CSE 4095 – Special Topics in Computer Engineering III: Introduction to Embedded Systems
Project Report

Ahmet Emre Sağcan 150119042

Mehmet Alper Karabay 150119044

Q1.

Source Code

main.s | startup_stm32l476xx.s

```
        INCLUDE core_cm4_constants.s
        INCLUDE stm32l476xx_constants.s


        AREA    data, DATA, READWRITE
str1    DCB "8AB", 0
str2    DCB "478", 0
result  DCD 0

        AREA    main, CODE, READONLY
        EXPORT  __main
        ENTRY

__main  PROC

        ;hex to dec first one
        LDR R2, = str1
        BL hexToDec
        MOV R4, R0

        ;hex to dec second one
        LDR R2, = str2
        BL hexToDec
        MOV R5, R0

        ;subtraction
        SUB R2, R4, R5
        LDR R0, =result
        ;written to the memory that result label corresponds t
        STR R2, [R0]

stop    B stop

hexToDec
        PUSH{LR}
        MOV R0, #0
parse
        LDRB R1, [R2], #1
        CMP R1, #'0' ;if reached end of string
        BLT done
        CMP R1, #'9' ;if it is numeric
        BLE numeric
        SUB R1, R1, #'A' - 10 ;then it is hexa
        B convert
numeric
        SUB R1, R1, #'0'
convert
        LSL R0, R0, #4
        ADD R0, R0, R1
        B parse
done
        POP {PC}

        ENDP

        END
```

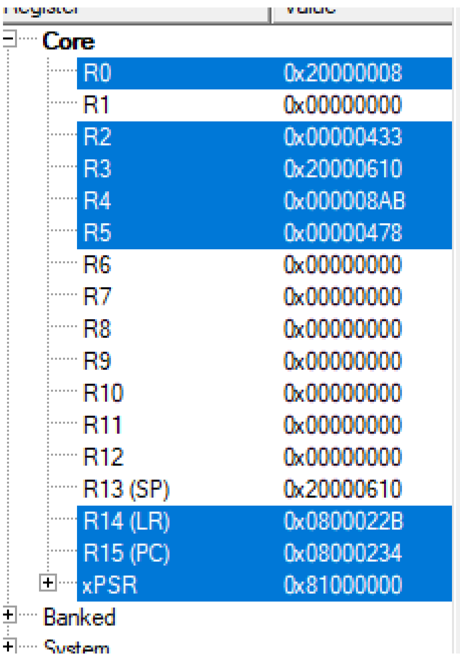
Code Explanation

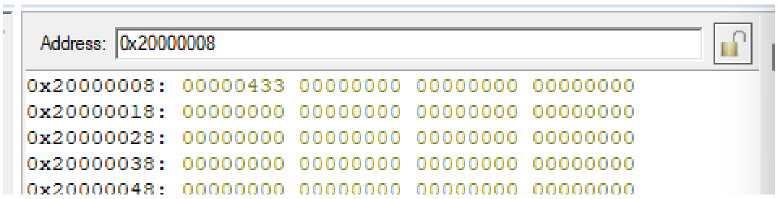
The code starts with two INCLUDE directives to import constants defined in two separate files 'core_cm4_constants.s' and 'stm32l476xx_constants.s'. These files define constants specific to the STM32L476 microcontroller. After the INCLUDE directives, the code defines three variables in the 'data' section: 'str1', 'str2', and 'result'. 'str1' and 'str2' are hexadecimal strings, and 'result' is a 32-bit integer initialized to zero.

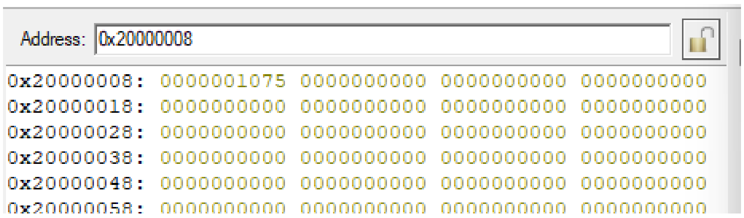
The main section starts with the __main label, which is the entry point of the program. The program first calls the 'hexToDec' function twice to convert the two hexadecimal strings to decimal numbers and stores the results in registers R4 and R5. After that, the program subtracts the second value from the first one and stores the result in register R2. Then, it stores the result in the 'result' variable by writing it to the memory address that corresponds to the 'result' label.

