# CARLETON UNIVERSITY

**Department of Systems and Computer Engineering**

**SYSC 5704** **Elements of Computer Systems** **Assignment 2**

Due date: Tuesday, October $14^{\text{th}}$, 18:00.

---

**2.7** [5] <§2.3> Show how the value `0xabcdef12` would be arranged in memory of a little-endian and big-endndian machine. Assume the data is stored starting at addesss 0.

**2.8** [5] <§2.4> Translate `0xabcdef12` into decimal.

**2.12** Assume that registers `$s0` and `$s1` hold the values `0x80000000` and `0D0000000`, respectively.

1. [5] <§2.4> What is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $sl
```

2. [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?
3. [5] <§2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
sub $t0, $s0, $sl
```

4. [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?
5. [5] <§2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $sl
add $t0, $t0, $s0
```

6. [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?

**2.13** Assume that `$s0` holds the value $128_{\texttt{ten}}$.

1. [5] <§2.4> For the instruction `add $t0, $s0, $s1`, what is the range(s) of values for `$s1` that would result in overflow?
2. [5] <§2.4> For the instruction `sub $t0, $s0, $s1`, what is the range(s) of values for `$s1` that would result in overflow?
3. [5] <§2.4> For the instruction `sub $t0, $s1, $s0`, what is the range(s) of values for `$s1` that would result in overflow?

**2.18** Assume that we would like to expand the MIPS register file to 128 registers and expand the instruction set to contain four times as many instructions.

1. [5] <§2.5> How this would this affect the size of each of the bit fields in the R-type instructions?
2. [5] <§2.5> How this would this affect the size of each of the bit fields in the I-type instructions?
3. [5] <§§2.5, 2.10> How could each of the two proposed changes decrease the size of an MIPS assembly program? On the other hand, how could the proposed change increase the size of an MIPS assembly program?

**2.24** [5] <§2.7> Suppose the program counter (PC) is set to `0x2000 0000`. Is it possible to use the jump (`j`) MIPS assembly instruction to set the PC to the address as `0x4000 0000`? Is it possible to use the branch-on-equal (`beq`) MIPS assembly instruction to set the PC to this same address?

**2.27** [5] <§2.7> Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `$s0`, `$s1`, `$t0`, and `$t1`, respectively. Also, assume that register `$s2` holds the base address of the array `D`.

```
for ( i = 0; i < a; i++ ) {
    for ( j = 0; j < b; j++ ) {
        D[4*j] = i + j;
    }
}
```

**2.33** [5] <§2.8> For each function call, show the contents of the stack after the function call is maked. Assume the stack pointer is opriginally at address `0x7ffffffc`, and follow the register conventions as specified below:

| Preserved | No Preserved |
|---|---|
| Saved registers `$s0-$s7` | Temporary registers `$t0-$t9` |
| Stack pointer register `$sp` | Argument registers `$ar0-$ar3` |
| Return address registers `$ra` | Return value registers `$v0-$v1` |
| Stack above the stack pointer | Stack below the stack pointer |

**2.34** [5] <§2.8> Translate function `f` into MIPS assembly language. If you need to use registers `$t0` through `$t7`, use the lower-numbered registers first. Assume the function declaration for `func` is "`int func(int a, int b);`". The code for function `f` is as follows:

```
int f(int a, int b, int c, int d ) {
    return func( func( a, b ), c + d );
}
```

**2.35** [5] <§2.8> Can we use the tail-call optimization in this function? If no, explain why not. If yes, what is the difference in the number of executed instriuctions in `f` with and without the optimization.

**2.38** [5] <§2.9> Consider the following code:

```
lbu $t0, 0($t1)
sw  $t0, 9($t2)
```

Assume that the register `$t1` contains the address `0x1000 0000` and the register `$t2` contains the address `0x1000 0010`. Note the MIPS architecture utilizes big-endian addressing. Assume that the data (in hexadecimal) at address `0x1000 0000` is: `0x1122 3344`. What value is stored at the address pointed to by register `$t2`?