ECE 375 LAB 8

Morse Code Transmitter

Lab Time: Wednesday 12 - 2

Marcus Wheeler

INTRODUCTION

This is the last lab of the term and a culmination of everything else so far. The only kind of new thing is interacting with Timer1 and being forced into polling for input rather than using interrupts again. There was a way to implement interrupts with the Timer, but I opted for watching the TIFR Flag rather than set up only a single interrupt.

PROGRAM OVERVIEW

The Morse Code transmitter takes input from buttons to make a word and then outputs that translation in morse code. The small space is 1 second and the large space is 3 seconds.

INITIALIZATION ROUTINE

Set up stack, initialized PORTB for output and PORTD for input, TCCR1 is set up to be stopped when we start, in PWM mode, and Set on match, clear on bottom.

MAIN ROUTINE

My program is divided into three pieces. I was originally going to rely on a state variable, so I named them State_Zero, State_One, and State_Two. In state zero, the program writes the initial welcome message and only pools for pd0. In state one, all four buttons are polled and we loop until the count of the letters is equal to 16 and then auto confirm. One we've either pressed pd4 to confirm our short message or reached 16 letters, we enter State_Two. In State_Two, the LEDs and timer are the only things interacting – I'm not polling for any input anymore. It basically just iterates through the string given and performs the conversions as necessary. I ended up doing a linear search rather than implementing a binary search for my letter finding.

SUBROUTINES

1. Get_Constant

This moves my constants from program memory to data memory.

2. Wait Routine

The usual Wait routine we've seen since lab 1. It's only used for debouncing and not for managing the morse code timing.

3. Write_State_Zero

This takes our initial welcome message and writes it to the LCDDisplay.

4. Write_State_One

Similarly, this takes our prompt for input and writes it to the LCDDisplay.

5. Clear

Clear deletes the first and second lines – I only call it between states.

6. Write

   Write is a helper function that the rest of the Write functions use – it takes an end address and writes whatever Y is pointing at to the correct address until it reaches the end of that line.

7. DOT

   DOT sets the timer up for a 1 second delay with the LEDs on and a 1 second delay with the LEDs off.

8. END_DOT

   END_DOT sets the timer up for a 1 second delay with the LEDs on and a 3 second delay with the LEDs off.

9. DASH

   DASH sets the timer up for a 3 second delay with the LEDs on and a 1 second delay with the LEDs off.

10. END_DASH

    END_DASH sets the timer up for a 3 second delay with the LEDs on and a 3 second delay with the LEDs off.

11. START_CLOCK

    I use this to clear out my TCNT1 (high then low), fill my values in OCCR1, and once everything is set up, provide a clock source to TCCR1B.

12. STOP_CLOCK

    I also use this to clear out TCNT1, but I do so after I stop the clock in TCCR1B. The idea is to make sure I'm getting the same delay every time that I call them.

13. CALL_(A, B, C, …, Z)

    These are 26 different functions (A-Z) that I use to define what patterns make up the letters. It's made up of DOT, DASH, and an END_DOT or END_DASH. It's potentially more accurate to say that these are a part of main, but if I included them in main it broke literally every single branching statement, so I did it like a disjointed switch statement.

Difficulties

Deciding how I was going to create a cohesive program was the hardest part, but once I realized it was just designing three different stages, I was able to start working and then change things as needed in the later parts once I thought of something.

Conclusion

This has easily been my favorite Lab overall of any class I've ever taken – it's a cool subject and I'm looking forward to doing projects of my own this winter break.

