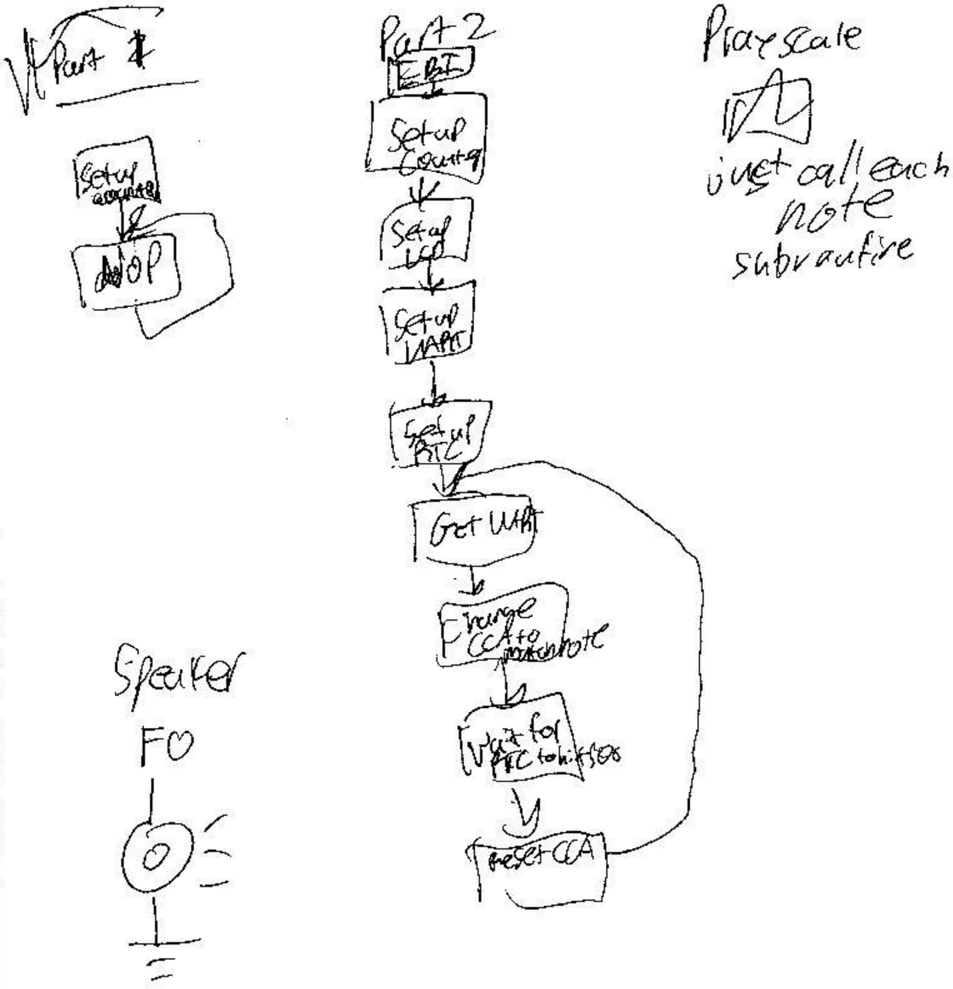
Branodn Pollack

Lab7 Report



Part 1

```
/* A collection of inits for the EBI and stack init at the bottom */
.equ IOPORT = 0x5000
.equ SRAMPORT = 0x370000
.equ LCDPORT_COM = 0x4000
.equ LCDPORT_DAT = 0x4001

.macro TRIPORT_ALE_ONE_INIT



	ldi R16, 0b01110111
	sts PORTH_DIR, R16 //set port pins as outputs for RE and ALE and WE CS1 and CS0

	ldi R16, 0b01110011
	sts PORTH_OUT, R16 //WE and RE is active low so it must be set

	ldi R16, 0xFF
	sts PORTJ_DIR, R16 //set datalines as outputs (manual says so)
	sts PORTK_DIR, R16 //set address lines as outputs

	ldi R16, 0x01
	sts EBI_CTRL, R16 //turn on 3 port SRAM ALE1 EBI
.endmacro



.macro CS0_INIT
	ldi ZH, HIGH(EBI_CS0_BASEADDR) //all the set up for CS0, since EBI won't work without it
	ldi ZL, LOW(EBI_CS0_BASEADDR)

	ldi R16, ((IOPORT>>8) & 0xF0)
	st Z+, R16

	ldi R16, ((IOPORT>>16) & 0xFF)
	st Z, R16

	ldi R16, 0x11
	sts EBI_CS0_CTRLA, R16
.endmacro

.macro CS1_INIT
	ldi ZH, HIGH(EBI_CS1_BASEADDR) //set up CS1 for the SRAM
	ldi ZL, LOW(EBI_CS1_BASEADDR)

	ldi R16, ((SRAMPORT>>8) & 0xF0)
	st Z+, R16

	ldi R16, ((SRAMPORT>>16) & 0xFF)
	st Z, R16

	ldi R16, 0b00011101
	sts EBI_CS1_CTRLA, R16
.endmacro

.macro CS2_INIT
	ldi ZH, HIGH(EBI_CS2_BASEADDR) //set up CS1 for the SRAM
```

```
    ldi ZL, LOW(EBI_CS2_BASEADDR)

    ldi R16, ((LCDPORT_COM>>8) & 0xF0)
    st Z+, R16

    ldi R16, ((LCDPORT_COM>>16) & 0xFF)
    st Z, R16

    ldi R16, 0x01
    sts EBI_CS2_CTRLA, R16
.endmacro


.macro STACK_INIT
    ldi R16, 0xFF
    out CPU_SPL, R16
    ldi R16, 0x3F
    out CPU_SPH, R16 //init stack pointer
.endmacro

.macro LCD_INIT
        ldi XH, high(LCDPORT_COM)
        ldi XL, low(LCDPORT_COM)

        call LCD_BF_WAIT

        ldi R16, 0b00111000 // two lines, bigger font, 8 bits
        st X, R16
        call LCD_BF_WAIT

        ldi R16, 0b00001111 // display on cursor on curor blink
        st X, R16

        call LCD_BF_WAIT

        ldi R16, 0b00000001 // clear disp
        st X, R16

        call LCD_BF_WAIT

        ldi R16, 0b00000011 // cursor home
        st X, R16
.endmacro

.macro ADC_8bit_INIT
        ldi R16, 1
        sts PORTA_DIRCLR, R16

        sts ADCA_CTRLA, R16 //enable the ADC

        ldi R16, 0b00011100 //turn on free run and set the conversion mode to 8 bit signed
        sts ADCA_CTRLB, R16

        ldi R16, 0b00010000 //set teh reference to VCC/1.6 ~= 2.0625
```

```
			sts ADCA_REFCTRL, R16 //which the 5 volts on the POT is divided by the board to fit the constraint of


			ldi R16, 0b00000011 // set the prescaler to div32 (2MHZ/32 = 62.5 KHZ)
			sts ADCA_PRESCALER, R16
  .endmacro


  .macro ADC_CH0_INIT
			ldi R16, 0b10000001
			sts ADCA_CH0_CTRL, R16 //start taking readings on CH0
  .endmacro
/*
 * Lab7_part1_BRP.asm
 *
 *  Created: 4/6/2013 4:38:47 PM
 *   Author: Brandon
 */



.include "Atxmega128a1udef.inc"


.org 0
rjmp main


.org 0x100
main:
		ldi R16, 0x01
		sts PORTF_DIRSET, R16 //make the pin an output
	ldi R16, 0b00010001 //make this FRQ mode turn on CCA
