Autumn 2022, Homework 4 Key

Q1. (5 pts)
Consider the following IT (IF-THEN) block-based 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>

int hw4_1( int R0, int R1, int R2 ) {
  int R3, R4;

  if ( R0 >= R1 ) {
    if ( ( R3 = R0 - R2 ) >= 0 ) {
      R0 = 25;
    }
  }
  else {
    if ( ( R4 = R2 - R0 ) < 0 )
      R0 = 15;
  }
  std::cout << "R0 = " << R0 << std::endl;

  return 0;
}

int main( ) {
  hw4_1( 20, 10, 5 );
  hw4_1( 10, 20, 5 );
}
```

`Grading: the code may look different, but the logic of the if-else statement must be correct.`

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
   25 or 0x19

`Grading: wrong result, -1.`

3. (1pt) Repeat part 2 for R0 = 10, R1 = 20, and R2 = 5.
   15 or 0xF

`Grading: wrong result, -1.`

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 = 0x0000 0008
               ADD     R0, R0, #1      ; R0 = 0x0000 0009
               LDRB    R1, [R0]        ; R1 = 0x0000 00FF
draw_mem1      PUSH    {R1}            ; SP = 0x2000 0400 -> 0x2000 03FC
               ADD     R0, #4          ; R0 = 0x0000 000D
               LDRSB   R2, [R0]        ; R2 = 0x0000 0000
draw_mem2      PUSH    {R2}            ; SP = 0x2000 03FC -> 0x2000 03F8
               MOV     R3, SP          ; R3 = 0x2000 03F8
               ADD     R3, R3, #4      ; R3 = 0x2000 03FC
               LDR     R4, [R3]        ; R4 = 0x0000 00FF
               SUB     R4, R4, #0x004C004C; R4 = 0xFFB4 00B3
draw_mem3      STRB    R4, [R3]        ; R3 = 0x2000 03FC
               POP     {R2}            ; SP = 0x2000 03F8 -> SP = 0x200003FC, R2=0x00000000
               POP     {R1}            ; SP = 0x2000 03FC -> SP = 0x20000400, R1=0x000000FF
```

Grading: any wrong value, -0.5; if there are many wrong values (more than 3), -2;

1. (3pts) 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}	STR R4, [R3]
0x200003F8	00	00	00
0x200003F9	00	00	00
0x200003FA	00	00	00
0x200003FB	00	00	00
0x200003FC	FF	FF	B3
0x200003FD	00	00	00
0x200003FE	00	00	00
0x200003FF	00	00	00
0x20000400	00	00	00

Grading: any wrong value in the table -0.5; if there are more than two mistakes, -1;

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:

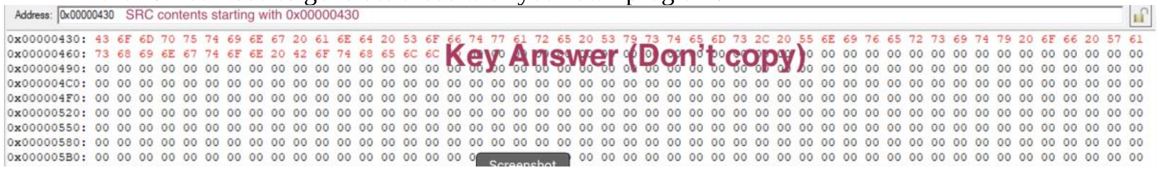
1. To run Keil uVision,
   a. Create the HW4 folder on your Windows desktop, start uVision, choose "new uVision projection".
   b. Select this HW4 folder.
   c. Use HW4 for HW4.uvproj/.uvprojx file names.
   d. Choose Texas Instruments/Tiva C Series/TM4C123x Series/TM4C1233H6PM (the same platform as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

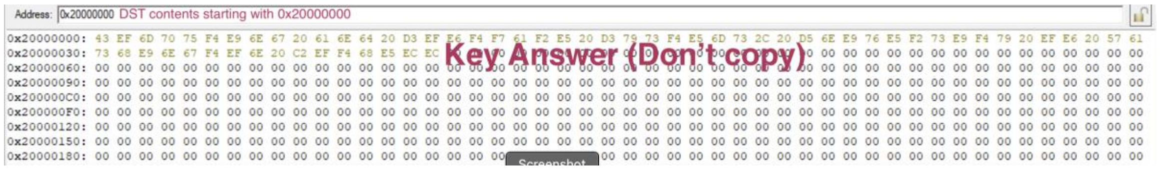
3. In your main.s, get started with the following code snippet:

```
        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.



```
0x00000430: 43 6F 6D 70 75 74 69 6E 67 20 61 6E 64 20 53 6F 66 74 77 61 72 65 20 53 79 73 74 65 6D 73 2C 20 55 6E 69 76 65 72 73 69 74 79 20 6F 66 20 57 61
0x00000460: 73 68 69 6E 67 74 6F 6E 20 42 6F 74 68 65 6C 6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000004C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000004F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000005B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [illegible] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



```
Address: 0x20000000 DST contents starting with 0x20000000
0x20000000: 43 EF 6D 70 75 F4 E9 6E 67 20 61 6E 64 20 D3 EF E6 F4 F7 61 F2 E5 20 D3 79 73 F4 E5 6D 73 2C 20 D5 6E E9 76 E5 F2 73 E9 F4 79 20 EF E6 20 57 61
0x20000030: 73 68 E9 6E 67 F4 EF 6E 20 C2 EF F4 68 E5 EC EC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200000F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

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. After the screenshots, copy the code in main.s into the same pdf file that you show your answers to Q1 and Q2.

Grading: If the source file is not submitted, 0 for this question. If the source code is not copied, -2.

```
        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_stmt
        LDRB    R2, [R0], #1    ; each src char
        CBZ     R2, end_of_main
        MOV     R3, #7          ; #bits (i.e., 7) in each ASCII
        MOV     R4, #0          ; clear #ones
loop_char
        RORS    R2, R2, #1      ; rotate left
        ADDCS   R4, R4, #1      ; #ones++
        SUBS    R3, R3, #1      ; counter--
        BNE     loop_char

        ROR     R2, R2, #25     ; retrieve the original ASCII char
        LSRS    R4, #1          ; #ones % 2 == 1? (odd?)
        BCS     odd_number      ; skip odd parity
        ORR     R2, #0x80       ; add a parity
odd_number
        STRB    R2, [R1], #1    ; store the ASCII char with a parity
        B       loop_stmt

end_of_main
        B       end_of_main
```

3. 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.

Grading: the starting address of the memory view for src and dst could be different. However, the resulted char sequence should match: any mismatch, -1; more than two mismatches, -2;

src:

```
43 6F 6D 70 75 74 69 6E 67 20 61 6E 64 20 53 6F 66 74 77 61 72 65 20 53 79 73 74 65 6D 73 2C 20 55 6E
69 76 65 72 73 69 74 79 20 6F 66 20 57 61 73 68 69 6E 67 74 6F 6E 20 42 6F 74 68 65 6C 6C 00 00 00 00
```

dst:

```
43 EF 6D 70 75 F4 E9 6E 67 20 61 6E 64 20 D3 EF E6 F4 F7 61 F2 E5 20 D3 79 73 F4 E5 6D 73 2C 20 D5 6E
E9 76 E5 F2 73 E9 F4 79 20 EF E6 20 57 61 73 68 E9 6E 67 F4 EF 6E 20 C2 EF F4 68 E5 EC EC 00 00 00 00
```