```
@ Assignment number 5
@ CPU architectures
@ Penoelo Thibeaud, student id: 40212017

@ Question 1)
@ Write a subroutine STRCAT to concatenate two strings.
@ The main program passes the addresses of both the strings by registers to the subroutine.
@ The second string is concatenated to the first string.
@ Note to reserve extra memory space after the first string to hold the resulting concatenated string.
@ The subroutine STRCAT is to be called twice by the main program.
@ in Arm assembly language

.syntax unified
.cpu cortex-m4
.thumb

.data
str1: .asciz "Hello "
str2: .asciz "World!"

.text
.global _start
_start:
    ldr r0, =str1
    ldr r1, =str2
    @ first call
    bl STRCAT
    @ will result in str1 = "Hello World!"
    @ Will result in str2 = "World!"

    @ second call
    bl STRCAT
    @ will result in str1 = "Hello World!World!"
    @ Will result in str2 = "World!"

    b .

STRCAT:
    push {lr}

    @ find end of str1
    find_end_str1:
        ldrb r2, [r0]
        adds r0, r0, #1
        cmp r2, #0
    bne find_end_str1

    @ decrement r0 to point to the last character of str1
    subs r0, r0, #1

    @ copy str2 to str1
    copy_str2:
        ldrb r2, [r1], #1
        strb r2, [r0], #1
        cmp r2, #0
```

```
    bne copy_str2

    pop {lr}
bx lr

@ ------------------------------------------------------------------------------------- @

@ Question 2)
@ Write a complete ARM assembly program to implement bubble sort algorithm of sorting a list of
@ n words stored consecutively in memory locations starting from ARRAY.
@ The program should include writing a subroutine bubble_sort whose input parameters
@ (n, ARRAY) are passed from the main program using the stack.
@ Use the bubble_sort algorithm given below:
@ for(i = n-1; i > 0; i--) {
@   for(j = 1; j <= i; j++) {
@ if(array[j-1] > array[j]){     //swap them    temp = array[j-1];    array[j-1] = array[j];    array[j] = temp; }
@   } }
@ in arm assembly language

.syntax unified
.cpu cortex-m4
.thumb

.data
array: .word 2, 3, 5, 7, 10, 11, 13, 17, 19, 0

.text
start:
    ldr r0, =array
    mov r1, #10     @ r1 is used as n
    bl bubble_sort
    b .

bubble_sort:
    push {r4-r7, lr}

    mov r4, r0     @ r4 is used as ARRAY
    mov r5, r1     @ r5 is used as i

outer_loop:
    mov r6, #0    @ r6 is used as j-1

inner_loop:
    cmp r6, r5
    bge outer_loop_end

    @ Load current and next elements
    ldr r0, [r4, r6, lsl #2]
    ldr r1, [r4, r6, lsl #2]!

    @ Compare and swap if needed
    cmp r0, r1
    blt no_swap
    str r1, [r4, r6, lsl #2]
    sub r4, r4, #4
```

```
    str r0, [r4, r6, lsl #2]
    add r4, r4, #4

no_swap:
    add r6, r6, #1
    b inner_loop

outer_loop_end:
    sub r5, r5, #1
    cmp r5, #0
    bgt outer_loop

    pop {r4-r7, pc}

@ ------------------------------------------------------------------------------------------ @

@ Question 3)
@ Rewrite the ARM assembly program to calculate prime number (as given in the class notes)
@ in the form of a macro. This program counts prime numbers in an array using macro.
@ The array ends with a 0 to indicate end of the array. The program takes every
@ element in the array, calls a macro ChkPrime to check if this element is prime,
@ then adds 1 to the counter. The macro shall have 2 parameters:
@ array element (the number) as input, and returns 1 if the number is prime or 0 if it is not.
@ The count of prime numbers is saved in memory location “result”.
@ Arm assenbly code

.syntax unified
.cpu cortex-m4
.thumb

.data
array: .word 145,6,15,4,7,5,101,8,9,105,11,47,12,0  // Array end with 0
result: .word 0  // store the count of prime numbers

.text
.global _start

.macro ChkPrime, inputNumber, outputFlag
    mov r4, #2            // r4 is the starting divisor
    udiv r5, \inputNumber, r4 // r5 = \inputNumber / 2, the max value for checking
    mov \outputFlag, #1    // Assume it’s a prime by setting the flag to 1

    prime_check_loop:
        cmp r4, r5        // If we’ve reached half the number, it’s a prime
        bhi prime_check_done
        udiv r6, \inputNumber, r4 // Divide the input number by r4
        mul r7, r6, r4     // Multiply the quotient by r4
        cmp r7, \inputNumber // If the product is equal to the input number, it’s not prime
        beq not_prime
        add r4, r4, #1     // Increment the divisor and continue
        b prime_check_loop

    not_prime:
        mov \outputFlag, #0 // Set the flag to 0 if not prime
```

```
    prime_check_done:
.endm

_start:
    ldr r0, =array    // Address of the array
    mov r2, #0        // Initialize the prime count

    // Loop through the array
    loop:
        ldr r3, [r0], #4
        cmp r3, #0
        beq finish        // If the number is 0, we reached the end
        ChkPrime r3, r1
        add r2, r2, r1
        b loop

    finish:
        ldr r0, =result
        str r2, [r0]      // Store the prime count

    b .
```