```
;     Evan Heinrich
;     CE2801 sect. 011
;     9/28/2021
;
;     File:
;           delay.S
;     Description of File:
;           Lab 3, Delay subroutine used in future labs
;     (opt) Dependencies:
;           N/A

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text

.global delay_ms

;     Function: delay_ms
;     Register-safe! Pushes all general purpose registers (R0-R12) & LR to the stack
;     Description:
;           Busy loop that takes the value stored in Register 0 and waits that many ms
;           R1 = 4000 (dec) = 1ms
;           R1 LSL 12 = R1 * 4096
;     Args:
;           R1 - Desired delay in milliseconds
;     Returns:
;           Void
;     Register Use:
;           R1    -     Argument
delay_ms:
      PUSH {R0-R12, LR} ; Back up all registers
      LSL R1, R1, #12         ; Conversion to milliseconds
      1:                      ; Delay loop
            SUBS R1, R1, #1   ; Decrement by 1
            BNE 1b            ; Loop if not zero
      POP {R0-R12, LR}  ; Restore registers
      BX LR             ; Return from subroutine
```

```
;     Evan Heinrich
;     CE2801 sect. 011
;     9/28/2021
;
;     File:
;           LED_init.S
;     Description of File:
;           Lab 3, LED initialization used in future labs
;     (opt) Dependencies:
;           N/A

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text

; Constants
.equ RCC_BASE, 0x40023800
.equ RCC_AHB1ENR, 0x30        ; Offset from RCC_BASE
.equ GPIOBEN, 1<<1            ; GPIOBEN lives on bit 1 of AHB1ENR so shift a 1 left by 1

.equ GPIOB_BASE, 0x40020400
.equ GPIOB_MODER, 0x00        ; Offset from GPIOB_BASE
.equ GPIOB_ODR, 0x14          ; Offset from GPIOB_BASE

.global num_to_LED_init

;     Function: num_to_LED_init
;     Register-safe! Pushes all general purpose registers (R0-R12) & LR to the stack
;     Description:
;           Enables RCC for GPIOB
;           Sets GPIOB_MODER for PB5-10 & PB12-15 as outputs
;     Args:
;           N/A
;     Returns:
;           Void
;     Register Use:
;           R1    -     Current main address
;           R2    -     Working register where masks will be applied to
;           R3    -     Masks
```

```
num_to_LED_init:
      PUSH {R0-R12, LR}            ; Backup registers

      ; Enable RCC For GPIOB
      LDR R1, =RCC_BASE            ; Load the RCC base address
      LDR R2, [R1, #RCC_AHB1ENR]   ; Load what is currently stored in the AHB1 Enabler
      ORR R2, R2, #GPIOBEN         ; Apply the mask to enable GPIOB
      STR R2, [R1, #RCC_AHB1ENR]   ; Write back the new AHB1 Enabler value

      ; Set GPIOB Pins as Output
      LDR R1, =GPIOB_BASE          ; Load base address
      LDR R2, [R1, #GPIOB_MODER]   ; Load the GPIOB mode status
      LDR R3, =0xFF3FFC00          ; Load the output clearing mask
      BIC R2, R2, R3               ; Clear the modes
      LDR R3, =0x55155400          ; Load the output setting mask
      ORR R2, R2, R3               ; Overwrite with output set mask
      STR R2, [R1, #GPIOB_MODER]   ; Write back to memory

      POP {R0-R12, LR}             ; Restore registers
      BX LR                        ; Return from subroutine
```

```
;     Evan Heinrich
;     CE2801 sect. 011
;     9/28/2021
;
;     File:
;           num_to_LED.S
;     Description of File:
;           Lab 3, Displays a number on the 10 LEDs on GPIOB
;     (opt) Dependancies:
;           N/A

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text

; Constants
.equ GPIOB_BASE, 0x40020400
.equ GPIOB_ODR, 0x14          ; Offset from GPIOB_BASE

.global num_to_LED

;     Function: num_to_LED
;     Register-safe! Pushes all general purpose registers (R0-R12) & LR to the stack
;     Description:
;           Displays a number provided in R1 using the GPIOB LEDs
;                 ->    Note: There are only 10 LEDs, so if the number in R0 uses
;                       more than 10 bits, bit 11+ will be masked off
;     Args:
;           R1 - Number to be displayed
;     Returns:
;           Void
;     Register Usage:
;           R1    -     Argument
;           R2    -     Working register; will contain the desired contents of the ODR
;           R3    -     Scratch/Addresses
;           R4    -     Masks/Offsets
```

```
num_to_LED:
      PUSH {R0-R12, LR} ; Backup registers
      MOV R2, #0        ; Clear register 2 since it stores the modified value

      LDR R4, =0xFFFFFC00     ; Mask to clear all but lower then bits
      BIC R3, R1, R4    ; Apply mask to Register 1 and store the result in a scratch register

      BFI R2, R3, #5, #6      ; Insert the lower portion of the pattern from R3 into R2
      LSR R3, R3, #6    ; Shift the number left by 6, giving us the last 4 bits in R3[0..3]
      BFI R2, R3, #12, #4     ; Insert the upper portion

      LDR R3, =GPIOB_BASE     ; Set the address for GPIOB
      MOV R4, #GPIOB_ODR      ; Set the offset for the ODR
      STR R2, [R3, R4]        ; Write the value

      POP {R0-R12, LR}        ; Restore registers
      BX LR                   ; Return from subroutine
```

```
;     Evan Heinrich
;     CE2801 sect. 011
;     10/2/2021
;
;     File:
;           num_to_ASCII.S
;     Description of File:
;           Lab 3, converts a provided integer value into ASCII characters
;           representing that number.
;     (opt) Dependencies:
;           N/A

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text

.equ MAX_VALUE, 0x270F  ; Maximum representable value of 9999
.equ ERR, 0x4572722E    ; Error code, "Err." in ASCII

.global num_to_ASCII

;     Function: num_to_ASCII
;     Register-safe! Pushes general purpose registers (R1-R12 & LR) to the stack
;     Description:
;           Converts a provided integer value into ASCII characters representing that number
;     Args:
;           R1    -     Integer to be converted to ASCII (4 chars MAX)
;     Returns:
;           R0    -     ASCII values representing the argument integer
;     Register Use:
;           R0    -     Return value
;           R1    -     Argument
;           R2    -     Scratch
;           R3    -     Ones
;           R4    -     Tens
;           R5    -     Hundreds
;           R6    -     Thousands
;           R7    -     Mask
```

