Memo to: Randy Larimer
From: Matthew Handley
Date: April 24, 2014
Regarding: EELE 465-01, Lab 6 – TEC Module

Summary:

This lab built on the previous ones, by adding a Thermo Electric Cooler (TEC) modeule and LM92 I2C temperature sensor to the system. The goal was to have the system run in two modes A or B, based on user input from the keypad and LCD. In mode A, the user would use the keypad to set the TEC module to either heat, cool, off. The temperature an aluminum block on one side of the TEC was read using the LM92 and displayed on the LCD, along with the current state and time in that state. In mode B, the user would enter a temperature and the microcontroller would control the TEC to drive the aluminum block to the entered temperature.

Preliminary Solutions:

As before, the TPM module was used to toggle the heartbeat led. However, instead of updating the LCD and LEDs in the TPM or continuously in the main loop, a set of update_devices subroutines was created. These subroutines would only be called when something on either the LEDs or LCD needed to be changed. Additionally these subroutines would be called at the connivance of the main loop, to avoid problems.

The high-level flow of the program is detailed in Figures 1 through 4 of Appendix A. The system would start by resetting everything to a default state and prompting the user for the desired mode. When mode A was selected, the system would go into the A_mainLoop. When mode B was selected, the system would first prompt for a temperature set-point, then go into the B_mainLoop. If at any time the '*' key was pressed, the system would return to the reset state.

Setup:

To begin programming, an LM92 mounted to the aluminum block was connected to the existing I2C bus on the breadboard, from the RTC lab. Also, the inputs to the TEC controller relays were connected to a ULN2003AN high-voltage Darlington transistor array, as shown in the lab 6 schematic.

Solution:

The lm92_driver.asm file was written to implement the subroutines needed with the LM92 over I2C and convert the data to temperature. This driver was fairly straight forward, as the only communications required was setting the register address on the LM92, then reading the data from that register.

The other main addition of code was to the main.asm file, as the flow of the user interface and automatic temperature control for mode B were implemented here. Writing these changes to the main.asm file took the majority of the time spent on the lab.

Summary Comments:

Having built up a large code base over the semester for the system bus, LEDs, keypad, LCD, and I2C bus most of the work for this lab was already done. Additionally, by building this code base an understanding and familiarity with the S08 assembly was developed. This familiarity meant much less time was spent implementing and debugging the code. The final implementation worked as designed and there were no big hurdles for this lab.

The following is a summary of the memory usage for this lab, as found in the project's .map file.