	sts TCF0_CTRLB, R16

	ldi ZL, low(TCF0_CCA)
	ldi ZH, high(TCF0_CCA) //load Z so we can write our compare value

	ldi R16, 0x38
	st Z+, R16

	ldi R16, 0x02
	st Z, R16 //set the compare A (also the period reg since we are in FRQ mode)

	ldi R16, 0x01
	sts TCF0_CTRLA, R16 //turn on the timer/counter and use 2e6 hz

	done: rjmp done //loop forever a frequency should play
/* Brandon Pollack
*  HW4
*  SCI Subroutines
*/


.macro SCI_C_INIT

	.equ BSEL = 51
	.equ BSCL = -2
	ldi R16, 0x18
	sts USARTC0_CTRLB, R16			;this buts a one in RXEN and TXEN, enabling transmission and receive


	ldi R16, 0x03
	sts USARTC0_CTRLC, R16			;No parity, 8 bit data, a single stop bit


	ldi R16, BSEL
	sts USARTC0_BAUDCTRLA, R16		;setting baud to 9600 HZ involves some calculation from the manual


	ldi R16, ((BSCL << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

	sts USARTC0_BAUDCTRLB, R16		;set the scale to -2 as per the formula to get 9600 HZ from the Fper and BSCL, upper 4 bits of BSEL stay the same


	; now begins the set up of the PORTC to output and input serial


	ldi R16, 0x08
	sts PORTC_DIR, R16
	sts PORTC_OUT, R16 ; set the direction of the TX line as out and default as 1 as per docs
.endmacro
/*SPI_C_INIT:
	.macro
	.equ BSEL = 51
	.equ BSCL = -2



	ldi R16, 0x18
	sts USARTC0_CTRLB, R16			;this buts a one in RXEN and TXEN, enabling
transmission and receive


	ldi R16, 0b01000011

	sts USARTC0_CTRLC, R16			;No parity, 8 bit data, a single stop bit,
synchronous transmission


	ldi R16, BSEL

	sts USARTC0_BAUDCTRLA, R16		;setting baud to 9600 HZ involves some
calculation from the manual


	ldi R16, ((BSCL << 4) & 0xF0) | ((BSel >> 8) & 0x0F)


	sts USARTC0_BAUDCTRLB, R16		;set the scale to -2 as per the formula to get 9600
HZ from the Fper and BSCL, upper 4 bits of BSEL stay the same


	; now begins the set up of the PORTC to output and input serial


	ldi R16, 0x08

	sts PORTC_DIR, R16

	sts PORTC_OUT, R16 ; set the direction of the TX line as out and default as 1 as per docs

	.endmacro

	*/


OUT_CHAR:

	.org 0x200

	push R17 ;save this value


	isdatasent:

		lds R17, USARTC0_STATUS
```

```
        sbrs R17, 6 ;poll TXIF in status register, if it is clear we are not done
        rjmp isdatasent

    sts USARTC0_DATA, R16

    pop R17

    ret

OUT_STRING:
    push R16 ;I chose to use z so this sub works for program or data memory (remember to shift left if program memory)

    beginwritingstring:
        ld R16, Z+ ;at the end of this sub, z will point to one address past the end of the string
        breq donewritingstring
        call OUT_CHAR
        rjmp beginwritingstring

    donewritingstring:
        pop R16
        ret

IN_CHAR:
    push R17

    isdatarecieved:
        lds R17, USARTC0_STATUS
        sbrs R17, 7
        rjmp isdatarecieved

    lds R16, USARTC0_DATA

    pop R17
    ret

IN_STRING:  ;be sure to have X point where you want this data to go
    push R16

    beginreadingstring:
        call IN_CHAR ;puts the character in R16
        cpi R16, 0
