgs (7 max)
.equ DefaultSpeed = 13                ;default code speed = 13 WPM
.equ MaxSpeed     = 30                ;highest WPM choice (30 max)
.equ MinSpeed     =  5                ;lowest WPM choice (5 min)

;***********************************************************
;*      I/O PIN DEFINITIONS
;***********************************************************

.equ Col2           = PB0             ;keypad column output
.equ Col3           = PB1             ;keypad column output
.equ Row4           = PB2             ;keypad row input
.equ Row3           = PB3             ;keypad row input
.equ Row2           = PB4             ;keypad row input
.equ Row1           = PB5             ;keypad row input
;    XTAL1          = PB6             ;crystal oscillator input
;    XTAL2          = PB7             ;crystal oscillator input

.equ LCD_DRS        = PC0             ;LCD output
.equ LCD_E          = PC1             ;LCD output
.equ LCD_CP         = PC2             ;LCD output
.equ LED            = PC3             ;LED output
.equ RPaddle        = PC4             ;Paddle input
.equ LPaddle        = PC5             ;Paddle input
;    RESET          = PC6             ;not available (Reset line)
;    NOPIN          = PC7             ;not available (no pin)

.equ DDSenable      = PD0             ;DDS output
.equ DDSclock       = PD1             ;DDS output
.equ STATE          = PD2             ;Encoder input
.equ Button         = PD3             ;Encoder button input
.equ PHASE          = PD4             ;Encoder input
.equ DDSdata        = PD5             ;DDS output
.equ KeyOut         = PD6             ;Keyer output
.equ Col1           = PD7             ;Keypad column output
```

```
;*********************************************************
;*      REGISTER DEFINITIONS
;*********************************************************

.def hold           = r15
.def temp1          = r16
.def temp2          = r17
.def temp3          = r18
.def temp4          = r19
.def temp5          = r20
.def release        = r21
.def StepRate       = r22
.def press          = r23
.def delay          = r24
.def encoder        = r25
;      XL           = r26
;      XH           = r27
;      YL           = r28
;      YH           = r29
;      ZL           = r30
;      ZH           = r31


;*********************************************************
;*     SRAM MEMORY USAGE
;*********************************************************

.dseg
.org $100

LCDrcve0:     .byte 8                 ;buffer for decimal freq readout
rcve0:        .byte 4                 ;buffer for DDS freq numbers
freq0:        .byte 4                 ;buffer for 1 freq numbers
freq1:        .byte 4                 ;buffer for 10 freq numbers
freq2:        .byte 4                 ;buffer for 100 freq numbers
freq3:        .byte 4                 ;buffer for 1,000 freq numbers
freq4:        .byte 4                 ;buffer for 10,000 freq numbers
freq5:        .byte 4                 ;buffer for 100,000 freq numbers
freq6:        .byte 4                 ;buffer for 1,000,000 freq numbers
freq7:        .byte 4                 ;buffer for 10,000,000 freq numbers
preset:       .byte 1                 ;buffer for current frequency preset#
mode:         .byte 1                 ;0=tuning mode
                                      ;1=load presets
                                      ;2=save preset
                                      ;3=keyer memories
                                      ;4=set code speed
speed:        .byte 1                 ;code speed, in words per minute
cwdelay:      .byte 1                 ;timing delay for morse code output
tunerate:     .byte 1                 ;default tuning rate (6=MHz,3=KHz,0=Hz)
ch:           .byte 1                 ;ASCII character buffer
msgnum:       .byte 1                 ;message number
msgbuf        .byte 33                ;keyer message buffer
flags:        .byte 1                 ;bit0 = hold in progress;
                                      ;bit1 = last element sent 'dah'
                                      ;bit2 = alternate submode
```

```
;*********************************************************
;*     INTERRUPT VECTOR TABLE
;*********************************************************
;  use RJMP instructions with ATmega88 chips
;  use JMP instructions for ATmega328 chips

.cseg

.org $000
       jmp    RESET                    ;Program Start
.org INT0addr
       jmp    EINT0                    ;External Interrupt Request 0
.org INT1addr
       jmp    EINT1                    ;External Interrupt Request 1
.org OVF0addr
       jmp    OVF0                     ;Timer/Counter0 Overflow
.org OVF2addr
       Jmp    OVF2                     ;Timer/Counter2 overflow
.org INT_VECTORS_SIZE


;*********************************************************
;*     PROGRAM START
;*********************************************************

RESET:                                 ;init everything here
       ldi    temp1,low(RAMEND)
       out    SPL,temp1                ;Set stack to last internal RAM location
       ldi    temp1,high(RAMEND)
       out    SPH,temp1

;init ports
       ldi    temp1,$0F
       out    DDRC,temp1               ;make all 4 PC0-PC3 pins an output
       cbi    PORTC,LCD_E              ;set LCD_E low
       cbi    PORTC,LCD_CP             ;set LCD_CP low
       sbi    PORTC,LED                ;turn LED off
       ldi    temp1,$BF                ;!! W8BH - removed pullup from PD6
       out    PORTD,temp1

;init misc
       ldi    StepRate,6               ;set to the 1,000,000's position
       clr    encoder                  ;clear encoder interrupt counts
       clr    press                    ;clear press interrupt counts
       ldi    temp1,$05                ;set timer0 prescale divisor to 1024
       out    TCCR0B,temp1             ;using 20.48 XTAL = 50uS

       ldi    temp1,$01
       sts    TIMSK0,temp1             ;enable TIMER0 overflow interrupts

       ldi    temp1,$03
       out    EIMSK,temp1              ;enable int0 and int1 interrupts

       ldi    temp1,0b00001011         ;int1 on falling edge & int0 on rising edge
       sbic   PIND,STATE               ;test state of encoder
       ldi    temp1,0b00001010         ;int1 on falling edge & int0 on falling edge
       sts    EICRA,temp1
       ldi    temp1,$21                ;reset AD9834 and init all registers
       ldi    temp2,$00
       rcall  SHIFT_16                 ;output to DDS chip

       ldi    temp1,$7F                ;freq0 ls 14 bits
       ldi    temp2,$29
```

```
        rcall   SHIFT_16                ;output to DDS chip
        ldi     temp1,$47               ;freq0 ms 14 bits
        ldi     temp2,$FF
        rcall   SHIFT_16                ;output to DDS chip
        ldi     temp1,$80               ;freq1 ls 14 bits
        ldi     temp2,$00
        rcall   SHIFT_16                ;output to DDS chip
        ldi     temp1,$80               ;freq1 ms 14 bits
        ldi     temp2,$80
        rcall   SHIFT_16                ;output to DDS chip
        ldi     temp1,$C0               ;clear phase0
        ldi     temp2,$00
        rcall   SHIFT_16                ;output to DDS chip
        ldi     temp1,$E0               ;clear phase1
        ldi     temp2,$00
        rcall   SHIFT_16                ;output to DDS chip
        ldi     temp1,$20               ;enable output
        ldi     temp2,$00
        rcall   SHIFT_16                ;output to DDS chip
        rcall   DEFAULT_FREQ            ;move default freq to buffers
        rcall   FREQ_OUT                ;output freq bits to DDS chip

        sei                             ;global all interrupt enable
        ldi     temp1,2                 ;blink LED 2x to show system is working
        rcall   BLINK_LED

        rcall   INIT_LCD

        ldi     ZH,high(2*msg1)
        ldi     ZL,low(2*msg1)
        ldi     temp1,$80
        rcall   LCDCMD
        rcall   DISPLAY_LINE
        rcall   ShowFreq


menu:                                   ;main program
        rjmp    W8BH                    ;!!GO TO NEW MAIN PROGRAM
        tst     encoder                 ;check for encoder pulses
        breq    menu5                   ;exit if no pulses
        brpl    menu1                   ;branch if positive
        cbi     PORTC,LED               ;turn LED on
        inc     encoder
        rcall   DecFreq0
        cpi     temp1,55                ;if 55 then all is OK
        brne    menu05
        rcall   IncFreq0                ;correct overflow
        rjmp    menu2
menu05:
        rcall   DecFreq9                ;update the DDS
        rcall   FREQ_OUT
        rcall   ShowFreq
        rjmp    menu2
menu1:
        cbi     PORTC,LED               ;turn LED on
        dec     encoder
        rcall   IncFreq0
        cpi     temp1,55                ;if 55 then all is OK
        brne    menu15
        rcall   DecFreq0                ;correct overflow
        rjmp    menu2
menu15:
        rcall   IncFreq9                ;update the DDS
```

```
        rcall   FREQ_OUT
        rcall   ShowFreq
menu2:
        ldi     delay,20
        rcall   wait
        sbi     PORTC,LED               ;turn LED off
        ldi     delay,20
        rcall   wait
menu5:
        tst     press
        breq    menu9
        dec     press
menu55:
        dec     StepRate
        brpl    menu6
        ldi     StepRate,7
        rjmp    menu6
menu6:
        rcall   ShowCursor
        cbi     PORTC,LED               ;turn  LED on
        ldi     delay,20
        rcall   wait
        sbi     PORTC,LED               ;turn LED off
        rcall   wait
menu9:
        rjmp    menu


;*********************************************************
;*      W8BH - INITIALIZATION CODE
;*********************************************************

W8BH:

;       PORT B SETUP
        ldi     temp1,$03               ;binary 0000.0011
        out     DDRB,temp1              ;set PB0,1 as output
        ldi     temp1,$3C               ;binary 0011.1100
        out     PORTB,temp1             ;set pullups on PB2-5

;       PORT C SETUP
        ldi     temp1,$0F               ;binary 0000.1111
        out     DDRC,temp1              ;set PC0-PC3 as outputs
        ldi     temp1,$38               ;binary 0011.1000
        out     PORTC,temp1             ;set pullups on PC4-5 & LED off

;       PORT D SETUP
        ldi     temp1,$E3               ;b1110.0011 (add bits 6&7)
        out     DDRD,temp1              ;set PD0,1,5,6,7 outputs

;       VARIABLES
        clr     temp1
        sts     mode,temp1              ;start mode0 = normal operation
        sts     flags,temp1             ;nothing to flag yet
        sts     preset,temp1            ;start with no presets
        sts     speed,temp1             ;start with default code speed
        sts     msgnum,temp1            ;start with no pending messages
        clr     release                 ;no button events on startup
        clr     hold                    ;no holds on startup

;       COUNTERS/TIMERS
        ldi     temp1, $07              ;set timer2 prescale divider to 1024
        sts     TCCR2B,temp1
```

```
        ldi     temp1, $01              ;enable TIMER2 overflow interrupt
        sts     TIMSK2,temp1

;       MISC STARTUP CODE
        rcall   CheckEE                 ;make sure EEPROM is initialized
        sbis    PinD,Button             ;is button down on startup?
        rcall   ProgramEE               ;yes, so initialize EEPROM
        rcall   LoadEEspeed             ;get stored code speed
        rcall   LoadEETuneRate          ;get stored tuning rate
        ldi     temp1,0                 ;mode 0 = tuning mode
        rcall   ChangeMode              ;setup display for tuning mode


;*********************************************************
;*  W8BH - REVISED MAIN PROGRAM LOOP
;*********************************************************