Flash Used: 2677 bytes
RAM Used: 113 bytes

Vectors Used:
_Vtpmovf
_Startup

Appendix A – Figures

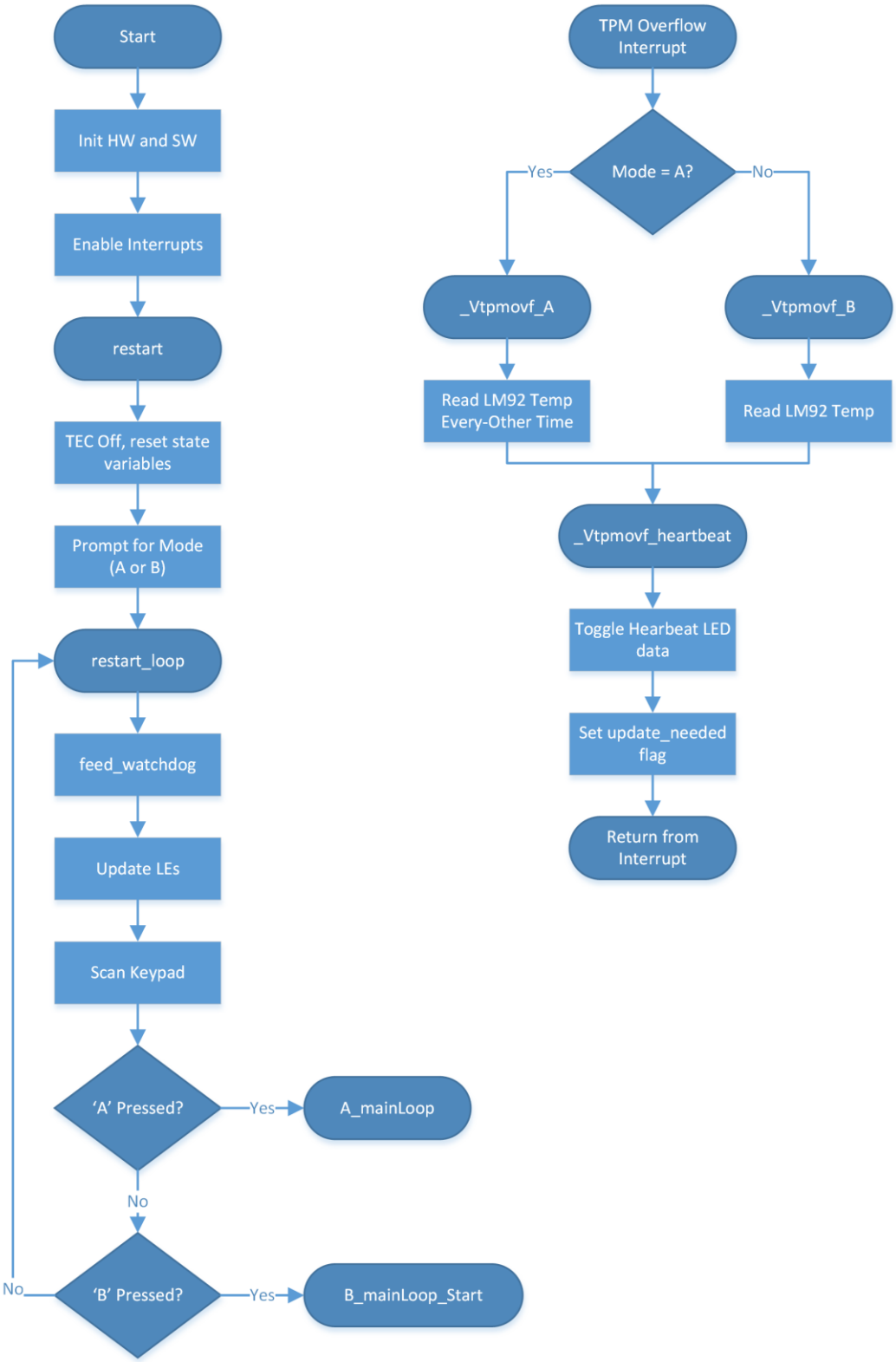

Figure 1: Top Level Flow Diagram

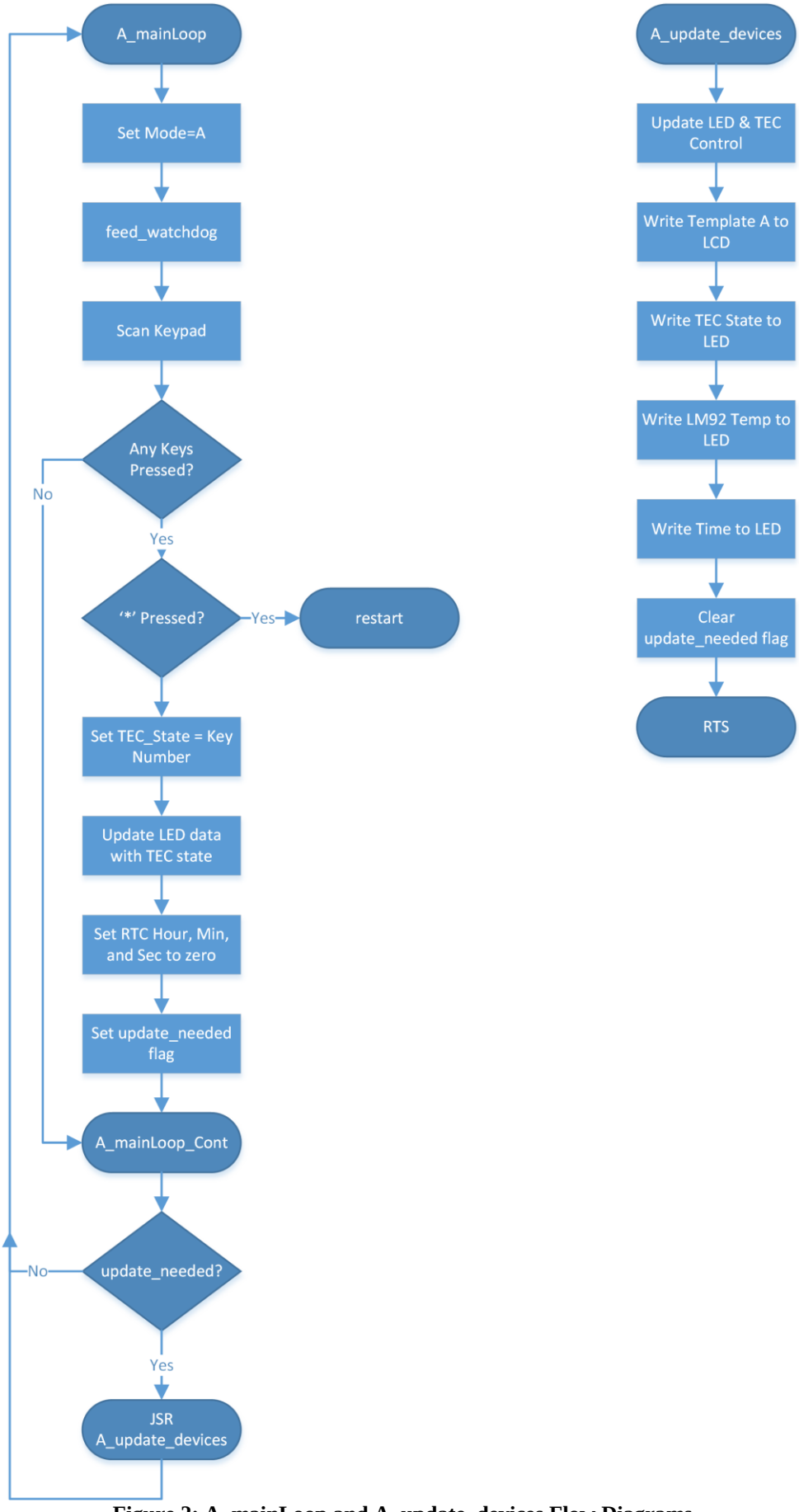

Figure 2: A_mainLoop and A_update_devices Flow Diagrams

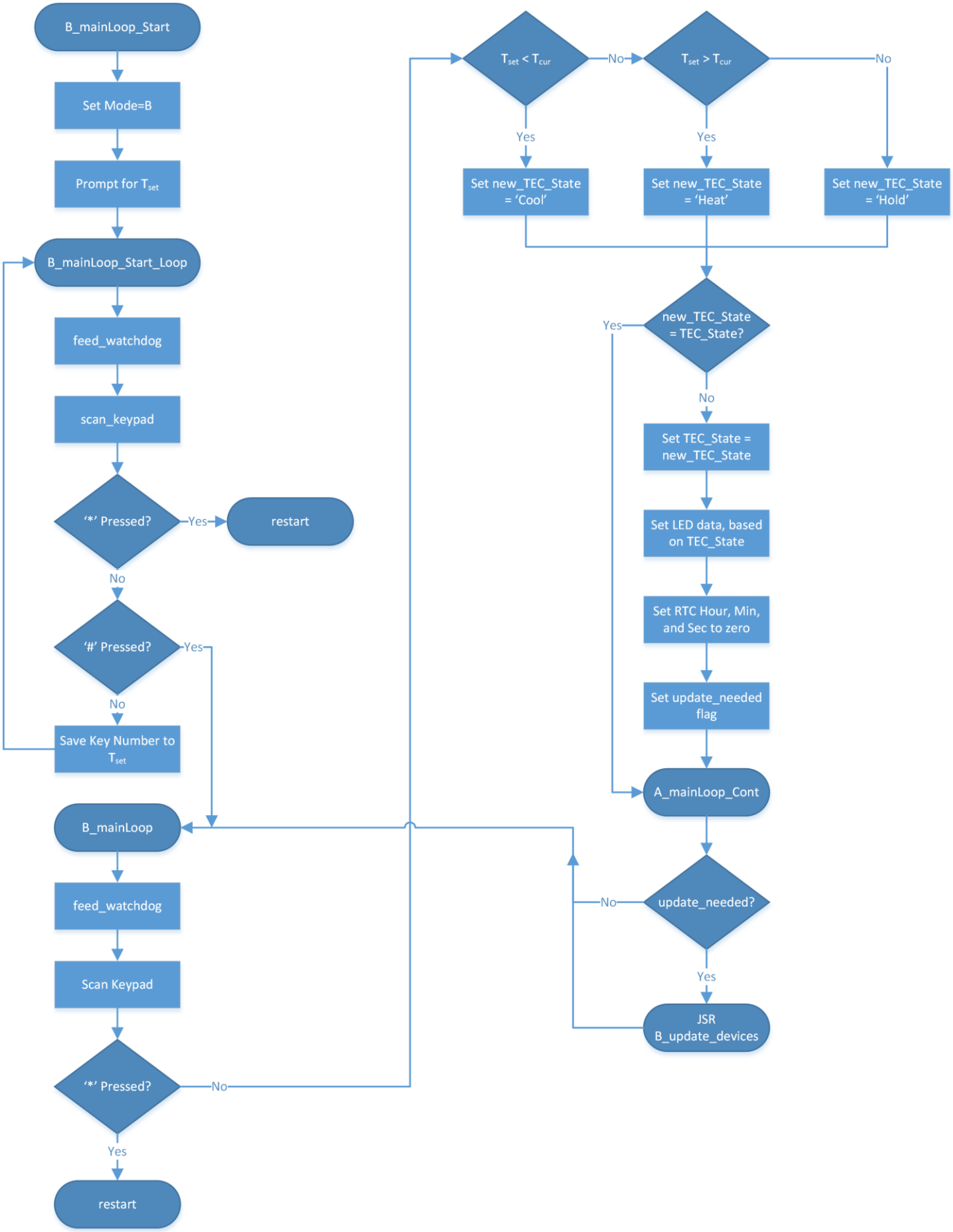

Figure 3: B_mainLoop Flow Diagrams

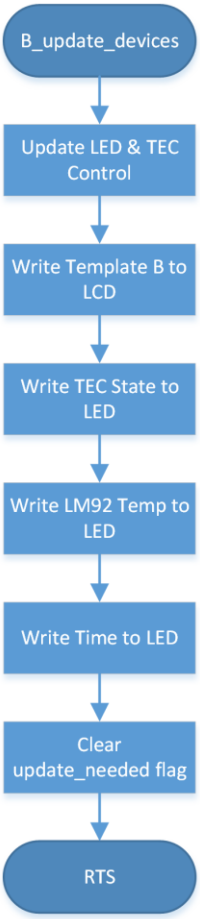

Figure 4: B_update_devices Flow Diagrams

Appendix B – Source Code

```
;*********************************************************************************
;* File Name    : main.asm
;* Program Name   : Lab#03 - TEC Module
;* Author Names   : Matthew Handley
;* Date                : 2014-04-24
;* Description    : Drives a TEC module directly from user input,
;*                or automatically based on a temperature set point.
;*
;*********************************************************************************
; Include derivative-specific definitions
            INCLUDE 'derivative.inc'


; export symbols
            XDEF _Startup, main, _Vtpmovf, SUB_delay, SUB_delay_cnt
            ; we export both '_Startup' and 'main' as symbols. Either can
            ; be referenced in the linker .prm file or from C/C++ later on


            XREF __SEG_END_SSTACK   ; symbol defined by the linker for the end of the
stack

            XREF bus_init, bus_read, bus_write, bus_addr, bus_data

            XREF led_write, led_data

            XREF keypad_interpret, keypad_scan, keypad_get_keypress
            XREF keypad_data_0, keypad_data_1

            XREF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char, lcd_clear,
lcd_goto_addr, lcd_goto_row0, lcd_goto_row1
            XREF lcd_data, lcd_char_data, lcd_col_idx

            XREF adc_init, adc_read_ch26_avg, adc_read_ch2_avg, adc_read_avg,
adc_data_0, adc_data_1

            XREF math_mul_16
            XREF INTACC1, INTACC2

            XREF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte

            XREF rtc_init, rtc_set_time_zero, rtc_calc_tod, rtc_write_tod,
rtc_set_time, rtc_get_time, rtc_display_data, rtc_prompt_time
            XREF Sec, Min, Hour, Date, Month, Year

            XREF lm92_init, lm92_read_temp, lm92_write_lcd_K, lm92_write_lcd_C


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  SUB_delay_cnt:          DS.B  3           ; counter for SUB_delay
subroutine

                  num_samples:            DS.B    1         ; number of samples to
take on the ADC

                  temp:                   DS.B  1           ; some space to hold stuff

                  temp_k:                       DS.B  1           ; some space to hold
stuff
```

```
                rtc_set:                DS.B  1           ; 0x01 when the rtc has
been set, 0x00 otherwise

                update_needed:          DS.B  1           ; 0x01 when an update of
the LEDs, LCD, or TEC is needed, 0x00 otherwise

                TEC_state:              DS.B  1           ; In Mode A: 0=off,
1=heat, 2=cool
                                                          ; In Mode B:
0=hold, 1=heat, 2=cool
                new_TEC_state:          DS.B  1

                mode:                   DS.B  1           ; Mode; 0x0A = A, 0x0B =
B, 0x00 = Waiting for mode

                Tset:                   DS.B  1           ; set point temperature,
for mode B
                Tcur:                   DS.B  1           ; current temperature, for
mode B

MY_CONST: SECTION
; Constant Values and Tables Section

                str_start:                    DC.B  "Mode: A,B?      "
                str_start_length:       DC.B  16

                str_A_top:                    DC.B  "TEC State:      "
                str_A_top_length:       DC.B  16
                str_A_bottom:                 DC.B  "T92:   K@T=000s "
                str_A_bottom_length:    DC.B  16


                str_B_prompt_top:       DC.B  "Target Temp?    "
                str_B_prompt_top_len:   DC.B  16
                str_B_prompt_bottom:    DC.B  "Enter 10-40C "
                str_B_prompt_bottom_len:DC.B  13

                str_B_top_heat:               DC.B  "TEC State:HeatXX"
                str_B_top_cool:               DC.B  "TEC State:CoolXX"
                str_B_top_hold:               DC.B  "TEC State:HoldXX"
                str_B_bottom:                 DC.B  "T92:   C@T=000s "

                str_tec_heat:                 DC.B  "Heat"
                str_tec_cool:                 DC.B  "Cool"
                str_tec_off:                  DC.B  "Off "
                str_tec_length:               DC.B  4

; code section
MyCode:     SECTION
main:
_Startup:
            LDHX   #__SEG_END_SSTACK ; initialize the stack pointer
            TXS

            ; init bus
            JSR         bus_init

            ;*** init LCD and RS, RW pins ***
                  JSR         lcd_init

                  LDA         #$00
```

```
                STA         lcd_col_idx

                ;*** init TPM module - for heartbeat LED ***
                ; TPMMODH:L Registers
                LDA   #$00
                STA         TPMMODH
                LDA   #$00
                STA         TPMMODL
                ; TPMSC Register
                LDA   #$4E                          ; TOIE clear, CLKS: Bus clock,
Prescale: 128
                STA         TPMSC

                ;*** init led_data variable ***
                LDA         #$00
                STA         led_data

                ; init i2c
                JSR         i2c_init

                ; init rtc
                JSR         rtc_init

                ; lm92_init
                JSR         lm92_init

                ; initially TEC off
                MOV         #$00, TEC_state

                ; set update_needed
                MOV         #$01, update_needed

                ; set mode

                CLI               ; enable interrupts
```

```
;*************************************************************
;* Subroutine Name: restart
;* Description: Restart loop for startup and when the user
;*                      wants to change modes.
;* Registers Modified: A,X,H
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
restart:
                ; reset mode & Tset
                MOV         #$00, mode
                MOV         #$00, Tset

                ; turn TEC off
                LDA         led_data
                AND         #$FC
                STA         led_data

                ; reset state
                MOV         #$00, TEC_state

                ; put prompt on LCD
                JSR         lcd_clear

                JSR         lcd_goto_row0
                LDHX  #str_start
                LDA         str_start_length
                JSR         lcd_str

                ; set update_needed
                MOV         #$01, update_needed

restart_loop:
                feed_watchdog

                ; update LEDs
                JSR         led_write

                ; scan keypad
                JSR         keypad_scan
                JSR         keypad_interpret

                ; was 'A' pressed?
                CBEQA #$0A, A_mainLoop

                ; was 'B' pressed?
                CBEQA #$0B, B_mainLoop_start

                BRA         restart_loop
```

```
;*************************************************************
;* Subroutine Name: A_mainLoop
;* Description: Main loop for mode A
;* Registers Modified: A,X,H
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
A_mainLoop:
                    ; set mode
                    MOV         #$0A, mode

                    feed_watchdog

                    ; scan keypad
                    JSR         keypad_scan
                    JSR         keypad_interpret

                    ; was a key pressed?
                    CBEQA #$FF, A_mainLoop_cont

                    ; was '*' pressed
                    CBEQA #$0E, restart

                    ; key was pressed, so consider it our new state
                    STA         TEC_state

                    ; update TEC data (led_data)
                    LDA         led_data
                    AND         #$FC
                    STA         led_data

                    LDA         TEC_state
                    AND         #$03
                    ORA         led_data
                    STA         led_data

                    ; since key was pressed, zero the time
                    JSR         rtc_set_time_zero

                    ; set update_needed
                    MOV         #$01, update_needed

A_mainLoop_cont:
                    ; do we need to update stuff?
                    LDA         update_needed
                    BEQ         A_mainLoop

                    JSR         A_update_devices

                    BRA         A_mainLoop

;*************************************************************

jmp_restart:
                    JMP         restart
```

```
;*************************************************************
;* Subroutine Name: B_mainLoop_start
;* Description: Main loop for mode B
;* Registers Modified: A,X,H
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
B_mainLoop_start:

                    ; set mode
                    MOV         #$0B, mode

                    ; set TEC_state so that state change forced on first iteration
                    MOV         #$10, TEC_state

                    ; prompt for Tset
                    JSR         lcd_goto_row0
                    LDHX  #str_B_prompt_top
                    LDA         str_B_prompt_top_len
                    JSR         lcd_str

                    JSR         lcd_goto_row1
                    LDHX  #str_B_prompt_bottom
                    LDA         str_B_prompt_bottom_len
                    JSR         lcd_str

B_mainLoop_start_loop:

                    feed_watchdog

                    ; scan keypad
                    JSR         keypad_scan
                    JSR         keypad_interpret

                    ; was '*' pressed
                    CBEQA #$0E, jmp_restart

                    ; was '#' pressed
                    CBEQA #$0F, B_mainLoop

                    ; was nothing presed
                    CBEQA #$FF, B_mainLoop_start_loop

                    ; else, something was pressed, save it
                    STA         temp

                    ; is this the second digit?
                    LDA         Tset
                    BNE         B_mainLoop_start_2nd_digit

B_mainLoop_start_1st_digit:

                    ; multiply by 10
                    LDHX  #$000A
                    LDA         temp
                    MUL               ; X:A <= (X) * (A)
                    STA         Tset

                    ; wait for next digit
                    BRA         B_mainLoop_start_loop

B_mainLoop_start_2nd_digit:
```

```
                    ; Add the digit to Tset
                    ADD         temp
                    STA         Tset

                    ; wait for '#' key press
                    BRA         B_mainLoop_start_loop

B_mainLoop:
                    feed_watchdog

                    ; scan keypad
                    JSR         keypad_scan
                    JSR         keypad_interpret

                    ; was '*' pressed
                    CBEQA #$0E, jmp_restart

                    ; compare Tset with Tcur
                    LDA         Tset
                    CMP         Tcur

                    ; if Tset < Tcur, cool
                    BHI         B_mainLoop_cool

                    ; else if Tset > Tcur, heat
                    BLO         B_mainLoop_heat

                    ; else, hold
                    BRA         B_mainLoop_hold

B_mainLoop_heat:
                    MOV         #$02, new_TEC_state
                    BRA         B_mainLoop_check_state

B_mainLoop_cool:
                    MOV         #$01, new_TEC_state
                    BRA         B_mainLoop_check_state

B_mainLoop_hold:
                    MOV         #$00, new_TEC_state

B_mainLoop_check_state:
                    ; new_TEC_state == TEC_state ?
                    LDA         TEC_state
                    CBEQ  new_TEC_state, B_mainLoop_cont

                    ; else new_TEC_state != TEC_state

B_mainLoop_state_changed:

                    ; save new state to current state
                    MOV         new_TEC_state, TEC_state

                    ; merge TEC_state with LED_data
                    LDA         led_data
                    AND         #$FC
                    ORA         TEC_state
                    STA         led_data