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

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

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

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

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

      MOV R3, R2              ; Whatever is left is the ones place
      MOV R2, #0              ; Clear register 2
```

```
    MOV R7, #0x30           ; Load mask for numeric ASCII values

    ORR R2, R2, R7          ; Apply base mask
    ORR R2, R2, R6          ; Apply thousands place
    LSL R2, R2, #8          ; Shift left 8 for the hundreds place
    ORR R2, R2, R7          ; Apply base mask
    ORR R2, R2, R5          ; Apply hundreds place
    LSL R2, R2, #8          ; Shift left 8 more for the tens place
    ORR R2, R2, R7          ; Apply base mask
    ORR R2, R2, R4          ; Apply tens place
    LSL R2, R2, #8          ; Shift left last 8 time for the ones place
    ORR R2, R2, R7          ; Apply base mask
    ORR R2, R2, R3          ; Apply ones place

    MOV R0, R2              ; Move return value

    POP {R1-R12, LR}  ; Restore registers
    BX LR

error:
    LDR R0, =ERR
    POP {R1-R12, LR}
    BX LR
```

```
;     Evan Heinrich
;     CE2801 sect. 011
;     10/2/2021
;
;     File:
;           test.S
;     Description of File:
;           Lab 3, Driver to test all methods
;     (opt) Dependencies:
;           delay.S
;           LED_init.S
;           num_to_LED.S
;           num_to_ASCII.S

; Assembler Directives
.syntax unified
.cpu cortex-m4
.thumb
.section .text

.global main

.equ MAX_LOOP, 0x400    ; Loop if less than 1024
.equ DELAY, 0x7D        ; 125ms delay
.equ ASCII, 0x4D2       ; Number to be converted to ASCII, 1234

main:
      MOV R1, #ASCII          ; Load the integer for the ASCII test
      BL num_to_ASCII         ; Test the integer to ASCII funct
      BL num_to_LED_init      ; Configure the LEDs
      MOV R1, #0              ; Start counting at 0
      LDR R7, =MAX_LOOP       ; Load the maximum value
1:    BL num_to_LED           ; Display R1
      PUSH {R1}               ; Backup R1, the current count value
      LDR R1, =DELAY          ; Load the 125ms delay
      BL delay_ms             ; Start the delay
      POP {R1}                ; Restore the count value
      ADD R1, R1, #1          ; Increment the count value
      CMP R1, R7              ; Compare to the maximum loop value
      BNE 1b                  ; Continue loop if not at maximum value
end: B end                    ; Infinite loop at end of program
```