ECE 375 LAB 8

Introduction to AVR Development Tools

Lab Time: Friday 2-4

Hao Truong

INTRODUCTION

The purpose of this lab is to utilize the 16 bit Timer/Counter 1, write AVR assembly code to transmit precisely timed patterns, and combine the timing and individual patterns to create Morse code messages. Morse code is a communication method that is used to transmit messages via sound or light. In this lab, we will be designing a system to broadcast Morse code messages using LEDs.

PROGRAM OVERVIEW

The program provides the functionality for the Morse code transmitter using LEDs on the microcontroller board to represent light. The full table of Morse code characters is provided from Wikipedia. When a dot is active, the 3 LEDs (PB5-PB7) light up for one second. When a dash is active, the LEDs turn on for three seconds. The LEDs will be disabled for a second during the pause before the next dot or dash, of the same letter, occurs. The LEDs will be disabled for 3 seconds between letters. When the CPU first boosts up, the LCD will display "Welcome!" on the first line and "Please press PD0" on the second line. This will remain indefinitely until the user presses PD0. When PD0 is first pressed, the user is able to select the characters that they want to send; one character is selected once at a time. The LCD needs to be updated whenever user provides inputs. When PD7 is pressed, the CPU will change the current character and iterate through the alphabet in reverse order. For example, if the current character is A and PD7 is pressed, the current character is then changed to Z. Pressing PD6 will also change the current character and iterate through the alphabet in forward order. When PD0 is pressed again, the CPU will confirm the current character and moves to the right by one unit. When PD4 is pressed, the message will be transmitted.

Besides the standard. INIT and MAIN routines within the program, additional routines were created and used. These routines will be described later in this paper.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization that allows the program to execute correctly. Inside the INIT routine, important components were initialized such as Stack Pointer, LCD display, input port (PORT D), and output port (PORT B). Besides that, Timer/Counter0 and Timer/Counter1 were configured, and the LCD was also configured to display the content when the CPU first boosted.

MAIN ROUTINE

The Main routine executes an indefinite simple pooling loop that checks to see which a button was pressed and then call the corresponding routine.

SUBROUTINES

1. DisplayPromt

The DisplayPromt routine displays the promt ("Welcome!" on first line and "Please press PD0" on second line) on the LCD after the CPU first boosts up.

2. ButtonPD0

The ButttonPD0 routine confirms the current character and moves to the right. When PD0 is pressed for the first time after the CPU first boost, the LCD will display “Enter word:” on the first line and “A” on the second line.

3. ButtonPD6

The ButtonPD6 routine changes the current character and iterates through the alphabet in forward order.

4. ButtonPD7

The ButtonPD6 routine changes the current character and iterates through the alphabet in reverse order.

5. CheckORC0

The CheckORC0 routine checks the current value of OCR0, assigns corresponding character to the current character, and displays it on the LCD

6. Waiting

The Waiting function provides a delay for switch debouncing

7. Delay1Sec

The Delay1Sec routine configures Timer/Counter1 to trigger a delay of 1 second

8. ButtonPD4

The ButtonPD4 routine transmits the message that is displayed on the LCD using LEDs. After the message is done being transmitted, the LCD will display “Enter word:” on the first line and “A” on the second line.

9. Letters

The Letters routine compares the current to each character in the alphabet and performs the corresponding Morse code patterns.

10. Dot

The Dot routine illuminates the LEDs (PB7-PB5) for one second

11. Dash

The Dash routine illuminates the LEDs for three seconds.

12. Disabled1Sec

The Disabled1Sec disables the LEDs for 1 second.

13. Disabled3Sec

The Disabled3Sec disables the LEDs for 3 seconds.

CONCLUSION

In this lab, we were required to utilize the 16 bit Timer/Counter 1, write AVR assembly code to transmit precisely timed patterns, and combine the timing and individual patterns to create Morse code messages. We designed a system to broadcast Morse code messages using LEDs since we didn't have spot light.

