ECE 375 Lab 8

Morse Code Transmitter

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

TA Signature

Introduction

Program Overview

Additional Questions

Difficulties

Conclusion

Source Code

```
;***********************************************************
;*
;*  Robert Detjens & David Headrick Lab 8 Source Code
;*
;***********************************************************
;*
;*   Author: Robert Detjens
;*           David Headrick
;*     Date: 11/30/21
;*
;***********************************************************

.include "m128def.inc"    ; Include definition file

; ====  REGISTER CONSTANTS  ====

.def    mpr           = r16
.def    curr_letter   = r23
.def    zeroreg          = r24

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

;***********************************************************
;*  Interrupt Vectors
;***********************************************************
.org  $0000
  rjmp  INIT    ; reset interrupt
```

```
.org  $0046      ; end of interrupt vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
  ; Initialize the 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

  ; Configure I/O ports

  ; Initialize Port B for output
  ldi mpr, $FF ; Set Port B Directional Register
  out DDRB, mpr ; for output
  ldi mpr, $00 ; Initialize Port B for outputs
  out PORTB, mpr ; Port B outputs low

  ; Initialize Port D for input
  ldi mpr, $00 ; Set Port D Directional Register
  out DDRD, mpr ; for inputs
  ldi mpr, $FF ; Initialize Port D for inputs
  out PORTD, mpr ; with pull-up

  ; init LCD
  call  LCDInit

  ; configure Timer 1 for 1|3s sleep
  ; set timer to normal mode
  ; ldi   mpr,      0
  ; out   TCCR1A,   mpr
  ldi   mpr,      0b00000101  ; clk/1024 prescaling
  ; ldi   mpr,      0b00000100  ; TODO: remove, clk/256
  out   TCCR1B,   mpr
  ; ldi   mpr,      0
  ; out   TCCR1C,   mpr

  ; Enable global interrupts (if any are used)
  ; sei

  ; clear zero register
  clr     zeroreg
```

```
  ; Display intro message & wait for button
  rcall   INTRO

;***********************************************************
;*  Main Program
;***********************************************************
MAIN:
  rcall   PROMPT
  rcall   MORSE

  rjmp    MAIN    ; return to top of MAIN


;***********************************************************
;*  Functions
;***********************************************************


; INTRO()
;   Displays a welcome message on the LCD
;   and waits for the user to press button 0.
INTRO:
  ; load intro strings to lcd memory

  ; Move strings from Program Memory to Data Memory
  ; location of string in program memory
  ldi     ZL,   low(WELCOME_L1_S << 1)
  ldi     ZH,   high(WELCOME_L1_S << 1)
  ; dest addr in data memory (0x0100)
  ldi     YL,   low(LCD_Line1)
  ldi     YH,   high(LCD_Line1)
  istr1_l:
    lpm     mpr,  Z+
    st      Y+,   mpr
    cpi     YL, low(WELCOME_L1_E << 1)
    brne    istr1_l

  ; String 2
  ldi     ZL,   low(WELCOME_L2_S << 1)
  ldi     ZH,   high(WELCOME_L2_S << 1)
  ; dest addr in data memory (0x0100)
  ldi     YL,   low(LCD_Line2)
  ldi     YH,   high(LCD_Line2)
  istr2_l:
```

```
    lpm     mpr,  Z+
    st      Y+,   mpr
    cpi     YL, low(WELCOME_L2_E << 1)
    brne    istr2_l

  ; display the strings
  call LCDWrite

  ; now wait for pd0 button
  wait_for_b0:

    ; get button inputs
    in    mpr,    PIND
    sbrc  mpr,    0
    jmp   wait_for_b0

  rcall DEBOUNCE

  ret


; PROMPT()
;   Main function for getting the word input from the user.
;   Polls buttons:
;     rcall   DASH 6/7 step through letters
;     rcall   DASH 0 confirms the letter
