```
//     Evan Heinrich
//     CE2801 sect. 011
//     11/12/2021
//
//     File:
//           main.S
//     Description of File:
//           Main program of Lab 8
//           Temperature Monitor
//     (opt) Dependencies:
//           timer_delay.S
//           LCD_Control.S
//           keypad.S
//           ASCII.S
//           tone.S
//           temperature.S
.syntax unified
.cpu cortex-m4
.thumb
.section .text
.global main
main:
       // Required API Initializations
       BL delay_Init
       BL LCD_Init
       BL tone_Init
       BL key_Init

       // This program's hardware initialization
       BL RCC_Setup

       BL GPIOB_Setup

       BL ADC1_Setup

       BL NVIC_Setup

       BL TIM5_Setup

       // Start ADC timer
       LDR R1, =TIM5_BASE
       LDR R2, [R1, #TIM_CR1]     // Read current controls
       ORR R2, #CEN               // Enable count
       STR R2, [R1, #TIM_CR1]     // Write
       // Main program loop
       loop:
       // Check if a conversion is ready
       LDR R1, =EndOfConversion
       LDR R2, [R1]
       CMP R2, #0
       IT NE
              BLNE Display


       // If we are in continuious mode, constantly
       // update display
       LDR R1, =IsContinuious
       LDR R2, [R1]
       CMP R2, #1
       IT EQ
              BLEQ BufferCont
```

```
	// Check if any keys were pressed
	LDR R1, =press
	LDRB R2, [R1]
	CMP R2, #1
	IT EQ
		BLEQ KeyPressed

	B loop


// Displays the stored sample
Display:
	PUSH {R1-R3, LR}

	// Pause Interrupts
	LDR R1, =NVIC_BASE

	// ICER1, AKA ADC
	LDR R2, [R1, #0x84]
	ORR R2, #1<<18
	STR R2, [R1, #0x84]

	// Clear Display
	BL LCD_Home

	// Load value to be displayed
	LDR R0, =Buffer
	LDR R1, [R0]

	// Convert the sampled value into millivolts by
	BL ADC_to_mV

	// Convert millivolts into temperature
	// The function returns the whole number of the temp
	// In R0, decimal value in R1. Argument is R2
	MOV R2, R0   // Move return value from prev. funct
	BL mV_to_C

	// R0 contains whole number of temp
	// R1 contains decimal val. of temp
	// This method pushes and pops those
	BL DispCel

	// Move values to where they will need to be
	// After newline
	MOV R2, R0
	MOV R3, R1

	// Newline
	MOV R0, #1
	MOV R1, #0
	BL LCD_MoveCursor

	// Convert Celsius (stored in R2&R3) to Fahrenheit
	BL C_to_F
```

```
        // Display Fahrenheit
        BL DispFah

        // Clear flag that got us here
        LDR R1, =EndOfConversion
        MOV R2, #0
        STR R2, [R1]

        // Resume Interrupts
        LDR R1, =NVIC_BASE

        // ICER1, AKA ADC
        LDR R2, [R1, #NVIC_ISER1]
        ORR R2, #1<<18
        STR R2, [R1, #NVIC_ISER1]

        POP {R1-R3, PC}


// Display the line for celsius
DispCel:
        PUSH {R0, R1, LR}

        // Backup whole number
        MOV R7, R0

        // Convert decimal value to ASCII
        BL num_to_ASCII
        MOV R6, R0

        // Convert whole number to ASCII
        MOV R1, R7
        BL num_to_ASCII

        // Clear scratch register
        MOV R2, #0

        // Write tens to memory
        LDR R1, =StringBuffer
        LDR R3, =0xFF00
        AND R2, R3, R0
        LSR R2, #8
        STRB R2, [R1, #0]

        // Write ones to memory
        MOV R2, #0
        LSR R3, #8
        AND R2, R3, R0
        STRB R2, [R1, #1]

        // Write a decimal point
        MOV R2, #'.'
        STRB R2, [R1, #2]

        // Write the decimal value
        STRB R6, [R1, #3]
```

```
        // Write "[degree symbol]C"
        MOV R2, #'C'
        LSL R2, #8
        ORR R2, 0xDF
        STR R2, [R1, #4]

        BL LCD_PrintString

        POP {R0, R1, PC}


// Display the line for fahrenheit
DispFah:
        PUSH {R0, R1, LR}

        // Backup whole number
        MOV R7, R0

        // Convert decimal value to ASCII
        BL num_to_ASCII
        MOV R6, R0

        // Convert whole number to ASCII
        MOV R1, R7
        BL num_to_ASCII

        // Clear scratch register
        MOV R2, #0

        // Write tens to memory
        LDR R1, =StringBuffer
        LDR R3, =0xFF00
        AND R2, R3, R0
        LSR R2, #8
        STRB R2, [R1, #0]

        // Write ones to memory
        MOV R2, #0
        LSR R3, #8
        AND R2, R3, R0
        STRB R2, [R1, #1]

        // Write a decimal point
        MOV R2, #'.'