SOURCE CODE

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting. Make sure to reformate the code so it looks nice and is readable.

```
;*******;***********************************************************
;*
;*     Marcus_Wheeler_Lab8_Source
;*
;*
;*
;*
;*
;***********************************************************
;*
;*      Author: Marcus Wheeler
;*        Date: 11/25/2021
;*
;***********************************************************

.include "m128def.inc"                ; Include definition file

;***********************************************************
;*     Internal Register Definitions and Constants
;***********************************************************
.def   mpr = r16                             ; Multipurpose register

.def    lastAddress = r1
.def    currentLetter = r2

;Necessary for Wait routine
.def    waitcnt = r25
.def    olcnt = r24
.def    ilcnt = r23

.equ    ONE_SECOND_DELAY = 15625
.equ    THREE_SECOND_DELAY = 46875

.equ    ALetter = 0x41
.equ    ZLetter = 0x5A

.equ    ENDFIRSTLINE = 0x10
.equ    ENDSECONDLINE = 0x20

.equ    ENDWELCOMEDATA = 0x30
.equ    ENDPRESSDATA = 0x40
.equ    ENDENTERDATA = 0x50

;LCD Address locations
.equ    LCDFirstLine = 0x0100
.equ    LCDSecondLine = 0x0110

.equ    FirstLineData = 0x0200
.equ    SecondLineData = 0x0210

.equ    WelcomeAddress = 0x0220
.equ    PressAddress = 0x0230
.equ    EnterAddress = 0x0240
;***********************************************************
;*     Start of Code Segment
;***********************************************************
.cseg                                                ; Beginning of code segment

;***********************************************************
;*     Interrupt Vectors
;***********************************************************
.org   $0000                                 ; Beginning of IVs
               rjmp   INIT                   ; Reset interrupt
               ; Set up interrupt vectors for any interrupts being used

.org    $0046                               ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:                                               ; The initialization routine
               ; Initialize Stack Pointer
               ldi mpr, LOW(RAMEND)
               out SPL, mpr

               ldi mpr, HIGH(RAMEND)
               out SPH, mpr
               ; Initialize Port B for output
               ldi mpr, 0x00
               out PORTB, mpr

               ldi mpr, 0xFF
               out DDRB, mpr

               ; Initialize Port D for input
               ldi mpr, 0xFF
               out PORTD, mpr

               ldi mpr, 0x00
               out DDRD, mpr

               ;Have to set up the timer
               ;TCCR1A 00000000\
               ldi mpr, 0b00000000
               out TCCR1A, mpr

               ;TCCR1B 00011000   I'm going to keep it stopped right now, and I'll start and stop
it as needed in my functions.
               ldi mpr, 0b00011000
               out TCCR1B, mpr
               ;This should take care of it. I want it with 1024 prescaling, FAST PWM mode, and
set on match clear at bottom

               rcall Get_Constant

               rcall LCDInit


;***********************************************************
;*      Main Program
;***********************************************************
;***************************************************************************
;* I'm going to use count as my address accumulator, but the LCDDriver names it
;***************************************************************************
MAIN:                ;If I'm in state zero, I only need to poll for my PD0
        rcall Write_State_Zero
STATE_ZERO_LOOP:
        in mpr, PIND           ;PIND is active low, so I'm looking for the value that's 0 out of
the ones
               andi mpr, 0b00000001    ;If any of my values are 0, there will be a zero in that
location - otherwise, it'll still be a one and unpressed
               cpi mpr, 0b00000001    ;The only input I care about in this iteration is PD0 - If
it's still one, skip incrementing my currentState
               breq STATE_ZERO_LOOP
