All grades will be posted on the website as a single spreadsheet file. For anonymity please provide class id (which you will remember) which will be used instead of your name or university ID when posting grades. The class id could be any combination of symbols of <8 length.

Your class ID: ______________
Problem #1 (20 points)

Write MIPS assembler code which would check whether the data are stored in little- or big-endian formats in external memory. After execution of your code register $t0 should have value 0 if the external memory is little-endian and 1 if it is big-endian. Any temporary registers ($t0-$t9) and instructions from green card can be used for this problem. Only “valid” memory locations can be used for testing purposes, e.g. those used for register spilling. (Note that, in general, assembler language is a platform dependent so such code is rarely needed)

There could be various ways to do that. The main idea is to store word with test content, i.e. having hex value 0x000 0001, using stack pointer ($sp) and then check the order of how it was stored by loading byte value from the stack and performing simple comparison. For example, the following code will do the job even without any comparison

```
addi $t0, $0, 0x0001
sw $t0, -4($sp)
lb $t0, -1($sp)
```

Note that stack grows from higher address down and to use stack you have to advance the pointer first, i.e. subtract -4 or use immediate offset -4. It is unsafe to use stack without advancing the pointer 0($sp) because you may destroy the stored value needed by other part of the program.
Problem #2 (30 points)

(a) (20 points) Translate the following MIPS code into equivalent C code

```mips
addi $v0, $0, 0
Loop:  andi $t0, $a0, 1
       add $v0, $v0, $t0
       srl $a0, $a0, 1
       bne $a0, $0, Loop
```

Here the same variable names are used in c code as that of corresponding registers in assembler
```c
v0 = 0;
do {
    v0 = v0 + a0 & 1;
    a0 = a0 >> 1;
} while (a0!=0);
```

Note that “do(body) while(cond)” is not the same as “while(cond)(body)”!

(b) (5 points) Briefly explain in English what the code does
Counts the number of 1’s in register $a0, i.e. “population count”

(c) (5 points) Assuming 32-bit MIPS architecture what is the maximum and the minimum number of instructions (i.e. dynamic count) which can be executed by this code?

Minimum is when value of register $a0 is 0, i.e. $1 + 4*1 = 5$
Maximum is when value of the most significant bit of register $a0 is 1, i.e. $1 + 4*32 = 129$
Problem 3 (30 points)

Translate the following C code into equivalent MIPS one:

for ( i = 0; i < 10; i++)  b[i] = ( a[i] + a[i+10] ) << 1;

Assume the base addresses for the arrays a and b are in $a0 and $a1, correspondingly.

Do-while constructions style (for-loop is also okay for full credit)

```
addi $t0, $a0, 40
LOOP:   lw $t1, 0($a0)
        lw $t2, 40($a0)
        addi $a0, $a0, 4
        addi $a1, $a1, 4
        add $t1, $t1, $t2
        sll $t1, $t1, 1
        sw $t1, -4($a1)
        bne $a0, $t0, LOOP
```
Problem 4 (20 points)

Given your understanding of PC-relative addressing, explain why an assembler might have problems directly implementing the branch instruction in the following code sequence:

here: beq $s0, $s2, there
...
there: add $t2, $t1, $t0

Show how the assembler might rewrite this code sequence to solve these problems.

There is a limitation to how far bne instruction can jump because the immediate field is only 16 bits. More specifically, for any I-type control flow instruction the range is $2^{16}$ instructions. Note the 4 times larger range because of the shifting of the immediate value when calculating the physical address (this is due to the fact that all instructions are word aligned in memory). The range is much larger for J-type instructions. With 26 bits in J-type instruction it is $2^{28}$ which is still not the maximum possible one. The maximum range is provided by “jr rs” instruction which jumps to the address provided by 32-bit value in rs register. Therefore, e.g., assuming J-type instruction, assembler could rewrite the following sequence as

here:  bne $s0, $s2, skip 
     j there
skip: 
...
there: add $t2, $t1, $t0
Bonus Problem 5 (10 points)

Suppose that you need to implement swapping of the content between registers $t0$ and $t1$ and only these two registers are available. (For example, it could be a situation when one wants to avoid register spilling to improve execution time). Write your code for this operation.

**version 1:**
- xor $t0$, $t0$, $t1$
- xor $t1$, $t0$, $t1$
- xor $t0$, $t1$, $t0$

**version 2:**
- addu $t0$, $t0$, $t1$
- subu $t1$, $t0$, $t1$
- subu $t0$, $t0$, $t1$