CprE 381 Homework 2

[Note: This homework gives you practice with MIPS assembly language. When you are asked to provide a program, try running it on MARS to confirm it works. MARS can also check your understanding of code tracing, etc. Plus you get more practice with the tools.]

1. MIPS Control Flow
   a. Assume `$t0` holds the value `0x0000FFFF`, `$t1` holds the value `0xEFF01270`, and `$t2` holds the value `0x00000001`. What is the value of `$t2` after the following instructions?

```
          lui   $t0, 0xFFFF
          slt   $t1, $t1, $t0
          beq   $t1, $0, SOMEPLACE
          sra   $t2, $t1, 31
          j EXIT
     SOMEPLACE:
          add $t2, $t2, $t1
     EXIT:
```

   b. Translate the following C-style loop into MIPS assembly, assuming the following variables to register mappings:

`int *a`	`$s0`
`int max`	`$s1`
`int i`	`$s2`
`int N`	`$s3`

   All variables are 4 byte words. How many instructions are executed if `N`=1, `N`=10, `N`=100, `N`=1000? *[If your code depends on the values of a — which it does not have to — you may assume an approximate expected number of changes to max is $lnN + 0.577$]*

```
int max = a[0];
for (i=1; i<N; i++) {
   max = a[i] > max ? a[i] : max;
}
```

   c. Write MIPS assembly for the following statement. Assume `score` is in `$a0` and `grade` is in `$v0`. Do NOT use a jump table as a compiler might, but rather use conditional branches.

```
if (score >= 90) {
    grade = 'A';
} else if (score >= 80) {
    grade = 'B';
} else if (score >= 70) {
    grade = 'C';
} else if (score >= 60) {
    grade = 'D';
} else {
```

```
        grade = 'F';
    }
```

2. MIPS Assembly Language Design
    a. P&H(2.21) <§2.6>. Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstruction *[We've talked about pseudoinstructions before with **mov**. Effectively they are assembly instructions that aren't actually in the ISA of hardware, so the assembler has to translate them into another machine instruction or series of machine instructions.]*:
    ```
    not $t1, $t2    # bit-wise invert
    ```
    b. You are tasked with adding a new pseudo-instruction that performs a left rotational shift.
    ```
    rolr $t0, $t1, sign_ext_imm
    # rotates $t1 left by sign_ext_imm bits
    # (a negative immediate implies right shift)
    ```
    Give a reasonably minimal implementation (i.e., don't use control flow instructions). Should this instruction produce any exceptions (i.e., can it produce any unexpected behaviors)? If so, what are they and does your version cause the same exceptions or introduce any more?
    c. Why does MIPS not have **add** `label_dst,label_src1,label_src2,` instructions in its ISA (there are at least two reasons for this)? *[Read this instruction's function as M[label_dst] = M[label_src1] + M[label_src2].]* Provide a concrete technical justification—you should have ideas both from lab and lecture.

3. MIPS Programming *[I suggest you actually run these programs to confirm that they work. Use MARS for MIPS runtime simulation.]*
    a. Write a simple (you do not need to optimize—just use the direct implementation) C code snippet that implements the strrchr function from string.h (https://cplusplus.com/reference/cstring/strrchr/)

```
char * strrchr( char * str, int character);
```

    b. Translate your answer to part a into MIPS assembly. Make sure you use correct System-V ABI conventions as described in lecture (https://refspecs.linuxfoundation.org/elf/mipsabi.pdf). All variables in your C code should be initialized within your code.
    c. Add code that *calls* strrchr and prints the result for testing purposes. Provide *three* reasonable test cases for your MIPS assembly (inputs and expected outputs) and justify why you have included each one.