MAIN:
        rcall   CheckEncoder            ;check for encoder action
        rcall   CheckButton             ;check for button events
        rcall   CheckHold               ;check for button holds
        rcall   CheckKey                ;check for paddle action
        rcall   CheckKeypad             ;check for keypad action
        rjmp    Main                    ;loop forever


CHECKENCODER:
        tst     encoder                 ;any encoder requests?
        breq    ce9                     ;no, so quit
        lds     temp1,mode
        cpi     temp1,0                 ;are we in normal mode (0)?
        brne    ce1                     ;no, skip
        rcall   EncoderMode0            ;yes, handle it
        rjmp    ce9
ce1:    cpi     temp1,1                 ;are we in mode 1?
        brne    ce2                     ;no, skip
        rcall   EncoderMode1            ;yes, handle it
        rjmp    ce9
ce2:    cpi     temp1,2                 ;are we in mode 2?
        brne    ce3                     ;no, skip
        rcall   EncoderMode2            ;yes, handle it
        rjmp    ce9
ce3:    cpi     temp1,3                 ;are we in mode 3?
        brne    ce4                     ;no, skip
        rcall   EncoderMode3            ;yes, handle it
        rjmp    ce9
ce4:    cpi     temp1,4                 ;are we in mode 4?
        brne    ce5                     ;no, skip
        rcall   EncoderMode4            ;yes, handle it
        rjmp    ce9
ce5:
ce9:    ret


CHECKBUTTON:
        tst     encoder                 ;any encoder requests?
        brne    cb4                     ;wait until encoder is done
        tst     press                   ;any button down events?
        breq    cb1                     ;no, check for button up events?
        rcall   ButtonTapDown           ;do the button down
        dec     press                   ;one less button tap to do
cb1:    tst     release                 ;any button up events?
        breq    cb4                     ;no, so quit
```

```
       lds    temp1,flags
       sbrs   temp1,0                  ;is there a hold in progress?
       rjmp   cb2                      ;no
       cbr    temp1,$01                ;yes, remove hold flag
       sts    flags,temp1              ;save un-held state
       rcall  ButtonHoldUp             ;do hold release
       rjmp   cb3
cb2:   rcall  ButtonTapUp              ;to the Tap Release
cb3:   dec    release                  ;one less release to do
cb4:   ret


CHECKHOLD:
       tst    hold                     ;any new hold event?
       brpl   ck1                      ;no, so quit
       lds    temp1,flags
       sbr    temp1,$01                ;flag the hold
       sts    flags,temp1              ;save it
       rcall  ButtonHoldDown           ;do the hold event
       clr    hold                     ;reset = allow future holds
ck1:   ret


BUTTONTAPUP:
       lds    temp1,mode               ;get mode
       cpi    temp1,0                  ;are we in mode0?
       brne   tu1                      ;no, skip
       rcall  TapUp0                   ;yes, handle it
       rjmp   tu9
tu1:   cpi    temp1,1                  ;are we in mode1?
       brne   tu2                      ;no, skip
       rcall  TapUp1                   ;yes, handle it
       rjmp   tu9
tu2:   cpi    temp1,2                  ;are we in mode2?
       brne   tu3                      ;no, skip
       rcall  TapUp2                   ;yes, handle it
       rjmp   tu9
tu3:   cpi    temp1,3                  ;are we in mode3?
       brne   tu4                      ;no, skip
       rcall  TapUp3                   ;yes, handle it
       rjmp   tu9
tu4:   cpi    temp1,4                  ;are we in mode4?
       brne   tu5                      ;no, skip
       rcall  TapUp4                   ;yes, handle it
       rjmp   tu9
tu5:
tu9:   ret


BUTTONTAPDOWN:
       lds    temp1,mode               ;get mode
       cpi    temp1,0                  ;are we in mode0?
       brne   td1                      ;no, skip
       rcall  TapDown0                 ;yes, handle it
       rjmp   td9
td1:   cpi    temp1,1                  ;are we in mode1?
       brne   td2                      ;no, skip
;      rcall  TapDown1                 ;yes, handle it
       rjmp   td9
td2:   cpi    temp1,2                  ;are we in mode2?
       brne   td3                      ;no, skip
;      rcall  TapDown2                 ;yes, handle it
       rjmp   td9
```

```
td3:    cpi     temp1,3                 ;are we in mode3?
        brne    td4                     ;no, skip
        rcall   TapDown3                ;yes, handle it
        rjmp    td9
td4:    cpi     temp1,4                 ;are we in mode4?
        brne    td5                     ;no, skip
;       rcall   TapDown4                ;yes, handle it
        rjmp    td9
td5:
td9:    ret


BUTTONHOLDUP:
        lds     temp1,mode              ;get mode
        cpi     temp1,0                 ;are we in mode0?
        brne    hu1                     ;no, skip
        rcall   HoldUp0                 ;yes, handle it
        rjmp    hu9
hu1:    cpi     temp1,1                 ;are we in mode1?
        brne    hu2                     ;no, skip
        rcall   HoldUp1                 ;yes, handle it
        rjmp    hu9
hu2:    cpi     temp1,2                 ;are we in mode2?
        brne    hu3                     ;no, skip
        rcall   HoldUp2                 ;yes, handle it
        rjmp    hu9
hu3:    cpi     temp1,3                 ;are we in mode3?
        brne    hu4                     ;no, skip
        rcall   HoldUp3                 ;yes, handle it
        rjmp    hu9
hu4:    cpi     temp1,4                 ;are we in mode4?
        brne    hu5                     ;no, skip
        rcall   HoldUp4                 ;yes, handle it
        rjmp    hu9
hu5:
hu9:    ret


BUTTONHOLDDOWN:
        lds     temp1,mode              ;get mode
        cpi     temp1,0                 ;are we in mode0?
        brne    hd1                     ;no, skip
        rcall   HoldDown0               ;yes, handle it
        rjmp    td9
hd1:    cpi     temp1,1                 ;are we in mode1?
        brne    hd2                     ;no, skip
        rcall   HoldDown1               ;yes, handle it
        rjmp    hd9
hd2:    cpi     temp1,2                 ;are we in mode2?
        brne    hd3                     ;no, skip
        rcall   HoldDown2               ;yes, handle it
        rjmp    hd9
hd3:    cpi     temp1,3                 ;are we in mode3?
        brne    hd4                     ;no, skip
        rcall   HoldDown3               ;yes, handle it
        rjmp    hd9
hd4:    cpi     temp1,4                 ;are we in mode4?
        brne    hd5                     ;no, skip
        rcall   HoldDown4               ;yes, handle it
        rjmp    hd9
hd5:
hd9:    ret
```

```
CHANGEMODE:
;       call this routine with new mode in temp1
;       main action is to change the message on Line 1
        sts     mode,temp1              ;save the new mode
        cpi     temp1,0                 ;mode 0?
        brne    cm1                     ;no, skip
        ldi     temp1,1
        rcall   DisplayLine1            ;yes, show normal title
        rcall   ShowTuning              ;display tuning freq
        rjmp    cm9
cm1:    cpi     temp1,1                 ;mode 1?
        brne    cm2                     ;no, skip
        ldi     temp1,2
        rcall   DisplayLine1            ;yes, show mode 1 title
        rjmp    cm9
cm2:    cpi     temp1,2                 ;mode 2?
        brne    cm3                     ;no, skip
        ldi     temp1,3
        rcall   DisplayLine1            ;yes, show mode 2 title
        rjmp    cm9
cm3:    cpi     temp1,3                 ;mode 3?
        brne    cm4                     ;no, skip
        ldi     temp1,4
        rcall   DisplayLine1            ;yes, show mode 3 title
        rjmp    cm9
cm4:    cpi     temp1,4                 ;mode 4?
        brne    cm5                     ;no, skip
        ldi     temp1,5
        rcall   DisplayLine1            ;yes, show mode 4 title
        rjmp    cm9
cm5:
cm9:    ret


QUICKBLINK:
        cbi     PORTC,LED               ;turn  LED on
        ldi     delay,15                ;keep on 15 ms
        rcall   wait
        sbi     PORTC,LED               ;turn LED off
        ret


ENCODERVALUE:
;       increment/decrement a value, depending on encoder rotation
;       call with temp1 = current value
;                 temp2 = lower limit (0-254)
;                 temp3 = upper limit (0-255)
;       returns with new value in temp1

        tst     encoder                 ;which way did encoder turn?
        brmi    ev1                     ;negative = CCW rotation
        cp      temp1,temp3             ;CW rotation------------
        brge    ev2                     ;hard stop at upper limit
        inc     temp1                   ;go to next higher preset
        rjmp    ev2
ev1:    cp      temp2,temp1             ;CCW rotation------------
        brge    ev2                     ;hard stop at lower limit
        dec     temp1                   ;go to next lower preset
ev2:    clr     encoder                 ;ignore any more requests
        ret
```

```
;*********************************************************
;*  W8BH - MODE 0 (VFO TUNING) ROUTINES
;*********************************************************

ENCODERMODE0:
;       This code taken from original program loop.
;       Called when there is a non-zero value for encoder variable.
;       Negative encoder values = encoder has turned CCW
;       Positive encoder values = encoder has turned CW
;       In mode 0, encoder should increase/decrease the DDS freq

        tst     encoder
        brpl    e02                     ;which way did encoder rotate?
        inc     encoder                 ;remove 1 negative rotation
        rcall   DecFreq0                ;reduce displayed frequency
        cpi     temp1,55                ;55 = all OK
        brne    e01
        rcall   IncFreq0                ;correct freq. underflow
        rjmp    e05
e01     :rcall DecFreq9                 ;reduce magic number
        rjmp    e04
e02:    dec     encoder                 ;remove 1 positive rotation
        rcall   IncFreq0                ;increase displayed frequency
        cpi     temp1,55                ;55 = all OK
        brne    e03
        rcall   DecFreq0                ;correct freq. overflow
        rjmp    e05
e03:    rcall   IncFreq9                ;increase magic number
e04:    rcall   FREQ_OUT                ;update the DDS
        rcall   ShowFreq                ;display new frequency
e05:    rcall   QuickBlink
        ret

TAPDOWN0:
;       This code taken from original program loop.
;       Called when there is a non-zero value for press variable.
;       Non-zero value = number of times button has been pressed
;       In mode 0, button should advance cursor to the right

        dec     StepRate                ;advance cursor position variable
        brpl    b01                     ;position >= 0 (Hz position)
        ldi     StepRate,7              ;no, so go back to 10MHz position
b01:    rcall   ShowCursor
        rcall   QuickBlink              ;flash the LED
        ret

TAPUP0:
        lds     temp1,msgnum            ;get message counter
        tst     temp1                   ;was a message number entered?
        breq    t98                     ;no
        rcall   SendMorseMsg            ;send the message
        clr     temp1                   ;done, so remove msg#
        sts     msgnum,temp1            ;and save it.
t98:    ret

HOLDDOWN0:
;       Called when button has been held down for about 1.6 seconds.
;       In mode 0, action should be to invoke mode1 = scrolling freq. presets
        ldi     temp1,1
        rcall   ChangeMode              ;go to next mode
        ret

HOLDUP0:
```

```
;       Called when entering this mode from another mode
        rcall  ShowTuning
        ret


;*********************************************************
;*  W8BH - MODE 1 (SCROLL FREQUENCY PRESET) ROUTINES
;*********************************************************

