Aviles, Jean-Ralph
EEL3744
Section 1539
October 27th, 2015
Lab 5

Questions

1. List the XMEGA's USART registers used in your programs and briefly describe their functions.

   (a) CTRLA
   Determines whether interrupts will fire for specific USART events.

   (b) CTRLB
   Enables Tx and Rx pins for USART

   (c) CTRLC
   Sets the serial's parity, stop bits, character length, etc.

   (d) BAUDCTRLA
   Lower 8 bits of BSEL

   (e) BAUDCTRLAB
   Upper 8 bits of BSEL and BSCALE

2. What is the difference between serial and parallel communication?
   Serial communication transfers information – bits – one at a time, while a parallel communication transfers all bits at the same time.

3. List the number of bounces from part A of this lab. How long (in ms) is your delay routine for debouncing?
   I was getting about 2-3 bounces per flip. However I set my delay to 1ms because my finger touching the switch would cause more bounces for a longer period of time.

4. What is the maximum possible baud you can use **for asyncronous communication** if your board runs at 2Mhz? Support your answer with the values you would place in any special registers that are needed.
   Using double clock speed mode, our baud rate is bounded by the equation...

$$f_{baud} \leq \frac{f_{per}}{8}$$

   Plugging in our clock rate of 2MHz, we get a maximum baud rate of **250 kbps**.

Problems Encountered

I tried using GNU screen as my tty, but I couldn't get it configured correctly.

Future Work/Applications

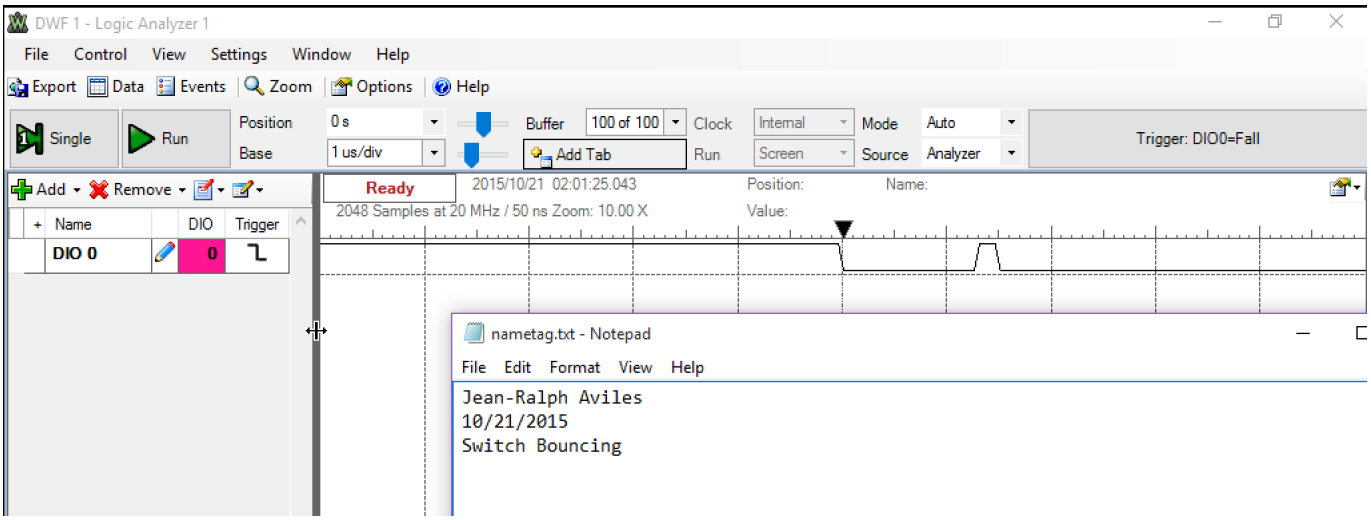
Serial can be used to communicate between our AVR and our laptops or even other microcontrollers. The possibilities with serial communication are endless.