        breq donereadingstring
        st X+, R16
        rjmp beginreadingstring

    donereadingstring:
    pop R16
    ret
```

Part 2

```
/* A collection of inits for the EBI and stack init at the bottom */
.equ IOPORT = 0x5000
.equ SRAMPORT = 0x370000
.equ LCDPORT_COM = 0x4000
.equ LCDPORT_DAT = 0x4001
```

```
.macro TRIPORT_ALE_ONE_INIT


        ldi R16, 0b01110111
        sts PORTH_DIR, R16 //set port pins as outputs for RE and ALE and WE CS1 and CS0

        ldi R16, 0b01110011
        sts PORTH_OUT, R16 //WE and RE is active low so it must be set

        ldi R16, 0xFF
        sts PORTJ_DIR, R16 //set datalines as outputs (manual says so)
        sts PORTK_DIR, R16 //set address lines as outputs

        ldi R16, 0x01
        sts EBI_CTRL, R16 //turn on 3 port SRAM ALE1 EBI
.endmacro



.macro CS0_INIT
        ldi ZH, HIGH(EBI_CS0_BASEADDR) //all the set up for CS0, since EBI won't work without it
        ldi ZL, LOW(EBI_CS0_BASEADDR)

        ldi R16, ((IOPORT>>8) & 0xF0)
        st Z+, R16

        ldi R16, ((IOPORT>>16) & 0xFF)
        st Z, R16
        ldi R16, 0x11
        sts EBI_CS0_CTRLA, R16
.endmacro

.macro CS1_INIT
        ldi ZH, HIGH(EBI_CS1_BASEADDR) //set up CS1 for the SRAM
        ldi ZL, LOW(EBI_CS1_BASEADDR)

        ldi R16, ((SRAMPORT>>8) & 0xF0)
        st Z+, R16

        ldi R16, ((SRAMPORT>>16) & 0xFF)
        st Z, R16

        ldi R16, 0b00011101
        sts EBI_CS1_CTRLA, R16
.endmacro

.macro CS2_INIT
        ldi ZH, HIGH(EBI_CS2_BASEADDR) //set up CS1 for the SRAM
        ldi ZL, LOW(EBI_CS2_BASEADDR)

        ldi R16, ((LCDPORT_COM>>8) & 0xF0)
        st Z+, R16

        ldi R16, ((LCDPORT_COM>>16) & 0xFF)
        st Z, R16
```

```
    ldi R16, 0x01
    sts EBI_CS2_CTRLA, R16
.endmacro


.macro STACK_INIT
    ldi R16, 0xFF
    out CPU_SPL, R16
    ldi R16, 0x3F
    out CPU_SPH, R16 //init stack pointer
.endmacro

.macro LCD_INIT
        ldi XH, high(LCDPORT_COM)
        ldi XL, low(LCDPORT_COM)

        call LCD_BF_WAIT

        ldi R16, 0b00111000 // two lines, bigger font, 8 bits
        st X, R16

        call LCD_BF_WAIT

        ldi R16, 0b00001111 // display on cursor on curor blink
        st X, R16

        call LCD_BF_WAIT
```

```
                ldi R16, 0b00000001 // clear disp
                st X, R16

                call LCD_BF_WAIT

                ldi R16, 0b00000011 // cursor home
                st X, R16
  .endmacro

  .macro ADC_8bit_INIT
                ldi R16, 1
                sts PORTA_DIRCLR, R16

                sts ADCA_CTRLA, R16 //enable the ADC

                ldi R16, 0b00011100 //turn on free run and set the conversion mode to 8 bit signed
                sts ADCA_CTRLB, R16

                ldi R16, 0b00010000 //set teh reference to VCC/1.6 ~= 2.0625
                sts ADCA_REFCTRL, R16 //which the 5 volts on the POT is divided by the board to fit the constraint of

                ldi R16, 0b00000011 // set the prescaler to div32 (2MHZ/32 = 62.5 KHZ)
                sts ADCA_PRESCALER, R16
  .endmacro