INITMODE1:
        ldi    temp1,1                  ;start with first preset
        sts    preset,temp1
        rcall  EELoadMem                ;get it from EEPROM
        rcall  ClearLine2
        rcall  ShowPreset               ;and show it on LCD
        ret

ENCODERMODE1:
        ldi    temp2,1                  ;set lower limit
        ldi    temp3,NumPresets         ;set upper limit
        lds    temp1,preset             ;get current preset#
        rcall  EncoderValue             ;change preset up or down
        sts    preset,temp1             ;save current preset#
        rcall  EELoadMem                ;get the preset in LCD buffer
        rcall  ShowPreset               ;and display it
        ret

TAPUP1:
        rcall  LoadNewFreq              ;DDS output new frequency
        ldi    temp1,0
        rcall  ChangeMode               ;go to mode 0 = normal op.
        ret

HOLDDOWN1:
        ldi    temp1,2
        rcall  ChangeMode
        ret

HOLDUP1:
        rcall  InitMode1
        ret


;*********************************************************
;*  W8BH - MODE 2 (SAVE NEW PRESET) ROUTINES
;*********************************************************

ENCODERMODE2:
        ldi    temp2,1                  ;set lower limit
        ldi    temp3,NumPresets         ;set upper limit
        lds    temp1,preset             ;get current preset#
        rcall  EncoderValue             ;change preset up/down
        sts    preset,temp1             ;save current preset#
        rcall  ShowPresetNum            ;display preset number
        ret

TAPUP2:
        lds    temp1,preset
        rcall  EESaveMem                ;save preset to EEPROM
        ldi    temp1,9
        rcall  DisplayLine2             ;display 'SAVED'
        ldi    temp1,2
        rcall  Blink_LED                ;blink for user feedback
```

```
       ldi    temp1,0
       rcall  ChangeMode                ;return to tuning mode
       ret

HOLDDOWN2:
;      called when leaving this mode
       ldi    temp1,3
       rcall  ChangeMode
       ret

HOLDUP2:
;      called when this entering this mode
       ldi    temp1,1
       sts    preset,temp1              ;start with first preset
       rcall  ClearLine2                ;erase line 2
       rcall  ShowMemFreq               ;show frequency on line2
       rcall  ShowPresetNum             ;show preset number on line2
       ret


;*********************************************************
;*  W8BH - MODE 3 (KEYER MESSAGES) ROUTINES
;*********************************************************

.equ   space   = $20                    ;ASCII space character
.equ   ASCIImin = $20                   ;lowest displayed char
.equ   ASCIImax = $7A                   ;ASCII 'z' character

;this mode has two submodes, which determine how the button
;and encoder controls behave.  Bit2 of the flag variable
;determines which of the two submodes is active.  By default,
;the 'Norm' submode is active

;Norm submode = scrolling list of keyers messages
;Alt submode  = editable display of the current message

;temp5 is used to keep track of alt-mode cursor position

SETALTMODE:
       lds    temp1,flags               ;ALT MODE =
       sbr    temp1,$04                 ;set flag bit2
       sts    flags,temp1               ;save it
       ret

SETNORMMODE:
       lds    temp1,flags               ;NORM MODE =
       cbr    temp1,$04                 ;clear flag bit 2
       sts    flags,temp1               ;save it
       ret

ENCODERMODE3:
       lds    temp1,flags
       sbrs   temp1,2                   ;check for alternate submode
       rjmp   NormEncoder3
       rjmp   AltEncoder3

TAPUP3:
       lds    temp1,flags
       sbrs   temp1,2                   ;check for alternate submode
       rjmp   NormTU3
       rjmp   AltTU3

HOLDDOWN3:
```

```
        lds     temp1,flags
        sbrs    temp1,2                 ;check for alternate submode
        rjmp    NormHD3
        rjmp    AltHD3

HOLDUP3:
        lds     temp1,flags
        sbrs    temp1,2                 ;check for alternate submode
        rjmp    NormHU3
        rjmp    AltHU3

TAPDOWN3:
        ret


NORMENCODER3:
        ldi     temp2,1                 ;set lower limit
        ldi     temp3,NumMessages       ;set upper limit
        lds     temp1,msgnum            ;get current message #
        rcall   EncoderValue            ;update message # by encoder
        sts     msgnum,temp1            ;save message #
        rcall   ShowMsgIndex            ;and display it
        ret

NORMTU3:
        rcall   SetAltMode              ;alternate control behavior
        rcall   InitAlt3                ;init submode display
        ret

NORMHU3:
;       called when entering this mode
        rcall   ClearLine2
        ldi     temp1,1                 ;start with 1st msg
        sts     msgnum,temp1
        rcall   ShowMsgIndex            ;show beginning of 1st msg
        ret

NORMHD3:
;       called when leaving this mode
        ldi     temp1,4
        rcall   ChangeMode
        ret

ALTENCODER3:
        ldi     temp2,AsciiMin          ;set character limits
        ldi     temp3,AsciiMax
        lds     temp1,ch                ;get current character
        rcall   EncoderValue            ;update character
        sts     ch,temp1                ;save it
        rcall   LCDCHR                  ;and display it
        mov     temp1,temp5
        rcall   SetCursor               ;place cursor under char
        ret

ALTTU3:
        lds     temp1,ch                ;get current character
        st      Z+,temp1                ;store in message buffer
        inc     temp5                   ;update cursor position
        cpi     temp5,32                ;did we pass end of msg?
        brlo    ta2                     ;no, continue
        sbiw    Z,32                    ;yes, reset msg pointer
        ldi     temp5,0                 ;reset cursor pointer
ta2:    mov     temp1,temp5
```

```
       rcall  SetCursor                ;advance cursor
       ld     temp1,Z                  ;get next char from buffer
       tst    temp1                    ;is it 0?
       brne   ta3                      ;no, continue
       ldi    temp1,$20                ;yes, convert to a space
ta3:   sts    ch,temp1                 ;buffer current character
       rcall  LCDCHR                   ;output new char
       mov    temp1,temp5
       rcall  SetCursor                ;put cursor under char
       ret

ALTHD3:
       rcall  SaveMessage              ;save message to EEPROM
       rcall  SetNormMode              ;normal control behavior
       ldi    temp1,3
       rcall  ChangeMode               ;init normal mode3 display
       ret

AltHU3:
       ret

INITALT3:
       rcall  ClearDisplay
       lds    temp1,msgnum             ;get message#
       rcall  DisplayMsg               ;show current message
       rcall  HomeCursor
       ldi    temp5,0                  ;track cursor position
       ldi    ZH,high(msgbuf)          ;point to message buffer
       ldi    ZL,low(msgbuf)
       ld     temp1,Z                  ;get first char of msg
       sts    ch,temp1                 ;buffer it
       ret

SAVEMESSAGE:
;      save current message to EEPROM
       ldi    temp1,9                  ;display 'SAVED'
       rcall  DisplayLine1
       lds    temp1,msgnum             ;load message#
       rcall  SaveEEMsg                ;save msg to EEPROM
       ldi    temp1,2
       rcall  Blink_LED
       ret

MARKEND:
;      puts a zero character at the end of the message
       ldi    ZH,high(msgbuf+32)  ;point to end of msg
       ldi    ZL,low(msgbuf+32)
       ld     temp1,-Z                 ;get last character
       cpi    temp1,space              ;is it used?
       brne   me2                      ;no, msg is full
me1:   ld     temp1,-Z                 ;get prior char
       cpi    temp1,space+1            ;is it a space/null?
       brlo   me1                      ;yes, keep looping
       clr    temp1                    ;found non-space char
       adiw   Z,1                      ;point to next char
       st     Z,temp1                  ;add terminating zero
me2:   ret
```

```
;*********************************************************
;*  W8BH - MODE 4 (SET CODE SPEED) ROUTINES
;*********************************************************

ENCODERMODE4:
       ldi    temp2,MinSpeed            ;set speed limits
       ldi    temp3,MaxSpeed
       lds    temp1,speed               ;get speed
       rcall  EncoderValue              ;update speed based on encoder
       sts    speed,temp1               ;save new speed value
       rcall  ShowSpeed                 ;display speed
       ret

TAPUP4:
       rcall  SaveEESpeed               ;save code speed to EEPROM
       ldi    temp1,9
       rcall  DisplayLine2              ;display 'SAVED'
       ldi    temp1,2
       rcall  Blink_LED                 ;blink for user feedback
       ldi    temp1,0
       rcall  ChangeMode                ;return to tuning mode
       ret

HOLDDOWN4:
;      called when leaving this mode
       ldi    temp1,0
       rcall  ChangeMode
       ret

HOLDUP4:
;      called when this entering this mode
       rcall  ClearLine2
       rcall  ShowSpeed
       ret



;*********************************************************
;*  W8BH - KEYPAD ROUTINES
;*********************************************************
;
;      KEYPAD CONNECTIONS (7 wires)
;      Row1 to PB5, Row2 to BP4,
;      Row3 to PB3, Row4 to PB2,
;      Col1 to PD7, Col2 to PB0, Col3 to PB1
;
;      FUNCTIONS
;      # = cursor right
;      * = frequency presets.


CHECKKEYPAD:
       tst    encoder                   ;is encoder busy?
       brne   kp0                       ;wait for encoder to finish
       cbi    PORTD,Col1                ;take column1 low
       ldi    temp1,2                   ;col1 value is 2
       rcall  ScanRows                  ;see if a row went low
       sbi    PORTD,Col1                ;restore column1 high

       cbi    PORTB,Col2                ;take column2 low
       ldi    temp1,1                   ;col2 value is 1
       rcall  ScanRows                  ;see if a row went low
       sbi    PORTB,Col2                ;restore col2 high
```

```
       cbi    PORTB,Col3              ;take column3 low
       ldi    temp1,0                 ;col3 value is 0
       rcall  ScanRows                ;see if a row went low
       sbi    PORTB,Col3              ;restore column3 high
kp0:   ret


SCANROWS:
       clc                            ;clear carry
       sbis   pinB,Row1               ;is row1 low?
       subi   temp1,3                 ;yes, subtract 3
       sbis   pinB,Row2               ;is row2 low?
       subi   temp1,6                 ;yes, subtract 6
       sbis   pinB,Row3               ;is row3 low?
       subi   temp1,9                 ;yes, subtract 9
       sbis   pinB,Row4               ;is row4 low?
       subi   temp1,12                ;yes, subtract 12
       brcc   kp1                     ;no carry = no keypress
       neg    temp1                   ;negate answer
       rcall  PadCommand              ;do something
kp1:   ret