                    ; reset RTC counter
                    JSR         rtc_set_time_zero
```

```
                ; set update_needed flag
                MOV         #$01, update_needed

B_mainLoop_cont:
                ; do we need to update stuff?
                LDA         update_needed
                BEQ         B_mainLoop

                JSR         B_update_devices

                BRA         B_mainLoop

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: _Vtpmovf
;* Description: Interrupt service routine for the TPM overflow
;*                      interrupt. Toggles the heartbeat LED (PortA[0])
;*                      and resets TPM overflow flag.
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
_Vtpmovf:
                ; check mode
                LDA         mode
                CBEQA #$0A, _Vtpmovf_A
                CBEQA #$0B, _Vtpmovf_B

                ; not in Mode A or B, so do nothing, except update heartbeat LED
                BRA         _Vtpmovf_heartbeat

_Vtpmovf_A:
                ; read LM92 every-other time (when heartbeat LED is On)
                LDA         led_data
                AND         #$80
                BEQ         _Vtpmovf_heartbeat
                JSR         lm92_read_temp
                STA         Tcur

                BRA         _Vtpmovf_heartbeat

_Vtpmovf_B:
                ; always read LM92
                JSR         lm92_read_temp
                STA         Tcur

                ;BRA        _Vtpmovf_heartbeat

_Vtpmovf_heartbeat:
                ; Toggle Heartbeat LED
                LDA         led_data                ; load current LED pattern
                EOR         #$80                    ; toggle bit 7
                STA         led_data                ; Store pattern to var

                ; clear TPM ch0 flag
                LDA         TPMSC                   ; read register
                AND         #$4E                    ; clear CH0F bit, but leav
others alone
                STA         TPMSC                   ; write back register

                ; set update_needed
                MOV         #$01, update_needed

                ; Done, Return from Interrupt
                RTI


;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: A_update_devices
;* Description:
;*
;* Registers Modified: A, update_needed
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
A_update_devices:

; Update LEDs and TEC
                    JSR         led_write

; write lcd template string
                    JSR         lcd_clear

                    JSR         lcd_goto_row0
                    LDHX  #str_A_top
                    LDA         str_A_top_length
                    JSR         lcd_str

                    JSR         lcd_goto_row1
                    LDHX  #str_A_bottom
                    LDA         str_A_bottom_length
                    JSR         lcd_str

; write TEC state
                    ; set LCD cursor position
                    LDA         #$8A
                    JSR         lcd_goto_addr

                    LDA         TEC_state
                    CBEQA #$01, A_update_devices_tec_heat
                    CBEQA #$02, A_update_devices_tec_cool

A_update_devices_tec_off:
                    LDHX  #str_tec_off
                    BRA         A_update_devices_tec_write

A_update_devices_tec_heat:
                    LDHX  #str_tec_heat
                    BRA         A_update_devices_tec_write

A_update_devices_tec_cool:
                    LDHX  #str_tec_cool
                    BRA         A_update_devices_tec_write

A_update_devices_tec_write:
                    LDA         str_tec_length
                    JSR         lcd_str


; write LM92 temp
                    ; set LCD cursor position
                    LDA         #$C4
                    JSR         lcd_goto_addr
                    JSR         lm92_write_lcd_K

; write time
                    ; if TEC is off, don't overwrite the 000s for time
                    LDA         TEC_state
                    CBEQA #$00, A_update_devices_done
```

```
                    ; set LCD cursor position
                    LDA         #$CB
                    JSR         lcd_goto_addr

                    ; read time from RTC
                    JSR         rtc_get_time

                    ; calc and write TOD
                    JSR         rtc_calc_tod
                    JSR         rtc_write_tod

                    BRA   A_update_devices_done

A_update_devices_done:
;*** done
                    MOV         #$00, update_needed
                    RTS

;*************************************************************
;*************************************************************
;* Subroutine Name: B_update_devices
;* Description:
;*
;* Registers Modified: A, update_needed
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
B_update_devices:

; Update LEDs and TEC
                    JSR         led_write

; write to lcd template string
                    JSR         lcd_clear

                    ; set LCD cursor position
                    JSR         lcd_goto_row0

                    ; write top row depending on state
                    LDA         TEC_state
                    CBEQA #$01, B_update_devices_tec_heat
                    CBEQA #$02, B_update_devices_tec_cool

B_update_devices_tec_hold:
                    LDHX  #str_B_top_hold
                    BRA         B_update_devices_tec_write

B_update_devices_tec_heat:
                    LDHX  #str_B_top_heat
                    BRA         B_update_devices_tec_write

B_update_devices_tec_cool:
                    LDHX  #str_B_top_cool
                    BRA         B_update_devices_tec_write

B_update_devices_tec_write:
                    LDA         #$10
                    JSR         lcd_str

                    ; write LCD bottom row
                    JSR         lcd_goto_row1
```

```
                    LDHX  #str_B_bottom
                    LDA         #$10
                    JSR         lcd_str

; write LM92 temp
                    ; set LCD cursor position
                    LDA         #$C5
                    JSR         lcd_goto_addr
                    JSR         lm92_write_lcd_C

; write Tset temp
                    ; set LCD cursor position
                    LDA         #$8E
                    JSR         lcd_goto_addr

                    ; write upper number to LCD
                    LDA         Tset
                    LDHX  #$000A
                    DIV                                 ; A <= (H:A)/(X), H <=
(remainder)
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    ; write upper number to LCD
                    PSHH
                    PULA
                    JSR         lcd_num_to_char
                    JSR         lcd_char

; write time

                    ; set LCD cursor position
                    LDA         #$CB
                    JSR         lcd_goto_addr

                    ; read time from RTC
                    JSR         rtc_get_time

                    ; calc and write TOD
                    JSR         rtc_calc_tod
                    JSR         rtc_write_tod

                    BRA   B_update_devices_done

B_update_devices_done:
;*** done
                    MOV         #$00, update_needed
                    RTS

;************************************************************
```

```
;*************************************************************
;* Subroutine Name: SUB_delay
;* Description: Decrements SUB_delay_cnt until it reaches zero.
;*                      1 count in SUB_delay_cnt is approx 4.019 us
;*
;* Registers Modified: None.
;* Entry Variables: SUB_delay_cnt - 3 byte variable, determines length
;*                           of time the SUB_delay routine will take to execute.
;* Exit Variables: SUB_delay_cnt - will be zero at exit.
;*************************************************************
SUB_delay:
                    ; save the existing values of registers
                    PSHH
                    PSHX
                    PSHA

                    ; load address of SUB_delay_cnt
                    LDHX #SUB_delay_cnt

SUB_delay_loop_0:

                    feed_watchdog

                    ; if byte[0] == 0
                    LDA   0, X
                    BEQ         SUB_delay_loop_1        ; jump to SUB_delay_outer_loop

                    ;else
                    DECA                                ; decrement byte[0]
                    STA         0, X

                    ;repeat
                    BRA SUB_delay_loop_0

SUB_delay_loop_1:

                    ; if byte[1] == 0
                    LDA   1, X
                    BEQ         SUB_delay_loop_2        ; branch to done

                    ;else
                    DECA                                ; decrement byte[1]
                    STA         1, X

                    LDA         #$FF                    ; reset byte[0]
                    STA         0,X

                    ;repeat
                    BRA SUB_delay_loop_0

SUB_delay_loop_2:

                    ; if byte[2] == 0
                    LDA   2, X
                    BEQ         SUB_delay_done          ; branch to done

                    ;else
                    DECA                                ; decrement byte[2]
                    STA         2, X

                    LDA         #$FF                    ; reset byte[1]
                    STA         1, X
```

```
                LDA         #$FF                        ; reset byte[0]
                STA         0, X

                ;repeat
                BRA SUB_delay_loop_0

SUB_delay_done:

                ; restore registers to previous values
                PULA
                PULX
                PULH

                RTS
;***********************************************************
```

```
;*************************************************************
;* File Name    :       bus.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for controlling the
;*                            bus.
;*
;*************************************************************

; EQU statements
mDataBus          EQU   $F0         ; Mask for the data bus pins on PortB
mAddrBus          EQU   $0F         ; Mask for the address bus pins on PortB


; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF bus_init, bus_write, bus_read, bus_addr, bus_data

; import symbols
                  XREF _Startup, main, _Vtpmov


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  bus_addr:         DS.B  1     ; only use lower 3 bits
                  bus_data:         DS.B  1     ; only use lower 4 bits

; code section
MyCode:     SECTION

;*************************************************************
;* Subroutine Name: bus_init
;* Description: Reads data from the device whose address is
;*                      the lower 3 bits of bus_addr, and store the
;*                      data to the lower 4 bits of bus_data.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
bus_init:
                  ; preserve registers
                  PSHA

                  ;*** init Data & Address Busses ***
                  LDA         mAddrBus                        ; Set Address Bus pins as
output by default, leave data as input
                  STA         PTBDD
                  LDA         $00                                   ; Leave all of PortB
as input at start
                  STA         PTBD

                  ; restore registers
                  PULA

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: bus_read
;* Description: Reads data from the device whose address is
;*                      the lower 3 bits of bus_addr, and store the
;*                      data to the lower 4 bits of bus_data.
;*
;* Registers Modified: None
;* Entry Variables: bus_addr
;* Exit Variables: bus_data
;*************************************************************
bus_read:
                  ; preserve accumulator A
                  PSHA

                  ; make address bus output, data bus an input
            LDA         #mAddrBus
            STA         PTBDD

                  ; pull the address low
            LDA   bus_addr                ; load address
            AND         #$07                    ; mask off the lower 3 bits to be
sure, will leave G2A low
            STA         PTBD                    ; write data to address bus, and clear
data bus

            ; read data from the bus
            LDA         PTBD
            NSA                                 ; shift data down to the lower 4 bits