SOURCE CODE

```
;***********************************************************
;*
;*     Hao_Truong_Lab8_sourcecode.asm
;*
;*     Morse Code Transmitter
;*
;*
;***********************************************************
;*
;*      Author: Hao Truong
;*        Date: 12/03/21
;*
;***********************************************************

.include "m128def.inc"                  ; Include definition file

;***********************************************************
;*     Internal Register Definitions and Constants
;***********************************************************
.def   mpr = r16                        ; Multipurpose register is
                                                ; required for LCD Driver

.def   temp = r23
.def   charNum = r24
.def   isPromt = r25 ; if isPromt = 0, then the code will display the promt as below:
                                                ;
       Enter word:
                                                ;
       A

;***********************************************************
;*     Start of Code Segment
;***********************************************************
.cseg                                           ; Beginning of code segment

;***********************************************************
;*     Interrupt Vectors
;***********************************************************
.org   $0000                            ; Beginning of IVs
              rjmp INIT                         ; Reset interrupt

.org   $0046                            ; End of Interrupt Vectors

;***********************************************************
;*     Program Initialization
;***********************************************************
INIT:                                           ; The initialization routine
              ; Initialize Stack Pointer
```

```
		ldi		mpr, low(RAMEND)
		out		SPL, mpr		; Load SPL with low byte of RAMEND
		ldi		mpr, high(RAMEND)
		out		SPH, mpr		; load SPH with high byte of RAMEND

		; Set X to begginning address of Message
		ldi		XL, low(Message)	; Load low byte
		ldi		XH, high(Message)	; Load high byte

		; Initialize LCD Display
		RCALL		LCDInit

		; initialize isPromt
		ldi		isPromt, 0		; load 0 into isPromt

		ldi		charNum, 0

		; Initialize Timer/Counter1 (This will be used for delay)
		ldi			mpr, 0b00000100		; set prescalar to 64,
WGM13:12=00
		out			TCCR1B, mpr		; by default WGM11:10=00,
normal mode



		; Configure 8-bit Timer/Counters  (This will be used to keep track of
current state(character))
		ldi		mpr, 0b01111001			; Setting up the Fast PWM
mode
		out		TCCR0, mpr			; No prescaling (And
inverting)
		ldi		mpr, 0				; Initialize to be 0 at
first
		out		OCR0, mpr


		; Initialize Port B for output
		ldi			mpr, $FF		; set Port B Data
Direction Register
		out			DDRB, mpr		; for output
		ldi			mpr, $00		; initialize Port B
Data Register
		out			PORTB, mpr		; so all port B data
outputs are low



		; Initialize Port D for input
		ldi			mpr, $00		; set Port D Data Direction
Register
		out			DDRD, mpr		; for input
		ldi			mpr, $FF		; initialize Port D Data
Direction Register
		out			PORTD, mpr		; so all Port D inputs are
Tri-State

		; move first string from Program Memory to Data Memory
```

```
            ldi         ZL, low(STRING_1 << 1)          ; extract low byte of
STRING_BEG
            ldi         ZH, high(STRING_1 << 1)         ; extract high byte of
STRING_BEG
                                                                        ; Z
now points to first char (W)
            ldi         YL, $00                                     ; Y <-
data memory address of character destination (line 1)
            ldi         YH, $01
            ldi         temp, 8                                 ; used to
count down chars, first line contains 8 letters
Line1:
            lpm         mpr, Z+                             ; load Z to mpr
            st          Y+, mpr                             ; load mpr to Y
            dec         temp                            ; count down # of words to
add
            BRNE   Line1                            ; if not done loop

            ; Move second string from Program Memory to Data Memory
            ldi         ZL, low(STRING_2 << 1)          ; extract low byte of
STRING_BEG
            ldi         ZH, high(STRING_2 << 1)         ; extract high byte of
STRING_BEG
                                                                        ; Z
now points to first char (P)
            ldi         YL, $10                                     ; Y <-
data memory address of character destination (line 2)
            ldi         YH, $01
            clr         temp
            ldi         temp, 16                                ; used to
count down chars, first line contains 16 letters
Line2:
            lpm         mpr, Z+                             ; load Z to mpr
            st          Y+, mpr                             ; load mpr to Y
            dec         temp                            ; count down # of words to
add
            BRNE   Line2                            ; if not done loop

