Autumn 2022, Homework 4 (20 points in total)

Q1. (5 pts)
Consider the following IT (IF-THEN) block-based ARM assembly program.

```
CMP    R0, R1
ITTEE  GE
SUBSGE R3, R0, R2
MOVGE  R0, #25
SUBSLT R4, R2, R0
MOVLT  R0, #15
```

1. (3pts) Write the equivalent C or C++ program. The C or C++ program variable names must be the same as the register labels. You code must be executable, i.e., I expect to run "./a.out" after compiling your code through command "g++ testq1.cpp". You should simply include your code in the pdf file that you show your answers to Q1 and Q2.

```
#include <iostream>
using namespace std;

void ifThen (int r0 ,int r1, int r2){
    if( r0 >= r1 ) { // if r0 is greater than or equal to r1
    int r3 = r0 - r2; // SUBSGE  R3, R0, R2
    if( r0 >= r2 ){
      r0 = 25; // MOVGE   R0, #25
    }
  } else {
    int r4 = r2 - r0; // SUBSLT  R4, R2, R0
    if( r2 < r0 ){
      r0 = 15; // MOVLT   R0, #15
    }
  }
  std::cout << "R0: " << r0 << '\n';
  std::cout << "R1: " << r1 << '\n';
  std::cout << "R2: " << r2 << '\n';
}

int main () {
  std::cout << "CASE 1: R0 = 20, R1 = 10, R2 = 5" << '\n';
  ifThen(20, 10, 5);
  std::cout << '\n';
  std::cout << "CASE 2: R0 = 10, R1 = 20, R2 = 5" << '\n';
  ifThen(10, 20, 5);
  return 0;
}
```

2. (1pt) For the initial values of R0 = 20, R1 = 10, and R2 = 5, what will be the final value of R0 after the execution of the above program?

CASE 1: R0 = 20, R1 = 10, R2 = 5
R0: 25
R1: 10
R2: 5

(1pt) Repeat part 2 for R0 = 10, R1 = 20, and R2 = 5.

CASE 2: R0 = 10, R1 = 20, R2 = 5
R0: 15
R1: 20
R2: 5

Q2. (7 pts)
Assume that **src1 DCD 0xFF00FF00** and **src2 DCD 0x00AA00AA** are located consecutively in the READONLY area of memory (see the code below). Also, assume that **src1 starts at address 0x00000008** and **SP = 0x20000400**.

1. For each of the following instructions, calculate the values of registers involved. **Write the values in the comment spaces of the following code as instructed**. For example, "**R0 =**"means you need to fille out like *R0 = 0x00000008*; "**SP =** → " means you need to show the transition of SP contents like *SP = 0x20000400* → *0x20003FC*.

```
                THUMB
StackSize       EQU     0x00000400

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
MyStackMem      SPACE   StackSize

                AREA    RESET, READONLY
                EXPORT  __Vectors
__Vectors
                DCD     MyStackMem + StackSize
                DCD     Reset_Handler

                AREA    MYDATA, DATA, READWRITE

                AREA    MYDCODE, CODE, READONLY
src1            DCD     0xFF00FF00
src2            DCD     0x00AA00AA

                ALIGN
                ENTRY
                EXPORT  Reset_Handler
Reset_Handler
                ; WHAT ARE THE CONTENTS OF EACH REGISTER BELOW? Please fill in as comments
HW4_2           LDR     R0, =src1       ; R0 = 0x00000008
                ADD     R0, R0, #1      ; R0 = 0x00000009
                LDRB    R1, [R0]        ; R1 = 0x000000FF
draw_mem1       PUSH    {R1}            ; SP = 0x20000400  ->      0x200003FC
                ADD     R0, #4          ; R0 = 0x0000000D
                LDRSB   R2, [R0]        ; R2 = 0x00000000
draw_mem2       PUSH    {R2}            ; SP = 0x200003FC    -> 0x200003F8
                MOV     R3, SP          ; R3 = 0x200003F8
                ADD     R3, #4          ; R3 = 0x200003FC
                LDR     R4, [R3]        ; R4 = 0x000000FF
                SUB     R4, #0x004C004C; R4 = 0xFFB400B3
draw_mem3       STRB    R4, [R3]        ; R3 = 0x200003FC
                POP     {R2}            ; SP = 0x200003F8 ->  0x200003FC,  R2 = 0x00000000
                POP     {R1}            ; SP = 0x200003FC ->  0x20000400,  R1 = 0x000000B3
```

2. Stack memory: fill out blanks upon the completion of each event: draw_mem1, draw_mem2, and draw_mem3. Each cell should store only one byte.