  .macro ADC_CH0_INIT
```

```
                ldi R16, 0b10000001
                sts ADCA_CH0_CTRL, R16 //start taking readings on CH0
 .endmacro
/*
 * Lab7_part2.asm
 *
 *  Created: 4/6/2013 5:51:08 PM
 *   Author: Brandon
 */

.include "Atxmega128a1udef.inc"
.include "EBI_INITS.asm"
.include "USART_FUNCTIONS.asm"

.org 0
rjmp main

.org USARTC0_RXC_vect
        jmp USARTC0_RXC_ISR
.org RTC_COMP_vect
        jmp RTC_COMP_ISR

.org 0x100
main:
        /*ldi R16, CLK_STATUS
        sbrs R16, 1
        rjmp main
        ldi R16, 2
```

```
sts CLK_CTRL, R16 //switches the clock, dear god please don't explode
*/

TRIPORT_ALE_ONE_INIT
CS0_INIT
CS1_INIT
CS2_INIT
LCD_INIT
SCI_C_INIT //19200 baud
ldi R16, 0x10
sts USARTC0_CTRLA, R16
STACK_INIT


ldi R16, 0x01
sts PORTF_DIRSET, R16 //make the pin an output

ldi R16, 0b00010001 //make this FRQ mode turn on CCA
sts TCF0_CTRLB, R16

ldi R16, 0x01
sts TCF0_CTRLA, R16 //turn on the timer/counter and use 2e6 hz

ldi R16, (1<<2)
sts RTC_INTCTRL, R16 //turn on the RTC interupt compare for 500 ms

ldi ZH, high(RTC_COMP)
ldi ZL, low(RTC_COMP)
```

```
	ldi R16, low(500)
	st Z+, R16
	ldi R16, high(500)
	st Z, R16
	ldi R16, 0xFF
	sts RTC_PER, R16
	sts RTC_PER+1, R16 // make the period FFFF
	ldi R16, 1 | 0b010 << 1
	sts CLK_RTCCTRL, R16

	ldi R16, 1
	sts PMIC_CTRL, R16
	sei
	clr R16

mainloop:
	cpi R16, '1'
	breq b5j
	cpi R16, '2'
	breq c6j
	cpi R16, '3'
	breq c6shj
	cpi R16, '4'
	breq d6j
	cpi R16, '5'
	breq d6shj
	cpi R16, '6'
	breq e6j
```

```
        cpi R16, '7'
        breq f6j
        cpi R16, '8'
        breq f6shj
        cpi R16, '9'
        breq g6j
        cpi R16, 'A'
        breq A6j
        cpi R16, 'B'
        breq a6shj
        cpi R16, 'C'
        breq b6j
        cpi R16, 'D'
        breq c7j
        cpi R16, '*'
        breq ascendingscalej
        cpi R16, '#'
        breq descendingscalej
        rjmp mainloop
b5j:
        call b5
        rjmp mainloop
c6j:
        call c6
        rjmp mainloop
c6shj:
        call c6sh
        rjmp mainloop
```

```
d6j:
        call d6
        rjmp mainloop
d6shj:
        call d6sh
        rjmp mainloop
e6j:
        call e6
        rjmp mainloop
f6j:
        call f6
        rjmp mainloop
f6shj:
        call f6sh
        rjmp mainloop
g6j:
        call g6
        rjmp mainloop
G6shj:
        call g6sh
        rjmp mainloop
A6j:
        call A6
        rjmp mainloop
A6shj:
        call A6sh
        rjmp mainloop
B6j:
	call b6
	rjmp mainloop
C7j:
	call C7
	rjmp mainloop
ascendingscalej:
	call ascendingscale
	rjmp mainloop
descendingscalej:
	call descendingescale
	rjmp mainloop



	//check the R16 register for note value, when you have it play it through subroutines, and make scales call those as well
	//make each note subroutine reset and start RTC counting to 500, then when that ends, ISR it to stop, reset its value, and stop the note
	//by writing CCA to 0


// inside of all these, be sure to clear R16
//also be sure to reset the RTC count so the interrupt doesnt trigger early, and when it is done turn off RTC