            rcall       LCDWrite                    ; write both lines of the LCD

;***********************************************************
;*     Main Program
;***********************************************************
MAIN:                                           ; The Main program
            ; Check for inputs (PD0, PD4, PD6, PD7)


;           rcall       ButtonPD0
;           rcall       ButtonPD0
;           rcall       ButtonPD4                           ; call subroutine
PD6
;           rcall       ButtonPD0


            in                  temp, PIND              ; get input from Port D
            cpi                 temp, (0b11111110)  ; check for input PD0
            brne        Next                        ; continue with next check if not
equal
```

```
            rcall       ButtonPD0                           ; call subroutine
PD0
            rjmp        MAIN
Next:
            cpi                 temp, (0b11101111)  ; check for PD4
            brne        Next1                       ; continue with next check if not
equal
;           rcall       LCDClr                      ; clear both lines of LCD
            rcall       ButtonPD4                           ; call subroutine
PD4
            rjmp        MAIN
Next1:
            cpi                 temp, (0b10111111)  ; check for PD6
            brne        Next2                       ; continue with next check if not
equal
            rcall       ButtonPD6                           ; call subroutine
PD6
            rjmp        MAIN
Next2:
            cpi                 temp, (0b01111111)  ; check for Button1
            brne        MAIN                        ; continue with next check if not
equal
;           rcall       LCDClr                      ; clear both lines of LCD
            rcall       ButtonPD7                           ; call subroutine
PD7
            rjmp        MAIN



            rjmp   MAIN             ; jump back to main and create an infinite
                                                    ; while loop.  Generally, every
main program is an
                                                    ; infinite while loop, never let
the main program
                                                    ; just run off

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

;-----------------------------------------------------------
; sub: DisplayPromt
; Desc: This function displays the promt when the PD0 is pressed
;             after the CPU first boosts up
;-----------------------------------------------------------
DisplayPromt:
;           push   isPromt
            ; move third string from Program Memory to Data Memory
            ldi         ZL, low(STRING_3 << 1)          ; extract low byte of
STRING_BEG
            ldi         ZH, high(STRING_3 << 1)         ; extract high byte of
STRING_BEG
                                                                        ; Z
now points to first char (E)
            ldi         YL, $00                                         ; Y <-
data memory address of character destination (line 1)
            ldi         YH, $01
```

```
            ldi         temp, 12                                    ; used to
count down chars, first line contains 12 letters
Line3:
            lpm         mpr, Z+                                     ; load Z to mpr
            st          Y+, mpr                                     ; load mpr to Y
            dec         temp                                ; count down # of words to
add
            BRNE  Line3                             ; if not done loop
            rcall LCDWrLn1                          ; display line 1

            rcall LCDClrLn2                         ; clear line 2

            ldi         YL, $10                                                 ; Y <-
data memory address of character destination (line 2)
            ldi         YH, $01
            ldi         mpr, 'A'                            ; load character ‘A’ into
mpr
            st          Y, mpr
            rcall   LCDWrLn2                ; write character to LCD

            inc         isPromt

;           pop         isPromt
            ret                                     ; Return from Subroutine



;-----------------------------------------------------------
; sub: ButtonPD0
; Desc: Pressing PD0 confirms the current character and moves to the right.
;           When PD0 is pressed the first time after the CPU boosts up,
;            this information will be displayed on the screen:
;                 Enter word:
;                 A
;-----------------------------------------------------------
ButtonPD0:                                                  ; Begin a function
with a label
            ; switch debouncing delay
            ldi         r17, 20
            rcall Waiting

;           rcall       Delay

            cpi         isPromt, 0                          ; if isPromt is 0
            breq  Do
            cpi         charNum, 16
            breq  CallPD4
            rjmp  Done
CallPD4:
            rcall ButtonPD4
            rjmp  SKIP1
Do:
            rcall DisplayPromt
            rjmp  Skip1

Done:
            inc         charNum                             ; increment the number of
characters
```

```
                st              X+, mpr


                ; Set Z to begginning address of Message
                ldi             ZL, low(Message)    ; Load low byte
                ldi             ZH, high(Message)   ; Load high byte


                ldi             YL, $10                                         ; Y <-
data memory address of character destination (line 2)
                ldi             YH, $01
                clr             temp
                mov             temp, charNum                           ; used to
count down chars, first line contains 16 letters
loop1:
                ld              mpr, Z+                                 ; load Z to mpr
                st              Y+, mpr                                 ; load mpr to Y
                dec             temp                            ; count down # of words to
add
                BRNE    loop1                           ; if not done loop

                ldi             temp, 0
                out             OCR0, temp

                ldi             mpr, 'A'
                st              Y, mpr

;               rcall   LCDClrLn2
                rcall   LCDWrLn2


Skip1:
                ret


;***********************************************************
; sub: ButtonPD6
; Desc: This function changes the current character and
;               iterates through the alphabet in forward order
;***********************************************************
ButtonPD6:

;               rcall   Delay
                ldi             r17, 15
                rcall   Waiting
                cpi             isPromt, 0                      ; if CPU first boosts up
                breq    SKIP6                           ; ignore PD6
                in              temp, OCR0                      ; Else Read in OCR0
                cpi             temp, 225                       ; Compare if OCR0 is 234
(Z)
                breq    WrapAround2                     ; wrap around to 0 (A)
                ldi             mpr, 9
                add             temp, mpr
                out             OCR0, temp
                rjmp    Done1
WrapAround2:
                ldi             temp, 0
                out             OCR0, temp
```

```
Done1:
              in            temp, OCR0
              rcall   CheckOCR0
SKIP6:
              ret


;-----------------------------------------------------------
; sub: ButtonPD7
; Desc: This function changes the current character and
;             interates through the alphabet in forward order
;-----------------------------------------------------------
ButtonPD7:
;             rcall   Delay
;             rcall   Delay
;             rcall   Delay
              ldi           r17, 15
              rcall   Waiting
              cpi           isPromt, 0                      ; if CPU first boosts up
              breq    SKIP7                           ; ignore PD7
              in            temp, OCR0                      ; Else Read in OCR0
              cpi           temp, 0                         ; Compare if OCR0 is 0 (Z)
              breq    WrapAround3                     ; wrap around  (Z)
              ldi           mpr, 9
              sub           temp, mpr
              out           OCR0, temp
              rjmp    Done2
WrapAround3:
              ldi           temp, 225
              out           OCR0, temp
Done2:
              in            temp, OCR0
              rcall   CheckOCR0
SKIP7:
              ret

;-----------------------------------------------------------
; sub: CheckORC0
; Desc: This function checks the current value of OCR0, assigns corresponding character,
;             and display the character on LCD
;-----------------------------------------------------------
CheckOCR0:
              push    temp
              cpi           temp, 0
              breq    A
              cpi           temp, 9
              breq    B
              cpi           temp, 18
              breq    C
              cpi           temp, 27
              breq    D
              cpi           temp, 36
              breq    E
              cpi           temp, 45
              breq    F
              cpi           temp, 54
              breq    G
              cpi           temp, 63
```

```
                breq    H
                cpi             temp, 72
                breq    I
                cpi             temp, 81
                breq    J
                cpi             temp, 90
                breq    K
                cpi             temp, 99
                breq    L
                cpi             temp, 108
                breq    M
                cpi             temp, 117
                breq    N
                cpi             temp, 126
                breq    O
                cpi             temp, 135
                breq    P