        STRB R2, [R1, #2]

        // Write the decimal value
        STRB R6, [R1, #3]

        // Write "[degree symbol]F"
        MOV R2, #'F'
        LSL R2, #8
        ORR R2, 0xDF
        STR R2, [R1, #4]

        BL LCD_PrintString

        POP {R0, R1, PC}
```

```
// Check the keycode if a key was pressed
KeyPressed:
	PUSH {R0-R2, LR}

	// Get the key that was pressed
	LDR R1, =button
	LDRB R2, [R1]
	MOV R1, R2

	// Convert the keycode to a hex char
	BL key_ToChar

	// Compare to C
	CMP R0, #'C'
	IT EQ
		BLEQ ContMode

	CMP R0, #'1'
	ITT EQ
		MOVEQ R1, #1000-1
		BLEQ AdjustTime

	CMP R0, #'2'
	ITT EQ
		MOVEQ R1, #2000-1
		BLEQ AdjustTime

	CMP R0, #'3'
	ITT EQ
		MOVEQ R1, #3000-1
		BLEQ AdjustTime

	CMP R0, #'4'
	ITT EQ
		MOVEQ R1, #4000-1
		BLEQ AdjustTime

	CMP R0, #'5'
	ITT EQ
		MOVEQ R1, #5000-1
		BLEQ AdjustTime

	CMP R0, #'6'
	ITT EQ
		MOVEQ R1, #6000-1
		BLEQ AdjustTime

	CMP R0, #'7'
	ITT EQ
		MOVEQ R1, #7000-1
		BLEQ AdjustTime

	CMP R0, #'8'
	ITT EQ
		MOVEQ R1, #8000-1
		BLEQ AdjustTime

	CMP R0, #'9'
	ITT EQ
		MOVEQ R1, #9000-1
		BLEQ AdjustTime
```

```
        LDR R1, =press
        MOV R2, #0
        STRB R2, [R1]

        POP {R0-R2, PC}


// Toggle continuious mode if the key pressed was C
ContMode:
        PUSH {R1, R2, LR}
        // Toggle the beeper when sampling, if this beeped during
        // constant, that would be irritating.
        LDR R1, =ShouldBeep
        LDR R2, [R1]
        EOR R2, #1
        STR R2, [R1]

        // Change the continuious flag
        LDR R1, =IsContinuious
        LDR R2, [R1]
        EOR R2, #1
        STR R2, [R1]

        // Toggle cont. mode
        LDR R1, =ADC_BASE
        LDR R2, [R1, #ADC_CR2]
        EOR R2, #CONT
        ORR R2, #SWSTART
        STR R2, [R1, #ADC_CR2]

        // Toggle ADC EOC IRQ
        LDR R1, =ADC_BASE
        LDR R2, [R1, #ADC_CR1]
        EOR R2, #EOCIE                  // Toggle EOC interrupt
        STR R2, [R1, #ADC_CR1]

        POP {R1, R2, PC}


// Adjust the sample interval if the key pressed was a number
// Interval gets changed to N seconds where N is the number of
// the key that was pressed
AdjustTime:
        PUSH {R0, R1, LR}

        MOV R0, R1
        LDR R1, =TIM5_BASE
        STR R0, [R1, #TIM_ARR]
        MOV R0, #0
        STR R0, [R1, #TIM_CNT]

        POP {R0, R1, PC}
```

```
// When the buffer is in continuous mode, EOC doesn't trigger
// so manually poll the DR and update the display
BufferCont:
	PUSH {R1, R2, LR}

	// Read the data
	LDR R1, =ADC_BASE
	LDR R2, [R1, #ADC_DR]

	LDR R1, =Buffer
	STR R2, [R1]

	BL Display

	MOV R0, #0
	MOV R1, #17
	BL LCD_MoveCursor

	POP {R1, R2, PC}



.equ RCC_BASE,      0x40023800
.equ AHB1ENR,0x30
.equ APB1ENR,0x40
.equ APB2ENR,0x44
.equ GPIOBEN,1 << 1
.equ ADC1EN, 1 << 8
.equ TIM5EN, 1 << 3
RCC_Setup:
	PUSH {R1, R2, LR}

	// Enable GPIOB in RCC
	LDR R1, =RCC_BASE
	LDR R2, [R1, #AHB1ENR]
	ORR R2, #GPIOBEN
	STR R2, [R1, #AHB1ENR]

	// Enable ADC1 in RCC
	LDR R2, [R1, #APB2ENR]
	ORR R2, #ADC1EN
	STR R2, [R1, #APB2ENR]

	// Enable TIM5 in RCC
	LDR R2, [R1, #APB1ENR]
	ORR R2, #TIM5EN
	STR R2, [R1, #APB1ENR]

	POP {R1, R2, PC}
```

```
.equ GPIOB_BASE,    0x40020400
.equ GPIO_MODER,    0
.equ PIN_ANALOG,    0b11
GPIOB_Setup:
       PUSH {R1-R3, LR}

       // Set GPIOB MODER
       LDR R1, =GPIOB_BASE
       MOV R3, #PIN_ANALOG
       LDR R2, [R1, #GPIO_MODER]
       BFI R2, R3, #0, #2
       STR R2, [R1, #GPIO_MODER]

       POP {R1-R3, PC}