            AND         #$0F                    ; mask off the lower 4 bits to be sure
            STA         bus_data                ;

                  ; pull the address high
            LDA   #$08                    ; G2A_not high
            STA         PTBD                    ; write, clears address bus

                  ; restore accumulator A
                  PULA

                  ; return from subroutine bus_read
                  RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: bus_write
;* Description: Writes the lower 4 bits of bus_data to the
;*                      device on whose address is the lower 3 bits
;*                      of bus_addr.
;* Registers Modified: None
;* Entry Variables: bus_addr, bus_data
;* Exit Variables: None
;*************************************************************
bus_write:
                ; preserve accumulator A
                PSHA

                ; make data and address busses outputs
            LDA         #$FF
            STA         PTBDD

            ; prep data for the bus
            LDA         bus_data
            NSA                                 ; swap the lower 4 bits to be the
upper 4 bits
            AND         #$F0                    ; mask off the upper 4 bits to be sure

                ; prep the addr, G2A_not low, Yx goes low
            ORA         bus_addr                ; add in the address
            STA         PTBD                    ; write data and address bus, with
G2A_not low

            ORA         #$08                    ; leave data and address, set G2A_not
high - Yx goes high
            STA         PTBD

                ; restore accumulator A
                PULA

                ; return from subroutine bus_write
                RTS

;*************************************************************
```

```
;*************************************************************
;* File Name    :       i2c_driver.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-25
;* Description  :       Contains subroutines for a bit-banging
;*                            software I2C driver, based on AN1820.
;*
;*************************************************************

; EQU statements
SCL         EQU 3             ;Serial clock bit number
SDA         EQU 2             ;Serial data bit number

; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte
            ;XDEF

; import symbols
                  ;XREF


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  BitCounter:       DS.B  1           ; Used to count bits in a Tx
                  Value:                  DS.B  1           ; Used to store rx data
value

; code section
MyCode:     SECTION

;*************************************************************
;* Subroutine Name: i2c_init
;* Description: Initilizes the software I2C driver.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_init:
                  ;Initialize variables
                  CLR   Value                   ;Clear all RAM variables
                  CLR   BitCounter

                  ;*** init SDA and SCL pins as outputs
                  BSET  SDA, PTADD
                  BSET  SCL, PTADD

                  ;*** init SDA and SCL pins to high
                  BSET  SDA, PTAD
                  BSET  SCL, PTAD

                  RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: i2c_start
;* Description: Generate a START condition on the bus.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_start:
                ; crate falling edge on SDA while SCL high
                BCLR  SDA, PTAD
                JSR   i2c_bit_delay
                BCLR  SCL, PTAD
                RTS

;*************************************************************

;*************************************************************
;* Subroutine Name: i2c_stop
;* Description: Generate a STOP condition on the bus.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_stop:
                ; crate rising edge on SDA while SCL high
                BCLR  SDA, PTAD
                BSET  SCL, PTAD
                BSET  SDA, PTAD
                JSR   i2c_bit_delay
                RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: i2c_tx_byte
;* Description: Transmit the byte in Acc to the SDA pin
;*                      (Acc will not be restored on return)
;*
;*                      Must be careful to change SDA values only
;*                      while SCL is low, otherwise a STOP or START
;*                      could be implied.
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_tx_byte:
                ;Initialize variable
                LDX   #$08
                STX   BitCounter

tx_nextbit:
                ROLA                                    ; Shift MSB into Carry
                BCC         tx_send_low                 ; Send low bit or high bit

tx_send_high:
                BSET  SDA, PTAD                 ; set the data bit value
                JSR         i2c_setup_delay             ; Give some time for data

tx_setup:
                BSET  SCL, PTAD                 ; clock in data
                JSR         i2c_bit_delay               ; wait a bit
                BRA         tx_continue                 ; continue

tx_send_low:
                BCLR  SDA, PTAD                 ; set the data bit value
                JSR         i2c_setup_delay             ; Give some time for data
                BRA         tx_setup                        ; clock in the bit

tx_continue:
                BCLR  SCL, PTAD                 ; Restore clock to low state
                DEC         BitCounter                  ; Decrement the bit counter
                BEQ         tx_ack_poll                 ; Last bit?
                BRA         tx_nextbit                  ; Do the next bit

tx_ack_poll:
                BSET  SDA, PTAD
                BCLR  SDA, PTADD                ; Set SDA as input
                JSR   i2c_setup_delay           ; wait

                BSET  SCL, PTAD                 ; clock the line
                JSR         i2c_bit_delay               ; wait

                BRCLR SDA, PTAD, tx_done        ; check SDA for ack

tx_no_ack:
                ; do error handling here

tx_done:
                BCLR  SCL, PTAD                 ; restore the clock line
                BSET  SDA, PTADD                ; SDA back to output
                RTS                                             ; done
;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: i2c_rx_byte
;* Description: Recieves a byte from the I2C bus.
;*                      Will Ack the byte if Accu A != 0
;*                      Data returned in Accu A
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_rx_byte:

                ; clear output var
                CLR         Value

                ; set BitCounter
                LDX   #$08
                STX   BitCounter

                ; set SDA to input and pull clock low
                BCLR  SDA, PTADD
                BCLR  SCL, PTAD

rx_nextbit:
                ; wait for a bit
                JSR         i2c_bit_delay

                ; shift the last bit recieved left (and fill LSB with zero)
                LSL         Value

                ; clock the line and wait
                BSET  SCL, PTAD
                JSR         i2c_setup_delay

                ; grab bit from bus
                BRCLR SDA, PTAD, rx_low

rx_high:
                ; store a 1 to Value
                BSET  0, Value
                BRA         rx_continue

rx_low:
                ; do nothing since LSL fills with 0

rx_continue:
                BCLR  SCL, PTAD               ; Restore clock to low state
                DEC         BitCounter              ; Decrement the bit counter
                BNE         rx_nextbit              ; More bits?


                ; set SDA back to output
                BSET  SDA, PTADD

                ; test Accu A == 0
                CBEQA #$00, rx_nack
                BRA         rx_ack

rx_ack:
                ; clear data bit to acknowledge
                BCLR  SDA, PTAD
                BRA         rx_done
```

```
rx_nack:
                    ; set data bit to not acknowledge
                    BSET  SDA, PTAD

rx_done:
                    ; let ack/nack settle
                    JSR         i2c_setup_delay

                    ;clock the ack/nack
                    BSET  SCL, PTAD
                    JSR         i2c_bit_delay

                    ; retun clock to low
                    BCLR  SCL, PTAD

                    ; load Value into Accu A
                    LDA         Value

                    RTS


;*************************************************************

;*************************************************************
;* Subroutine Name: i2c_setup_delay
;* Description: Provide some data setup time to allow
;*                      SDA to stabilize in slave device
;*                      Completely arbitrary delay (10 cycles?)
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_setup_delay:

                    NOP
                    NOP
                    RTS

;*************************************************************

;*************************************************************
;* Subroutine Name: i2c_setup_delay
;* Description: Bit delay to provide (approximately) the desired
;*                      SCL frequency
;*                      Again, this is arbitrary (16 cycles?)
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
i2c_bit_delay:

                    NOP
                    NOP
                    NOP
                    NOP
                    NOP
                    RTS

;*************************************************************
```

```
;*****************************************************************
;* File Name    :       keypad.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for reading the
;*                            keypad.
;*
;*****************************************************************

; EQU statements


; Include derivative-specific definitions
            INCLUDE 'derivative.inc'

; export symbols
            XDEF keypad_interpret, keypad_scan, keypad_data_0, keypad_data_1,
keypad_data_0_old, keypad_data_1, keypad_data_cmp, keypad_get_keypress

; import symbols
                  XREF bus_read, bus_write, bus_addr, bus_data
                  XREF led_write, led_data
                  XREF lcd_char


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  keypad_data_0:    DS.B  1     ; bit flags representing what keys are
pressed on they 4x4 keypad
                  keypad_data_1:    DS.B  1

                  keypad_data_0_old:      DS.B  1     ; bit flags representing which
keys were pressed on the keypad, the last time it was scanned
                  keypad_data_1_old:      DS.B  1

                  keypad_data_cmp:  DS.B  1     ; tempory holder for keypad data
comparison in keypad_interpret

; code section
MyCode:     SECTION
```

```
;*************************************************************
;* Subroutine Name: keypad_scan
;* Description: Scans the greyhill 4x4 keypad, and saves the
;*                      result to variable.
;*                      Note that this method will overwrite values in
;*                      the bus_addr and bus_data variables.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: keypad_data_0, keypad_data_1
;*************************************************************
keypad_scan:
                  ; preserve registers
                  PSHA

;*** save old value of keypad_data, before we overwrite it

                  LDA         keypad_data_0
                  STA         keypad_data_0_old
                  LDA         keypad_data_1
                  STA         keypad_data_1_old

;*** scan row 0 ***

            ;* set row 0 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00001110
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                  LDA         bus_data          ; load in data nibble
                  COMA                          ; compliment bits, so 1=button press
                  AND         #$0F              ; mask off the lower 4 bits
                  STA         keypad_data_0     ; store to vairable


;*** scan row 1 ***

            ;* set row 1 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr
```

```
            ; set the data
            LDA   #%00001101
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                  LDA         bus_data          ; load in data nibble
                  COMA                          ; compliment bits, so 1=button press