Appendix

Part A

1. How many times did your interrupt service routine get called when you pulled the switch?
   About 2-3 times per flip.

Switch Bouncing



Part B

1. What pins on PORTD are used for USART0?
   (a) Pin1 - XCK0
   (b) Pin2 - RXD0
   (c) Pin3 - TXD0

2. USART Configurations
   CPU_Clock = 2MHz
   BSCALE = -7
   F_{baud} = 14.4kHz
   $BSEL = \frac{1}{2^{BSCALE}}\left(\frac{CPU\_CLOCK}{16 \cdot BAUD - 1}\right) = \frac{1}{2^{-7}}\left(\frac{2e6}{230400} - 1\right) = 983.\overline{11}$

Pseudocode/Flowcharts

Part A

Counter Pseudocode

```
def main():
    InitLeds()
    EnableInterrupts()
    while True:
        pass  # Do nothing forever

def Counter_Interrupt():
    sleep(1ms)
    if read_pin() is not 0:
        return
    LED = LED + 1
    clear_interrupt_flag()
    return
```

Part C

Out Char Pseudocode

```
def out_char(c):
  while USARTD0_STATUS & 0x10 is 0:
    pass
  output(c)
  return
```

Out String Pseudocode

```
def out_string(string):
  for c in string:
    out_char(c)
   return
```

In Char Pseudocode

```
def in_char():
  while USARTD0_STATUS & 0x80 is 0:
    pass
  c = read_char()
```

```
  return c
```

Part D

PartD Pseudocode

```
def main():
        x = 0xC3
        while True:
                output(led, x)
                x = ~x

def USARTD0_RXC_isr():
        x = read(serial)
        output(serial, x)

if __name__ == "__main__":
        main()
```