//then make sure the RTC ISR sets the CCA to 0
b5:
	call CLEAR_LCD
	ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

	ldi ZL, low(b5string<<1)
	ldi ZH, high(b5string<<1) //load the location i am outpitting to lcd
```

```
        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(1012)
        st Z+, R16
        ldi R16, high(1012)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
C6:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(c6string<<1)
        ldi ZH, high(c6string<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD
        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(956)
        st Z+, R16
        ldi R16, high(956)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16 //turn on the RTC
        call checkifdone
        clr R16
        ret
c6sh:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(c6shstring<<1)
        ldi ZH, high(c6shstring<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)

        ldi R16, low(902)

        st Z+, R16

        ldi R16, high(902)

        st Z, R16 //this block sets the timer's period to make the freq


        ldi ZH, high(RTC_CNT)

        ldi ZL, low(RTC_CNT)

        ldi R16, 0

        st Z+, R16

        st Z, R16

        ldi R16, 1

        sts RTC_CTRL, R16

        call checkifdone

        clr R16

        ret

D6:

        call CLEAR_LCD

        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes


        ldi ZL, low(d6string<<1)

        ldi ZH, high(d6string<<1) //load the location i am outpitting to lcd


        //call OUT_STRING_LCD


        ldi ZH, high(TCF0_CCA)

        ldi ZL, low(TCF0_CCA)

        ldi R16, low(851)
        st Z+, R16
        ldi R16, high(851)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
D6sh:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(d6shstring<<1)
        ldi ZH, high(d6shstring<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(804)
        st Z+, R16
        ldi R16, high(804)
```

```
	st Z, R16 //this block sets the timer's period to make the freq

	ldi ZH, high(RTC_CNT)
	ldi ZL, low(RTC_CNT)
	ldi R16, 0
	st Z+, R16
	st Z, R16
	ldi R16, 1
	sts RTC_CTRL, R16
	call checkifdone
	clr R16
	ret
e6:
	call CLEAR_LCD
	ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

	ldi ZL, low(e6string<<1)
	ldi ZH, high(e6string<<1) //load the location i am outpitting to lcd

	//call OUT_STRING_LCD

	ldi ZH, high(TCF0_CCA)
	ldi ZL, low(TCF0_CCA)
	ldi R16, low(758)
	st Z+, R16
	ldi R16, high(758)
	st Z, R16 //this block sets the timer's period to make the freq
```

```
    ldi ZH, high(RTC_CNT)
    ldi ZL, low(RTC_CNT)
    ldi R16, 0
    st Z+, R16
    st Z, R16
    ldi R16, 1
    sts RTC_CTRL, R16
    call checkifdone
    clr R16
    ret
F6:
    call CLEAR_LCD
    ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

    ldi ZL, low(f6string<<1)
    ldi ZH, high(f6string<<1) //load the location i am outpitting to lcd

    //call OUT_STRING_LCD

    ldi ZH, high(TCF0_CCA)
    ldi ZL, low(TCF0_CCA)
    ldi R16, low(716)
    st Z+, R16