.equ TIM5_BASE,     0x40000C00
.equ TIM_CR1,0
.equ TIM_DIER,      0x0C
.equ TIM_PSC,0x28
.equ TIM_ARR,0x2C
.equ TIM_EGR,       0x14
.equ TIM_CNT,0x24
.equ TIM_SR, 0x10
.equ CEN,    1 << 0
.equ DIR,    1 << 4
.equ OPM,    1 << 3
.equ UIE,    1 << 0
.equ TIM_UG, 1 << 0
.equ TIM_UIF,       1 << 0
.equ MILLISECONDS,  16000
TIM5_Setup:
       PUSH {R1, R2, LR}

       LDR R1, =TIM5_BASE        // Timer 5 base address

       LDR R2, =16000                  // 16MHz / 16kHz = 1kHz aka 1ms
       STR R2, [R1, #TIM_PSC]    // Apply prescale to 1ms per count

       // This is the prescaler fix Dr. Livingston provided
       // From what I understand, it forces an update event
       // on the timer, which somehow forces the prescale into effect.
       MOV R2, #TIM_UG
       LDR R1, =TIM5_BASE
       STR R2, [r1, #TIM_EGR]

       LDR R1, =TIM5_BASE
       LDR R2, [R1, #TIM_SR]
       BIC R2, #TIM_UIF
       STR R2, [R1, #TIM_SR]

       // These should be cleared by default but better safe than sorry
       LDR R2, [R1, #TIM_CR1]
       BIC R2, #DIR
       BIC R2, #OPM
       STR R2, [R1, #TIM_CR1]
```

```
        // Enable timer to generate interrupts
        LDR R2, [R1, #TIM_DIER]
        ORR R2, #UIE
        STR R2, [R1, #TIM_DIER]

        // Load default delay of 1sec
        LDR R2, =1000-1
        STR R2, [R1, #TIM_ARR]

        POP {R1, R2, PC}



.equ ADC_BASE,      0x40012000
.equ ADC_CR1,0x04
.equ ADC_CR2,0x08
.equ ADC_SQR1,      0x2C
.equ ADC_SQR3,      0x34
.equ ADC_DR, 0x4C
.equ ADC_10BIT,     0b01
.equ EOCIE,         1 << 5
.equ ADON,          1 << 0
.equ CONT,          1 << 1
.equ CH_8,          8
.equ SWSTART,1 << 30
ADC1_Setup:
        PUSH {R1-R3, LR}

        // Set 10bit resolution and enable EOC interrupt
        LDR R1, =ADC_BASE
        MOV R3, #ADC_10BIT
        LDR R2, [R1, #ADC_CR1]
        ORR R2, #EOCIE                  // Enable EOC interrupt
        BFI R2, R3, #24, #2       // Insert 10bit code
        STR R2, [R1, #ADC_CR1]

        // Turn on ADC and default to NOT continuious
        LDR R2, [R1, #ADC_CR2]
        ORR R2, #ADON
        BIC R2, #CONT
        STR R2, [R1, #ADC_CR2]

        // Set scan count
        LDR R1, =ADC_BASE
        LDR R2, [R1, #ADC_SQR1]
        BFC R2, #20, #4
        STR R2, [R1, #ADC_SQR1]

        // Set the one channel to be scanned
        LDR R1, =ADC_BASE
        MOV R3, #8
        LDR R2, [R1, #ADC_SQR3]
        BFI R2, R3, #0, #5
        STR R2, [R1, #ADC_SQR3]


        POP {R1-R3, PC}
```

```
.equ NVIC_BASE,            0xE000E100
.equ NVIC_ISER0,    0x00
.equ NVIC_ISER1,    0x04
.equ TIM5_INT,             1 << 18
.equ ADC_INT,       1 << 18
NVIC_Setup:
       PUSH {R1, R2, LR}

       LDR R1, =NVIC_BASE

       // ADC Interrupt is slot 18
       // which lives in the first ISER
       LDR R2, [R1, #NVIC_ISER0]
       ORR R2, #ADC_INT
       STR R2, [R1, #NVIC_ISER0]

       // TIM5 Interrupt is slot 50
       // which lives in the second ISER
       LDR R2, [R1, #NVIC_ISER1]
       ORR R2, #TIM5_INT
       STR R2, [R1, #NVIC_ISER1]

       POP {R1, R2, PC}



.global TIM5_IRQHandler
.thumb_func
TIM5_IRQHandler:
       PUSH {LR}

       // Clear flag that hardware uses to generate IRQ
       LDR R1, =TIM5_BASE
       LDR R2, [R1, #TIM_SR]
       BIC R2, #TIM_UIF
       STR R2, [R1, #TIM_SR]

       // Start conversion
       LDR R1, =ADC_BASE
       LDR R2, [R1, ADC_CR2]
       ORR R2, #SWSTART
       STR R2, [R1, ADC_CR2]

       1:
       // Return from ISR
       POP {LR}
       BX LR
```

```
.global ADC_IRQHandler
.thumb_func
ADC_IRQHandler:
	PUSH {LR}

	// Reading from the DR clears the EOC flag which causes
	// IRQ generation, so there's nothing to clear
	LDR R1, =ADC_BASE
	LDRH R2, [R1, #ADC_DR]

	// Store value
	LDR R1, =Buffer
	STRH R2, [R1]

	// Beep if not in continuious mode
	LDR R1, =ShouldBeep
	LDR R2, [R1]
	CMP R2, #1
	IT EQ
		BLEQ tone_Notify

	// Update interrupt flag for main
	LDR R1, =EndOfConversion
	MOV R2, #1
	STR R2, [R1]

	POP {LR}
	BX LR

.section .data
// Flag to poll between conversions
EndOfConversion:
.word  0

// Flag to control if the speaker should beep every conversion
ShouldBeep:
.word  1

// Determines if the ADC is in continuious mode
IsContinuious:
.word 0

// Buffer for display text
StringBuffer:
.word  0
.word  0

// Where the next free entry is in the buffer
BufferIndex:
.word  0

// What index in the buffer is the display showing
DisplayIndex:
.word  0

// 10 half-words for temp values
Buffer:
.space 2*(10)
```

```
//     Evan Heinrich
//     CE2801 sect. 011
//     11/12/2021
//
//     File:
//           temperature.S
//     Description of File:
//           Contains functions to convert ADC samples
//           into degrees F and C
//     (opt) Dependencies:
//           N/A


.syntax unified
.cpu cortex-m4
.thumb
.section .text

.global mV_to_C
.global ADC_to_mV
.global C_to_F


// Convert millivolts into degrees C according to our
// development board's TMP36 sensor
// R0 = Return, Whole number value of degrees C
// R1 = Return, Decimal value of degrees C
// R2 = Argument, mV
mV_to_C:
       PUSH {R2, R3, LR}

       // Convert number to ASCII to extract
       // Decimal value
       MOV R1, R2
       BL num_to_ASCII

       // R0 contains the ASCII now

       // Extract the number from the last digit
       // All ASCII numbers are 0x3N where N is the number,
       // so it can be extracted by and-ing with 0xF
       MOV R1, R0
       MOV R3, #0x0F
       AND R1, R3
       // R1 now contains decimal value of temp

       // Divide millivolt value by 10
       MOV R0, R2          // Move millivolts into R0
       MOV R3, #10         // Prepare to divide by 10
       UDIV R0, R0, R3     // Divide by 10

       // Subtract our offset
       SUB R0, #45
       // R0 now contains the whole number portion of temp

       // Return
       POP {R2, R3, PC}
```

```
// Converts a celsius number to Fahrenheit
// R0 = Whole Number Fahrenheit
// R1 = Decimal val. Fahrenheit
// R2 = Whole Number Celsius
// R3 = Decimal val. Celsius
C_to_F:
	PUSH {R2-R4, LR}

	// Celsius * 18
	MOV R4, #18
	MUL R2, R4

	// Above / 10
	MOV R4, #10
	UDIV R2, R4

	// Offset for C to F
	ADD R2, #32

	// R2 now contains Whole number F
	MOV R0, R2

	// Apply the same conversion to the decimal value
	MOV R4, #18
	MUL R3, R4

	MOV R4, #10
	UDIV R3, R4

	// Move into return register
	MOV R1, R3

	POP {R2-R4, PC}


// Converts an ADC sample value into millivolts
// R0 = Return, mV
// R1 = Argument, ADC sample
// This assumes 10-bit sampling
ADC_to_mV:
	PUSH {R1, R2, LR}

	MOV R2, #3
	MOV R0, R1
	MUL R0, R2

	POP {R1, R2, PC}
```

Slight Modification to num_to_ASCII to remove leading zeroes

```
num_to_ASCII:
	PUSH {R1-R12, LR}		// Backup registers

	LDR R2, =MAX_VALUE		// Load max value
	CMP R1, R2			// Compare the argument to the maximum value
	BGE error			// Return the error code if the argument is larger than the max.