        jmp STATE_ONE_LOOP
               ;There's no way to get from 0x00 to 0x02, so I don't need to include the option
STATE_ONE_LOOP:                 ;If I'm in state one, I need to poll for PD0, PD4, PD6, and PD7

                                ;PD0 will confirm the current letter
                                                            ;PD4  will  send  us  to  state  two  and
we'll be ready to transmit the message
                                                            ;PD6   will   increment   the   current
letter in forward order from A - Z and then back to A
```

```
                                                            ;PD7 will decrement the current
letter in backward order from Z - A and then back to Z
               rcall Clear
               rcall Write_State_One

               ldi waitcnt, 15
               call Wait_Func
               clr count               ;Named in LCDDriver - I'm going to use it as the number of
letters I've confirmed.

PD0_CHECK:
        in mpr, PIND            ;This is going to load in my value for the rest of the checks
               andi mpr, 0b00000001
               cpi mpr, 0b00000001       ;PD0 is pressed. I can increment the address that I'm
working with unless I'm at 16 letters - Then I can skip to the PD4 result
               ;First thing I need to do is load an A into the new address
               ;After that I can increment it or decrement it as needed, so I should increment
the current address in this one.

               breq PD6_CHECK          ;If they are equal, then PD0 is not currently pressed.
                                       ;If PD0 is currently pressed and this is the sixteenth
letter, then skip to PD4
                                                            ;0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
15
                                                            ;1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16
               inc count                   ;We've confirmed our last letter, so I'm going to go
forward one and put down an A.
        cpi count, 16               ;If I this is my 16th confirmed letter , then I must have
rolled over.
               breq PD4_RELATIVE_WORKAROUND       ;If equal, skip to the end of PD4

        ldi YH, HIGH(LCDSecondLine)
               ldi YL, LOW(LCDSecondLine)

               add YL, count           ;Orients me to where I'm currently at.

               ldi mpr, ALetter

               st Y, mpr
               call LCDWrite       ;This should write the specific byte and letter that I have to
wherever I want in the second line

               ldi waitcnt, 15
               call Wait_Func
               jmp PD0_CHECK           ;Otherwise, we're done with this call and we can start
checking our buttons again.
PD4_RELATIVE_WORKAROUND:
               jmp PD4_CONFIRMED   ;I think this is a workaround to the relative branch being out
of reach. It will never be reached because of the prior jump call
                                   ;And the only way to get to this spot is from my breq


PD6_CHECK:
               in mpr, PIND
               andi mpr, 0b01000000
               cpi mpr, 0b01000000      ;PD6 is pressed. I can increment the current letter and
then write it to the screen

               breq PD7_CHECK          ;If they are equal then we don't have a button press and I
can skip to my next button check
        ;Otherwise, what I need to do now is increment  my letter unless it's already equal to A
in which case I can set it equal to Z

               ldi YH, HIGH(LCDSecondLine)
               ldi YL, LOW(LCDSecondLine)

               add YL, count           ;Orients me to where I'm currently at.
               ld mpr, Y

               cpi mpr, ZLetter
               breq LOOP_LETTER_INCREMENT  ;If my currentLetter is equal to Z, decrement it

               inc mpr
               jmp UPDATE_LETTER_INCREMENT

LOOP_LETTER_INCREMENT:
        ldi mpr, ALetter
               jmp UPDATE_LETTER_INCREMENT


UPDATE_LETTER_INCREMENT:
        st Y, mpr
               call LCDWrite
               ldi waitcnt, 15