PADCOMMAND:
       cpi    temp1,11                ;special case: is it 0?
       brne   kp2                     ;no, continue
       ldi    temp1,0                 ;yes, replace with real zero
kp2:   cpi    temp1,12                ;special case: "#" command?
       brne   kp3                     ;no, try next command
       inc    press                   ;emulate button press = cursor right
       ldi    temp1,1                 ;1 blink for switch debouncing
       rjmp   kp6                     ;done with '#'
kp3:   cpi    temp1,10                ;special case:"*" command
       brne   kp4                     ;no, try next command
       rcall  LoadNextPreset          ;yes, get next preset
       rjmp    kp6                    ;done with '*'
kp4:   mov    temp2,StepRate          ;no, get current cursor position
       ldi    ZH,high(rcve0)          ;point to frequency value in memory
       ldi    ZL,low(rcve0)           ;16 bits, so need two instructions
kp5:   dec    ZL                      ;advance through frequency digits
       dec    temp2                   ;and advance through cursor positions
       brpl   kp5                     ;until we get to current digit
       ld     temp3,Z                 ;load value at cursor
       sub    temp1,temp3             ;subtract from keypad digit
       mov    encoder,temp1           ;set up difference for encoder routines.
       inc    press                   ;advance cursor position
kp6:   ldi    delay,150               ;simple key debouncer
       rcall  wait                    ;give the LED a rest!
       ret


;*********************************************************
;*  W8BH - FREQUENCY PRESET ROUTINES
;*********************************************************

ZeroMagic:
       ldi    ZH,high(rcve0)          ;point to magic#
       ldi    ZL,low(rcve0)
       ldi    temp1,0
       st     Z+,temp1                ;zero first byte (MSB)
       st     Z+,temp1                ;zero second byte
       st     Z+,temp1                ;zero third byte
```

```
       st     Z+,temp1                ;zero fourth byte (LSB)
       ret

ShowMagic:
       ldi    ZH,high(rcve0)          ;point to magic number
       ldi    ZL,low(rcve0)           ;2 byte pointer
       ldi    temp3,4                 ;counter for 4 byte display
       ldi    temp1,$80               ;display on line1
       rcall  LCDCMD
sh1:   ld     temp1,Z+                ;point to byte to display
       rcall  SHOWHEX                 ;display first nibble
       ldi    temp1,' '               ;add a space
       rcall  LCDCHR                  ;display the space
       dec    temp3                   ;all 4 bytes displayed yet?
       brne   sh1                     ;no, so do the rest
       ret

AddMagic:
;      adds one component to magic according to StepRate
;      0 = Hz rate, 3=Khz rate, 6=MHz rate, 7=10MHz rate
       rcall  IncFreq9
       ret

BuildMagic:
       push   StepRate                ;save StepRate
       ldi    XH,high(LCDrcve0)       ;point to LCD digits
       ldi    XL,low(LCDrcve0)        ;16bit pointer
       ldi    StepRate,7              ;Start with 10MHz position
bm1:   ld     temp3,X+                ;get next LCD digit
       tst    temp3                   ;is it zero?
       breq   bm3                     ;yes, so go to next digit
bm2:   rcall  AddMagic                ;no, so add magic component
       dec    temp3                   ;all done with this component
       brne   bm2                     ;no, add some more
bm3:   dec    StepRate                ;all done with freq. positions?
       brne   bm1                     ;no, go to next (lowest) position
       pop    StepRate                ;restore StepRate
       ret

LoadPMmem:
       ldi    ZH,high(freqLCD*2)      ;point to the preset table (-8 bytes)
       ldi    ZL,low(freqLCD*2)       ;16bit pointer
lp1:   adiw   ZL,8                    ;point to next frequency preset
       dec    temp1                   ;get to the right preset yet?
       brne   lp1                     ;no, keep looking
       ldi    YH,high(LCDrcve0)       ;yes, point to LCD digits
       ldi    YL,low(LCDrcve0)        ;16bit pointer
       ldi    temp2,8                 ;there are 8 frequency digits
lp2:   lpm    temp1,Z+                ;get an LCD digit from FLASH mem
       st     Y+,temp1                ;and put into LCD display buffer
       dec    temp2                   ;all digits done?
       brne   lp2                     ;not yet
       ret

LoadNewFreq:
       rcall  ZeroMagic               ;clear out old magic number
       rcall  BuildMagic              ;build new one based on current freq
       rcall  Freq_Out                ;send new magic to DDS
;      rcall  ShowMagic               ;show magic# on line 1 (debugging)
;nf1:  tst    encoder                 ;wait for encoder twist
;      breq   nf1
       ret
```

```
LoadNextPreset:
       lds    temp1,preset              ;check current preset
       cpi    temp1,NumPresets          ;at top of list?
       brne   ln1                       ;no, continue
       clr    temp1                     ;yes, start at beginning
ln1:   inc    temp1                     ;go to next preset#
       sts    preset,temp1              ;save save it
       rcall  EELoadMem                 ;get preset from EE
       rcall  LoadNewFreq               ;update DDS with new freq
       rcall  ShowTuning                ;display it
       ret



;*********************************************************
;*  W8BH - Timer 2 Overflow Interrupt Handler
;*********************************************************
;      This handler is called every 8 ms @ 20.48MHz clock
;      Increments HOLD counter (max 128) when button held
;      Resets HOLD counter if button released before hold met
;      Sets hold & down flags in button state register.

OVF2:
       push   temp1
       in     temp1,SREG                ;save status register
       push   temp1
       ldi    temp1,90                  ;256-90=160; 160*50us = 8ms
       sts    TCNT2,temp1               ;reduce cycle time to 8 ms
       tst    hold                      ;counter at max yet?
       brmi   ov1                       ;not yet
       sbic   pinD,BUTTON
       clr    hold                      ;if button is up, then clear
       sbis   pinD,BUTTON
       inc    hold                      ;if button is down, then count
ov1:   pop    temp1
       out    SREG,temp1                ;restore status register
       pop    temp1
       reti

;*********************************************************
;*  W8BH - External Interrupt 1 Handler
;*********************************************************
;      This handler is replaces the original EXT_INT1 code
;      It is called when a logic-level change on the
;      external interrupt 1 (pushbutton) pin occurs.
;      Press is incremented on button-down events.
;      Release is incremented on button-up events.

EINT1:
       push   temp1                     ;save temp1 register
       in     temp1,SREG
       push   temp1                     ;save status register
       lds    temp1,EICRa               ;get interrupt control register
       sbrs   temp1,2                   ;bit2: rising edge =0, falling edge =1
       rjmp   ei1                       ; -- here is the falling-edge code --
       cbr    temp1,$04                 ;falling edge '11' -> rising edge '10'
       inc    release                   ;count the button-up
       rjmp   ei2                       ; -- here is the rising-edge code --
ei1:   sbr    temp1,$04                 ;rising edge '10' -> falling edge '11'
       inc    press                     ;count the button-down
ei2:   sts    EICRa,temp1               ;save interrupt control register
       pop    temp1
       out    SREG,temp1                ;restore status register
```

```
       pop    temp1                    ;restore temp1 register
       reti

;*********************************************************
;*  W8BH - External Interrupt 0 Handler
;*********************************************************
;      This handler is replaces the original EXT_INT0 code
;      It is called when a logic-level change on the
;      external interrupt 0 (encoder state) pin occurs.
;      Press is incremented on button-down events.
;      Release is incremented on button-up events.

EINT0:
       push   temp1                    ;save temp1 register
       in     temp1,SREG               ;save the status register
       push   temp1
       lds    temp1,EICRA              ;get current interrupt mode
       sbrs   temp1,0                  ;is mode rising-edge?
       rjmp   i02                      ;no, so go to falling edge (bit0=0)
       cbr    temp1,$01                ;yes, clear bit 0
       sts    EICRA,temp1              ;change mode to falling-edge
       sbis   PIND,PHASE               ;is PHASE=1?
       rjmp   i01                      ;no, increase encoder (CW rotation)
       dec           encoder           ;yes, decrease encoder (CCW rotation)
       rjmp   i04
i01:   inc           encoder
       rjmp   i04
i02:                                   ;current mode = falling-edge
       sbr    temp1,$01                ;set bit 0
       sts    EICRA,temp1              ;change mode to rising-edge
       sbis   PIND,PHASE               ;is PHASE=1?
       rjmp   i03                      ;no, decrease encoder (CCW rotation)
       inc    encoder                  ;yes, increase encoder (CW rotation)
       rjmp   i04
i03:   dec    encoder
i04:   pop    temp1
       out    SREG,temp1               ;restore the status register
       pop    temp1                    ;restore temp1 register
       reti

;*********************************************************
;*  W8BH - LCD Display routines
;*********************************************************


HOMECURSOR:
;      puts the cursor at top-left
       push   temp1                    ;preserve register
       ldi    temp1,$80                ;cursor at top-left
       rcall  LCDCMD                   ;do it
       pop    temp1                    ;restore register
       ret

HOMELINE2:
;      puts the cursor at beginning of line2
       push   temp1                    ;preserve register
       ldi    temp1,$C0                ;cursor on line2
       rcall  LCDCMD                   ;do it
       pop    temp1                    ;restore register
       ret


CLEARDISPLAY:
```

```
;       clears the LCD display & puts cursor at top-left
        push    temp1                   ;save register
        ldi     temp1,1                 ;clear display command
        rcall   LCDCMD                  ;do it
        rcall   HomeCursor              ;put cursor @ top-left
        pop     temp1                   ;restore register
        ret

DISPLAYMSG:
;       call with keyer msg# in temp1
;       will display full 32 byte message on LCD
        push    temp1
        push    temp2                   ;preserve registers
        push    ZH
        push    ZL
        rcall   LoadEEmsg               ;get msg from EEPROM
        clr     temp2                   ;top-left cursor
        ldi     ZH,high(msgbuf)         ;point to message
        ldi     ZL,low(msgbuf)
dm1:    mov     temp1,temp2
        rcall   SetCursor               ;set cursor position
        ld      temp1,Z+                ;get next char in msg
        tst     temp1                   ;is it 0=done?
        breq    dm2                     ;yes
        rcall   LCDCHR                  ;no, display it on LCD
        inc     temp2                   ;next cursor position
        cpi     temp2,32                ;are we done?
        brne    dm1                     ;no, get next char
dm2:    pop     ZL                      ;restore registers
        pop     ZH
        pop     temp2
        pop     temp1
        ret

DISPLAYLINE1:
;       displays a 16-character msg on line 1
;       call with msg# in temp1
        push    temp1
        mov     temp2,temp1
        ldi     temp1,$80               ;use line 1
        rcall   LCDCMD
        rcall   DISPLAY16               ;send 16 characters
        pop     temp1
        ret

DISPLAYLINE2:
;       displays a 16-character msg on line 2
;       call with msg# in temp1
        push    temp1
        mov     temp2,temp1
        ldi     temp1,$C0               ;use line 2
        rcall   LCDCMD
        rcall   DISPLAY16               ;send 16 characters
        pop     temp1
        ret

SETCURSOR:
;       call with cursor position (0-31) in temp1
;       will place LCD cursor at desired position
;       (0 = row 1, column 1;  31 = row 2, column 16)
;       Assumes 2x16 LCD display
        push    temp2                   ;preserve registers
        push    temp1
```

```
       andi   temp1,$1F                ;allow only 0-31 input
       cpi    temp1,16                 ;is it >=16?
       brge   sc1                      ;yes: on 2nd line
       ldi    temp2,$80                ;no, on first line
       rjmp   sc2
sc1:   subi   temp1,16                 ;get 2nd line offset
       ldi    temp2,$C0                ;start of second line
sc2:   add    temp1,temp2              ;add for cursor posn
       rcall  LCDCMD                   ;set the cursor
       pop    temp1                    ;restore registers
       pop    temp2
       ret