    ldi R16, high(716)
    st Z, R16 //this block sets the timer's period to make the freq

    ldi ZH, high(RTC_CNT)
    ldi ZL, low(RTC_CNT)
```

```
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
F6sh:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(f6shstring<<1)
        ldi ZH, high(f6shstring<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(676)
        st Z+, R16
        ldi R16, high(676)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
```

```
	st Z, R16
	ldi R16, 1
	sts RTC_CTRL, R16
	call checkifdone
	clr R16
	ret
G6:
	call CLEAR_LCD
	ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

	ldi ZL, low(g6string<<1)
	ldi ZH, high(g6string<<1) //load the location i am outpitting to lcd

	//call OUT_STRING_LCD

	ldi ZH, high(TCF0_CCA)
	ldi ZL, low(TCF0_CCA)
	ldi R16, low(638)
	st Z+, R16
	ldi R16, high(638)
	st Z, R16 //this block sets the timer's period to make the freq

	ldi ZH, high(RTC_CNT)
	ldi ZL, low(RTC_CNT)
	ldi R16, 0
	st Z+, R16
	st Z, R16
	ldi R16, 1
```

```
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
G6sh:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(g6shstring<<1)
        ldi ZH, high(g6shstring<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(602)
        st Z+, R16
        ldi R16, high(602)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
A6:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(a6string<<1)
        ldi ZH, high(a6string<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(568)
        st Z+, R16
        ldi R16, high(568)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
```

```
A6sh:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(a6shstring<<1)
        ldi ZH, high(a6shstring<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(536)
        st Z+, R16
        ldi R16, high(536)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
B6:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(b6string<<1)
        ldi ZH, high(b6string<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(506)
        st Z+, R16
        ldi R16, high(506)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        call checkifdone
        clr R16
        ret
C7:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes
        ldi ZL, low(c7string<<1)
        ldi ZH, high(c7string<<1) //load the location i am outpitting to lcd

        //call OUT_STRING_LCD


        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(478)
        st Z+, R16
        ldi R16, high(478)
        st Z, R16 //this block sets the timer's period to make the freq


        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        clr R16
        call checkifdone
        ret
D7:
        call CLEAR_LCD
        ldi R17, 1 // a simple flag used in scales, cleared when the RTC compare completes

        ldi ZL, low(d7string<<1)
        ldi ZH, high(d7string<<1) //load the location i am outpitting to lcd
        //call OUT_STRING_LCD

        ldi ZH, high(TCF0_CCA)
        ldi ZL, low(TCF0_CCA)
        ldi R16, low(426)
        st Z+, R16