               call Wait_Func
               JMP PD0_CHECK          ;Now I'm done with this call



PD7_CHECK:
        in mpr, PIND
               andi mpr, 0b10000000
               cpi mpr, 0b10000000      ;PD7 is pressed. I can decrement the current letter and
then write it to the screen

               breq PD4_CHECK          ;If they are equal then we don't have a button press and I
can skip to my next button check
        ;Otherwise, what I need to do now is decrement my letter unless it's already equal to Z
in which case I set it equal to A

               ;All we have to do is get the letter from my LCDSecondLine + count, check if it
needs to roll over, and if not inc/dec as needed then write

               ldi YH, HIGH(LCDSecondLine)
               ldi YL, LOW(LCDSecondLine)

               add YL, count           ;Orients me to where I'm currently at.
               ld mpr, Y

               cpi mpr, ALetter
               breq LOOP_LETTER_DECREMENT  ;If my currentLetter is equal to Z, decrement it

               dec mpr
               jmp UPDATE_LETTER_DECREMENT

LOOP_LETTER_DECREMENT:
        ldi mpr, ZLetter
               jmp UPDATE_LETTER_DECREMENT


UPDATE_LETTER_DECREMENT:
               ;mpr holds my updated letter and now I can replace the Y I got it from
               st Y, mpr
               call LCDWrite
               ldi waitcnt, 15
               call Wait_Func
               JMP PD0_CHECK          ;Now I'm done with this call


PD4_CHECK:
        in mpr, PIND
               andi mpr, 0b00010000
               cpi mpr, 0b00010000     ;PD4 is pressed. I can increment my current state and then
skip to the end.
               breq PD0_CHECK_WORKAROUND          ;If they're equal then nothing is pressed and I
can start checking buttons at PD0 again
               ;I'm going to delete any unconfirmed bytes and then rewrite it
        inc count               ;Then make it so I confirm whatever letter is written in spot 0
                                ;And skip the deletion because nothing will be written in spot 1
               rjmp PD4_CONFIRMED
PD0_CHECK_WORKAROUND:
       jmp PD0_CHECK

PD4_CONFIRMED:                   ;Otherwise they are pressed and everything is confirmed - we can
skip to transmission now
               ldi waitcnt, 15
               call Wait_Func
               jmp STATE_TWO_LOOP

STATE_TWO_LOOP:                   ;If I'm in state two, I'm transmitting the message, so I don't
need to poll for anything.
               clr currentLetter
               ;Turn on the transmission LED here
               ldi mpr, 0b00010000
               out PORTB, mpr
SEARCH_LOOP:
        cpi count, 0x00
               breq TRANSMISSION_WORKAROUND

        ldi YL, LOW(LCDSecondLine)
               ldi YH, HIGH(LCDSecondLine)
               ADD YL, currentLetter

        ld mpr, Y                  ;May need to be Y+ depending on how I want to do things
               cpi mpr, ALetter
               breq CASE_A
               cpi mpr, ALetter + 1
               breq CASE_B
               cpi mpr, ALetter + 2
               breq CASE_C
               cpi mpr, ALetter + 3
               breq CASE_D
               cpi mpr, ALetter + 4
               breq CASE_E
               cpi mpr, ALetter + 5
               breq CASE_F
               cpi mpr, ALetter + 6
               breq CASE_G
               cpi mpr, ALetter + 7
               breq CASE_H
               cpi mpr, ALetter + 8
               breq CASE_I
               cpi mpr, ALetter + 9
               breq CASE_J
               cpi mpr, ALetter + 10
               breq CASE_K
               cpi mpr, ALetter + 11
               breq CASE_L
               cpi mpr, ALetter + 12
               breq CASE_M
               cpi mpr, ALetter + 13
               breq CASE_N
               cpi mpr, ALetter + 14
               breq CASE_O
               cpi mpr, ALetter + 15
               breq CASE_P
               cpi mpr, ALetter + 16
               breq CASE_Q
               cpi mpr, ALetter + 17
               breq CASE_R
               cpi mpr, ALetter + 18
               breq CASE_S