DISPLAY16:
;      displays a 16-character msg
;      call with msg# in temp2
       push   ZH
       push   ZL
       push   temp3
       ldi    ZH,high(messages*2-16)
       ldi    ZL,low(messages*2-16)
di1:   adiw   Z,16                     ;add 16 for each message
       dec    temp2                    ;add enough?
       brne   di1                      ;no, add some more
       ldi    temp3,16                 ;16 characters
di2:   lpm    temp1,Z+                 ;get the next character
       rcall  LCDCHR                   ;put character on LCD
       dec    temp3                    ;all 16 chars sent?
       brne   di2                      ;no, so repeat
       pop    temp3
       pop    ZL
       pop    ZH
       ret

SHOWDECIMAL:
;displays a number 00-99 on the LCD
       push   temp1                    ;preserve registers
       push   temp2
       push   temp3
       clr    temp2                    ;10's counter
sd1:   cpi    temp1,10                 ;at least 10 remaining?
       brlo   sd2                      ;no, done counting 10's
       inc    temp2                    ;count the next 10
       subi   temp1,10                 ;remove the next 10
       brpl   sd1                      ;loop until all 10's gone
sd2:   mov    temp3,temp1              ;save 10's counter
       mov    temp1,temp2
       rcall  ShowDec                  ;display 10's digit
       mov    temp1,temp3              ;get 1's digit
       rcall  ShowDec                  ;and display it
       ldi    temp1,' '
       rcall  LCDCHR                   ;put a space after number
       pop    temp3                    ;restore registers
       pop    temp2
       pop    temp1
       ret

SHOWMEMFREQ:
;      Displays the frequency in a more compact form: 'XX.XXXXXX'
       ldi    temp1,$C5                ;second line, indented
       rcall  LCDCMD
       ldi    ZH,high(LCDrcve0)        ;point to LCD freq buffer
       ldi    ZL,low(LCDrcve0)
```

```
        ld      temp1,Z+                ;get first digit (10 MHz posn)
        rcall   ShowDec                 ;and show it
        ld      temp1,Z+                ;get second digit (1 MHz posn)
        rcall   howDec                  ;and show it
        ldi     temp1,'.'               ;decimal point
        rcall   LCDCHR                  ;and show it
        ldi     temp2,6                 ;for the next 6 digits
cf1:    ld      temp1,Z+                ;get them from buffer
        rcall   SHOWDEC                 ;and show them on LCD
        dec     temp2                   ;done all of them?
        brne    cf1                     ;not yet
        ret

SHOWINDEX:
;       displays a two-digit index number at the beginning of line 2
;       call with number in temp1
        push    temp1                   ;preserve register
        rcall   HomeLine2               ;start beginning of line2
        rcall   ShowDecimal             ;show the two-digit number
        ldi     temp1,17                ;return cursor under number
        rcall   SetCursor
        pop     temp1                   ;restore register
        ret

SHOWPRESETNUM:
;       displays current preset# on line2
        lds     temp1,preset            ;get preset#
        rcall   ShowIndex
        ret

SHOWMSGNUM:
;       displays current msg# on line2
        lds     temp1,msgnum
        rcall   ShowIndex
        ret

SHOWMSGPART:
;       displays the first part of the current message on line2
        ldi     temp1,19
        rcall   SetCursor
        ldi     temp2,13                ;do first 13 chars of msg
        ldi     ZH,high(msgbuf)         ;point to msg
        ldi     ZL,low(msgbuf)
sp0:    ld      temp1,Z+                ;get character in msg
        rcall   LCDCHR                  ;put it on LCD
        dec     temp2                   ;all 13 done yet?
        brne    sp0                     ;no, not yet
        ret

SHOWMSGINDEX:
;       will display msg# & part of msg on line2
        lds     temp1,msgnum            ;get msg#
        rcall   LoadEEmsg               ;load msg from EEPROM
        rcall   ShowMsgPart
        rcall   ShowMsgNum
        ret

SHOWPRESET:
        rcall   ShowMemFreq             ;show preset frequency
        rcall   ShowPresetNum
        ret

SHOWSPEED:
```

```
;       displays current code speed 'xx WPM' on line2
        ldi     temp1,$C4                   ;second line, indented
        rcall   LCDCMD
        lds     temp1,speed
        rcall   ShowDecimal                 ;display number
        ldi     temp1,'W'                   ;display ' WPM'
        rcall   LCDCHR
        ldi     temp1,'P'
        rcall   LCDCHR
        ldi     temp1,'M'
        rcall   LCDCHR
        ldi     temp1,$C5                   ;put cursor under number
        rcall   LCDCMD
        ret

SHOWTUNING:
        rcall   ClearLine2
        rcall   ShowFreq
        lds     StepRate,TuneRate
        rcall   ShowCursor
        ret

CLEARLINE2:
        push    temp1
        push    temp2
        ldi     temp1,$C0
        rcall   LCDCMD
        ldi     temp2,16
cl1:    ldi     temp1,' '
        rcall   LCDCHR
        dec     temp2
        brne    cl1
        pop     temp2
        pop     temp1
        ret


;*********************************************************
;*  W8BH - EEPROM routines
;*********************************************************

;Data is transferred to/from temp1 (single byte) or Z (multiple bytes)
;EE address must be put into Y prior to call
;See ATMEL application note "AVR100"

.equ            SigByte1 = 'B'              ;first signature byte
.equ            SigByte2 = 'H'              ;second signature byte

READEE:
        sbic    EECR,EEPE                   ;busy writing EEPROM?
        rjmp    ReadEE                      ;yes, so wait
        out     EEARH,YH                    ;set up address reg.
        out     EEARL,YL
        sbi     EECR,EERE                   ;strobe the read bit
        in      temp1,EEDR                  ;get the data
        ret

WRITEEE:
        sbic    EECR,EEPE                   ;busy writing EEPROM?
        rjmp    WriteEE                     ;yes, so wait
        out     EEARH,YH                    ;set up address reg.
        out     EEARL,YL
        out     EEDR,temp1                  ;put data in data reg.
```

```
        cli                             ;dont interrupt the write
        sbi     EECR,EEMPE              ;master write enable
        sbi     EECR,EEPE               ;strobe the write bit
        sei                             ;interrupts OK now
        ret

READ8E:                                 ;read 8 bytes from EE
        ldi     temp2,8                 ;counter=8
r81:    rcall   ReadEE                  ;get byte from EE
        st      Z+,temp1                ;move byte to destination
        adiw    Y,1                     ;go to next EE addr
        dec     temp2
        brne    r81
        ret

WRITE8E:                                ;write 8 bytes to EE
        ldi     temp2,8                 ;counter=8
r82:    ld      temp1,Z+                ;get byte from source
        rcall   WriteEE                 ;store byte in EE
        adiw    Y,1                     ;go to next EE addr
        dec     temp2
        brne    r82
        ret

FillEE:
;       repeatedly writes a byte to EE
;       call with byte in temp1, count in temp2
;       starting address in YH:YL

        rcall   WriteEE                 ;write temp1 value to EEPROM
        adiw    Y,1                     ;go to next address
        dec     temp2                   ;count finished?
        brne    FillEE                  ;loop until done
        ret

INITEEPAGE0:
        clr     YH                      ;go to EEPROM page 0
        clr     YL                      ;at beginning of page
        ldi     temp1,SigByte1          ;load first signature byte
        rcall   WriteEE                 ;write it
        inc     YL                      ;go to byte 1
        ldi     temp1,SigByte2          ;load second signature byte
        rcall   WriteEE                 ;write it

        inc     YL
        ldi     temp1,DefaultSpeed
        rcall   WriteEE                 ;write default CW speed

        inc     YL
        ldi     temp1,DefTuneRate
        rcall   WriteEE                 ;write default tuning rate

        inc     YL
        clr     temp1
        ldi     temp2,28
        rcall   FillEE                  ;fill next 28 bytes with 0

        ldi     temp2,NumPresets*8      ;load # of preset bytes
        ldi     ZH,high(presets*2)      ;point to preset bytes
        ldi     ZL,low(presets*2)
pe1:    lpm     temp1,Z+                ;get byte from program memory
        rcall   WriteEE                 ;store byte in EE
        adiw    Y,1                     ;go to next EE addr
```

```
        dec     temp2                   ;all preset bytes written?
        brne    pe1                     ;loop until all written
        ret

INITEEPAGE1:
        ldi     YH,1                    ;go to EEPROM page 1
        clr     YL                      ;at beginning of page
        ldi     temp2,32                ;create space for 32 values
        clr     temp1
        rcall   FillEE                  ;fill next 32 bytes with 0
        ldi     ZH,high(cwmsg*2)        ;point to keyer messages
        ldi     ZL,low(cwmsg*2)
        ldi     temp2,224               ;7 messages * 32 bytes/msg
pe2:    lpm     temp1,Z+                ;get next message char
        rcall   WriteEE                 ;store byte in EE
        adiw    Y,1                     ;go to next EE addr
        dec     temp2                   ;all bytes written yet?
        brne    pe2                     ;loop until done
        ret

ProgramEE:
;       copy default memories from program FLASH to EE
        ldi     temp1,8                 ;'FACTORY RESET' funny
        rcall   DisplayLine1            ;display it
        rcall   InitEEPage0             ;initialize VFO presets
        rcall   InitEEPage1             ;initialize keyer messages
        ret

CHECKEE:
;       looks to see if EE has been loaded with default presets
;       if not, defaults are programmed into the EE
        clr     YH
        clr     YL                      ;go to byte 00
        rcall   ReadEE                  ;look at first signature byte
        cpi     temp1,SigByte1          ;is it correct?
        brne    ee1                     ;no, so store defaults
        inc     YL                      ;go to byte 01
        rcall   ReadEE                  ;look at second signature byte
        cpi     temp1,SigByte2          ;is it correct?
        brne    ee1                     ;no, so store defaults
        rjmp    ee2                     ;signature byte OK, so done
ee1:    rcall   ProgramEE               ;write defaults to EE
ee2:    ret


EESETUPMEM:
;       called by LoadMem & SaveMem
;       to set up source/destination EEPROM addresses
;       call with preset# in temp1
        clr     YH
        ldi     YL,24                   ;point to EEPROM
        ldi     ZH,high(LCDrcve0)       ;point to LCD buffer
        ldi     ZL,low(LCDrcve0)
ge0:    adiw    Y,8                     ;increment 8 bytes/preset
        dec     temp1                   ;correct preset yet?
        brne    ge0                     ;loop until done
ge1:    ret

EELOADMEM:
;       specify the preset# in temp1
;       will return the EE memory into LCDrcve0
        rcall   EESetupMem
        rcall   Read8E
```

```
        ret

EESAVEMEM:
;       specify the preset# in temp1
;       will save frequency in LCDrcve0 to EE
        rcall  EESetupMem
        rcall  Write8E
        ret

LOADEESPEED:
;       loads the code speed stored in EEPROM
        clr    YH
        ldi    YL,2                     ;point to CW speed
        rcall  ReadEE
        sts    speed,temp1
        rcall  GetCWDelay               ;convert WPM into timing delay
        ret

LOADEETUNERATE:
;       loads the tune rate stored in EEPROM
        clr    YH
        ldi    YL,3                     ;point to tune rate addr
        rcall  ReadEE                   ;get it from EEPROM
        sts    TuneRate,temp1           ;and load it into SRAM
        ret