Programs

Part A

Counter Assembly

```
/*
* Jean-Ralph Aviles
* Lab 5 Program - Interrupt Counting
* 10/21/2015
* TA Khaled
*/

.include "ATxmega128A1Udef.inc"

.equ START = 0x8000  ; Start Address for IO
.equ SRAM_STARTA = 0x472000  ; Start Address for SRAM
.equ KEYTAB_WIDTH = 4  ; Width for Keypad Table
.equ KEYTAB_HEIGHT = 4  ; Height for Keypad Table

.org 0x0000
        rjmp CONFIGURE

.org PORTE_INT0_VECT
```

```
        rjmp COUNT_ISR

CONFIGURE:
.org 0x0200
        ; Initialize Stack
  ldi r16, 0xFF  ; Lower Byte of Stack Pointer
  out CPU_SPL, r16  ; Set lower byte
  ldi r16, 0x3F  ; Upper byte of Stack Pointer
  out CPU_SPH, r16  ; Set Upper byte

  ldi r16, 0xF0  ; PortF dir, upper bits output, lower input.
  sts PORTF_DIR, r16  ; ""
  ldi r16, 0x0F  ; Set mask for pins to configure 3:0
  sts PORTCFG_MPCMASK, r16  ; Load mask into MPCMASK register.
  ldi r16, 0x10  ; Configure input pins pull up resistors.
  sts PORTF_PIN0CTRL, r16  ; Pin0 pullup (Applies to 3:0).

  ; Mem mapped I/O
  ldi R16, 0b110111  ;set /WE, /RE, /CS0, /CS1 to Output
  sts PORTH_DIR, R16
  ldi R16, 0b110011  ;set /RE, /WE, /CS0, /CS1 to defaultH
  sts PORTH_OUTSET, R16
  ldi R16, 0xFF  ;set all PORTJ pins (D0-D7) to be outputs.
                 ;As requried
  sts PORTJ_DIR, R16  ;in the data sheet.
  ldi R16, 0xFF  ;set all PORTK pins (A0-A15) to be outputs.
  sts PORTK_DIR, R16  ;As required in the data sheet.
  ldi R16, 0x01  ;Store 0x01 in EBI_CTRL register to select
                 ;3 port EBI(H, J, K)  and SRAM ALE1 mode
  sts EBI_CTRL, R16

  ; Switches and Leds /CS0
  ldi r16, 0b0010101  ; Set CTRL A to 8K address size
  sts EBI_CS0_CTRLA, r16  ; (0b00101) and SRAM Mode (0b01)
              ; pg 335 8331
  ldi ZL, LOW(EBI_CS0_BASEADDR) ;  Load Z with BASEADDR
  ldi ZH, HIGH(EBI_CS0_BASEADDR) ;  We will load the upper 12
  ldi r16, byte2(START) ;  bits of the START address
  st Z+, r16  ; into BASEADDR register.
  ldi r16, byte3(START)
  st Z, r16

  ldi r16, byte3(START)  ;  Set third byte of X
  sts CPU_RAMPX, r16
  ldi XL, LOW(START)  ; Load X with START address
```

```
  ldi XH, HIGH(START)

  ldi r16, 0x0 ; Zero out the LED
  st X, r16

  ; Init Interupt 0 on falling edge on PORTE_7
  ldi r16, 0x01  ; Load 0x01 into r16 (Low Priority)
  sts PORTE_INTCTRL, r16  ; Set PORTE INT0 as Low Priority
  sts PORTE_DIRCLR, r16  ; Set PORTE_7 as Input
  ldi r16, 0x80  ; Load Bit7 mask into r16
  sts PORTE_OUT, r16  ; Default output on PORTE
  sts PORTE_INT0MASK, r16  ; Enable interrupts on Pin7
  ldi r16, 0x00  ; Load 0x00 into r16
  sts PORTE_INT1MASK, r16  ; Disable interrupt 1 on PORTE
  ldi r16, 0x02  ; Load Bit2 mask into r16
  sts PORTE_PIN7CTRL, r16  ; Set Pin7 trigger on falling edges
  ldi r16, 0x01  ; Load 0x01 into r16
  sts PMIC_CTRL, r16  ; Enable low level interrupts.
  sei  ; Interrupts Activated!
  rjmp MAIN

MAIN:
  ldi r16, 0x00  ; Initialize counter
DONE:
  sleep  ; Wait for an interrupt
  rjmp DONE

COUNT_ISR:
  push r17  ; Save r17
  push r16  ; Save r16
  ldi r16, 0x01  ; Load 0x01 into r16
  call DELAY  ; Delay for 1ms
  pop r16  ; Restore r16
  lds r17, PORTE_IN  ; Read PORTE after delay bounces
  andi r17, 0x80  ; Read Pin7
  tst r17  ; Compare Pin7 to 0x00
  brne COUNT_ISR_EXIT  ; False positive
  inc r16  ; Increment our counter r16
  st X, r16  ; Store the counter into the LEDS
COUNT_ISR_EXIT:
  ldi r17, 0x01  ; Load a 0x01 into r17
  sts PORTE_INTFLAGS, r17  ; Clear interrupt flag
  pop r17  ; Restore r17
  reti  ; Return from interrupt
```

```
DELAY:                                  ; Delays by r16 x 10ms
                                        ; 66*100 instructions=10ms
  push r16                              ; Push r16 onto the stack
  cpi r16, 0                            ; Compare counter
  breq DELAY_RET                        ; If counter is 0 return
DELAY_LOOP:
  push r16                              ; Counter onto the stack
  ldi r16, 66                           ; Load outer time loop
DELAY_OUTERLOOP:
  push r16                              ; Push outer loop to stack
  ldi r16, 100                          ; Load inner time loop
DELAY_INNERLOOP:
  dec r16                               ; Decrement inner counter
  brne DELAY_INNERLOOP
  pop r16                               ; Pop outer time loop
  dec r16                               ; Decrement outer counter
  brne DELAY_OUTERLOOP
  pop r16                               ; Pop counter off of stack
  dec r16                               ; Decrement Counter
  brne DELAY_LOOP                       ; If counter != 0 loop again
DELAY_RET:
  pop r16                               ; Pop r16 off
  ret                                   ; Return
```