                  NSA                                 ; swap our data to the upper
nibble
                  AND         #$F0              ; mask off the data
                  ORA         keypad_data_0     ; add the lower 4 bits in
                  STA         keypad_data_0     ; store to vairable


;*** scan row 2 ***

            ;* set row 2 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00001011
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                  LDA         bus_data          ; load in data nibble
                  COMA                          ; compliment bits, so 1=button press
                  AND         #$0F              ; mask off the lower 4 bits
                  STA         keypad_data_1     ; store to vairable


;*** scan row 3 ***

            ;* set row 3 to low, other rows to high *
```

```
            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00000111
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                  LDA         bus_data          ; load in data nibble
                  COMA                          ; compliment bits, so 1=button press
                  NSA                                 ; swap our data to the upper
nibble
                  AND         #$F0              ; mask off the data
                  ORA         keypad_data_1     ; add the lower 4 bits in
                  STA         keypad_data_1     ; store to vairable


;*** done ***

                  ; restore registers
                  PULA

                  ; return from subroutine keypad_scan
                  RTS

;************************************************************
```

```
;*************************************************************
;* Subroutine Name: keypad_interpret
;* Description: Checks if a numeric key (1..9) was pressed.
;*                      When a key is pressed, it writes it to the LCD
;*                      and returns the numeric value in Accu A.
;*                      Returns 0xFF when (1..9) was not pressed.
;*
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: Accu A
;*************************************************************
keypad_interpret:

;*** was a key pressed in the first 2 rows ? ***

                    LDA         keypad_data_0_old
                    COMA
                    AND         keypad_data_0
                    CBEQA #$00, keypad_interpret_lower_rows_jump

                    ; key was pressed
                    STA         keypad_data_cmp

keypad_interpret_1:

                    ; was '1' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11111110
                    BNE         keypad_interpret_2

                    ; write a '1' to the LCD
                    LDA         #'1'
                    JSR         lcd_char

                    ; return 0x01
                    LDA         #$01
                    RTS


keypad_interpret_2:

                    ; was '2' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11111101
                    BNE         keypad_interpret_3

                    ; write a '2' to the LCD
                    LDA         #'2'
                    JSR         lcd_char

                    ; return 0x02
                    LDA         #$02
                    RTS

keypad_interpret_3:

                    ; was '3' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11111011
                    BNE         keypad_interpret_A
```

```
                    ; write a '3' to the LCD
                    LDA         #'3'
                    JSR         lcd_char

                    ; return 0x03
                    LDA         #$03
                    RTS


keypad_interpret_A:

                    ; was 'A' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11110111
                    BNE         keypad_interpret_4

                    ; write a 'A' to the LCD
                    LDA         #'A'
                    JSR         lcd_char

                    ; return 0x0A
                    LDA         #$0A
                    RTS

                    BRA         keypad_interpret_4
keypad_interpret_lower_rows_jump:
                    BRA         keypad_interpret_lower_rows


keypad_interpret_4:

                    ; was '4' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11101111
                    BNE         keypad_interpret_5

                    ; write a '4' to the LCD
                    LDA         #'4'
                    JSR         lcd_char

                    ; return 0x04
                    LDA         #$04
                    RTS


keypad_interpret_5:

                    ; was '5' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11011111
                    BNE         keypad_interpret_6

                    ; write a '5' to the LCD
                    LDA         #'5'
                    JSR         lcd_char

                    ; return 0x05
                    LDA         #$05
                    RTS


keypad_interpret_6:
```

```
                    ; was '6' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%10111111
                    BNE         keypad_interpret_B

                    ; write a '6' to the LCD
                    LDA         #'6'
                    JSR         lcd_char

                    ; return 0x06
                    LDA         #$06
                    RTS


keypad_interpret_B:

                    ; was 'B' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%01111111
                    BNE         keypad_interpret_lower_rows

                    ; write a 'B' to the LCD
                    LDA         #'B'
                    ;JSR        lcd_char

                    ; return 0x0B
                    LDA         #$0B
                    RTS



keypad_interpret_lower_rows:
;*** was a key pressed in the second 2 rows ? ***

                    LDA         keypad_data_1_old
                    COMA
                    AND         keypad_data_1
                    CBEQA #$00, keypad_interpret_done_jump

                    ; key was pressed
                    STA         keypad_data_cmp


keypad_interpret_7:

                    ; was '7' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11111110
                    BNE         keypad_interpret_8

                    ; write a '7' to the LCD
                    LDA         #'7'
                    JSR         lcd_char

                    ; return 0x07
                    LDA         #$07
                    RTS


keypad_interpret_8:
```

```
                    ; was '8' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11111101
                    BNE         keypad_interpret_9

                    ; write a '8' to the LCD
                    LDA         #'8'
                    JSR         lcd_char

                    ; return 0x08
                    LDA         #$08
                    RTS


keypad_interpret_9:

                    ; was '9' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11111011
                    BNE         keypad_interpret_C

                    ; write a '9' to the LCD
                    LDA         #'9'
                    JSR         lcd_char

                    ; return 0x09
                    LDA         #$09
                    RTS


keypad_interpret_C:

                    ; was 'C' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11110111
                    BNE         keypad_interpret_E

                    ; write a 'C' to the LCD
                    LDA         #'C'
                    ;JSR        lcd_char

                    ; return 0x0C
                    LDA         #$0C
                    RTS


                    BRA   keypad_interpret_E
keypad_interpret_done_jump:
                    BRA         keypad_interpret_done


keypad_interpret_E:

                    ; was 'E'/'*' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11101111
                    BNE         keypad_interpret_0

                    ; write a 'E' to the LCD
                    LDA         #'*'
                    JSR         lcd_char
```

```
                    ; return 0x0E
                    LDA         #$0E
                    RTS


keypad_interpret_0:

                    ; was '0' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%11011111
                    BNE         keypad_interpret_F

                    ; write a '0' to the LCD
                    LDA         #'0'
                    JSR         lcd_char

                    ; return 0x00
                    LDA         #$00
                    RTS


keypad_interpret_F:

                    ; was 'F'/'#' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%10111111
                    BNE         keypad_interpret_D

                    ; write a 'F' to the LCD
                    LDA         #'#'
                    JSR         lcd_char

                    ; return 0x00
                    LDA         #$0F
                    RTS


keypad_interpret_D:

                    ; was 'D' pressed ?
                    LDA         keypad_data_cmp
                    AND         #%01111111
                    BNE         keypad_interpret_done

                    ; write a 'D' to the LCD
                    LDA         #'D'
                    JSR         lcd_char

                    ; return 0x0D
                    LDA         #$0D
                    RTS



keypad_interpret_done:
;*** done ***

                    ; return $FF to indicate no key pressed
                    LDA         #$FF
                    RTS

;************************************************************
```

```
;*************************************************************
;* Subroutine Name: keypad_get_keypress
;* Description: Continously scans and interprets the keypad
;*                      until a key is pressed.
;*
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: Accu A
;*************************************************************
keypad_get_keypress:

                ; feed watchdog
                feed_watchdog

                ; update heatbeat led
                JSR         led_write

                ; scan the keypad
                JSR         keypad_scan

                ; check for keypress
                JSR         keypad_interpret

                ; if no key pressed, repeat
                CBEQA #$FF, keypad_get_keypress

                ; key was pressed, so we're done
                RTS


;*************************************************************
```

```
;*****************************************************************
;* File Name    :       lcd.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for controlling the
;*                            lcd.
;*
;*****************************************************************

; EQU statements


; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char, lcd_clear,
lcd_goto_addr, lcd_goto_row0, lcd_goto_row1
            XDEF lcd_data, lcd_char_data, lcd_col_idx

; import symbols
                  XREF SUB_delay, SUB_delay_cnt
                  XREF bus_read, bus_write, bus_addr, bus_data


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  lcd_data:         DS.B  1     ; lower 4 bits = LCD data lines, bit 6
= RS, bit 5 = RW
                  lcd_char_data:    DS.B  1     ; used by lcd_char subroutine to store
a character
                  lcd_col_idx:      DS.B  1     ; index of the column of the LCD that
the cursor is currently in

                  lcd_addr:         DS.B  1     ; holds an address for lcd_goto_addr

                  str_length:       DS.B  1     ; holds the offset into a string for
lcd_str

; code section
MyCode:     SECTION
```

```
;*************************************************************
;* Subroutine Name: lcd_init
;* Description: Initilizes the LCD.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
lcd_init:
                ; preserve registers
                PSHA

;*** init RS and RW pins as outputs
                LDA         PTADD
                ORA         #$03
                STA         PTADD

;*** wait for 15 ms
                ; load address of SUB_delay_cnt
                LDHX #SUB_delay_cnt

                ; configure loop delays: 0x001388 = 20 ms
                LDA         #$00
                STA         2,X
                LDA         #$13
                STA         1,X
                LDA         #$88
                STA         0,X

                ; jump to the delay loop
                JSR         SUB_delay


;*** Send Init Command

                LDA         #$03
                JSR         lcd_write