               cpi mpr, ALetter + 19
               breq CASE_T
               cpi mpr, ALetter + 20
               breq CASE_U
               cpi mpr, ALetter + 21
               breq CASE_V
               cpi mpr, ALetter + 22
               breq CASE_W
               cpi mpr, ALetter + 23
               breq CASE_X
               cpi mpr, ALetter + 24
               breq CASE_Y
               cpi mpr, ALETTER + 25
               breq CASE_Z            ;It has to be Z at this point. I could do a binary search
here
TRANSMISSION_WORKAROUND:
        jmp TRANSMISSION_COMPLETE
CASE_A:   ;DOT DASH
  jmp CALL_A
CASE_B:   ;DASH DOT DOT DOT
  jmp CALL_B
CASE_C:   ;DASH DOT DASH DOT
  jmp CALL_C
CASE_D:   ;DASH DOT DOT
  jmp CALL_D
CASE_E:   ;DOT
  jmp CALL_E
CASE_F:   ;DOT DOT DASH DOT
  jmp CALL_F
CASE_G:   ;DASH DASH DOT
  jmp CALL_G
CASE_H:   ;DOT DOT DOT DOT
  jmp CALL_H
CASE_I:   ;DOT DOT
  jmp CALL_I
CASE_J:   ;DOT DASH DASH DASH
  jmp CALL_J
CASE_K:   ;DASH DOT DASH
  jmp CALL_K
CASE_L:   ;DOT DASH DOT DOT
  jmp CALL_L
CASE_M:   ;DASH DASH
  jmp CALL_M
CASE_N:   ;DASH DOT
  jmp CALL_N
CASE_O:   ;DASH DASH DASH
  jmp CALL_O
CASE_P:   ;DOT DASH DASH DOT
  jmp CALL_P
CASE_Q:   ;DASH DASH DOT DASH
  jmp CALL_Q
CASE_R:   ;DOT DASH DOT
  jmp CALL_R
CASE_S:   ;DOT DOT DOT
  jmp CALL_S
CASE_T:   ;DASH
  jmp CALL_T
CASE_U:   ;DOT DOT DASH
  jmp CALL_U
CASE_V:   ;DOT DOT DOT DASH
  jmp CALL_V
CASE_W:   ;DOT DASH DASH
  jmp CALL_W
CASE_X:   ;DASH DOT DOT DASH
  jmp CALL_X
CASE_Y:   ;DASH DOT DASH DASH
  jmp CALL_Y
CASE_Z:   ;DASH DASH DOT DOT
  jmp CALL_Z
TRANSMISSION_COMPLETE:
  ldi mpr, 0b00000000
  out PORTB, mpr
  jmp STATE_ONE_LOOP       ;If I'm not going to jump to main, I must be jumping to State_TWO_LOOP

;***********************************************************
;*      Functions and Subroutines
;***********************************************************

Get_Constant:
    push mpr
        ldi ZL, LOW(WELCOME_BEG << 1)
        ldi ZH, HIGH(WELCOME_BEG << 1)
        ldi YL, LOW(WelcomeAddress)
        ldi YH, HIGH(WelcomeAddress)
        ldi mpr, ENDWELCOMEDATA
        mov lastAddress, mpr

        Welcome_Loop:
          lpm mpr, Z+
          st Y+, mpr
          cpse YL, lastAddress
          jmp Welcome_Loop


        ldi ZL, LOW(PRESS_BEG << 1)
        ldi ZH, HIGH(PRESS_BEG << 1)
        ldi YL, LOW(PressAddress)
        ldi YH, HIGH(PressAddress)
        ldi mpr, ENDPRESSDATA
        mov lastAddress, mpr

        Press_Loop:
          lpm mpr, Z+
          st Y+, mpr
          cpse YL, lastAddress
          jmp Press_Loop

        ldi ZL, LOW(ENTER_BEG << 1)
        ldi ZH, HIGH(ENTER_BEG << 1)
        ldi YL, LOW(EnterAddress)
        ldi YH, HIGH(EnterAddress)
        ldi mpr, ENDENTERDATA
        mov lastAddress, mpr

        Enter_Loop:
          lpm mpr, Z+
          st Y+, mpr
          cpse YL, lastAddress
          jmp Enter_Loop
  pop mpr
  ret

Write_State_Zero:
    push mpr
        ldi XL, LOW(WelcomeAddress)
        ldi XH, HIGH(WelcomeAddress)
        ldi YL, LOW(LCDFirstLine)
        ldi YH, HIGH(LCDFirstLine)
        ldi mpr, ENDFIRSTLINE
        mov lastAddress, mpr
        rcall Write

        ldi XL, LOW(PressAddress)
        ldi XH, HIGH(PressAddress)
        ldi YL, LOW(LCDSecondLine)
        ldi YH, HIGH(LCDSecondLine)
        ldi mpr, ENDSECONDLINE
        mov lastAddress, mpr
        rcall write

        rcall LCDWrite
        pop mpr
    ret

  Write_State_One:
    push mpr
        ldi XL, LOW(EnterAddress)
        ldi XH, HIGH(EnterAddress)

        ldi YL, LOW(LCDFirstLine)
        ldi YH, HIGH(LCDFirstLine)
        ldi mpr, ENDFIRSTLINE
        mov lastAddress, mpr      ;Only filling in one line and one letter
        rcall Write
        ldi XL, LOW(LCDSecondLine)
        ldi XH, HIGH(LCDSecondLine)

        ldi YL, LOW(LCDSecondLine)
        ldi YH, HIGH(LCDSecondLine)

        ldi mpr, ALetter
        st Y+, mpr            ;Put the A in the second line and then print out the rest as spaces
        inc XL
        ldi mpr, ENDSECONDLINE
        mov lastAddress, mpr
        rcall Write
        rcall LCDWrite
        pop mpr
        ret