SAVEEESPEED:
;       saves the code speed in EEPROM
        clr    YH
        ldi    YL,2                     ;point to CW speed addr
        lds    temp1,speed              ;get current CW rate
        rcall  GetCWDelay               ;convert WPM into timing delay
        rcall  WriteEE                  ;save it to EEPROM
        ret

SAVEEETUNERATE:
;       saves the tune rate stored in EEPROM
        clr    YH
        ldi    YL,3                     ;point to tune rate addr
        lds    temp1,TuneRate           ;get current tune rate
        rcall  WriteEE                  ;and save it.
        ret

SETUPEEMSG:
;       called by Load/Save EEMsg routines
;       to set up source/destination addresses
        ldi    YH,1                     ;keyer memories are in page 1
        ldi    YL,0                     ;start of page
        ldi    ZH,high(msgbuf)          ;point to message buffer
        ldi    ZL,low(msgbuf)
sa0:    cpi    temp1,0
        breq   sa1
        adiw   Y,32                     ;add 32 bytes for each message
        dec    temp1                    ;loop until done
        brne   sa0
sa1:    ret

SAVEEEMSG:
;       saves the keyer message in EEPROM
;       call with message# in temp1
        rcall  SetupEEMsg               ;set source/destination pointers
        rcall  Write8E                  ;write 32 bytes
        rcall  Write8E
```

```
        rcall  Write8E
        rcall  Write8E
        ret

LOADEEMSG:
;       loads a keyer message from EEPROM
;       call with message# in temp1
        rcall  SetupEEMsg               ;set source/destination pointers
        rcall  Read8E                   ;read 32 bytes
        rcall  Read8E
        rcall  Read8E
        rcall  Read8E
        ret



;*********************************************************
;*  W8BH - Iambic Keyer routines
;*********************************************************
;
; Left paddle (dit) = Port C, bit 5
; Right paddle (dah) = Port C, bit 4
; Keyer output line = Port D, bit 6

.equ          DahFlag       = 1         ;0=dit, 1=dah

CHECKKEY:
;       Checks to see if either of the paddles have been pressed.
;       Paddle inputs are active low
        lds    temp2,flags              ;get flags in register
        sbis   PinC,LPaddle             ;dit (left) paddle pressed?
        rcall  LPaddleDown              ;yes, so do it
        sbis   PinC,RPaddle             ;dah (right) paddle pressed?
        rcall  RPaddleDown              ;yes, so do it
        sts    flags,temp2              ;save flags
        ret

LPADDLEDOWN:
;       Come here is the left (dit) paddle is pressed
        sbis   PinD,Button              ;is encoder button down?
        rjmp   QueueMsg1                ;yes, so do message1
        sbis   PinC,RPaddle             ;are both paddles pressed?
        rjmp   Iambic                   ;yes, so iambic mode
        rcall  Dit                      ;no, so just send a dit
        ret

RPADDLEDOWN:
; Come here is the left (dit) paddle is pressed
        sbis   PinD,Button              ;is encoder button down?
        rjmp   QueueMsg2                ;yes, so do message2
        sbis   PinC,LPaddle             ;are both paddles pressed?
        rjmp   Iambic                   ;yes, so iambic mode
        rcall  Dah                      ;no, so just send a dah
        ret

IAMBIC:
;       Come here if both paddles are pressed
        sbrc   temp2,DahFlag            ;was the last element a Dah?
        rjmp   Dit                      ;yes, so do a dit now
        rjmp   Dah                      ;no, so do a dah now

DIT:
        rcall  KeyDown
```

```
        rcall   DitWait                 ;key down for 1 dit
        rcall   KeyUp
        rcall   DitWait                 ;key up for 1 dit
        cbr     temp2,1<<DahFlag        ;remember dit sent
        ret

DAH:
        rcall   KeyDown
        rcall   DahWait                 ;key down for 1 dah
        rcall   KeyUp
        rcall   DitWait                 ;key up for 1 dit
        sbr     temp2,1<<DahFlag        ;remember dah sent
        ret

KEYDOWN:
        sbi     PortD,KeyOut            ;turn on output line
        cbi     PortC,LED               ;turn on LED
        ret

KEYUP:
        cbi     PortD,KeyOut            ;turn off output line
        sbi     PortC,LED               ;turn off LED
        ret

GETCWDELAY:
; convert WPM value into CW timing delay
; call with WPM in SPEED variable
; will set timing value in CWDELAY variable
        push    temp1                   ;preserve registers
        push    ZH
        push    ZL
        lds     temp1,speed
        cpi     temp1,MinSpeed          ;is speed >= 5?
        brge    gd1                     ;yes
        ldi     temp1,92                ;no, use default = 13 WPM
        rjmp    gd2
gd1:    subi    temp1,MinSpeed          ;index into speed table
        ldi     ZH,high(2*ctable)       ;point to table
        ldi     ZL,low(2*ctable)
        add     ZL,temp1                ;add in index
        clr     temp1
        adc     ZH,temp1
        lpm     temp1,Z                 ;get the value
gd2:    sts     cwdelay,temp1           ;and save it
        pop     ZL                      ;restore registers
        pop     ZH
        pop     temp1
        ret

DITWAIT:
        lds     delay,cwdelay           ;get # of milliseconds for dit
        rcall   wait                    ;and wait that long
        ret

DAHWAIT:                                ;wait for 3 dits
        rcall   DitWait
        rcall   DitWait
        rcall   DitWait
        ret

WORDWAIT:
        ldi     temp1,4                 ;wait for 4 addl dits
wd1:    rcall   DitWait                 ;(in addn to 3 post-char dits
```

```
       dec    temp1                    ;for a total of 7 dits)
       brne   wd1
       ret


;*********************************************************
;*  W8BH - Memory Keyer routines
;*********************************************************

.equ          ContChar = '+'            ;continuation character

MORSEOUT:
;      call this routine with the encoded morse byte in temp1
       cpi    temp1,$80                 ;found stop bit yet:
       breq   mo2                       ;yes, so quit
       lsl    temp1                     ;no, get next bit into carry
       brcs   mo1                       ;is the bit a dit? (bit=1)
       rcall  dah                       ;no, so send a dah
       rjmp   MorseOut
mo1:   rcall  dit                       ;yes, so send a dit
       rjmp   MorseOut
mo2:   rcall  DitWait                   ;end of char spacing
       rcall  DitWait
mo3:   ret

CQTEST:
;      sends a CQ
       ldi    temp1,$58                 ;binary 0101.1000 = 'C'
       rcall  MorseOut
       ldi    temp1,$28                 ;binary 0010.1000 = 'Q'
       rcall  MorseOut
       ret

ASCIITOMORSE:
;      Call with an ASCII character in temp1
;      This routine will convert it into a coded morse character
;      If input is control or graphic character, output = $80
       push   ZH                        ;preserve Z pointer
       push   ZL
       ldi    ZH,high(2*mtable)         ;point to morse table
       ldi    ZL,low(2*mtable)
       cpi    temp1,$20                 ;is it a space character?
       brne   am1                       ;no
       rcall  WordWait                  ;yes, so wait appropriate time
       rjmp   am3
am1:   cpi    temp1,$2A                 ;ignore control chars
       brmi   am3
       cpi    temp1,$7A                 ;ignore graphic chars
       brpl   am3
       cpi    temp1,$60                 ;is it an lower-case char?
       brmi   am2                       ;no
       andi   temp1,$DF                 ;yes, convert to upper-case
am2:   subi   temp1,$2A                 ;start table at $2A='*'
       add    ZL,temp1                  ;add char offset to table pointer
       clr    temp1                     ;keep only the carry bit
       adc    ZH,temp1                  ;add carry, if any, to ZH
       lpm    temp1,Z                   ;get character from table
       rjmp   am4                       ;done
am3:   ldi    temp1,$80                 ;output stop-bit for invalid chars
am4:   pop    ZL                        ;restore Z pointer
       pop    ZH
       ret
```

```
CQTEST2:
       ldi    temp1,'c'                ;send a 'c'
       rcall  AsciiToMorse
       rcall  MorseOut
       ldi    temp1,12                 ;send invalid char (FORM FEED)
       rcall  AsciiToMorse
       rcall  MorseOut
       ldi    temp1,'q'                ;send a 'q'
       rcall  AsciiToMorse
       rcall  MorseOut
       ret

RESETLINE2:
;      used by MorseMsg to prep LCD line 2
       rcall  ClearLine2               ;erase line2
       ldi    temp1,$C0                ;set cursor to start of line
       rcall  LCDCMD
       clr    temp3                    ;clear char counter
       ret

MORSEMSG:
;      call with Z pointing to message
;      will output Morse and show it on LCD line 2
       rcall  ResetLine2               ;prep line2 for display
mm1:   ld     temp1,Z+                 ;get next ASCII character
       tst    temp1                    ;look for 0=stop byte
       breq   mm2                      ;done
       cpi    temp1,ContChar           ;is it a continuation character?
       breq   mm2                      ;yes, so quit this msg
       push   temp1                    ;save char
       rcall  LCDCHR                   ;put char on LCD
       pop    temp1                    ;retrieve char
       rcall  AsciiToMorse             ;convert char to morse
       rcall  MorseOut                 ;and send it
       inc    temp3                    ;incr character counter
       cpi    temp3,16                 ;is line2 full=16 chars?
       breq   MorseMsg                 ;yes, clear it & continue
       rjmp   mm1                      ;no, keep going
mm2:   ret

SENDMORSEMSG:
;      sends the current message in morse code
       ldi    temp1,10                 ;Show 'Sending Message'
       rcall  DisplayLine1             ;on LCD line 1
       lds    temp1,msgnum             ;get current msg#
mr0:   rcall  LoadEEmsg                ;load message from EEPROM
       rcall  MarkEnd                  ;remove trailing spaces
       ldi    ZH,high(msgbuf)          ;point to msg
       ldi    ZL,low(msgbuf)
       rcall  MorseMsg                 ;send out the morse code
       cpi    temp1,ContChar           ;was last char a continuation?
       brne   mr1                      ;no, so done
       lds    temp1,msgnum             ;grab msg#
       inc    temp1                    ;go to next msg#
       sts    msgnum,temp1             ;and remember it
       rjmp   mr0                      ;send next message
mr1:   ldi    temp1,0                  ;restore LCD
       rcall  ChangeMode               ;to tuning mode
       ret

QUEUEMSG1:
;      puts the next available cw message into message buffer
;      displays the pending message on line2
```

```
       clr    hold                        ;dont trigger hold while keying in #
       lds    temp1,msgnum                ;get message number
       inc    temp1                       ;go to next number
       cpi    temp1,NumMessages           ;have we passed the last msg?
       brlo   ss1                         ;no, continue
       clr    temp1                       ;yes, so allow escape
       rcall  ShowTuning                  ;and erase queued msg display
ss1:   sts    msgnum,temp1                ;get queued value
       tst    temp1                       ;retrieve message number
       breq   ss2                         ;zero = nothing was queued
       rcall  ShowMsgIndex                ;show queued message on LCD
ss2:   rcall  DitWait                     ;keyer debouncer
       rcall  DitWait
       ret