                cpi             temp, 144
                breq    QQ
                cpi             temp, 153
                breq    RR
                cpi             temp, 162
                breq    S
                cpi             temp, 171
                breq    T
                cpi             temp, 180
                breq    U
                cpi             temp, 189
                breq    V
                cpi             temp, 198
                breq    W
                cpi             temp, 207
                breq    XX
                cpi             temp, 216
                breq    YY
                cpi             temp, 225
                breq    ZZ
A:              ldi             mpr, 'A'
                rjmp    Print
B:              ldi             mpr, 'B'
                rjmp    Print
C:              ldi             mpr, 'C'
                rjmp    Print
D:              ldi             mpr, 'D'
                rjmp    Print
E:              ldi             mpr, 'E'
                rjmp    Print
F:              ldi             mpr, 'F'
                rjmp    Print
G:              ldi             mpr, 'G'
                rjmp    Print
H:              ldi             mpr, 'H'
                rjmp    Print
I:              ldi             mpr, 'I'
                rjmp    Print
J:              ldi             mpr, 'J'
                rjmp    Print
K:              ldi             mpr, 'K'
```

```
                rjmp    Print
L:              ldi             mpr, 'L'
                rjmp    Print
M:              ldi             mpr, 'M'
                rjmp    Print
N:              ldi             mpr, 'N'
                rjmp    Print
O:              ldi             mpr, 'O'
                rjmp    Print
P:              ldi             mpr, 'P'
                rjmp    Print
QQ:             ldi             mpr, 'Q'
                rjmp    Print
RR:             ldi             mpr, 'R'
                rjmp    Print
S:              ldi             mpr, 'S'
                rjmp    Print
T:              ldi             mpr, 'T'
                rjmp    Print
U:              ldi             mpr, 'U'
                rjmp    Print
V:              ldi             mpr, 'V'
                rjmp    Print
W:              ldi             mpr, 'W'
                rjmp    Print
XX:             ldi             mpr, 'X'
                rjmp    Print
YY:             ldi             mpr, 'Y'
                rjmp    Print
ZZ:             ldi             mpr, 'Z'
                rjmp    Print
Print:
;               st              X, mpr


                ldi             line, 2
                mov             count, charNum
;               rcall   LCDClrLn2
                rcall   LCDWriteByte

                pop             temp
                ret



;-----------------------------------------------------------------
; Sub: Waiting
; 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
;-----------------------------------------------------------------
Waiting:
                push    r17                     ; Save wait register
                push    r18                     ; Save ilcnt register
                push    r19                     ; Save olcnt register
```

```
Loop:  ldi          r19, 224            ; load olcnt register
OLoop: ldi          r18, 237            ; load ilcnt register
ILoop: dec          r18                 ; decrement ilcnt
             brne   ILoop               ; Continue Inner Loop
             dec          r19           ; decrement olcnt
             brne   OLoop               ; Continue Outer Loop
             dec          r17           ; Decrement wait
             brne   Loop                ; Continue Wait loop

             pop          r19           ; Restore olcnt register
             pop          r18           ; Restore ilcnt register
             pop          r17           ; Restore wait register
             ret                        ; Return from subroutine


;-----------------------------------------------------------
; sub: Delay
; Desc: configure Timer/Counter1 so that your unit delay
;             is one second.
;-----------------------------------------------------------
Delay1Sec:
       push   mpr
       ldi          mpr, high(3036)          ; Load the value for delay
       out          TCNT1H, mpr                    ; Must load high byte first
       ldi          mpr, low(3036)
       out          TCNT1L, mpr
LOOP3:
       IN           mpr, TIFR                      ; Skip if TOV1 flag in TIFR is set
       sbrs   mpr, TOV1                    ; TOV1 = bit 2
       rjmp   LOOP3                        ; Loop if TOV1 not set
       ldi          mpr, 0b00000100                ; Reset OCF1A
       out          TIFR, mpr                      ; Note - write 1 to reset

       pop          mpr
       ret


;-----------------------------------------------------------
; sub: ButtonPD4
; Desc: Transmit message that is displayed
;
;-----------------------------------------------------------
ButtonPD4:
             ldi          r17, 30
