Final Project

FinalProject.s

```
             .include "Equates.s"
             .include "Button Drivers.s"
             .include "LED Drivers.s"

// Array of 20 unsigned hexadecimal numbers
NUMBERS1:
             .word  0x08, 0x13, 0x1E, 0x2B, 0x3E, 0x49, 0x55, 0x67
             .word  0x70, 0x84, 0x91, 0xA0, 0xBD, 0xCA, 0xD5, 0xE3
             .word  0xF0, 0x1C, 0x2F, 0x3A

// Array of 20 signed hexadecimal numbers
NUMBERS2:
             .word  0x08, -0x13, 0x1E, -0x2B, 0x3E, -0x49, 0x55, -0x67
             .word  0x70, -0x84, 0x91, -0xA0, 0xBD, -0xCA, 0xD5, -0xE3
             .word  0xF0, 0x1C, -0x2F, 0x3A

// Array of 20 unsigned hexadecimal numbers initialized to 0
NUMBERS3:    .space 80

// Variables in data memory
MINU:        .word  0
MAXU:        .word  0
AVGU:        .word  0
PHASE:       .word  0

// Main function
             .syntax unified
             .section .text
             .global main
             .global TIM6_Init
main:
      // Initialize devices
      bl          InitLEDs                   // initialize LEDs
      bl          InitButton                 // initialize button
      bl          InitTimer                  // initialize timer

      // Turn off all LEDs
      mov         r0, #0                     // LED_OffOn parameter: off
      bl          LED_OffOn                  // turn off LEDs

      // Continuous while loop for main loop
Loop: ldr         r1, =PHASE
      ldr         r2, [r1, #0]        // r2 = PHASE
Loop1: ldr        r1, =PHASE
      ldr         r3, [r1, #0]        // r3 = PHASE (updated)
      cmp         r3, r2
```

```
       beq          Loop1                     // wait for PHASE change
       mov          r2, r3                    // r2 = PHASE

       // Call task based on PHASE
       cmp          r2, #0
       beq          Task0                     // main loop
       cmp          r2, #1
       beq          Task1                     // sort NUMBERS1 in ascending order,
green LED
       cmp          r2, #2
       beq          Task2                     // sort NUMBERS2 in descending order,
orange LED
       cmp          r2, #3
       beq          Task3                     // add NUMBERS1 and NUMBERS2, store in
NUMBERS3, blue LED
       cmp          r2, #4
       beq          Task4                     // find MINU, MAXU, and AVGU of
NUMBERS3, red LED
       cmp          r2, #5
       beq          Task5                     // stop timer and turn on all LEDs

       b            Loop                      // continue in main loop

// Task 0: main loop
Task0:   ldr        r0, =PHASE          // load address of PHASE
        ldr         r1, [r0]            // load PHASE value
Loop0:   cmp        r1, #0              // check for phase 0
        bne     Task1           // if not 0, go to next task
        b       Loop0           // if 0, continue loop
       // Task 1: sort NUMBERS1 array in ascending order and blink green LED every
0.5s
Task1:
    bl SortAscending             // sort NUMBERS1 array
    ldr r0, =TIM6                // configure timer 6 for blinking
    bl InitTIM6forLEDs
    ldr r0, =LED_OffOn           // turn green LED on
    mov r1, #1
    mov r2, #1                  // use bit 1 (green LED)
    blx r0
    b TaskDone

// Task 2: sort NUMBERS2 array in descending order and blink orange LED every 1s
Task2:
    bl SortDescending            // sort NUMBERS2 array
    ldr r0, =TIM6                // configure timer 6 for blinking
    bl InitTIM6forLEDs
    ldr r0, =LED_OffOn           // turn orange LED on
    mov r1, #1
    mov r2, #2                  // use bit 2 (orange LED)
    blx r0
    b TaskDone

// Task 3: add NUMBERS1 and NUMBERS2, store result in NUMBERS3, and blink blue LED
every 0.5s
Task3:
    bl AddArrays                 // add NUMBERS1 and NUMBERS2, store result in
NUMBERS3
    ldr r0, =TIM6                // configure timer 6 for blinking
    bl InitTIM6forLEDs
    ldr r0, =LED_OffOn           // turn blue LED on
    mov r1, #1
    mov r2, #4                  // use bit 4 (blue LED)
    blx r0
    b TaskDone

// Task 4: find MINU, MAXU, and AVGU of NUMBERS3 array and blink red LED every 1s
Task4:
    bl FindMinMaxAvg             // find MINU, MAXU, and AVGU of NUMBERS3 array
    ldr r0, =TIM6                // configure timer 6 for blinking
    bl InitTIM6forLEDs
    ldr r0, =LED_OffOn           // turn red LED on
    mov r1, #1
    mov r2, #8                  // use bit 8 (red LED)
    blx r0
    b TaskDone

// Task 5: stop the timer and turn on all LEDs
Task5:
    ldr r0, =TIM6                // stop timer 6
    bl StopTIM6
    ldr r0, =LED_OffOn           // turn all LEDs on
    mov r1, #1
    mov r2, #0x0F               // use bits 9-6 (all LEDs)
    blx r0
    b TaskDone

// Task done: return to main loop
TaskDone:
    b Task0

TIM6_Init:
    // Enable clock for TIM6
    ldr r0, =RCC
    ldr r1, [r0, #APB1ENR]
    orr r1, r1, #TIM6EN
    str r1, [r0, #APB1ENR]

    // Configure TIM6
    ldr r0, =TIM6
    mov r1, #0
    str r1, [r0, #CR1]     // Disable the counter
    ldr r1, [r0, #CR2]
    and r1, r1, #0         // Clear CR2
    str r1, [r0, #CR2]

    // Set prescaler and auto-reload
    ldr r1, =0x1
    str r1, [r0, #PSC]
    ldr r1, =0x3E8
    str r1, [r0, #ARR]
    // Enable update interrupt (UIE)
    mov r1, #1
    str r1, [r0, #DIER]

    // Clear update interrupt flag (UIF)
    mov r1, #1
    str r1, [r0, #SR]

    // Enable the counter
    mov r1, #1
    str r1, [r0, #CR1]

    bx lr
```