;     rcall   DASH 4 confirms the word and returns (for transmission)
PROMPT:
  ; load prompt string into line 1
  ; location of string in program memory
  ldi     ZL,   low(PROMPT_S << 1)
  ldi     ZH,   high(PROMPT_S << 1)
  ; dest addr in data memory (0x0100)
  ldi     YL,   low(LCD_Line1)
  ldi     YH,   high(LCD_Line1)
  str1_l:
    lpm     mpr,  Z+
    st      Y+,   mpr
    cpi     YL, low(PROMPT_E << 1)
    brne    str1_l

  ; display string
  call    LCDWrLn1
  call    LCDClrLn2
```

```
; set Y to line 2
; second line in data memory (0x0110)
ldi     YL,   low(LCD_Line2)
ldi     YH,   high(LCD_Line2)

; turn on cursor on LCD
; no proc for this, so send command manually
; ldi     mpr,    0b00001110 ;  0  0  0  0  1  D  C  B
; call    LCDWriteCmd

; start with A
ldi     curr_letter,  'A'

PROMPT_LOOP:
  ; store current letter to LCD memory
  st      Y,  curr_letter
  ; update LCD
  call  LCDWrLn2

  ; get button inputs
  in    mpr,    PIND

  sbrs  mpr,    7 ; bit 7: decrement letter
  jmp   BIT_7
  sbrs  mpr,    6 ; bit 6: increment letter
  jmp   BIT_6
  sbrs  mpr,    0 ; bit 0: confirm letter
  jmp   BIT_0
  sbrs  mpr,    4 ; bit 4: confirm whole word
  jmp   BIT_4

  jmp   BIT_NONE  ; skip bit handling

  BIT_7:
    ; decrement letter
    dec   curr_letter
    ; wrap around if it underflowed
    cpi   curr_letter,   64   ; 'A' is 65
    brne  DEC_NOOP
      ; only decrement if above min
      ldi   curr_letter,  90  ; 'Z' is 90
    DEC_NOOP:
    jmp BIT_DONE

  BIT_6:
```

```
      ; increment letter
      inc   curr_letter
      ; wrap around if it overflowed
      cpi   curr_letter,   91   ; 'z' is 90
      brne  INC_NOOP
        ; only decrement if above min
        ldi   curr_letter,  65  ; 'A' is 65
      INC_NOOP:
      jmp BIT_DONE

    BIT_0:
      ; letter confirmed, increment dest addr

      ; store a final time with post-increment
      st      Y+,   curr_letter

      ; if 16 chars have been entered, start transmission (button 4)
      cpi     YL,   low(LCD_End)
      breq    BIT_4

      ; start with A once more
      ldi     curr_letter,  65

      jmp BIT_DONE

    BIT_4:
      ; word confirmed, exit prompt

      ; but make sure the string is space-terminated first
      st    Y+,           curr_letter
      ldi   curr_letter,  ' '
      st    Y+,           curr_letter

      ret

    BIT_DONE:
    rcall DEBOUNCE
    BIT_NONE:

  ; keep looping until button 4 is hit
  jmp   PROMPT_LOOP

DEBOUNCE:
  ; wait for a small delay to do some real basic debouncing
```

```
  ; wait a bit for debounce
  ; reuse exising wait func for inner loop
  ldi   mpr,    100
  debounce_l:
    ldi   wait,   255
    call  LCDWait
    dec   mpr
  brne  debounce_l

  ret

; MORSE()
;   Broadcasts the characters in data memory $1010:1020
;   as Morse code over the top 3 LEDS on port B
MORSE:
  ; turn on PIN/LED 4 to signal broadcasting
  ldi   mpr,    0b00010000
  out   PORTB,  mpr

  ; go through chars in line 2
  ; second line in data memory (0x0110)
  ldi     YL,   low(LCD_Line2)
  ldi     YH,   high(LCD_Line2)

  ; for some reason the first dot/dash is ignored,
  ; so "display" a letter with only one dot (e.g. E)
  ; and now subsequent letters (the actual message) works
  ; I Love Programming:tm:
  ldi     curr_letter,  'E'
  morse_loop:
    ; print current char
    rcall PRINT_MORSE

    ; load next one
    ld    curr_letter,  Y+

    ; if the the next character is not ' ', loop back
    cpi   curr_letter,  ' '
    brne  morse_loop

  ; turn off PIN/LED 4 when broadcast is done
  ldi   mpr,    0
  out   PORTB,  mpr

  ret
```

```
; PRINT_MORSE():
;   Prints the ascii char in curr_letter as morse code. This uses an jump into