QUEUEMSG2:
;      skip the next available cw message, and go to following one
;      display the queued message on line2
       lds    temp1,msgnum
       inc    temp1                       ;advance message# by 2
       sts    msgnum,temp1                ;save it
       rcall  QueueMsg1
       rcall  DahWait                     ;longer wait on right paddle
       ret


;*********************************************************
;*  W8BH - END OF INSERTED CODE
;*********************************************************



DISPLAY_LINE:
       lpm    temp1,Z+
       tst    temp1
       breq   DISPLAY_LINE_FINIS
       rcall  LCDCHR
       rjmp   DISPLAY_LINE
DISPLAY_LINE_FINIS:
       ret

;***********************************
;* INIT_LCD
;* uses temp1
;***********************************
INIT_LCD:
       ldi    delay,20                    ;wait 20 milliseconds for system to stabilize
       rcall  WAIT
       ldi    temp1,$30
       rcall  LCDCMD
       ldi    delay,4                     ;wait 4 milliseconds
       rcall  WAIT
       ;ldi   temp1,$30
       ;rcall LCDCMD
       ;ldi   temp1,$30
       ;rcall LCDCMD
       ldi    temp1,$38                   ;set for 8 bits, 2 lines, default font
       rcall  LCDCMD
       ldi    temp1,$01                   ;Clear screen, cursor home
       rcall  LCDCMD
       ldi    temp1,$0E                   ;Display on, cursor on, blink off
       rcall LCDCMD
       ret
```

```
;***********************************
;* WAIT
;* temp1 = WAIT time in milliseconds
;***********************************
WAIT:
       tst          delay
       brne   WAIT
       ret

;***********************************
;* BLINK_LED
;* temp1 = number of times LED BLINKS
;***********************************
BLINK_LED:
       cbi    PORTC,LED                ;turn LED on
       ldi    delay,200                ;200 mSec
       rcall  WAIT
       sbi    PORTC,LED                ;turn LED off
       ldi    delay,200                ;200 mSec
       rcall  WAIT
       dec    temp1
       brne   BLINK_LED
       ret

;***********************************
;* LCDCMD - Send a command to the LCD
;* temp1 = cmd
;***************************************
;* LCDCHR - Display a character on the LCD
;* temp1 = data
;***********************************
LCDCMD:
       rcall  SENDBYTE
       cbi    PORTC,LCD_DRS            ;set RS low
       rjmp   LCDEXEC
LCDCHR:
       rcall  SENDBYTE
       sbi    PORTC,LCD_DRS            ;set RS High
LCDEXEC:
       sbi    PORTC,LCD_E              ;E clocks on rising edge
       ldi    delay,2                  ;wait 2 milliseconds
       cbi    PORTC,LCD_E              ;set E low
       rcall  WAIT
       ret

;*********************************
;* SENDBYTE clock byte into 74hc164
;* temp1 = data
;* temp2 = bitcounter
;*********************************
SENDBYTE:
       push   temp2
       ldi    temp2,8                  ;bit counter
ShiftBits:
       cbi    PORTC,LCD_DRS            ;set LCD Data bit to 0
       lsl    temp1                    ;shift bits in AReg into carry
       brcc   ClockBit                 ;if zero, clock it out
       sbi    PORTC,LCD_DRS            ;if 1, set LCD Data bit to 1
ClockBit:
       sbi    PORTC,LCD_CP             ;bit clocks on rising edge
       cbi    PORTC,LCD_CP             ;bring clock back to 0
       dec    temp2                    ;shift out 8 bits
```

```
        brne    ShiftBits
        pop     temp2
        ret

IncFreq0:
        mov     temp2,StepRate
        ldi     ZH,high(rcve0)
        ldi     ZL,low(rcve0)
        ldi     temp1,9
IncFreq1:
        dec     temp1
        dec     ZL
        dec     temp2
        brpl    IncFreq1
IncFreq2:
        ld      temp3,Z
        inc     temp3
        cpi     temp3,10
        clc
        brne    IncFreq3
        sec
        clr     temp3
IncFreq3:
        st      Z,temp3
        brcc    IncFreq7
        dec     ZL
        dec     temp1
        rjmp    IncFreq2
IncFreq7:
        ldi     ZH,high(LCDrcve0)
        ldi     ZL,low(LCDrcve0)
        ld      temp1,Z
        cpi     temp1,3                 ;check for 30MHz
        brlo    IncFreq8
        ldi     temp1,55                ;55 is an error status flag
IncFreq8:
        ret

IncFreq9:
        ldi     ZH,high(freq0)
        ldi     ZL,low(freq0)
        mov     temp2,StepRate
        lsl     temp2
        lsl     temp2
        add     ZL,temp2
        ldi     YH,high(rcve0)
        ldi     YL,low(rcve0)
        ld      temp1,Z+
        ld      temp2,Y
        add     temp1,temp2
        st      Y+,temp1
        ld      temp1,Z+
        ld      temp2,Y
        adc     temp1,temp2
        st      Y+,temp1
        ld      temp1,Z+
        ld      temp2,Y
        adc     temp1,temp2
        st      Y+,temp1
        ld      temp1,Z+
        ld      temp2,Y
        adc     temp1,temp2
        st      Y+,temp1
```

```
        ret

DecFreq0:
        mov     temp2,StepRate
        ldi     ZH,high(rcve0)
        ldi     ZL,low(rcve0)
        ldi     temp1,9
DecFreq1:
        dec     temp1
        dec     ZL
        dec     temp2
        brpl    DecFreq1
DecFreq2:
        ld      temp3,Z
        dec     temp3
        cpi     temp3,255
        clc
        brne    DecFreq3
        sec
        ldi     temp3,9
DecFreq3:
        st      Z,temp3
        brcc    DecFreq7
        dec     ZL
        dec     temp1
        rjmp    DecFreq2
DecFreq7:
        ldi     ZH,high(LCDrcve0)
        ldi     ZL,low(LCDrcve0)
        ld      temp1,Z
        cpi     temp1,3                 ;check for 30MHz
        brlo    DecFreq8
        ldi     temp1,55                ;55 is an error status flag
DecFreq8:
        ret

DecFreq9:
        ldi     ZH,high(freq0)
        ldi     ZL,low(freq0)
        mov     temp2,StepRate
        lsl     temp2
        lsl     temp2
        add     ZL,temp2
        ldi     YH,high(rcve0)
        ldi     YL,low(rcve0)
        ld      temp2,Z+
        ld      temp1,Y
        sub     temp1,temp2
        st      Y+,temp1
        ld      temp2,Z+
        ld      temp1,Y
        sbc     temp1,temp2
        st      Y+,temp1
        ld      temp2,Z+
        ld      temp1,Y
        sbc     temp1,temp2
        st      Y+,temp1
        ld      temp2,Z+
        ld      temp1,Y
        sbc     temp1,temp2
        st      Y+,temp1
        ret
```

```
;**********************************
;* ShowFreq
;* uses temp1 & temp2
;**********************************
ShowFreq: ;display freq on LCD
       ldi    temp1,$C1
       rcall  LCDCMD
       ldi    ZH,high(LCDrcve0)
       ldi    ZL,low(LCDrcve0)
       clr    temp2
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq1
       ldi    temp1,' '
ShowFreq1:
       rcall  SHOWDEC
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq2
       ldi    temp1,' '
ShowFreq2:
       rcall  SHOWDEC
       ldi    temp1,','
       tst    temp2
       brne   ShowFreq3
       ldi    temp1,' '
ShowFreq3:
       rcall  SHOWDEC
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq4
       ldi    temp1,' '
ShowFreq4:
       rcall SHOWDEC
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq5
       ldi    temp1,' '
ShowFreq5:
       rcall  SHOWDEC
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq6
       ldi    temp1,' '
ShowFreq6:
       rcall  SHOWDEC
       ldi    temp1,','
       tst    temp2
       brne   ShowFreq7
       ldi    temp1,' '
ShowFreq7:
       rcall  SHOWDEC
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq8
       ldi    temp1,' '
ShowFreq8:
       rcall  SHOWDEC
       ld     temp1,Z+
       add    temp2,temp1
       brne   ShowFreq9
       ldi    temp1,' '
```

```
ShowFreq9:
       rcall  SHOWDEC
       ld     temp1,Z+
       rcall  SHOWDEC
       ldi    temp1,' '
       rcall  LCDCHR
       ldi    temp1,'H'
       rcall  LCDCHR
       ldi    temp1,'z'
       rcall  LCDCHR
       rcall  ShowCursor
       ret

ShowCursor:                            ;position cursor to active position
       ldi    temp1,$CA
       sub    temp1,StepRate
       cpi    temp1,$C8
       brsh   ShowCursor1
       dec    temp1
ShowCursor1:
       cpi    temp1,$C4
       brsh   ShowCursor2
       dec    temp1
ShowCursor2:
       rcall  LCDCMD
       ret

EXT_INT0:
       push   temp1                    ;save temp1 register
       in     temp1,SREG               ;save the status register
       push   temp1
       lds    temp1,EICRA
       cpi    temp1,0b00001010         ; test falling edge
       breq   int05
       rjmp   int05
       ldi    temp1,0b00001010         ; set int0 falling edge and int1 falling edge

       sts    EICRA,temp1
       sbis   PIND,PHASE               ; test PHASE
       rjmp   int01
       dec    encoder
       rjmp   int09
int01:
       inc    encoder
       rjmp   int09
int05:
       ldi    temp1,0b00001011         ; set int0 falling edge and int1 falling edge
       sts    EICRA,temp1
       sbis   PIND,PHASE               ; test PHASE
       rjmp   int06
       inc    encoder
       rjmp   int09
int06:
       dec    encoder
int09:
       pop    temp1
       out    SREG,temp1               ;restore the status register
       pop    temp1                    ;restore temp1 register
       reti


EXT_INT1:
       push   temp1                    ;save temp1 register
```

```
       in     temp1,SREG
       push   temp1                    ;save status register
       inc    press                    ;count the button-down
       out    SREG,temp1               ;restore status register
       pop    temp1                    ;restore temp1 register
       reti


;***********************************
;* Timer 0 Overflow
;* gets here every 1 millisecond
;* decrements the DELAY counter
;***********************************
OVF0:
       push   temp1
       in     temp1,SREG
       push   temp1
       ldi    temp1,256-20             ;1 ms using a 20.48 MHz Xtal
       out    TCNT0,temp1              ;set for next overflow
       tst    delay
       breq   OVF_EXIT
       dec    delay
OVF_EXIT:
       pop    temp1
       out    SREG,temp1
       pop    temp1
       reti

;***********************************
;* SHOWHEX
;* display HEX byte from temp1
;* at active LCD position
;***********************************
SHOWHEX:
;preserves temp1 and temp2 values
       push   temp1
       push   temp2
       push   temp1
       swap   temp1
       andi   temp1,$0F
       ori    temp1,$30
       cpi    temp1,$3A
       brlo   SHOWHEX2
       ldi    temp2,$07
       add    temp1,temp2
SHOWHEX2:
       rcall  LCDCHR
       pop    temp1
       andi   temp1,$0F
       ori    temp1,$30
       cpi    temp1,$3A
       brlo   SHOWHEX3
       ldi    temp2,$07
       add    temp1,temp2
SHOWHEX3:
       rcall  LCDCHR
       pop    temp2
       pop    temp1
       ret