        ldi R16, high(426)
        st Z, R16 //this block sets the timer's period to make the freq

        ldi ZH, high(RTC_CNT)
        ldi ZL, low(RTC_CNT)
        ldi R16, 0
        st Z+, R16
        st Z, R16
        ldi R16, 1
        sts RTC_CTRL, R16
        clr R16
        call checkifdone
        ret
ascendingscale:
        call c6
        call d6
        call e6
        call f6
        call g6
        call a6
        call b6
        call c7
        ret
descendingescale:
        call d6
        call d6
        //call d6
        call e6
        call f6
        call g6
        call g6
        call c7
        call d7
        call d7
        call g6
        call e6
        call e6
        //call d6
        call d6
        call d6
        call g6
        call e6
        call e6
        call d6
        call c6
        call d6
        call d6
        call g6
        call e6
```

```
            call e6
            ret

USARTC0_RXC_ISR:
            cli
            lds R16, USARTC0_DATA
            sei
            reti

 LCD_BF_WAIT:
            push R16
            push r17
            ldi R16, 0
            ldi R17, 0

            AGAINLCD:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
```

```
    NOP

    NOP

    NOP

    INC R16

    CPI R16, 0

    BREQ CARRYLCD


    BACKLCD:

    CPI R17, 0x01

    BRNE AGAINLCD

    BREQ RETURNLCD


    CARRYLCD:

    INC R17

    rjmp BACKLCD


    RETURNLCD:

        pop r17

        pop r16

        RET



// put all of the table data here so you can print it to LCD

B5STRING:

.db "B5 987.77 HZ", 0

C6STRING:

.db "C6 1046.50 HZ", 0

C6SHSTRING:
```

```
.db "C6#/D6b 1108.73 HZ", 0
D6STRING:
.db "D6 1174.66 HZ", 0
D6SHSTRING:
.db "D6#/E6b 1244.51 HZ", 0
E6STRING:
.db "E6 1318.51 HZ", 0
F6STRING:
.db "F6 1396.91 HZ", 0
F6SHString:
.db "F6#/G6b 1479.98 HZ", 0
G6STRING:
.db "G6 1567.98 HZ", 0
G6SHSTRING:
.db "G6#/A6b 1661.22 HZ", 0
A6STRING:
.db "A6 1760.00 HZ", 0
A6SHSTRING:
.db "A6#/B6b 1864.66 HZ", 0
B6STRING:
.db "B6 1975.53 HZ", 0
C7STRING:
.db "C7 2093.00 HZ", 0
D7STRING:
.db "D7 2349.32 HZ", 0


//LCD out subroutines
```

```
OUT_CHAR_LCD: //outs R16 to LCD
                call LCD_BF_WAIT
                ldi XH, high(LCDPORT_DAT)
                ldi XL, low(LCDPORT_DAT)
                st X, R16
                ret

 OUT_STRING_LCD: //put address of string in Z register
                push R16

                stringloop:
                        lpm R16, Z+

                        cpi R16, 0
                        breq string_done

                        call OUT_CHAR_LCD
                        rjmp stringloop

                string_done:
                        pop R16
                        ret
Clear_LCD:
        push R16
        ldi R16, 1
        ldi XL, low(LCDPORT_COM)
        ldi XH, high(LCDPORT_COM)
        st X, R16
        pop R16
        ret

RTC_COMP_ISR:
        cli
        push R16
        clr R17 //flag register for scales
        ldi R16, 0
        sts RTC_CTRL, R16
        sts RTC_CNT, R16