	MOV R2, R1			// Copy the argument for modification


	MOV R6, #0			// Clear thousands counter
mod1000:
	SUBS R2, R2, #0x3E8			// Subtract 1000, update flags
	ITET PL					// If positive
		ADDPL R6, R6, #1		// Increment thousands counter
		ADDMI R2, R2, #0x3E8		// Add back 1000 if negative
		BPL mod1000			// Otherwise continue looping


	MOV R5, #0			// Clear hundreds counter
mod100:
	SUBS R2, R2, #0x64		// Subtract 100, update flags
	ITET PL				//If positive
		ADDPL R5, R5, #1	// Increment hundreds counter
		ADDMI R2, R2, #0x64	// Add back 100 if negative
		BPL mod100		// Otherwise continue looping


	MOV R4, #0			// Clear tens register
mod10:
	SUBS R2, R2, #0xA	// Subtract 10, update flags
	ITET PL				// If positive
		ADDPL R4, R4, #1	// Increment tens counter
		ADDMI R2, R2, #0xA	// Add back 10 if negative
		BPL mod10	// Otherwise continue looping

	MOV R3, R2		// Whatever is left is the ones place

	MOV R0, #0		// Clear R0

	CMP R6, #0
	BGT thousands		// If thousands > 1, start from thousands

	CMP R5, #0
	BGT hundreds		// If hundreds > 1, start from hundreds

	CMP R4, #0
	BGT tens		// If tens > 1, start from tens

	B ones			// No matter what, do ones

	thousands:
	ORR R6, #0x30
	// R6 now contains an ASCII number representing thousands
	BFI R0, R6, #24, #8
```

```
hundreds:
ORR R5, #0x30
// R5 now contains an ASCII number representing hundreds
BFI R0, R5, #16, #8

tens:
ORR R4, #0x30
// R4 now contains an ASCII number representing tens
BFI R0, R4, #8, #8

ones:
ORR R3, #0x30
// R3 now contains an ASCII number representing ones
BFI R0, R3, #0, #8

B return

error:
LDR R0, =ERR

return:
POP {R1-R12, LR}
BX LR
```

Slight modification to LCD_PrintNum to remove leading zeroes

```
LCD_PrintNum:
	PUSH {R0-R4, LR}

	BL num_to_ASCII		// Stores ASCII representing chars in R0
	MOV R4, R0		// Backup number

	MOV R0, #0		// Prep an iterator
	LDR R3, =0xFF000000 // Prep mask

	// First pass
	AND R1, R4, R3			// Apply mask, store into R1
	LSR R1, #(3*8)			// Shift char into lsb
	CMP R1, #0
	IT NE				// If the char isn't null
		BLNE WriteData		// Write char
	LSR R3, R3, #8			// Shift mask to next char

	// Second pass
	AND R1, R4, R3			// Apply mask, store into R1
	LSR R1, #(2*8)			// Shift char into lsb
	CMP R1, #0
	IT NE				// If the char isn't null
		BLNE WriteData		// Write char
	LSR R3, R3, #8			// Shift mask to next char

	// Third pass
	AND R1, R4, R3			// Apply mask, store into R1
	LSR R1, #(1*8)			// Shift char into lsb
	CMP R1, #0
	IT NE				// If the char isn't null
		BLNE WriteData		// Write char
	LSR R3, R3, #8			// Shift mask to next char

	// Fourth pass
	AND R1, R4, R3			// Apply mask, store into R1
	CMP R1, #0
	IT NE
		BLNE WriteData		// Write char

	POP {R0-R4, PC}
```