The 'hexToDec' function is defined after the main section. It takes one argument in register R2, which is a pointer to a null-terminated hexadecimal string. It uses a loop to iterate over each character of the string and convert it to a decimal number. It then returns the decimal value in register R0. The function checks if the character is a valid hexadecimal digit. If it is a digit between 0 and 9, it calculates the decimal value of that digit by subtracting the ASCII value of '0' from the ASCII value of the digit, and then multiplying that value by 16 raised to the power of the current digit position. This value is then added to R0. If the character is a letter between A and F, the function calculates the decimal value of that letter in the same way, but adds 10 to the result to account for the fact that A represents 10, B represents 11, and so on.If the character is not a valid hexadecimal digit, the function returns -1 to indicate an error. Once the loop has processed all of the characters in the string, the function returns with R0, which is the decimal value represented by the hexadecimal string.

Output







The result of 8AB-478 in hexadecimal is 433.
Which is 2219 – 1144 = 1075.

Q2.

Source Code

main.s startup_stm32l476xx.s

```
        INCLUDE core_cm4_constants.s
        INCLUDE stm32l476xx_constants.s


        AREA    data, DATA, READWRITE
str1        DCB   "This is a test", 0
str2        DCB   "This is a test", 0
str1_new    SPACE 64
str2_new    SPACE 64

        AREA    main, CODE, READONLY

        EXPORT  __main
        ENTRY

__main  PROC

        LDR R0, = str1
        LDR R1, = str1_new
        BL upper_to_lower

        LDR R0, = str2
        LDR R1, = str2_new
        BL lower_to_upper

stop    B stop

upper_to_lower
        PUSH{LR}
        MOV R2, #0
        MOV R3, #0

loop1
        LDRB R2, [R0], #1
        CMP R2, #0
        BEQ finish

        CMP R2, #'A'
        BLT other1

        CMP R2, #'Z'
        BGT other1

        ADD R2, R2, #'a' - 'A'
        STRB R2, [R1], #1
        B loop1

other1
        STRB R2, [R1], #1
        B loop1

lower_to_upper
        PUSH{LR}
        MOV R2, #0
        MOV R3, #0

loop2
        LDRB R2, [R0], #1
        CMP R2, #0
```

```
        CMP R2, #0
        BEQ finish

        CMP R2, #'a'
        BLT other2

        CMP R2, #'z'
        BGT other2

        ADD R2, R2, #'A' - 'a'
        STRB R2, [R1], #1
        B loop2

other2
        STRB R2, [R1], #1
        B loop2

finish
        STRB R3, [R1], #1
        POP{PC}

        ENDP

        END
```

Code Explanation

The code defines three data items in the .data section:

str1 and str2: Two null-terminated strings that are initialized to "This is a test". These strings are used as inputs for the string manipulation functions.

str1_new and str2_new: Two empty buffers that are used to store the manipulated strings.

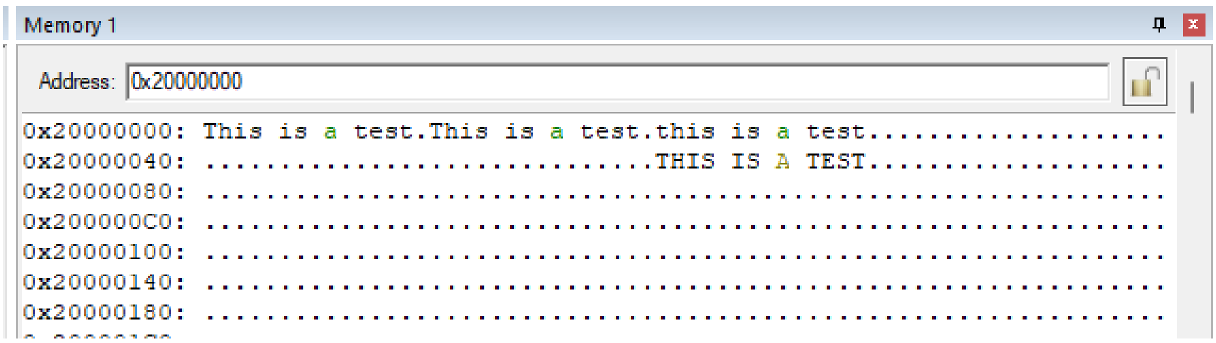
The __main function first calls the upper_to_lower function to convert the str1 string to lower case and store the result in str1_new. The __main function then calls the lower_to_upper function to convert the str2 string to upper case and store the result in str2_new.