        sts RTC_CNT+1, R16

        ldi ZL, low(TCF0_CCA)
        ldi ZH, high(TCF0_CCA) //load Z so we can write our compare value

        ldi R16, 0
        st Z+, R16

        ldi R16, 0
        st Z, R16 // stop the wave

        pop R16
        sei
        reti
checkifdone:
        sbrc R17, 0
        rjmp checkifdone
        ret
/* Brandon Pollack
*  HW4
*  SCI Subroutines
*/


.macro SCI_C_INIT

        .equ BSEL = 11
        .equ BSCL = -1

        ldi R16, 0x18
        sts USARTC0_CTRLB, R16          ;this buts a one in RXEN and TXEN, enabling
transmission and receive

        ldi R16, 0x03
        sts USARTC0_CTRLC, R16          ;No parity, 8 bit data, a single stop bit

        ldi R16, BSEL
        sts USARTC0_BAUDCTRLA, R16      ;setting baud to 9600 HZ involves some
calculation from the manual

        ldi R16, ((BSCL << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

        sts USARTC0_BAUDCTRLB, R16      ;set the scale to -2 as per the formula to get 9600
HZ from the Fper and BSCL, upper 4 bits of BSEL stay the same

        ; now begins the set up of the PORTC to output and input serial
        ldi R16, 0x08
        sts PORTC_DIR, R16
        sts PORTC_OUT, R16 ; set the direction of the TX line as out and default as 1 as per docs
.endmacro
/*SPI_C_INIT:
        .macro
        .equ BSEL = 51
        .equ BSCL = -2



        ldi R16, 0x18
        sts USARTC0_CTRLB, R16              ;this buts a one in RXEN and TXEN, enabling
transmission and receive


        ldi R16, 0b01000011
        sts USARTC0_CTRLC, R16              ;No parity, 8 bit data, a single stop bit,
synchronous transmission


        ldi R16, BSEL
        sts USARTC0_BAUDCTRLA, R16          ;setting baud to 9600 HZ involves some
calculation from the manual


        ldi R16, ((BSCL << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

        sts USARTC0_BAUDCTRLB, R16          ;set the scale to -2 as per the formula to get 9600
HZ from the Fper and BSCL, upper 4 bits of BSEL stay the same


        ; now begins the set up of the PORTC to output and input serial
        ldi R16, 0x08
        sts PORTC_DIR, R16
        sts PORTC_OUT, R16 ; set the direction of the TX line as out and default as 1 as per docs
        .endmacro
        */

OUT_CHAR:
        .org 0x1000
        push R17 ;save this value

        isdatasent:
                lds R17, USARTC0_STATUS
                sbrs R17, 6 ;poll TXIF in status register, if it is clear we are not done
                rjmp isdatasent

        sts USARTC0_DATA, R16

        pop R17

        ret

OUT_STRING:
        push R16 ;I chose to use z so this sub works for program or data memory (remember to shift left if program memory)

        beginwritingstring:
                ld R16, Z+ ;at the end of this sub, z will point to one address past the end of the string
                breq donewritingstring
                call OUT_CHAR
                rjmp beginwritingstring

        donewritingstring:
                pop R16
                ret
```