Part C

```
/*
* Jean-Ralph Aviles
* Lab 5 Program - Asynchronuous Serial Prompt
* 10/22/2015
* TA Khaled
*/

.include "ATxmega128A1Udef.inc"

.equ START = 0x8000                     ; Start Address for IO
.equ SRAM_STARTA = 0x472000             ; Start Address for SRAM
.equ KEYTAB_WIDTH = 4                   ; Width for Keypad Table
.equ KEYTAB_HEIGHT = 4                  ; Height for Keypad Table
.equ BSCALE = -7                        ; BSCALE = -7
.equ BSEL = 983                         ; BSEL = 983

.org 0x0000
```

```
        rjmp CONFIGURE

.org PORTE_INT0_VECT
        rjmp COUNT_ISR

CONFIGURE:
.org 0x0200
        ; Initialize Stack
        ldi r16, 0xFF               ; Lower Byte of Stack Pointer
        out CPU_SPL, r16            ; Set lower byte
        ldi r16, 0x3F               ; Upper byte of Stack Pointer
        out CPU_SPH, r16            ; Set Upper byte

        ldi r16, 0xF0               ; PortF pin dir, upper bits output, lo
        sts PORTF_DIR, r16          ; ""
        ldi r16, 0x0F               ; Set mask for pins to configure 3:0
        sts PORTCFG_MPCMASK, r16    ; Load mask into MPCMASK register.
        ldi r16, 0x10               ; Configure input pins to have pull up
        sts PORTF_PIN0CTRL, r16     ; Pin0 pullup (Applies to 3:0 due to M

        ; Mem mapped I/O
        ldi R16, 0b110111           ;set /WE, /RE, /CS0, /CS1 to Output
        sts PORTH_DIR, R16
        ldi R16, 0b110011           ;set /RE, /WE, /CS0, /CS1 to default o
        sts PORTH_OUTSET, R16
        ldi R16, 0xFF               ;set all PORTJ pins (D0-D7) to be outp
        sts PORTJ_DIR, R16          ;in the data sheet.
        ldi R16, 0xFF               ;set all PORTK pins (A0-A15) to be out
        sts PORTK_DIR, R16          ;in the data sheet.
        ldi R16, 0x01               ;Store 0x01 in EBI_CTRL register to se
                                    ;mode and SRAM ALE1 mode
        sts EBI_CTRL, R16

        ; Switches and Leds /CS0
        ldi r16, 0b0010101          ; Set CTRL A to 8K address size
        sts EBI_CS0_CTRLA, r16      ; (0b00101) and SRAM Mode (0b01)
                                                    ; pg 335 8331
        ldi ZL, LOW(EBI_CS0_BASEADDR) ; Load Z with BASEADDR
        ldi ZH, HIGH(EBI_CS0_BASEADDR) ; We will load the upper 12
        ldi r16, byte2(START)       ; bits of the START address
        st Z+, r16                  ; into BASEADDR register.
        ldi r16, byte3(START)
        st Z, r16

        ldi r16, byte3(START)       ; Set third byte of X
```

```
        sts CPU_RAMPX, r16
        ldi XL, LOW(START)          ; Load X with START address
        ldi XH, HIGH(START)

        ldi r16, 0x0                ; Zero out the LED
        st X, r16

        call INIT_USART

        ldi r16, 0x01               ; Load 0x01 into r16
        sts PMIC_CTRL, r16          ; Enable low level interrupts.
        sei                         ; Interrupts Activated!
        rjmp MAIN

.equ CR = 0x0D
.equ LF = 0x0A
.equ TAB = '\t'
.equ ESC = 0x1B
PROMPT:
        .DB "JR's favorite:", CR, LF,
  .DB "1.", TAB, "Food", CR, LF,
  .DB "2.", TAB, "Actor", CR, LF,
  .DB "3.", TAB, "Book", CR, LF,
  .DB "4.", TAB, "Pizza Topping", CR, LF,
  .DB "5.", TAB, "Ice cream", CR, LF,
  .DB "6.", TAB, "Refresh", CR, LF,
  .DB "ESC", TAB, "exit", CR, LF, 0x00
ANS1:
        .DB "JR's favorite food is Liver", CR, LF, 0x00
ANS2:
        .DB "JR's favorite actor is Nicholas Cage", CR, LF, 0x00
ANS3:
        .DB "JR's favorite book is Justin Bieber: His World", CR, LF, 0x0
ANS4:
        .DB "JR's favorite topping is Sausage", CR, LF, 0x00
ANS5:
        .DB "JR's favorite ice cream is Horse Raddish", CR, LF, 0x00
ESCMSG:
  .DB "Done", CR, LF, 0x00

MAIN:
        ldi ZL, byte1(PROMPT<<1)    ; Load ZL
        ldi ZH, byte2(PROMPT<<1)    ; Load ZH