             rcall  Waiting

             ; have Z point at Message
             ldi          ZL, low(Message)
             ldi          ZH, high(Message)

             clr          temp                   ; clear temp
             mov          temp, charNum          ; copy mpr to temp

loop4:
             ld           mpr, Z+                      ; load character in mpr
             rcall  Letters
             rcall  Disable3Sec
;            clr          temp
```

```
;               st              Z+, temp
                dec             charNum
                brne    loop4


                ; have Z point at Message
                ldi             ZL, low(Message)
                ldi             ZH, high(Message)
.               clr             mpr                             ; clear temp
;               mov             temp, charNum                   ; copy mpr to temp
loop5:
                ;clear message after transmitting
;               clr             mpr
                st              Z+, mpr
                dec             temp
                brne    loop5

                clr             charNum


                rcall   LCDClrLn2
                ldi             YL, $10                                                 ; Y <-
data memory address of character destination (line 2)
                ldi             YH, $01
                ldi             mpr, 'A'                        ; load character ‘A’ into
mpr
                st              Y, mpr

                ldi             temp, 0
                out             OCR0, temp
                rcall    LCDWrLn2                   ; write character to LCD

                ; Set X to begginning address of Message
                ldi             XL, low(Message)    ; Load low byte
                ldi             XH, high(Message)   ; Load high byte



                ret



;-----------------------------------------------------------
; sub: Letters
; Desc: This routine compares the current character to each character
;               in the alphabet and perform the corresponding Morse code patterns
;
;-----------------------------------------------------------
Letters:
                push    mpr
                cpi             mpr, 'A'
                brne    n1
                rjmp    Letter_A
n1:             cpi             mpr, 'B'
                brne    n2
                rjmp    Letter_B
n2:             cpi             mpr, 'C'
                brne    n3
```

```
            rjmp    Letter_C
n3:         cpi         mpr, 'D'
            brne    n4
            rjmp    Letter_D
n4:         cpi         mpr, 'E'
            brne    n5
            rjmp    Letter_E
n5:         cpi         mpr, 'F'
            brne    n6
            rjmp    Letter_F
n6:         cpi         mpr, 'G'
            brne    n7
            rjmp    Letter_G
n7:         cpi         mpr, 'H'
            brne    n8
            rjmp    Letter_H
n8:         cpi         mpr, 'I'
            brne    n9
            rjmp    Letter_I
n9:         cpi         mpr, 'J'
            brne    n10
            rjmp    Letter_J
n10:  cpi           mpr, 'K'
            brne    n11
            rjmp    Letter_K
n11:  cpi           mpr, 'L'
            brne    n12
            rjmp    Letter_L
n12:  cpi           mpr, 'M'
            brne    n13
            rjmp    Letter_M
n13:  cpi           mpr, 'N'
            brne    n14
            rjmp    Letter_N
n14:  cpi           mpr, 'O'
            brne    n15
            rjmp    Letter_O
n15:  cpi           mpr, 'P'
            brne    n16
            rjmp    Letter_P
n16:  cpi           mpr, 'Q'
            brne    n17
            rjmp    Letter_Q
n17:  cpi           mpr, 'R'
            brne    n18
            rjmp    Letter_R
n18:  cpi           mpr, 'S'
            brne    n19
            rjmp    Letter_S
n19:  cpi           mpr, 'T'
            brne    n20
            rjmp    Letter_T
n20:  cpi           mpr, 'U'
            brne    n21
            rjmp    Letter_U
n21:  cpi           mpr, 'V'
            brne    n22
            rjmp    Letter_V
```

```
n22:    cpi         mpr, 'W'
        brne    n23
        rjmp    Letter_W
n23:    cpi         mpr, 'X'
        brne    n24
        rjmp    Letter_X
n24:    cpi         mpr, 'Y'
        brne    n25
        rjmp    Letter_Y
n25:    cpi         mpr, 'Z'
        rjmp    Letter_Z

Letter_A:
        rcall   Dot
        rcall   Delay1Sec
        rcall   Disable1Sec
        rcall   Dash
        rjmp    done5

Letter_B:
;       rcall   Disable3Sec
        rcall   Dash
        rcall   Disable1Sec
        rcall   Dot
        rcall   Disable1Sec
        rcall   Dot
        rcall   Disable1Sec
        rcall   Dot
        rjmp    done5


Letter_C:
;       rcall   Disable3Sec
        rcall   Dash
        rcall   Disable1Sec
        rcall   Dot
        rcall   Disable1Sec
        rcall   Dash
        rcall   Disable1Sec
        rcall   Dot
        rjmp    done5
;       rcall   Disable1Sec

Letter_D:
        rcall   Dash
        rcall   Disable1Sec
        rcall   Dot
