# CprE 381 Homework 3

*[Note: This homework gives you more practice with the MIPS assembly language, in particular the implementation of more complex control flow. When you are asked to assemble a program, you can try running it on MARS to confirm it works. However, make sure you have it running in without 'Settings → Delayed branching' selected.]*

1. MIPS Machine Code
   a. The following instruction (count or cnt) is not included in the MIPS instruction set:

```
cnt $t0, $t1
# The first operand is rt, the second operand is rd
# if (M[R[rt]](7:0) >= 0)
#   R[rt] = R[rt]+1, R[rd] = R[rd] + 1, PC=PC
```

   If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format? Explain why. Provide a sequence of MIPS instructions that performs the same operation. Ungraded, but exam-worthy: Postulate why this instruction wasn't included the MIPS ISA (there is a general philosophical reason and at least one very specific technical reason).

   b. Translate the following MIPS assembly into machine code providing the following for each instruction. First, identify the instruction's format. Second, provide the decimal value for each instruction field. Third, provide the hex encoding of the entire instruction. Assume `begin` is at `0x00400abc` (start of the text/code segment in the default memory configuration of MARS). **No credit will be given without the field-by-field work.**

```
begin:
    nor $t0, $s0, $a0
    addiu $t1, $zero, 127
loop:
    lw $t2, 64($t0)
    sltu $t3, $t2, $t1
    beq $t3, $zero, loop
    j begin
```

   c. We've discussed in class that you cannot load an arbitrary 32 bit integer (e.g., `0xFEED2050`) using a single instruction. Look up the **`lui`** instruction (e.g., on the green sheet from your textbook) and provide a two-instruction sequence that loads `0xFEED2050` into `$t0`. Then, assuming that **`lui`** is not supported by the ISA, provide a valid three-instruction sequence that loads `0xFEED2050` into `$t0`. Translate these into MIPS machine code providing the same steps as part 1.b.

2. MIPS Programming with procedures *[You should actually simulate this program using the provided version of MARS to confirm that they work. Do not simply copy these from online examples or from the result of a compiler.* ***YOU MUST COMMENT WHAT YOU ARE DOING****.]*
    a. Write a MIPS program that iteratively (i.e., using a for loop) calculates the Fibonacci number, $F_N$, for an inputted number, N. Have N be an integer entered by a user and print $F_N$ to the console. *[See MARS lecture companion files for an example of how to read an integer in MARS and print an output.]* Submit your work as a .s or .asm assembly file.
    b. Write a second MIPS program that recursively calculates $F_N$ (i.e., using a procedure that calls itself with an updated argument). Make sure to follow the convention presented in lecture. Specifically, use the appropriate saved vs temporary registers, argument passing registers, return value registers, and a basic stack frame with appropriate alignment. Submit your work as a .s or .asm assembly file.
    c. How many instructions (i.e., dynamic instructions) were executed in your two different programs? Briefly show your calculations. *[MARS has a tool that can count instructions, which I suggest you use to verify that your hand calculations are reasonably close.]*