;Need to write a function to clear both lines for prep between each state
Clear:
  push mpr
  ldi mpr, ENDFIRSTLINE
  mov lastAddress, mpr
  ldi mpr, 0x20
  ldi YL, LOW(0x0099)
  ldi YL, HIGH(0x0099)

  CLEAR_LOOP_ONE:
  st Y+, mpr
  cpse YL, lastAddress
  rjmp CLEAR_LOOP_ONE
  ldi mpr, ENDSECONDLINE
  mov lastAddress, mpr
  ldi mpr, 0x20
  ldi YL, LOW(LCDSecondLine)
  ldi YH, HIGH(LCDSecondLine)
  CLEAR_LOOP_TWO:
  st Y+, mpr
  cpse YL, lastAddress
  rjmp CLEAR_LOOP_TWO

  call LCDWrite
  pop mpr
  ret

Write:
  push mpr
  WRITE_LOOP:
  ld mpr, X+
  st Y+, mpr
  cpse YL, lastAddress
  rjmp WRITE_LOOP
  pop mpr
  ret

DOT:
  push mpr
  in mpr, SREG
  push mpr
  ;I'm going to start and stop my clocks in these functions
  ldi mpr, 0b11110000
  out PORTB, mpr
  ldi mpr, HIGH(ONE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(ONE_SECOND_DELAY)
  out OCR1AL, mpr
  call START_CLOCK

DOT_WAIT:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne DOT_WAIT
  out PORTB, mpr
  call STOP_CLOCK
  out TIFR, mpr

  ldi mpr, HIGH(ONE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(ONE_SECOND_DELAY)
  out OCR1AL, mpr
  call START_CLOCK

DOT_DELAY:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne DOT_DELAY

  call STOP_CLOCK

  out TIFR, mpr
  pop mpr
  out SREG, mpr
  pop mpr
  ret

END_DOT:
  push mpr
  in mpr, SREG
  push mpr
  ;I'm going to start and stop my clocks in these functions
  ldi mpr, 0b11110000
  out PORTB, mpr
  ldi mpr, HIGH(ONE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(ONE_SECOND_DELAY)
  out OCR1AL, mpr
  call START_CLOCK

END_DOT_WAIT:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne END_DOT_WAIT
  out PORTB, mpr
  out TIFR, mpr
  call STOP_CLOCK

  ldi mpr, HIGH(THREE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(THREE_SECOND_DELAY)
  out OCR1AL, mpr
  call START_CLOCK

END_DOT_DELAY:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne END_DOT_DELAY

  call STOP_CLOCK
  out TIFR, mpr
  pop mpr
  out SREG, mpr
  pop mpr
  ret

DASH:
  push mpr
  in mpr, SREG
  push mpr
  ldi mpr, 0b11110000
  out PORTB, mpr
  ldi mpr, HIGH(THREE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(THREE_SECOND_DELAY)
  out OCR1AL, mpr
  call START_CLOCK
  DASH_WAIT:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne DASH_WAIT
  out PORTB, mpr
  call STOP_CLOCK
  out TIFR, mpr

  ldi mpr, HIGH(ONE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(ONE_SECOND_DELAY)
  out OCR1AL, mpr

  call START_CLOCK
  DASH_DELAY:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne DASH_DELAY
  call STOP_CLOCK

  out TIFR, mpr
  pop mpr
  out SREG, mpr
  pop mpr
  ret

END_DASH:
  push mpr
  in mpr, SREG
  push mpr
  ldi mpr, 0b11110000
  out PORTB, mpr
  ldi mpr, HIGH(THREE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(THREE_SECOND_DELAY)
  out OCR1AL, mpr
  call START_CLOCK
  END_DASH_WAIT:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne END_DASH_WAIT
  out PORTB, mpr
  call STOP_CLOCK
  out TIFR, mpr

  ldi mpr, HIGH(THREE_SECOND_DELAY)
  out OCR1AH, mpr
  ldi mpr, LOW(THREE_SECOND_DELAY)
  out OCR1AL, mpr

  call START_CLOCK
  END_DASH_DELAY:
  in mpr, TIFR
  andi mpr, 0b00010000
  cpi mpr, 0b00010000
  brne END_DASH_DELAY
  call STOP_CLOCK

  out TIFR, mpr
  pop mpr
  out SREG, mpr
  pop mpr
  ret

START_CLOCK: ;I just need to start with my clock source as 101 in TCCR1B after I've loaded in my
values
  push mpr
  in mpr, SREG
  push mpr
  clr mpr
  out TCNT1H, mpr
  out TCNT1L, mpr
  in mpr, TCCR1B
  ori mpr, 0b00000101
  out TCCR1B, mpr
  pop mpr
  out SREG, mpr
  pop mpr
  ret

STOP_CLOCK:
  push mpr
  in mpr, SREG
  push mpr
  in mpr, TCCR1B
  andi mpr, 0b11111000   ;Clear the bottom three bits and get rid of the clock
  out TCCR1B, mpr
  clr mpr                ;I'm going to clear the TCNT1H/L even though it's redundant here
  out TCNT1H, mpr
  out TCNT1L, mpr
  pop mpr
  out SREG, mpr
  pop mpr
  ret
CALL_A:
  call DOT
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_B:
  call DASH
  call DOT
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_C:
  call DASH
  call DOT
  call DASH
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_D:
  call DASH
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_E:   ;DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_F:   ;DOT DOT DASH DOT
  call DOT
  call DOT
  call DASH
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_G:   ;DASH DASH DOT
  call DASH
  call DASH
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_H:   ;DOT DOT DOT DOT
  call DOT
  call DOT
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_I:   ;DOT DOT
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_J:   ;DOT DASH DASH DASH
  call DOT
  call DASH
  call DASH
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_K:   ;DASH DOT DASH
  call DASH
  call DOT
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_L:   ;DOT DASH DOT DOT
  call DOT
  call DASH
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_M:   ;DASH DASH
  call DASH
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_N:   ;DASH DOT
  call DASH
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_O:   ;DASH DASH DASH
  call DASH
  call DASH
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_P:   ;DOT DASH DASH DOT
  call DOT
  call DASH
  call DASH
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_Q:   ;DASH DASH DOT DASH
  call DASH
  call DASH
  call DOT
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_R:   ;DOT DASH DOT
  call DOT
  call DASH
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_S:   ;DOT DOT DOT
  call DOT
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_T:   ;DASH
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_U:   ;DOT DOT DASH
  call DOT
  call DOT
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_V:   ;DOT DOT DOT DASH
  call DOT
  call DOT
  call DOT
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_W:   ;DOT DASH DASH
  call DOT
  call DASH
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_X:   ;DASH DOT DOT DASH
  call DASH
  call DOT
  call DOT
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_Y:   ;DASH DOT DASH DASH
  call DASH
  call DOT
  call DASH
  call END_DASH
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
CALL_Z:   ;DASH DASH DOT DOT
  call DASH
  call DASH
  call DOT
  call END_DOT
  inc currentLetter
  dec count
  jmp SEARCH_LOOP
;-----------------------------------------------------------------
; Sub:  Wait
; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
;              waitcnt*10ms.  Just initialize wait for the specific amount
;              of time in 10ms intervals. Here is the general eqaution
;              for the number of clock cycles in the wait loop:
;                      ((3 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------------
Wait_Func:
               push   waitcnt                ; Save wait register
               push   ilcnt                  ; Save ilcnt register
               push   olcnt                  ; Save olcnt register

Loop:   ldi           olcnt, 224             ; load olcnt register
OLoop:  ldi           ilcnt , 237            ; load ilcnt register
ILoop:  dec           ilcnt                  ; decrement ilcnt
               brne   ILoop                  ; Continue Inner Loop
               dec           olcnt           ; decrement olcnt
               brne   OLoop                  ; Continue Outer Loop
               dec           waitcnt         ; Decrement wait
               brne   Loop                   ; Continue Wait loop

               pop           olcnt           ; Restore olcnt register
               pop           ilcnt   ; Restore ilcnt register
               pop           waitcnt         ; Restore wait register
               ret                           ; Return from subroutine


;*********************************************************
;* Stored Program Data
;*********************************************************

WELCOME_BEG:
.DB  "Welcome!        "
WELCOME_END:

PRESS_BEG:
.DB   "Please Press PD0"
PRESS_END:

ENTER_BEG:
.DB   "Enter word:     "
ENTER_END:
;***********************************************************
;*     Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"              ; Include the LCD Driver
```