        ldi r16, byte3(PROMPT<<1)   ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
```

```
        rcall OUT_STRING            ; Output prompt
LOOP2:
        rcall IN_CHAR               ; Wait for input
        cpi r16, '1'                ; Compare r16 with '1'
        breq PRINT1                 ; If equal print ans 1
        cpi r16, '2'                ; Compare r16 with '2'
        breq PRINT2                 ; If equal print ans 2
        cpi r16, '3'                ; Compare r16 with '3'
        breq PRINT3                 ; If equal print ans 3
        cpi r16, '4'                ; Compare r16 with '4'
        breq PRINT4                 ; If equal print ans 4
        cpi r16, '5'                ; Compare r16 with '5'
        breq PRINT5                 ; If equal prin ans 5
        cpi r16, '6'                ; Compare r16 with '6'
        breq REFRESH                ; If equal refresh
        cpi r16, ESC                ; Compare r16 with ESC
        breq EXIT                   ; If equal refresh
        rjmp LOOP2                  ; Unrecognized, loop
EXIT:
        ldi ZL, byte1(ESCMSG<<1)    ; Load ZL
        ldi ZH, byte2(ESCMSG<<1)    ; Load ZH
        ldi r16, byte3(ESCMSG<<1)   ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
        rcall OUT_STRING            ; Output answer
        ldi r16, 0x03
        ldi r17, 0x00
EXIT2:
        st X, r17
        inc r17
        rcall DELAY
        rjmp EXIT2
PRINT1:
        ldi ZL, byte1(ANS1<<1)      ; Load ZL
        ldi ZH, byte2(ANS1<<1)      ; Load ZH
        ldi r16, byte3(ANS1<<1)     ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
        rcall OUT_STRING            ; Output answer
        rjmp MAIN                   ; Wait for another input
PRINT2:
        ldi ZL, byte1(ANS2<<1)      ; Load ZL
        ldi ZH, byte2(ANS2<<1)      ; Load ZH
        ldi r16, byte3(ANS2<<1)     ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
        rcall OUT_STRING            ; Output answer
        rjmp MAIN                   ; Wait for another input
```

```
PRINT3:
        ldi ZL, byte1(ANS3<<1)      ; Load ZL
        ldi ZH, byte2(ANS3<<1)      ; Load ZH
        ldi r16, byte3(ANS3<<1)     ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
        rcall OUT_STRING            ; Output answer
        rjmp MAIN                   ; Wait for another input
PRINT4:
        ldi ZL, byte1(ANS4<<1)      ; Load ZL
        ldi ZH, byte2(ANS4<<1)      ; Load ZH
        ldi r16, byte3(ANS4<<1)     ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
        rcall OUT_STRING            ; Output answer
        rjmp MAIN                   ; Wait for another input
PRINT5:
        ldi ZL, byte1(ANS5<<1)      ; Load ZL
        ldi ZH, byte2(ANS5<<1)      ; Load ZH
        ldi r16, byte3(ANS5<<1)     ; Load r16
        sts CPU_RAMPZ, r16          ; Load RAMPZ
        rcall OUT_STRING            ; Output answer
        rjmp MAIN                   ; Wait for another input
REFRESH:
        rjmp MAIN

OUT_CHAR:
        push r17                    ; Save r17
TX_POLL:
        lds r17, USARTD0_STATUS     ; Get USART status
        sbrs r17, 5                 ; Break when DREIF is empty
        rjmp TX_POLL                ; Loop again
        sts USARTD0_DATA, r16       ; Output r16
        pop r17                     ; Restore r17
        ret                         ; Return

OUT_STRING:
        push r16                    ; Save r16
        push ZL                     ; Save ZL
        push ZH                     ; Save ZH
        lds r16, CPU_RAMPZ          ; Save RAMPZ
        push r16                    ; Save RAMPZ
OUT_STRING_A:
        lpm r16, Z+                  ; Load r16 and inc Z
        tst r16                     ; Compare r16 with '\0'