Address	draw_mem1	draw_mem2	draw_mem3
	PUSH {R1}	PUSH {R2}	STRB R4, [R3]
0x200003F8	0x00	0x00	0x00
0x200003F9	0x00	0x00	0x00
0x200003FA	0x00	0x00	0x00
0x200003FB	0x00	0x00	0x00
0x200003FC	0xFF	0xFF	0xB3

0x200003FD	0x00	0x00	0x00
0x200003FE	0x00	0x00	0x00
0x200003FF	0x00	0x00	0x00

Q3. (8 pts) Parity-Bit Adding Program

Write a program in ARM assembly code that adds an odd parity to each ASCII character. While you may use parity.s runnable on VisUAL (which you can find in Canvas/files/folder/code/VisUAL), your HW4-Q3 program must run on Keil uVision. Your code must also satisfy the following specifications:

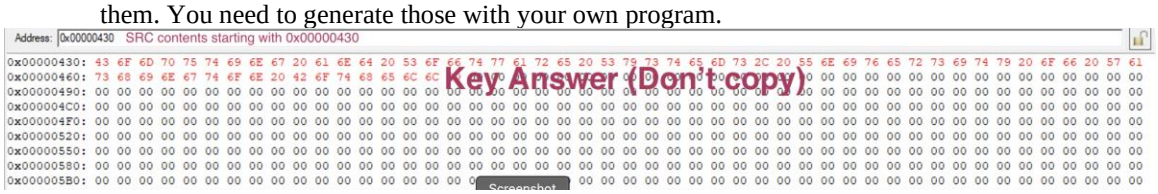
1. To run Keil uVision,
   a. Create the HW4 folder on your Windows desktop, start uVision, choose "new uVision projection".
   b. Select this HW4 folder.
   c. Use HW4 for HW4.uvproj/.uvprojx file names.
   d. Choose Texas Instruments/Tiva C Series/TM4C123x Series/TM4C1233H6PM (the same platform as in lab1b).
   e. Create main.s under Project HW4/Target 1/Source Group1/.

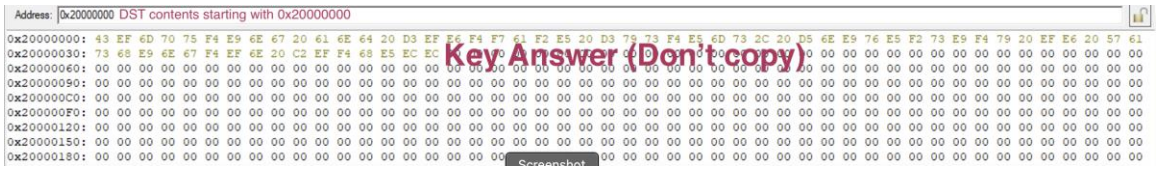
2. Simply comment out line 236 of startup_TM4C123.s: `236 ;  BLX  R0` so that startup_TM4C123.s skips calling SystemInit and directly invokes __main at line 238. (**This is only a modification you need to do in startup_TM4C123.s.**)

3. In your main.s, get started with the following code snippet:

```
        AREA    MyData, DATA, READWRITE
dst     SPACE   200

        AREA    MyCODE, DATA, READONLY
src     DCB     "Computing and Software Systems, University of Washington Bothell", 0

        AREA    |.text|, CODE, READONLY

        EXPORT  __main                  [WEAK]
        ENTRY
__main
        LDR             R0, =src        ; src address
        LDR             R1, =dst        ; dst address
        ;               R2  = the current ASCCI Char
        ;               R3  = counter from 7 to 0
        ;               R4  = #ones
```

4. Use R0, R1, R2, R3, and R4 as specified in the above code snipped:
   a. To read a new ASCII character from src, you should use: `LDRB  R2, [R0], #1`
   b. To write an ASCII character with an odd parity to dst, use: `STRB  R2, [R1], #1`
5. **If you use parity.s from Canvas, make sure:**
   a. R0 should be R2 in HW4.
   b. R1 should be R4 in HW4.
   c. R2 should be R3 in HW4.
6. To check if R4 is odd, use: `LSRS  R4, #1` If a carry is set, R4 was odd, in which case don't add a parity bit at bit 7 of R2. Otherwise it must have been even, and thus you need to set bit 7 of R2.
7. Unlike VisUAL, a real ARM processor does not stop with END. **The very last statement in main.s should be an infinite loop like:** `end_of_main B  end_of_main`
8. Run your program, capture snapshots of **src** and **dst** memory contents, and verify if your **dst** memory contents got correct odd parities. The following is the key answer's results. Don't copy them. You need to generate those with your own program.