;*** Wait for 4.1 ms
                ; load address of SUB_delay_cnt
                LDHX #SUB_delay_cnt

                ; configure loop delays: 0x001388 = 20 ms
                LDA         #$00
                STA         2,X
                LDA         #$13
                STA         1,X
                LDA         #$88
                STA         0,X

                ; jump to the delay loop
                JSR         SUB_delay


;*** Send Init command

                LDA         #$03
                JSR         lcd_write


;*** Wait for 100 us
                ; load address of SUB_delay_cnt
```

```
                LDHX #SUB_delay_cnt

                ; configure loop delays: 0x001388 = 20 ms
                LDA         #$00
                STA         2,X
                LDA         #$13
                STA         1,X
                LDA         #$88
                STA         0,X

                ; jump to the delay loop
                JSR         SUB_delay


;*** Send Init command

                LDA         #$03
                JSR         lcd_write

;*** Send Function set command

                LDA         #$02
                JSR         lcd_write

                LDA         #$02
                JSR         lcd_write

                LDA         #$08
                JSR         lcd_write


;*** Send display ctrl command

                LDA         #$00
                JSR         lcd_write

                LDA         #$0C
                JSR         lcd_write

;*** Send display clear command

                LDA         #$00
                JSR         lcd_write

;*** Wait for 5 ms
                ; load address of SUB_delay_cnt
                LDHX #SUB_delay_cnt

                ; configure loop delays: 0x001388 = 20 ms
                LDA         #$00
                STA         2,X
                LDA         #$13
                STA         1,X
                LDA         #$88
                STA         0,X

                ; jump to the delay loop
                JSR         SUB_delay

;*** Send display clear command

                LDA         #$01
```

```
                JSR         lcd_write

;*** Wait for 5 ms
                ; load address of SUB_delay_cnt
                LDHX #SUB_delay_cnt

                ; configure loop delays: 0x001388 = 20 ms
                LDA         #$00
                STA         2,X
                LDA         #$13
                STA         1,X
                LDA         #$88
                STA         0,X

                ; jump to the delay loop
                JSR         SUB_delay


;*** Send entry mode command

                LDA         #$00
                JSR         lcd_write

                LDA         #$06
                JSR         lcd_write

;*** done ***

                ; restore registers
                PULA

                ; return from subroutine lcd_init
                RTS

;***************************************************************
```

```
;*****************************************************************
;* Subroutine Name: lcd_write
;* Description: Sends data to the LCD.
;*
;* Registers Modified: Accu A
;* Entry Variables: Accu A
;* Exit Variables:
;*****************************************************************
lcd_write:
                    ; preserve HX register
                    PSHH
                    PSHX

                    ; store param to var for latter
                    STA         lcd_data

                    ; clear RS and RW pins on PTAD
                    LDA   PTAD
                    AND         #$FC
                    STA         PTAD

                    ; put RS an RW on PTAD
                    LDA         lcd_data
                    NSA
                    AND   #$03
                    ORA         PTAD
                    STA         PTAD

                    ; prep bus data
                    LDA         lcd_data
                    AND         #$0F
                    STA         bus_data
                    ; prep bus addr
                    LDA         #$04
                    STA         bus_addr
                    ; write data to bus (and clock the addr)
                    JSR         bus_write


;*** Wait for 40 us
                    ; load address of SUB_delay_cnt
                    LDHX #SUB_delay_cnt

                    ; configure loop delays: 0x00000A = 40 us
                    LDA         #$00
                    STA         2,X
                    LDA         #$00
                    STA         1,X
                    LDA         #$0A
                    STA         0,X

                    ; jump to the delay loop
                    JSR         SUB_delay

                    ; restore HX register
                    PULX
                    PULH

                    ; done
                    RTS

;*****************************************************************
```

```
;*************************************************************
;* Subroutine Name: lcd_char
;* Description: Writes a character to the LCD.
;*                      If lcd_col_idx is off of the first line, the
;*                      LCD will be cleared and the new char will be
;*                      written to the first column of row 0
;*
;* Registers Modified: Accu A
;* Entry Variables: Accu A
;* Exit Variables:
;*************************************************************
lcd_char:
                    ; preserve registers
                    PSHH
                    PSHX

                    ; save data
                    STA         lcd_char_data

                    ; write upper nibble
                    NSA
                    AND         #$0F
                    ORA         #$20
                    JSR         lcd_write

                    ; write lower nibble
                    LDA         lcd_char_data
                    AND         #$0F
                    ORA         #$20
                    JSR         lcd_write

;*** Wait for 1 ms ***
                    LDHX #SUB_delay_cnt

                    ; configure loop delays: 0x0000FA = 1 ms
                    LDA         #$00
                    STA         2,X
                    LDA         #$00
                    STA         1,X
                    LDA         #$FA
                    STA         0,X

                    ; jump to the delay loop
                    JSR         SUB_delay

                    ; done
                    PULX
                    PULH
                    RTS


;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lcd_str
;* Description: Writes a 0x00 terminated string of bytes to
;*                      the lcd, starting at the address in the HX
;*                      register. Does not keep track of location
;*                      on lcd.
;*
;* Registers Modified: Accu A, HX
;* Entry Variables: HX, A
;* Exit Variables:  none
;*************************************************************
lcd_str:
                ; save str length
                STA         str_length

lcd_str_loop:
                ; get data
                LDA         0,X

                ; write data to lcd
                JSR         lcd_char

                ; increament lower byte X
                PSHX
                PULA
                ADD         #01
                PSHA
                PULX

                ; increment upper byte H
                PSHH
                PULA
                ADC         #$00
                PSHA
                PULH

                ; decrement str_length
                LDA         str_length
                DECA
                STA         str_length

                ; repeat if length != 0
                BNE         lcd_str_loop

lcd_str_done:

                RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lcd_num_to_char
;* Description: Takes a number in Accu A and converts it to the
;*                      ASCII representation of that number. Only works
;*                      for lower for bits of Accu A.
;*
;* Registers Modified: None.
;* Entry Variables: Accu A
;* Exit Variables: Accu A
;*************************************************************
lcd_num_to_char:
                  ; Add 0x30
                  ADD         #$30
                  RTS

;*************************************************************


;*************************************************************
;* Subroutine Name: lcd_clear
;* Description: Sends the clear command to the lcd and waits
;*                      for it to clear (20 ms).
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
lcd_clear:

;*** Wait for 20 ms ***
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay

                  ; Send display clear command
                  LDA         #$00
                  JSR         lcd_write
                  LDA         #$01
                  JSR         lcd_write

;*** Wait for 20 ms ***
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay
```

```
                    ; done
                    RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lcd_goto_addr
;* Description: Commands the LCD to put the cursor at the
;*                      location given in Accu A.
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
lcd_goto_addr:

                  ; store addr
                  STA         lcd_addr

                  ; write upper nibble
                  NSA
                  AND         #$0F
                  JSR         lcd_write

                  ; write lower nibble
                  LDA         lcd_addr
                  AND         #$0F
                  JSR         lcd_write

                  RTS

;*************************************************************


;*************************************************************
;* Subroutine Name: lcd_goto_row0
;* Description: Commands the LCD to put the cursor at colum 0
;*                      of row 0.
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
lcd_goto_row0:

                  ; go back to first column and row of LCD
                  LDA         #$08
                  JSR         lcd_write
                  LDA         #$00
                  JSR         lcd_write

                  RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lcd_goto_row1
;* Description: Commands the LCD to put the cursor at colum 0
;*                      of row 1.
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
lcd_goto_row1:

                ; go back to first column and row of LCD
                LDA         #$0C
                JSR         lcd_write
                LDA         #$00
                JSR         lcd_write

                RTS

;*************************************************************
```

```
;*************************************************************
;* File Name    :       led.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for controlling the
;*                            DFF-driven LEDs.
;*
;*************************************************************

; EQU statements


; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF led_write, led_data

; import symbols
                  XREF bus_read, bus_write, bus_addr, bus_data


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  led_data:         DS.B  1     ; 8 bit value for the 8 LEDs

; code section
MyCode:     SECTION
```

```
;*************************************************************
;* Subroutine Name: led_write
;* Description: Writes the 8 bits of led_data two the 8 LEDs
;*                      on the DFFs at address 0 and 1 on the bus
;*
;* Registers Modified: None
;* Entry Variables: led_data
;* Exit Variables: None
;*************************************************************
led_write:
                ; preserve accumulator A
                PSHA

;*** write lower nibble LEDs ***
                ; set the address
            LDA   #$00
            STA         bus_addr

            ; set the data
            LDA   led_data
            AND         #$0F
            STA         bus_data

            ; write the data
            JSR         bus_write

;*** write upper nibble LEDs ***
                ; set the address
            LDA   #$01
            STA         bus_addr

            ; set the data
            LDA   led_data
            NSA
            AND         #$0F
            STA         bus_data

            ; write the data
            JSR         bus_write

;*** done ***
                ; restore accumulator A
                PULA
                RTS


;*************************************************************
```

```
;*************************************************************