        breq OUT_STRING_EXIT        ; At end of string, exit
        rcall OUT_CHAR              ; Call OUT_CHAR to output
```

```
        rjmp OUT_STRING_A           ; Loop
OUT_STRING_EXIT:
        pop r16                     ; Restore RAMPZ
        sts CPU_RAMPZ, r16          ; Restore RAMPZ
        pop ZH                      ; Restore ZH
        pop ZL                      ; Restore ZL
        pop r16                     ; Restore r16
        ret

IN_CHAR:
        lds r16, USARTD0_STATUS     ; Get USART status
        sbrs r16, 7                 ; Break if RXCIF is on
        rjmp IN_CHAR                ; Else jmp to IN_CHAR
        lds r16, USARTD0_DATA       ; Read Data
        ret                         ; Return

INIT_USART:
        ; GPIO
        push r16                    ; Save r16
        ldi r16, 0x08               ; Load r16 with Bit3 (Tx)
        sts PORTD_DIRSET, r16       ; Set Tx to output
        sts PORTD_OUTSET, r16       ; Set Tx default output
        ldi r16, 0x04               ; Load r16 with Bit2 (Rx)
        sts PORTD_DIRCLR, r16       ; Set Rx to input
        ldi r16, 0x0A               ; Load Bit3,1 into r16
        sts PORTQ_DIRSET, r16       ; Connect PORTD serial pins
        sts PORTQ_OUTCLR, r16       ; to USB
        ; USART
        ldi r16, 0x18               ; Load 0x18 into r16
        sts USARTD0_CTRLB, r16      ; Enable Rx and Tx on PortD
        ldi r16, 0x03               ; Configure USART for Async,
        sts USARTD0_CTRLC, r16      ; No Parity, 1 Stop Bit, 8bits
        ldi r16, (BSEL & 0xFF)      ; Load lower 8 bits of BSEL
        sts USARTD0_BAUDCTRLA, r16 ; Set lower 8 bits of Baud
        ldi r16, ((BSCALE << 4) & 0xF0) | ((BSEL>>8) & 0x0F)
        sts USARTD0_BAUDCTRLB, r16 ; Configure BSCALE and upper BSEL
        pop r16                     ; Restore r16
        ret                         ; Return

COUNT_ISR:
        push r17                  ; Save r17
        push r16                  ; Save r16
        ldi r16, 0x01             ; Load 0x01 into r16
        call DELAY                ; Delay for 1ms
        pop r16                   ; Restore r16
```

```
        lds r17, PORTE_IN          ; Read PORTE after delay bounces
        andi r17, 0x80             ; Read Pin7
        tst r17                    ; Compare Pin7 to 0x00
        brne COUNT_ISR_EXIT        ; False positive
        inc r16                    ; Increment our counter r16
        st X, r16                  ; Store the counter into the LEDS
COUNT_ISR_EXIT:
        ldi r17, 0x01              ; Load a 0x01 into r17
        sts PORTE_INTFLAGS, r17    ; Clear interrupt flag
        pop r17                    ; Restore r17
        reti                       ; Return from interrupt

DELAY:                             ; Delays by r16 x 10ms
                                   ; 66*100 instructions=10ms
  push r16                         ; Push r16 onto the stack
  cpi r16, 0                       ; Compare counter
  breq DELAY_RET                   ; If counter is 0 return
DELAY_LOOP:
  push r16                         ; Counter onto the stack
  ldi r16, 66                      ; Load outer time loop
DELAY_OUTERLOOP:
  push r16                         ; Push outer loop to stack
  ldi r16, 100                     ; Load inner time loop
DELAY_INNERLOOP:
  dec r16                          ; Decrement inner counter
  brne DELAY_INNERLOOP
  pop r16                          ; Pop outer time loop
  dec r16                          ; Decrement outer counter
  brne DELAY_OUTERLOOP
  pop r16                          ; Pop counter off of stack
  dec r16                          ; Decrement Counter
  brne DELAY_LOOP                  ; If counter != 0 loop again
DELAY_RET:
  pop r16                          ; Pop r16 off
  ret                              ; Return
```

