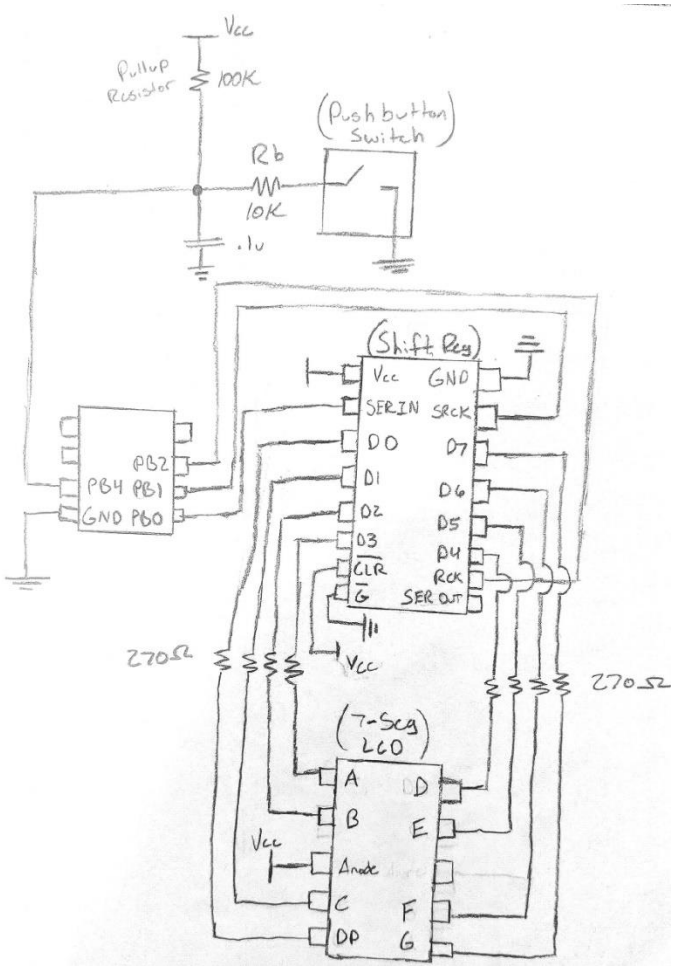
Team 23: Alex Powers and Ben Mitchinson
ECE:3360
Post-Lab Report 2

1. Introduction

The goal of Lab 2 was to create a counter with decrement, increment, and reset functionality. We accomplished this by using a 8 bit shift register, and a 7-segment LCD. We sampled the time that the button was pushed using a combination of subroutines for delays and register increments.

2. Schematic

Figure 1 shows the circuit design used to connect the 8 bit shift register, 7-segment LCD, and pushbutton. By adding a small grounded .1μF capacitor in parallel with a 100K resistor functioning as a pullup, we were able to remove any disruptive noise when referencing the PB4 status.



(Fig. 1, Circuit with two LEDs on PB2 and PB1)

3. Discussion

Our program implements the provided display subroutine, to load the register with byte sequences altered according to user input. 16 bytes of memory are reserved in SRAM that hold each value needed to represent digits on the display. These are then later indexed by utilizing a "lookup table", with the Z register (r30). In addition, register r20 is used to initialize those values in the lookup table, and register r19 is used to hold the state of whether the counter is in decrement mode.

Our "main" loop runs endlessly until a button press is measured. At that time, the "button_pressed" subroutine begins to iterate a register once every 10ms, using "sample_delay". Upon release, the program has an accurate measurement as to how long the button was held and can jump relatively to one of three subroutines. "update" decides between whether to jump to "reset_routine", "toggle_routine", or "move_routine". "toggle_routine" simply checks the current status of r19, and flips it's state. "move_routine" compares to make sure an overflow isn't about to occur, and if so, shifts the Z register to the proper position in order to account for the shift. The "reset_routine" shifts the Z register back to it's origin position, and toggles the decimal off before loading the display using previously described logic. These routines combine to fulfil the features defined below.

Implemented Features:

Counter Increment and Decrement	✓
Mode Selection Working	✓
Counter Reset Working	✓
Overflow to 0 and F when counting	✓

In order to appropriately time the user interaction based on 8MHz clock speed, we used the following equation to "nop" and create delays. This gave us high accuracy in being able to implement exact timing required by the project specifications and is done with several loops in the "sample_delay" subroutine.

$$C = \left(x_3 * \left(4 + \left(x_2 * \left(4 + (x_1 * 4)\right)\right)\right)\right) - (x_3 * x_2)$$

Figure 2: Our resultant calculation "sample_delay"

4. Conclusion

In this lab, we grew more comfortable navigating the flow of assembly branching and jumps, as well as managing memory to do so. Utilizing features of Atmel Studio to simulate IO pins along with viewing internal memory enabled us to problem solve much faster than originally anticipated. We look forward to using these strategies in the future.

5. Appendix A: Source Code

A-1: main.asm

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Assembly language file for Lab 2 in ECE:3360 (Embedded Systems)

; Spring 2018, The University of Iowa.