        rcall   Disable1Sec
        rcall   Dot
        rjmp    done5

Letter_E:
        rcall   Dot
        rjmp    done5

Letter_F:
        rcall   Dot
        rcall   Disable1Sec
```

```
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rjmp    done5

Letter_G:
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rjmp    done5

Letter_H:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rjmp    done5

Letter_I:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rjmp    done5

Letter_J:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_K:
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_L:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rjmp    done5
```

```
Letter_M:
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dash

Letter_N:
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dot
            rjmp   done5

Letter_O:
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dash
            rjmp   done5

Letter_P:
            rcall  Dot
            rcall  Disable1Sec
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dot
            rjmp   done5

Letter_Q:
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dot
            rcall  Disable1Sec
            rcall  Dash
            rjmp   done5

Letter_R:
            rcall  Dot
            rcall  Disable1Sec
            rcall  Dash
            rcall  Disable1Sec
            rcall  Dot
            rjmp   done5

Letter_S:
            rcall  Dot
            rcall  Disable1Sec
            rcall  Dot
            rcall  Disable1Sec
            rcall  Dot
            rjmp   done5

Letter_T:
            rcall  Dash
```

```
                rjmp    done5

Letter_U:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_V:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_W:
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_X:
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_Y:
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dash
                rjmp    done5

Letter_Z:
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dash
                rcall   Disable1Sec
                rcall   Dot
                rcall   Disable1Sec
                rcall   Dot
                rjmp    done5
```

```
done5:


            pop         mpr
            ret

;-----------------------------------------------------------
; sub: Dot
; Desc: This function illuminates LEDs for 1 second
;-----------------------------------------------------------
Dot:
            ldi         temp, 0b11100000                ;illuminate for LEDs
            out         PORTB, temp
;           rcall  Delay1Sec                        ; wait  for 1 sec
            rcall  Delay1Sec
            ret

;-----------------------------------------------------------
; sub: Dash
; Desc: This function illuminates LEDs for 3 second
;-----------------------------------------------------------
Dash:
            ldi         temp, 0b11100000                ;illuminate for LEDs
            out         PORTB, temp
            rcall  Delay1Sec                        ; wait for 3 sec
            rcall  Delay1Sec
            rcall  Delay1Sec
            ret

;-----------------------------------------------------------
; sub: Disable1Sec
; Desc: This function disables LEDs for 3 seconds
;-----------------------------------------------------------
Disable1Sec:
            ldi         temp, 0b00000000
            out         PORTB, temp
            rcall  Delay1Sec
            ret


;-----------------------------------------------------------
; sub: Disable3Sec
; Desc: This function disables LEDs for 3 seconds
;-----------------------------------------------------------
Disable3Sec:
            ldi         temp, 0b00000000
            out         PORTB, temp
            rcall  Delay1Sec
            rcall  Delay1Sec
            rcall  Delay1Sec
            ret

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;           beginning of your functions
;-----------------------------------------------------------
FUNC:                                       ; Begin a function with a label
```

```
                ; Save variables by pushing them to the stack

                ; Execute the function here

                ; Restore variables by popping them from the stack,
                ; in reverse order

                ret                                     ; End a function with RET

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

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
;-----------------------------------------------------------
STRING_1:
.DB             "Welcome!"              ; Declaring data in ProgMem
STRING_2:
.DB             "Please press PD0"
STRING_3:
.DB             "Enter word: "



;***********************************************************
;*     Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"            ; Include the LCD Driver

;***********************************************************
;*     Data Memory Allocation
;***********************************************************
.dseg
.org   $012B
Message:
                .byte 16
```