;   JUMP_TABLE to efficiently perform the correct sequence of dots/dashes based
;   on an index calculated from the current letter.
PRINT_MORSE:

  ; curr_letter = (curr_letter - 'A') * 5
  subi  curr_letter,  'A'
  mov   mpr,  curr_letter
  lsl   curr_letter
  lsl   curr_letter
  add   curr_letter,  mpr

  ; load address of jump table into Z for indirect call
  ldi   ZL,   LOW(JUMP_TABLE)
  ldi   ZH,   HIGH(JUMP_TABLE)
  ; add calculated letter offset
  add   ZL,   curr_letter
  adc   ZH,   zeroreg

  ; indirect call to JUMP_TABLE+offset
  icall

  ; wait 2 more units (for 3 total) between letters
  rcall WAIT_1
  rcall WAIT_1

  ret

DOT:
  ; wait 1 unit on, 1 unit off

  ; turn on signal leds
  ldi   mpr,    0b11110000
  out   PORTB,  mpr

  rcall   WAIT_1

  ; turn off signal pins
  ldi   mpr,    0b00010000
  out   PORTB,  mpr

  rcall   WAIT_1
```

```
  ret

DASH:
  ; wait 3 units on, 1 unit off

  ; turn on signal leds
  ldi   mpr,    0b11110000
  out   PORTB,  mpr

  rcall   WAIT_3

  ; turn off signal pins
  ldi   mpr,    0b00010000
  out   PORTB,  mpr

  rcall   WAIT_1

  ret

WAIT_1:

  ; 0xFFFF - 16000 (0x3E80) = 0xC17F
  ldi   mpr,      0xC1
  out   TCNT1H,   mpr
  ldi   mpr,      0x80
  out   TCNT1L,   mpr

  ; wait for timer overflow (reuse loop)
  rjmp  wait_for_timer

WAIT_3:
  ; 0xFFFF - 48000 (0xBB800) = 0x447F
  ldi   mpr,      0x44
  out   TCNT1H,   mpr
  ldi   mpr,      0x7f
  out   TCNT1L,   mpr

  wait_for_timer:
    ; check TOV1 bit in TIFR flag register
    in    mpr,      TIFR
    andi  mpr,      0b00000100
    breq  wait_for_timer  ; loop if not set

  ; clear overflow flag
  ldi   mpr,      0b00000100
```

```
  out   TIFR,     mpr

  ret

JUMP_TABLE:
  rcall   DOT
  rcall   DASH
  ret
  nop
  nop
  rcall   DASH
  rcall   DOT
  rcall   DOT
  rcall   DOT
  ret
  rcall   DASH
  rcall   DOT
  rcall   DASH
  rcall   DOT
  ret
  rcall   DASH
  rcall   DOT
  rcall   DOT
  ret
  nop
  rcall   DOT
  ret
  nop
  nop
  nop
  rcall   DOT
  rcall   DOT
  rcall   DASH
  rcall   DOT
  ret
  rcall   DASH
  rcall   DASH
  rcall   DOT
  ret
  nop
  rcall   DOT
  rcall   DOT
  rcall   DOT
  rcall   DOT
  ret
```

```
rcall   DOT
rcall   DOT
ret
nop
nop
rcall   DOT
rcall   DASH
rcall   DASH
rcall   DASH
ret
rcall   DASH
rcall   DOT
rcall   DASH
ret
nop
rcall   DOT
rcall   DASH
rcall   DOT
rcall   DOT
ret
rcall   DASH
rcall   DASH
ret
nop
nop
rcall   DASH
rcall   DOT
ret
nop
nop
rcall   DASH
rcall   DASH
rcall   DASH
ret
nop
rcall   DOT
rcall   DASH
rcall   DASH
rcall   DOT
ret
rcall   DASH
rcall   DASH
rcall   DOT
rcall   DASH
ret
```

```
rcall   DOT
rcall   DASH
rcall   DOT
ret
nop
rcall   DOT
rcall   DOT
rcall   DOT
ret
nop
rcall   DASH
ret
nop
nop
nop
rcall   DOT
rcall   DOT
rcall   DASH
ret
nop
rcall   DOT
rcall   DOT
rcall   DOT
rcall   DASH
ret
rcall   DOT
rcall   DASH
rcall   DASH
ret
nop
rcall   DASH
rcall   DOT
rcall   DOT
rcall   DASH
ret
rcall   DASH
rcall   DOT
rcall   DASH
rcall   DASH
ret
rcall   DASH
rcall   DASH
rcall   DOT
rcall   DOT
ret
```

```
;***********************************************************
;*  Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"  ; LCD stuff

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

WELCOME_L1_S:
.DB   "Welcome!        "
WELCOME_L1_E:
WELCOME_L2_S:
.DB   "Please press PD0"
WELCOME_L2_E:
PROMPT_S:
.DB   "Enter word:     "
PROMPT_E:

;***********************************************************
;* Data Memory Allocation
;***********************************************************

.dseg
.org $0100
LCD_Line1: .byte $10
.org $0110
LCD_Line2: .byte $10
.org $0120
LCD_End:
```