; Desc: runs the control logic for a button controlled counter

; B. Mitchinson, A. Powers

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.include "tn45def.inc"

.include "disp_values.inc" ; file with hex constants for 7-segment display

.dseg

.org 0x0060 ;Tiny45, Start in SRAM

disp_table: .byte 16

.cseg

.org 0x0000

; initialize SRAM

ldi r20, ZERO_DISP

sts disp_table, r20

ldi r20, ONE_DISP

sts disp_table+1, r20

ldi r20, TWO_DISP

sts disp_table+2, r20

ldi r20, THREE_DISP

sts disp_table+3, r20

ldi r20, FOUR_DISP

sts disp_table+4, r20

ldi r20, FIVE_DISP

sts disp_table+5, r20

ldi r20, SIX_DISP
```

```
sts disp_table+6, r20
ldi r20, SEVEN_DISP
sts disp_table+7, r20
ldi r20, EIGHT_DISP
sts disp_table+8, r20
ldi r20, NINE_DISP
sts disp_table+9, r20
ldi r20, A_DISP
sts disp_table+10, r20
ldi r20, B_DISP
sts disp_table+11, r20
ldi r20, C_DISP
sts disp_table+12, r20
ldi r20, D_DISP
sts disp_table+13, r20
ldi r20, E_DISP
sts disp_table+14, r20
ldi r20, F_DISP
sts disp_table+15, r20
clr r20
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ SER_IN = 0
.equ SRCK = 1
.equ RCK = 2
.equ PUSH_BUTTON = 4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.DEF DISP_REG = r16
.DEF DISP_INS_REG = r17
.DEF PRESS_TIME_REG = r18
.DEF DEC_REG = r19
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; ATiny - Register Relations
; PB0 = SER_IN
; PB1 = SRCK
; PB2 = RCK
; PB4 = button input
; Configure PB0, PB1, and PB2 as output pins.
; Configure PB4 as input pins
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sbi DDRB,0
sbi DDRB,1
sbi DDRB,2
cbi DDRB,4

ldi ZL, low(disp_table) ;r30
ldi ZH, high(disp_table) ;r31
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; main method -- infinite loop to keep the controller responding to input
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
rcall reset_routine ; reset display state
main:
nop
sbis PINB, PUSH_BUTTON ; if button pushed, next line will execute
rcall button_pressed ; react to the button press
rjmp main
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
button_pressed:
clr PRESS_TIME_REG ; set initial state of counter to zero
rcall count_press
rjmp update
; count press sets certain register values associated with the length of button press
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
count_press:
rcall sample_delay
cpi PRESS_TIME_REG, 255
brlo increment_time
sbis PINB, PUSH_BUTTON ; if button is still pressed, execute next line
rjmp count_press ; rjmp to this method
ret
increment_time:
inc PRESS_TIME_REG
sbis PINB, PUSH_BUTTON
rjmp count_press
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; sample_delay of 80000 cycles
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
sample_delay:
        ldi   r23,20      ; r23 <-- Counter for outer loop
  my_d1: ldi   r24,24     ; r24 <-- Counter for level 2 loop
  my_d2: ldi   r25,41     ; r25 <-- Counter for inner loop
  my_d3: dec   r25
        nop             ; no operation
        brne  my_d3
        dec   r24
        brne  my_d2
        dec   r23
        brne  my_d1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; update the state based on the contents of PRESS_TIME_REG
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
update:
cpi PRESS_TIME_REG, 200
```

```
brsh reset_routine


cpi PRESS_TIME_REG, 100
brsh toggle_routine


rjmp move_routine
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
reset_routine:
nop
ldi ZL, 0x60
ld DISP_REG, Z
rcall toggle_dec_off
rjmp main

toggle_routine:
nop
cpi DEC_REG, 0x00
breq toggle_dec_on
cpi DEC_REG, 0x00
brne toggle_dec_off
rjmp main
move_routine:
nop
; branching:
; if dec -> if overflow: edit r30 (or) else dec
; if inc -> if overflow: edit r30 (or) else inc
cpi r30, 0x6F
breq pos_overflow
cpi r30, 0x60
breq neg_overflow
cpi DEC_REG, 0x01
```

```
brne pos_counter
cpi DEC_REG, 0x00
brne neg_counter
rjmp main ; saftey
neg_overflow:
cpi DEC_REG, 0x01
breq bottom_overflow
cpi DEC_REG, 0x01
brne pos_counter
rjmp main ; saftey
pos_overflow:
cpi DEC_REG, 0x00
breq reset_routine
cpi DEC_REG, 0x00
brne neg_counter
rjmp main
pos_counter:
ld DISP_REG, Z+
ld DISP_REG, Z
rcall display
rjmp main
neg_counter:
ld DISP_REG, -Z
ld DISP_REG, Z
rcall display
rjmp main
bottom_overflow:
ldi ZL, 0x6F
ld DISP_REG, Z
rcall display
rjmp main
```

```
toggle_dec_on:
ldi DEC_REG, 0x01
rcall display
rjmp main
toggle_dec_off:
ldi DEC_REG, 0x00
rcall display
rjmp main
; Display subroutine that prints to the LCD the associate hex value in DISP_REG
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
add_dp:
sbr DISP_REG, 8
rjmp dp_return

remove_dp:
cbr DISP_REG, 8
rjmp dp_return
display:
cpi DEC_REG, 0x00
brne add_dp
cpi DEC_REG, 0x01
brne remove_dp
dp_return:
; backup used registers on stack
push DISP_REG
push DISP_INS_REG
in DISP_INS_REG, SREG
push DISP_INS_REG
ldi DISP_INS_REG, 8
; loop --> test all 8 bits
loop:
```

```
rol DISP_REG ;rotate left through Carry
BRCS set_ser_in_1
; branch if Carry set
; put code here to set SER_IN to 0
cbi PORTB, SER_IN
rjmp end
set_ser_in_1:
; put code here to set SER_IN to 1
sbi PORTB, SER_IN
end:
; put code here to generate SRCK pulse
sbi PORTB, SRCK
nop
nop
cbi PORTB, SRCK
dec DISP_INS_REG
brne loop
; put code here to generate RCK pulse
sbi PORTB, RCK
nop
nop
cbi PORTB, RCK
; restore registers from stack
pop DISP_INS_REG
out SREG, DISP_INS_REG
pop DISP_INS_REG
pop DISP_REG
ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; end of program -- should never be reached due to main infinite loop
.exit
```