The upper_to_lower function uses a loop to iterate over each character in the input string. For each character, the upper_to_lower function checks if it is an uppercase letter. If it is, the function converts it to lowercase by adding the difference between the ASCII codes of uppercase and lowercase letters ('a' - 'A') to the character. Then function writes the converted character to the output buffer. If the character is not an uppercase letter, the function writes the character as-is to the output buffer. The function ends by writing a null terminator to the output buffer and returning.

The lower_to_upper function is similar to upper_to_lower, but it converts lowercase letters to uppercase letters instead. The lower_to_upper function uses a loop to iterate over each character in the input string.

For each character, the lower_to_upper function checks if it is a lowercase letter. If it is, the function converts it to uppercase by adding the difference between the ASCII codes of lowercase and uppercase letters ('A' - 'a') to the character. The function writes the converted character to the output buffer. If the character is not a lowercase letter, the function writes the character as-is to the output buffer. The function ends by writing a null terminator to the output buffer and returning.

Output



The result of the first “This is a test” is “this is a test”.

The result of the second “This is a test” is “THIS IS A TEST”.

Q3.

Source Code

main.s | core_cm4_constants.s | startup_stm32l476xx.s

```
    INCLUDE core_cm4_constants.s
    INCLUDE stm32l476xx_constants.s


NUM_ROWS EQU 4
NUM_COLS EQU 3

    AREA matrices, DATA, READWRITE
input_matrix
    DCD 1, 2, 3
    DCD 4, 5, 6
    DCD 7, 8, 9
    DCD 10, 11, 12

output_matrix
    DCD 0, 0, 0, 0
    DCD 0, 0, 0, 0
    DCD 0, 0, 0, 0



    AREA    main, CODE, READONLY

    EXPORT  __main
    ENTRY

__main  PROC

    ; Initialize input matrix index
    MOV R4, #0

; Loop through columns of input matrix
outer_loop

    ; Initialize column index
    MOV R2, #0

    MOV R6, #12

    MOV R7, #16

; Loop through rows of input matrix
inner_loop

```

main.s | core_cm4_constants.s | startup_stm32l476xx.s

```
; Loop through rows of input matrix
inner_loop

    ; Calculate input matrix element address
    LDR R0, = input_matrix
    MOV R5, R4
    MUL R4, R4, R6
    ADD R0, R0, R4
    ADD R0, R0, R2, LSL #2
    MOV R4, R5


    ; Calculate output matrix element address
    LDR R1, = output_matrix
    MOV R5, R2
    MUL R2, R2, R7
    ADD R1, R1, R2
    ADD R1, R1, R4, LSL #2
    MOV R2, R5

    ; Load input matrix element
    LDR R3, [R0]

    ; Store input matrix element in output matrix
    STR R3, [R1]

    ;Increment column index
    ADD R2, R2, #1

    ;Check if row index has reached end of input matrix
    CMP R2, NUM_COLS
    BNE inner_loop

    ;end of inner loop body

    ; Increment row index
    ADD R4, R4, #1

    ;Check if column index has reached end of input matrix
    CMP R4, NUM_ROWS
    BNE outer_loop

;end of outer loop


stop B  stop
        ENDP

        END
```

Code Explanation

This code defines two matrices, input_matrix and output_matrix, and then copies the elements of input_matrix to output_matrix in a column-wise fashion. The dimensions of the matrices are defined by the constants NUM_ROWS and NUM_COLS.

The code first initializes the index R4 to 0 to keep track of the row index of the input matrix. It then enters an outer loop that iterates over the columns of the input matrix. Within this loop, it initializes the index R2 to 0 to keep track of the column index of the input matrix, and enters an inner loop that iterates over the rows of the input matrix.

For each element in the input matrix, the code calculates the memory address of the corresponding element in the output matrix, loads the value of the input matrix element into register R3 and stores the value into the output matrix. This is done by calculating the index of input matrix at that location by multiplying with 12 (which is the size of a row), then calculating the index of output matrix at that location by multiplying with 16 (which is the size of a row again) and then switching the column and row registers to transpose it.

After processing all rows of a column, the code increments the column index and checks if it has reached the end of the input matrix. If not, the code continues processing the next column. If the column index has reached the end of the input matrix, the code increments the row index and checks if it has reached the end of the input matrix. If not, the code continues processing the next column.

Output

The input matrix is

```
input_matrix
    DCD 1, 2, 3
    DCD 4, 5, 6
    DCD 7, 8, 9
    DCD 10, 11, 12
```

The output is

1 4 7 A

2 5 8 B

3 6 9 C

As it can be seen in memory.