LED_Drivers.s

```
             .include "Equates.s"           // peripheral addresses

// Functions in this file
             .global InitLEDs              // init GPIOB9-6 for LEDs
             .global LED_OffOn             // individual LED OFF/ON
             .global DisplayNum            // display 4-bit # on LEDs

// Global variables defined in main file

             .syntax unified
             .section .text.LEDdrivers

// GPIOB initialization for LEDs: PB9-8-7-6
InitLEDs:
             // enable clock to GPIOB
             ldr          r0, =RCC
             ldr          r1, [r0, #AHBENR]
             orr          r1, #GPIOBEN
             str          r1, [r0, #AHBENR]
             // configure PB9-6 as output pins
             ldr          r0, =GPIOB
             ldr          r1, [r0, #MODER]
             bic          r1, #0x000FF000
             orr          r1, #0x00055000
             str          r1, [r0, #MODER]
             // set initial output values to 0
             ldr          r1, [r0, #ODR]
             bic          r1, #0x03C0
             str          r1, [r0, #ODR]
             bx           lr

// r0 = bit for LED# 3-0, corresponds to PB9-6
// r1 = 0 for off, 1 for on
LED_OffOn:
             push   {r0-r4}
             add          r0, #6              // change 3:0 to 9:6 for PB9-6
             mov          r4, #1              // on value
             lsl          r4, r4, r0          // shift 1 to position in 9:6
             ldr          r2, =GPIOB          // GPIO port B
             ldrh  r3, [r2, #ODR]      // read current ODR value
             bic          r3, r4              // clear bit for PBx
             cmp          r1, #1              // ON?
             bne          L1                        // skip if ON
             orr          r3, r4              // set bit for PBx
L1:          strh  r3, [r2, #ODR]      // write new ODR value
             pop          {r0-r4}
             bx           lr                        // return
// Initialize Timer 6 for 1ms interrupt
InitTimer:
        // Enable clock to Timer 6
        ldr     r0, =RCC
        ldr     r1, [r0, #APB1ENR]
        orr     r1, #TIM6EN
        str     r1, [r0, #APB1ENR]
```

Button_Drivers.s

```
// Functions for input button on PA0

             .include "Equates.s"           // peripheral addresses

// Functions in this file
             .global InitButton             // initialize PA0
             .global Init_EXTI0             // init button as EXTI0
             .global CheckButton            // return button state
             .global EXTI0_IRQHandler

             .syntax unified
             .section .text.ButtonDriver

// Initialize the User Button with external interrupts
InitButton:
    // Enable clock to GPIOA
    ldr r0, =RCC
    ldr r1, [r0, #AHBENR]
    orr r1, #GPIOAEN
    str r1, [r0, #AHBENR]

    // Configure PA0 as input
    ldr r0, =GPIOA
    ldr r1, [r0, #MODER]
    bic r1, #0x00000003
    str r1, [r0, #MODER]

    // Enable EXTI0 interrupt
    ldr r0, =NVIC_ISER0
    mov r1, #1
    str r1, [r0]
    // Set EXTI0 to trigger on the rising edge
    ldr r0, =EXTI
    ldr r1, [r0, #RTSR]
    orr r1, #1
    str r1, [r0, #RTSR]

    // Unmask EXTI0 interrupt
    ldr r1, [r0, #IMR]
    orr r1, #1
    str r1, [r0, #IMR]

    bx lr

// EXTI0 is the interrupt source for the User Button on PA0
Init_EXTI0:
       ldr          r0, =SYSCFG               // SYSCFG register block
       ldr          r1, [r0, #APB1ENR] // read APB2ENR
       str          r1, [r0, #APB1ENR] // update APB2ENR
       ldr          r1, =EXTICR1       // EXTI0-3 are on EXTI_CR1 register
       ldr          r2, [r1]                  // read EXTI_CR1
       bic          r2, #0x000F               // clear EXTI0 (bits 0-3)
       orr          r2, #0x0000               // set EXTI0 to PA0
       str          r2, [r1]                  // write EXTI_CR1
       ldr          r2, =EXTI_IMR      // EXTI_IMR mask register
       mov          r3, #1                    // bit 0 is for EXTI0
       lsl          r3, r3, #0                // shift to position 0
       orr          r1, r3                    // set bit 0
       str          r1, [r2]                  // enable EXTI0
       ldr          r2, =EXTI_RTSR            // Rising Edge Trigger Selection
       mov          r3, #1                    // bit 0 is for EXTI0
       lsl          r3, r3, #0                // shift to position 0
       orr          r1, r3                    // set bit 0
       str          r1, [r2]                  // enable Rising Edge trigger
       bx           lr

// EXTI0 Interrupt Handler
EXTI0_IRQHandler:
    push {r4, lr}

    // Toggle global variable PHASE
    ldr r4, =PHASE
    ldr r1, [r4]
    eor r1, #1
    str r1, [r4]

    // Clear EXTI0 pending interrupt
    ldr r0, =EXTI
    mov r1, #1
    str r1, [r0, #PR]

    pop {r4, lr}
    bx lr
// CheckButton - return state of push button
// r0 = return value of 0 or 1
CheckButton:
```

```
ldr          r0, =GPIOA                // GPIO port A
ldrh   r0, [r0, #IDR]           // set bit
and          r0, #0x01                 // mask all but bit 0
bx           lr
```