;***********************************
;* SHOWDEC
;* display DEC byte from temp1
```

```
;* at active LCD position
;**********************************
SHOWDEC:
       cpi    temp1,10
       brsh   SHOWDEC1
       ori    temp1,$30
SHOWDEC1:
       rcall  LCDCHR
       ret

;**********************************
;* SHOWHEX_CHAR
;* display one HEX byte on line 2
;* byte count in temp2
;**********************************
SHOWHEX_CHAR:
       ldi    temp1,$C0                ;display on line 2
       rcall  LCDCMD
       rcall  SHOWHEX                  ;display both hex nibbles
       ret

;**********************************
;* SHOWHEX_LINE
;* display HEX bytes from non-program memory on line 2
;* byte count in temp2
;**********************************
SHOWHEX_LINE:
       ldi    temp1,$C0                ;display on line 2
       rcall  LCDCMD
       ldi    temp2,8                  ;display 8 hex bytes
SHOWHEX_LINE_2:
       ld     temp1,Z+                 ;get display byte
       rcall  SHOWHEX                  ;display both hex nibbles
       dec    temp2
       brne   SHOWHEX_LINE_2
       ret

FREQ_OUT:
       ldi    temp1,$20                ;28 bits FREQ0 to AD9834
       ldi    temp2,$00
       rcall  SHIFT_16                 ;output to DDS chip
       ldi    yl,low(rcve0)
       ld     temp4,y+                 ;LSB
       ld     temp3,y+                 ;
       ld     temp2,y+                 ;
       ld     temp1,y+                 ;MSB

       lsr    temp1                    ;MSB-high
       ror    temp2                    ;MSB-low
       ror    temp3                    ;LSB-high
       ror    temp4                    ;LSB-low

       lsr    temp1                    ;MSB-high
       ror    temp2                    ;MSB-low
       ror    temp3                    ;LSB-high
       ror    temp4                    ;LSB-low

       lsr    temp1                    ;MSB-high
       ror    temp2                    ;MSB-low
       ror    temp3                    ;LSB-high
       ror    temp4                    ;LSB-low

       lsr    temp3
```

```
       ror    temp4
       lsr    temp3
       ror    temp4

       push   temp1
       push   temp2
       mov    temp1,temp3
       mov    temp2,temp4

       ori    temp1,0b01000000
       rcall  SHIFT_16                  ;send 14 bits
       pop    temp2
       pop    temp1

       push   temp1
       push   temp2
       ori    temp1,0b01000000
       rcall  SHIFT_16                  ;send 14 bits

       mov    temp1,temp3
       mov    temp2,temp4
       ori    temp1,0b10000000
       rcall  SHIFT_16                  ;send 14 bits

       pop    temp2
       pop    temp1
       ori    temp1,0b10000000
       rcall  SHIFT_16                  ;send 14 bits

       ret

;*********************************
;* SHIFT_16
;* display HEX bytes
;* at active LCD position
;* byte count in temp2
;*********************************

SHIFT_16:
;16 bit serial out msb first in temp1, then lsb in temp2
       push   temp3
       cbi    PORTD,DDSenable           ;FSYNC goes LOW
       ldi    temp3,16                  ;16 bits bit counter
shift8:
       sbi    PORTD,DDSdata             ;set port bit
       rol    temp1                     ;shift dds address byte
       brcs   clockit                   ;check for 1/0
       cbi    PORTD,DDSdata             ;clear port bit
clockit:
       nop
       cbi    PORTD,DDSclock            ;clock dds
       nop
       sbi    PORTD,DDSclock
       dec    temp3                     ;decrement bit counter
       breq   sox                       ;exit if done
       cpi    temp3,8                   ;check byte counter
       brne   shift8                    ;output more bits
       mov    temp1,temp2               ;get lsb
       rjmp   shift8                    ;write data bits
sox:
       sbi    PORTD,DDSenable           ;FSYNC goes HIGH
       pop    temp3
       ret
```

```
DEFAULT_FREQ:                           ;FlashToRam
       ldi    ZH,high(FreqHex*2)        ;load default freq
       ldi    ZL,low(FreqHex*2)
       ldi    YH,high(rcve0)
       ldi    YL,low(rcve0)
       ldi    temp2,36
DF1:
       lpm    temp1,Z+
       st     Y+,temp1                  ;store in SRAM and increment Y-pointer
       dec    temp2                     ;decrememnt counter
       brne   DF1                       ;if not end of table, loop more

       ldi    ZH,high(FreqLCD*2)        ;load default freq
       ldi    ZL,low(FreqLCD*2)
       ldi    YH,high(LCDrcve0)
       ldi    YL,low(LCDrcve0)
       ldi    temp2,8
DF2:
       lpm    temp1,Z+
       st     Y+,temp1                  ;store in SRAM and increment Y-pointer
       dec    temp2                     ;decrememnt counter
       brne   DF2                       ;if not end of table, loop more
       ret

;calibrated freq reference table
FreqHex:
.db 0x33,0x33,0x33,0x33 ;10,000,000 MHz
.db 0x56,0x00,0x00,0x00 ;1
.db 0x5B,0x03,0x00,0x00 ;10
.db 0x8E,0x21,0x00,0x00 ;100
.db 0x8B,0x4F,0x01,0x00 ;1,000
.db 0x71,0x1B,0x0D,0x00 ;10,000
.db 0x6F,0x12,0x83,0x00 ;100,000
.db 0x52,0xB8,0x1E,0x05 ;1,000,000
.db 0x33,0x33,0x33,0x33 ;10,000,000

FreqLCD: .db 1,0,0,0,0,0,0,0 ;LCD for 10,000,000 Hz

;    12345678901234567890123456789012345678901234567890
msg1:
.db "Kits and Parts",0,0
```

```
;*******************************************
;*    USER-DEFINED FREQUENCY PRESETS
;*******************************************

;Up to 27 user-defined presets can be loaded into EEPROM
;The actual number must equal your value of NumPresets
;Enter the values that you want to program here

presets:                        ;##   DESCRIPTION       FREQUENCY
.db 0,3,5,6,0,0,0,0             ;01.  80M qrp calling =  3.560 MHz
.db 0,7,0,3,0,0,0,0             ;02.  40M qrp calling =  7.030 MHz
.db 1,0,0,0,0,0,0,0             ;03.  WWV             = 10.000 MHz
.db 1,0,1,0,6,0,0,0             ;04.  30M qrp calling = 10.106 MHz
.db 1,4,0,6,0,0,0,0             ;05.  20M qrp calling = 14.060 MHz
.db 1,8,0,9,6,0,0,0             ;06.  17M qrp calling = 18.096 MHz
.db 2,1,0,6,0,0,0,0             ;07.  15M qrp calling = 21.060 MHz
.db 2,4,9,0,6,0,0,0             ;08.  12M qrp calling = 24.906 MHz
.db 2,8,0,6,0,0,0,0             ;09.  10M qrp calling = 28.060 MHz
.db 0,1,0,0,0,0,0,0             ;10.                  = 01.000 MHz

.db 0,2,0,0,0,0,0,0             ;11.                  = 02.000 MHz
.db 0,3,0,0,0,0,0,0             ;12.                  = 03.000 MHz
.db 0,4,0,0,0,0,0,0             ;13.                  = 04.000 MHz
.db 0,5,0,0,0,0,0,0             ;14.                  = 05.000 MHz
.db 0,6,0,0,0,0,0,0             ;15.                  = 06.000 MHz
.db 0,7,0,0,0,0,0,0             ;16.                  = 07.000 MHz
.db 0,8,0,0,0,0,0,0             ;17.                  = 08.000 MHz
.db 0,9,0,0,0,0,0,0             ;18.                  = 09.000 MHz
.db 1,0,0,0,0,0,0,0             ;19.                  = 10.000 MHz
.db 1,2,3,4,5,6,7,8             ;20.  Test freq       = 12.345 MHz



;*******************************************
;*    MODE & STATUS DISPLAY MESSAGES
;*******************************************


messages:
.db "VFO Tuning Mode "          ;1
.db "Scroll Presets  "          ;2
.db "Save New Preset "          ;3
.db "Keyer Memories  "          ;4
.db "Set Code Speed  "          ;5
.db "Set Tuning Rate "          ;6
.db "Set IF offset   "          ;7
.db "FACTORY RESET***"          ;8
.db " SAVED          "          ;9
.db "SENDING MESSAGE "          ;10
```

```
mtable:
;This table converts ASCII characters into their morse equivalent
;The ASCII character is listed as a comment above each code
;Read the code from left to right, with 1=dit and 0=dah
;An extra, silent '1' is added at the end as a stop-bit

;       * (SK)      +           ,           - (BT)
.db 0b11101010, 0b00000000, 0b00110010, 0b01110100
;       .           /           0           1
.db 0b01010110, 0b01101100, 0b00000100, 0b10000100
;       2           3           4           5
.db 0b11000100, 0b11100100, 0b11110100, 0b11111100
;       6           7           8           9
.db 0b01111000, 0b00111100, 0b00011100, 0b00001100
;       :           ;           <           =
.db 0b00000000, 0b00000000, 0b00000000, 0b00000000
;       >           ?           @           A
.db 0b00000000, 0b11001110, 0b10010110, 0b10100000
;       B           C           D           E
.db 0b01111000, 0b01011000, 0b01110000, 0b11000000
;       F           G           H           I
.db 0b11011000, 0b00110000, 0b11111000, 0b11100000
;       J           K           L           M
.db 0b10001000, 0b01010000, 0b10111000, 0b00100000
;       N           O           P           Q
.db 0b01100000, 0b00010000, 0b10011000, 0b00101000
;       R           S           T           U
.db 0b10110000, 0b11110000, 0b01000000, 0b11010000
;       V           W           X           Y
.db 0b11101000, 0b10010000, 0b01101000, 0b01001000
;       Z           [           \           ] (AR)
.db 0b00110000, 0b00000000, 0b00000000, 0b10101100

ctable:
;This table converts code speed (in WPM) to the required length
;of each element (in milliseconds).  The table goes from 5 to
;30 WPM.  For example, the first table entry (for 5 WPM) is 240
;milliseconds, 6 WPM = 200 milliseconds, and so on.

.db 240, 200, 171, 150, 133, 120, 109, 100
.db  92,  86,  80,  75,  71,  67,  63,  60
.db  57,  55,  52,  50,  48,  46,  44,  43
.db  41,  40

cwmsg:
;Up to 7 user-defined keyer messages can be loaded into EEPROM
;The actual number of messages must equal your NumMessages value
;Enter your default 32-byte messages here:

;       12345678901234567890123456789012
.db "CQ CQ CQ de W8BH W8BH W8BH K    "         ;message 1
.db "TNX FER CALL - UR RST IS        "         ;message 2
.db "QTH DAYTON, OH ? DAYTON, OH - + "         ;message 3
.db "NAME BRUCE ? BRUCE - SO HW CPY? "         ;message 4
.db "RIG ELECRAFT K3 es ANT VERTICAL-"         ;message 5
.db "RIG HOMEBREW es ANT VERTICAL -  "         ;message 6
.db "TNX FER QSO - 73 73 * de W8BH K "         ;message 7
```