;* File Name    :       rtc_driver.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-27
;* Description  :       Contains subroutines talking to a DS1337
;*                            Real Time Clock, using i2c_driver.asm
;*
;*************************************************************

; EQU statements

LM92_ADDR_W             EQU $90     ; Slave address to write to LM92
LM92_ADDR_R             EQU $91     ; Slave address to read from LM92

LM92_REG_TEMP           EQU   $00         ; register address of the seconds register

; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF lm92_init, lm92_read_temp, lm92_write_lcd_K, lm92_write_lcd_C

; import symbols
                  XREF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte

                  XREF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char,
lcd_clear, lcd_goto_addr, lcd_goto_row0, lcd_goto_row1



; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  Temp_Data_Raw:          DS.B  2

                  Temp_c:                       DS.B  1
                  temp_k:                       DS.B  1

MY_CONST: SECTION
; Constant Values and Tables Section

                  ;str_date:              DC.B  "Date is "
                  ;str_date_length: DC.B  8


; code section
MyCode:     SECTION

;*************************************************************
;* Subroutine Name: lm92_init
;* Description: Initilizes the LM92 digital temperature
;*                      sensor driver.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
lm92_init:
                  ; nothing to see here
                  RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lm92_read_temp
;* Description: Read temperature from LM92 and converts to
;*                      degrees C. The result is returned in Accu A.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: Accu A
;*************************************************************
lm92_read_temp:

                    ; start condition
                    JSR         i2c_start

                    ; send rtc write addr
                    LDA         #LM92_ADDR_W
                    JSR   i2c_tx_byte

                    ; send register address
                    LDA         #LM92_REG_TEMP
                    JSR   i2c_tx_byte

                    ; stop condition
                    JSR         i2c_stop


                    ; start condition
                    JSR         i2c_start

                    ; send rtc read addr
                    LDA         #LM92_ADDR_R
                    JSR   i2c_tx_byte

                    ; read byte
                    LDA         #$01              ; ack the byte
                    JSR         i2c_rx_byte
                    STA         Temp_Data_Raw+0

                    ; read byte
                    LDA         #$00              ; nack the byte
                    JSR         i2c_rx_byte
                    STA         Temp_Data_Raw+1

                    ; stop condition
                    JSR         i2c_stop


                    ; divide by 16 to convert to degrees C
                    LDHX  Temp_Data_Raw+0
                    LDX         #$80
                    LDA         Temp_Data_Raw+1

                    DIV         ; A <- (H:A)/(X)

                    STA         Temp_c

                    ; done
                    RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lm92_write_lcd_K
;* Description: Writes the temperature in Accu A to the LCD
;*                      in degrees Kelvin
;*
;* Registers Modified: Accu A
;* Entry Variables: Temp_c
;* Exit Variables: None
;*************************************************************
lm92_write_lcd_K:

                    LDA         Temp_c

                    ; temp >= 27 C == 300 K?
                    CMP         #$1B
                    BLO         k_small

k_big:
                    ; convert to K
                    SUB         #$1B
                    STA         temp_k

                    ; write 3 for 300K
                    LDA         #'3'
                    JSR         lcd_char
                    BRA         cont

k_small:
                    ; convert to K
                    ADD         #$49
                    STA         temp_k

                    ; write 2 for 200K
                    LDA         #'2'
                    JSR         lcd_char

cont:
                    LDA         temp_k

                    ; write upper number to LCD
                    LDHX  #$000A
                    DIV                                 ; A <= (H:A)/(X), H <=
(remainder)
                    ; convert to ASCII char
                    JSR         lcd_num_to_char

                    ; write to LCD
                    JSR         lcd_char

                    ; move remainder from H to A
                    PSHH
                    PULA

                    ; convert to ASCII char
                    JSR         lcd_num_to_char

                    ; write to LCD
                    JSR         lcd_char

                    ; done
                    RTS
;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: lm92_write_lcd_K
;* Description: Writes the temperature in Accu A to the LCD
;*                      in degrees Kelvin
;*
;* Registers Modified: Accu A
;* Entry Variables: Temp_c
;* Exit Variables: None
;*************************************************************
lm92_write_lcd_C:

                  LDA         Temp_c

                  ; write upper number to LCD
                  LDHX  #$000A
                  DIV                                 ; A <= (H:A)/(X), H <=
(remainder)

                  ; convert to ASCII char
                  JSR         lcd_num_to_char

                  ; write to LCD
                  JSR         lcd_char

                  ; move remainder from H to A
                  PSHH
                  PULA

                  ; convert to ASCII char
                  JSR         lcd_num_to_char

                  ; write to LCD
                  JSR         lcd_char


                  ; done
                  RTS

;*************************************************************
```

```
;*************************************************************
;* File Name    :       rtc_driver.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-27
;* Description  :       Contains subroutines talking to a DS1337
;*                            Real Time Clock, using i2c_driver.asm
;*
;*************************************************************

; EQU statements

RTC_ADDR_W        EQU $D0     ; Slave address to write to RTC
RTC_ADDR_R        EQU $D1     ; Slave address to read from RTC
RTC_REG_SEC       EQU   $00         ; register address of the seconds register

; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF rtc_init, rtc_set_time_zero, rtc_calc_tod, rtc_write_tod,
rtc_set_time, rtc_get_time, rtc_display_data, rtc_prompt_time
            XDEF Sec, Min, Hour, Date, Month, Year

; import symbols
                  XREF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte

                  XREF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char,
lcd_clear, lcd_goto_addr, lcd_goto_row0, lcd_goto_row1
            XREF lcd_data, lcd_char_data, lcd_col_idx

            XREF keypad_get_keypress


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  Sec:                    DS.B  2           ; normally holds last time
read from RTC,
                  Min:                    DS.B  2           ; written to prior to
calling rtc_set_time
                  Hour:                   DS.B  2
                  Date:                   DS.B  2
                  Month:                        DS.B  2
                  Year:                   DS.B  2

                  TOD:                    DS.B  2           ; time of day in seconds
since midnight

                  rtc_epoch_delta:  DS.B  2

                  Byte_counter:           DS.B  1

MY_CONST: SECTION
; Constant Values and Tables Section

                  str_date:               DC.B  "Date is "
                  str_date_length:  DC.B  8
                  str_time:               DC.B  "Time is "
                  str_time_length:  DC.B  8

                  str_prompt_row0:              DC.B  "Set    : MM/DD/YY"
                  str_prompt_row0_length:       DC.B  16
```

```
str_prompt_row1:              DC.B  "Clock : HH:MM:SS"
str_prompt_row1_length:       DC.B  16


; code section
MyCode:     SECTION

;*************************************************************
;* Subroutine Name: rtc_init
;* Description: Initilizes the RTC driver.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_init:
                  ; set rtc
                  JSR         rtc_set_time_zero

                  RTS

;*************************************************************


;*************************************************************
;* Subroutine Name: rtc_set_time_zero
;* Description: Sets the RTC to time zero
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_set_time_zero:

                  MOV         #$00, Sec+0
                  MOV         #$00, Sec+1

                  MOV         #$00, Min+0
                  MOV         #$00, Min+1

                  MOV         #$00, Hour+0
                  MOV         #$00, Hour+1

                  MOV         #$00, Date+0
                  MOV         #$01, Date+1

                  MOV         #$00, Month+0
                  MOV         #$01, Month+1

                  MOV         #$00, Year+0
                  MOV         #$01, Year+1

                  JSR         rtc_set_time

                  ;done
                  RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: rtc_calc_tod
;* Description:
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_calc_tod:

                ; clear TOD var
                MOV         #$00, TOD+0
                MOV         #$00, TOD+1

;*** calculate seconds
                LDA         Sec+1
                STA         TOD+1

                LDA         Sec+0
                LDX         #$0A
                MUL                                 ; X:A <= (X) * (A)
                ADD         TOD+1
                STA         TOD+1

                PSHX
                PULA
                ADC         TOD+0
                STA         TOD+0

;*** calculate minutes
                LDA         Min+1
                LDX         #$3C            ; 0x3C = 60
                MUL                                 ; X:A <= (X) * (A)

                ; add lower byte of result
                ADD         TOD+1
                STA         TOD+1

                ; add upper byte of result
                PSHX
                PULA
                ADC         TOD+0
                STA         TOD+0

                ;done
                RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: rtc_write_tod
;* Description:
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: rtc_delta
;*************************************************************
rtc_write_tod:


;*** write upper number to LCD
                  LDHX  TOD+0
                  LDX         #$64
                  LDA         TOD+1
                  DIV                                 ; A <= (H:A)/(X), H <=
(remainder)

                  ; convert to ASCII char
                  JSR         lcd_num_to_char

                  ; write to LCD
                  JSR         lcd_char

;*** write middle number to LCD
                  PSHH
                  PULA

                  LDHX  #$000A
                  DIV                                 ; A <= (H:A)/(X), H <=
(remainder)

                  ; convert to ASCII char
                  JSR         lcd_num_to_char

                  ; write to LCD
                  JSR         lcd_char


;*** write lower number to LCD
                  PSHH
                  PULA

                  ; convert to ASCII char
                  JSR         lcd_num_to_char

                  ; write to LCD
                  JSR         lcd_char

                  ; done
                  RTS

;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: rtc_set_time
;* Description: Set the RTC with the current time in the Sec,
;*                      Min, etc var values
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_set_time:

                    ; start condition
                    JSR         i2c_start

                    ; send rtc write addr
                    LDA         #RTC_ADDR_W
                    JSR   i2c_tx_byte

                    ; send register address
                    LDA         #RTC_REG_SEC