Part D

```
/*
* Jean-Ralph Aviles
* Lab 5 Program - Asynchronuous Non blocking Serial Prompt
* 10/22/2015
* TA Khaled
*/
```

```
.include "ATxmega128A1Udef.inc"

.equ START = 0x8000              ; Start Address for IO
.equ SRAM_STARTA = 0x472000      ; Start Address for SRAM
.equ KEYTAB_WIDTH = 4            ; Width for Keypad Table
.equ KEYTAB_HEIGHT = 4           ; Height for Keypad Table
.equ BSCALE = -7                 ; BSCALE = -7
.equ BSEL = 983                  ; BSEL = 983

.org 0x0000
        rjmp CONFIGURE

.org USARTD0_RXC_vect
        rjmp USARTD0_RXC_isr

CONFIGURE:
.org 0x0200
        ; Initialize Stack
        ldi r16, 0xFF            ; Lower Byte of Stack Pointer
        out CPU_SPL, r16         ; Set lower byte
        ldi r16, 0x3F            ; Upper byte of Stack Pointer
        out CPU_SPH, r16         ; Set Upper byte

        ldi r16, 0xF0            ; PortF pin dir, upper bits output, lo
        sts PORTF_DIR, r16       ; ""
        ldi r16, 0x0F            ; Set mask for pins to configure 3:0
        sts PORTCFG_MPCMASK, r16 ; Load mask into MPCMASK register.
        ldi r16, 0x10            ; Configure input pins to have pull up
        sts PORTF_PIN0CTRL, r16  ; Pin0 pullup (Applies to 3:0 due to M

        ; Mem mapped I/O
        ldi R16, 0b110111        ;set /WE, /RE, /CS0, /CS1 to Output
        sts PORTH_DIR, R16
        ldi R16, 0b110011        ;set /RE, /WE, /CS0, /CS1 to default o
        sts PORTH_OUTSET, R16
        ldi R16, 0xFF            ;set all PORTJ pins (D0-D7) to be outp
        sts PORTJ_DIR, R16       ;in the data sheet.
        ldi R16, 0xFF            ;set all PORTK pins (A0-A15) to be out
        sts PORTK_DIR, R16       ;in the data sheet.
        ldi R16, 0x01            ;Store 0x01 in EBI_CTRL register to se
                                 ;mode and SRAM ALE1 mode
        sts EBI_CTRL, R16

        ; Switches and Leds /CS0
```

```
        ldi r16, 0b0010101          ; Set CTRL A to 8K address size
        sts EBI_CS0_CTRLA, r16      ; (0b00101) and SRAM Mode (0b01)
                                                    ; pg 335 8331
        ldi ZL, LOW(EBI_CS0_BASEADDR) ; Load Z with BASEADDR
        ldi ZH, HIGH(EBI_CS0_BASEADDR) ; We will load the upper 12
        ldi r16, byte2(START)       ; bits of the START address
        st Z+, r16                  ; into BASEADDR register.
        ldi r16, byte3(START)
        st Z, r16

        ldi r16, byte3(START)       ; Set third byte of X
        sts CPU_RAMPX, r16
        ldi XL, LOW(START)          ; Load X with START address
        ldi XH, HIGH(START)

        ldi r16, 0x0                ; Zero out the LED
        st X, r16

        call INIT_USART
        call INIT_USART_INT

        ldi r16, 0x01               ; Load 0x01 into r16
        sts PMIC_CTRL, r16          ; Enable low level interrupts.
        sei                         ; Interrupts Activated!
        rjmp MAIN

MAIN:
        ldi r16, 0x04           ; Load r16 with 0x04
        ldi r17, 0xC3           ; Load r17 with 0x0F
LOOP:
        st X, r17               ; Store r17 into LEDs
        com r17                 ; Flip r17 bits
        rcall DELAY             ; Delay for 40ms
        rjmp LOOP               ; Loop

USARTD0_RXC_isr:
        push r16                ; Save r16
        rcall IN_CHAR           ; Read character
        rcall OUT_CHAR          ; Echo character, will clear RXCIF
        pop r16                 ; Restore r16
        reti                    ; Resume execution

OUT_CHAR:
        push r17                    ; Save r17
TX_POLL:
```

```
        lds r17, USARTD0_STATUS     ; Get USART status
        sbrs r17, 5                 ; Break when DREIF is empty
        rjmp TX_POLL                ; Loop again
        sts USARTD0_DATA, r16       ; Output r16
        pop r17                     ; Restore r17
        ret                         ; Return

OUT_STRING:
        push r16                    ; Save r16
        push ZL                     ; Save ZL
        push ZH                     ; Save ZH
        lds r16, CPU_RAMPZ          ; Save RAMPZ
        push r16                    ; Save RAMPZ
OUT_STRING_A:
        lpm r16, Z+                 ; Load r16 and inc Z
        tst r16                     ; Compare r16 with '\0'
        breq OUT_STRING_EXIT        ; At end of string, exit
        rcall OUT_CHAR              ; Call OUT_CHAR to output
        rjmp OUT_STRING_A           ; Loop
OUT_STRING_EXIT:
        pop r16                     ; Restore RAMPZ
        sts CPU_RAMPZ, r16          ; Restore RAMPZ
        pop ZH                      ; Restore ZH
        pop ZL                      ; Restore ZL
        pop r16                     ; Restore r16
        ret

IN_CHAR:
        lds r16, USARTD0_STATUS     ; Get USART status
        sbrs r16, 7                 ; Break if RXCIF is on
        rjmp IN_CHAR                ; Else jmp to IN_CHAR
        lds r16, USARTD0_DATA       ; Read Data
        ret                         ; Return

INIT_USART:
        ; GPIO
        push r16                    ; Save r16
        ldi r16, 0x08               ; Load r16 with Bit3 (Tx)
        sts PORTD_DIRSET, r16       ; Set Tx to output
        sts PORTD_OUTSET, r16       ; Set Tx default output
        ldi r16, 0x04               ; Load r16 with Bit2 (Rx)
        sts PORTD_DIRCLR, r16       ; Set Rx to input
        ldi r16, 0x0A               ; Load Bit3,1 into r16
        sts PORTQ_DIRSET, r16       ; Connect PORTD serial pins
        sts PORTQ_OUTCLR, r16       ; to USB
```

```
        ; USART
        ldi r16, 0x18               ; Load 0x18 into r16
        sts USARTD0_CTRLB, r16      ; Enable Rx and Tx on PortD
        ldi r16, 0x03               ; Configure USART for Async,
        sts USARTD0_CTRLC, r16      ; No Parity, 1 Stop Bit, 8bits
        ldi r16, (BSEL & 0xFF)      ; Load lower 8 bits of BSEL
        sts USARTD0_BAUDCTRLA, r16 ; Set lower 8 bits of BSEL
        ldi r16, ((BSCALE << 4) & 0xF0) | ((BSEL>>8) & 0x0F)
        sts USARTD0_BAUDCTRLB, r16 ; Configure BSCALE and upper BSEL
        pop r16                     ; Restore r16
        ret                         ; Return

INIT_USART_INT:
        push r16                    ; Save r16
        ldi r16, 0x10               ; Load LO-level interupt mask
        sts USARTD0_CTRLA, r16      ; Set LO-level interupt on recieve
        pop r16
        ret

COUNT_ISR:
        push r17                  ; Save r17
        push r16                  ; Save r16
        ldi r16, 0x01             ; Load 0x01 into r16
        call DELAY                ; Delay for 1ms
        pop r16                   ; Restore r16
        lds r17, PORTE_IN         ; Read PORTE after delay bounces
        andi r17, 0x80            ; Read Pin7
        tst r17                   ; Compare Pin7 to 0x00
        brne COUNT_ISR_EXIT       ; False positive
        inc r16                   ; Increment our counter r16
        st X, r16                 ; Store the counter into the LEDS
COUNT_ISR_EXIT:
        ldi r17, 0x01             ; Load a 0x01 into r17
        sts PORTE_INTFLAGS, r17   ; Clear interrupt flag
        pop r17                   ; Restore r17
        reti                      ; Return from interrupt

DELAY:                            ; Delays by r16 x 10ms
                                  ; 66*100 instructions=10ms
  push r16                        ; Push r16 onto the stack
  cpi r16, 0                      ; Compare counter
  breq DELAY_RET                  ; If counter is 0 return
DELAY_LOOP:
  push r16                        ; Counter onto the stack
  ldi r16, 66                     ; Load outer time loop
```

```
DELAY_OUTERLOOP:
  push r16                          ; Push outer loop to stack
  ldi r16, 100                      ; Load inner time loop
DELAY_INNERLOOP:
  dec r16                           ; Decrement inner counter
  brne DELAY_INNERLOOP
  pop r16                           ; Pop outer time loop
  dec r16                           ; Decrement outer counter
  brne DELAY_OUTERLOOP
  pop r16                           ; Pop counter off of stack
  dec r16                           ; Decrement Counter
  brne DELAY_LOOP                   ; If counter != 0 loop again
DELAY_RET:
  pop r16                           ; Pop r16 off
  ret                               ; Return
```