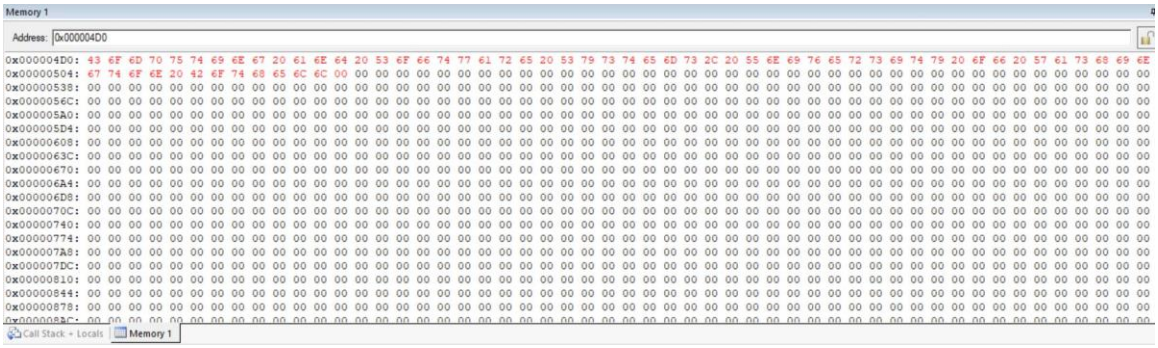




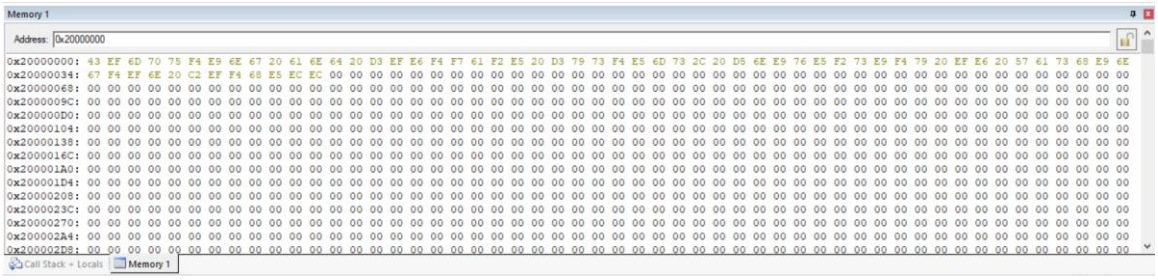
For this question, you need to submit the followings:

1. The source file: main.s. **(If no source file, then you get zero point for this question)**
2. The screenshots of memory view (**src** and **dst** contents) to support your correct results. Please include these screenshots in the same pdf file that you show your answers to Q1 and Q2.
3. After the screenshots, copy the code in main.s into the same pdf file that you show your answers to Q1 and Q2.

SRC:



DCD:



Main.s :

```
    AREA    MyData, DATA, READWRITE
dst     SPACE   200

        AREA    MyCODE, DATA, READONLY
src     DCB     "Computing and Software Systems, University of Washington Bothell", 0

        AREA    |.text|, CODE, READONLY

        EXPORT  __main                  [WEAK]
        ENTRY
__main
        LDR             R0, =src        ; src address
        LDR             R1, =dst        ; dst address
        ;               R2  = the current ASCCI Char
        ;               R3  = counter from 7 to 0
        ;               R4  = #ones
loop
        LDRB    R2, [R0], #1 ;load char into
        CMP             R2, #0 ; check if end of src
```

```
        BEQ             finish_end ; branch to end if at 0
        MOV             R3, #0 ;set counter to 0
        MOV             R4, R2 ;create copy of char
        MOV             R5, #0
        BL              loop_char ;branch into loop
        LSRS    R5, #1 ;shift right by 1 set flags
        ADDCC    R2, R2, #0x80 ; add carry bit
        STRB    R2, [R1], #1 ;store modified into dst
        B       loop ; branch to loop again

loop_char
        CMP R3, #8 ;check if null
        BXEQ LR ; branch back if null
        RORS R4, R4, #1 ; rotate right by 1
        ADDCS R5, R5, #1 ; add carry bit
        ADD R3, R3, #1 ;increment counter by 1
        B       loop_char ;start loop over again

finish_end
        B finish_end ;stop
        END ; end
```