                    JSR   i2c_tx_byte

                    ; send seconds data
                    LDA         Sec+0
                    NSA
                    AND         #$70
                    ORA         Sec+1
                    JSR   i2c_tx_byte

                    ; send minutes data
                    LDA         Min+0
                    NSA
                    AND         #$70
                    ORA         Min+1
                    JSR   i2c_tx_byte

                    ; send hours data
                    LDA         Hour+0
                    NSA
                    AND         #$30
                    ORA         Hour+1
                    JSR   i2c_tx_byte

                    ; send day of week (not used)
                    LDA         #$01
                    JSR   i2c_tx_byte

                    ; send date data
                    LDA         Date+0
                    NSA
                    AND         #$30
                    ORA         Date+1
                    JSR   i2c_tx_byte

                    ; send month data
                    LDA         Month+0
                    NSA
                    AND         #$10
                    ORA         #$80        ; set century bit
                    ORA         Month+1
                    JSR   i2c_tx_byte
```

```
                ; send year data
                LDA         Year+0
                NSA
                AND         #$F0
                ORA         Year+1
                JSR   i2c_tx_byte

                ; send stop condition
                JSR         i2c_stop


;*************************************************************
```

```
;*****************************************************************
;* Subroutine Name: rtc_get_time
;* Description: Get the RTC time and save to vars
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;*****************************************************************
rtc_get_time:

                    ; start condition
                    JSR         i2c_start

                    ; send rtc write addr
                    LDA         #RTC_ADDR_W
                    JSR   i2c_tx_byte

                    ; send register address
                    LDA         #RTC_REG_SEC
                    JSR   i2c_tx_byte

                    ; stop condition
                    JSR         i2c_stop


                    ; set byte counter to 6
                    MOV         #$06, Byte_counter

                    ; start condition
                    JSR         i2c_start

                    ; send rtc read addr
                    LDA         #RTC_ADDR_R
                    JSR   i2c_tx_byte

                    ; read seconds data
                    LDA         #$01              ; ack the byte
                    JSR         i2c_rx_byte
                    STA         Sec+1
                    NSA
                    STA         Sec+0

                    ; read minutes data
                    LDA         #$01              ; ack the byte
                    JSR         i2c_rx_byte
                    STA         Min+1
                    NSA
                    STA         Min+0

                    ; read hours data
                    LDA         #$01              ; ack the byte
                    JSR         i2c_rx_byte
                    STA         Hour+1
                    NSA
                    STA         Hour+0

                    ; read day of week data
                    LDA         #$01              ; ack the byte
                    JSR         i2c_rx_byte
                    ; we don't care about this

                    ; read date data
```

```
                LDA         #$01                ; ack the byte
                JSR         i2c_rx_byte
                STA         Date+1
                NSA
                STA         Date+0

                ; read month data
                LDA         #$01                ; ack the byte
                JSR         i2c_rx_byte
                STA         Month+1
                NSA
                STA         Month+0

                ; read Year data
                LDA         #$00                ; nack the byte
                JSR         i2c_rx_byte
                STA         Year+1
                NSA
                STA         Year+0

                ; stop condition
                JSR         i2c_stop

                ; mask off the recieved data
                JSR         rtc_mask_data

                RTS


;*************************************************************
```

```
;*************************************************************
;* Subroutine Name: rtc_display_data
;* Description: Takes the data in the Sec, Min, etc vars and
;*                      writes it to the lcd.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_display_data:

                    ; clear the lcd
                    JSR   lcd_clear

                    ; goto top row
                    JSR         lcd_goto_row0

                    ; write header
                    LDHX  #str_date
                    LDA         str_date_length
                    JSR         lcd_str

                    ; write month
                    LDA         Month+0
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    LDA         Month+1
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    ; write '/'
                    LDA         #'/'
                    JSR         lcd_char

                    ; write Date
                    LDA         Date+0
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    LDA         Date+1
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    ; write '/'
                    LDA         #'/'
                    JSR         lcd_char

                    ; write Year
                    LDA         Year+0
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    LDA         Year+1
                    JSR         lcd_num_to_char
                    JSR         lcd_char

                    ; goto second row on lcd
                    JSR         lcd_goto_row1

                    ; write header
                    LDHX  #str_time
```

```
                LDA         str_time_length
                JSR         lcd_str

                ; write hour
                LDA         Hour+0
                JSR         lcd_num_to_char
                JSR         lcd_char

                LDA         Hour+1
                JSR         lcd_num_to_char
                JSR         lcd_char

                ; write ':'
                LDA         #':'
                JSR         lcd_char

                ; write minute
                LDA         Min+0
                JSR         lcd_num_to_char
                JSR         lcd_char

                LDA         Min+1
                JSR         lcd_num_to_char
                JSR         lcd_char

                ; write ':'
                LDA         #':'
                JSR         lcd_char

                ; write minute
                LDA         Sec+0
                JSR         lcd_num_to_char
                JSR         lcd_char

                LDA         Sec+1
                JSR         lcd_num_to_char
                JSR         lcd_char

                ; done
                RTS


;***************************************************************
```

```
;*************************************************************
;* Subroutine Name: rtc_mask_data
;* Description: Takes the raw register values recieved in the
;*                      Sec, Min, etc vars and masks off the data we
;*                      want.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_mask_data:

                    ; Seconds
                    LDA         Sec+0
                    AND         #$07
                    STA         Sec+0

                    LDA         Sec+1
                    AND         #$0F
                    STA         Sec+1

                    ; Minutes
                    LDA         Min+0
                    AND         #$07
                    STA         Min+0

                    LDA         Min+1
                    AND         #$0F
                    STA         Min+1

                    ; Hours
                    LDA         Hour+0
                    AND         #$03
                    STA         Hour+0

                    LDA         Hour+1
                    AND         #$0F
                    STA         Hour+1

                    ; Date
                    LDA         Date+0
                    AND         #$03
                    STA         Date+0

                    LDA         Date+1
                    AND         #$0F
                    STA         Date+1

                    ; Month
                    LDA         Month+0
                    AND         #$01
                    STA         Month+0

                    LDA         Month+1
                    AND         #$0F
                    STA         Month+1

                    ; Year
                    LDA         Year+0
                    AND         #$0F
                    STA         Year+0
```

```
                LDA         Year+1
                AND         #$0F
                STA         Year+1

                ; done
                RTS

;***************************************************************
```

```
;*************************************************************
;* Subroutine Name: rtc_prompt_time
;* Description: Prompts the user to enter a date and time on
;*                       the LCD with the keypad, and saves the
;*                       user-entered time into the Sec, Min, etc vars.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_prompt_time:

;*** write promt to display

                    ; clear the lcd
                    JSR   lcd_clear

                    ; goto top row
                    JSR         lcd_goto_row0

                    ; write header
                    LDHX  #str_prompt_row0
                    LDA         str_prompt_row0_length
                    JSR         lcd_str

                    ; goto bottom row
                    JSR         lcd_goto_row1

                    ; write header
                    LDHX  #str_prompt_row1
                    LDA         str_prompt_row1_length
                    JSR         lcd_str

                    ; goto MM address
                    LDA         #$88
                    JSR         lcd_goto_addr

                    ; prompt for Month+0
                    JSR         keypad_get_keypress
                    STA         Month+0

                    ; prompt for Month+1
                    JSR         keypad_get_keypress
                    STA         Month+1

                    ; goto DD address
                    LDA         #$8B
                    JSR         lcd_goto_addr

                    ; prompt for Date+0
                    JSR         keypad_get_keypress
                    STA         Date+0

                    ; prompt for Date+1
                    JSR         keypad_get_keypress
                    STA         Date+1

                    ; goto YY address
                    LDA         #$8E
                    JSR         lcd_goto_addr

                    ; prompt for Year+0
```

```
                JSR         keypad_get_keypress
                STA         Year+0

                ; prompt for Year+1
                JSR         keypad_get_keypress
                STA         Year+1

                ; goto HH address
                LDA         #$C8
                JSR         lcd_goto_addr

                ; prompt for Hour+0
                JSR         keypad_get_keypress
                STA         Hour+0

                ; prompt for Hour+1
                JSR         keypad_get_keypress
                STA         Hour+1

                ; goto MM address
                LDA         #$CB
                JSR         lcd_goto_addr

                ; prompt for Min+0
                JSR         keypad_get_keypress
                STA         Min+0

                ; prompt for Min+1
                JSR         keypad_get_keypress
                STA         Min+1

                ; goto SS address
                LDA         #$CE
                JSR         lcd_goto_addr

                ; prompt for Sec+0
                JSR         keypad_get_keypress
                STA         Sec+0

                ; prompt for Sec+1
                JSR         keypad_get_keypress
                STA         Sec+1

                ; done
                RTS


;*************************************************************
```