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).

    #Get the value at register rt, Then go to the memory location of that value.

    #Look at the byte at that address, if it is greater than or equal to 0

    #increment the value in register rt and value in register rd by 1

    lb $at, 0 ($t0) # get the byte at the value of register rt which is $t0

    srl $at, $at, 31 # get msb in 0 location and 0s in other bit locations

    xor $at, $at, 1# negate the bit

    addu $t0, $t0, $at # add it to the registers, if negative signed goes from 1 to 0

    addu $t1, $t1, $at # adds 0 and if positive 0 goes to 1 and adds by 1

    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 `0x00400000` (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:
     andi $s1, $zero, 321
     addi $s0, $zero, -32768
loop:
     sra $s0, $s0, 5
     addiu $s1, $s1, 1
     slti $t0, $s0, -1
     bne $t0, $zero, loop
     j begin
```

```
   andi $s1, $zero, 321:
1. Type I
2. 12 opcode, 0 $zero, 17 $s1, 321 for 321 immediate
3. 0011|00 00|000 1|0001| 0000|0001|0100|0001
   0x30110141

   addi $s0, $zer0, -32768:
1. Type I
2. 8 opcode, 0 for $zero, 16 for $s0, -32768 for immediate
3. 0010|00 00|000 1|0000| 1000|0000|0000|0000
   0x20108000

   sra $s0, $s0, 5
1. Type R
2. 0 opcode, 0 for rs, 16 for $s0 rt register, 16 for $s0 rd
   register, 5 for shift amount, 3 for the function code
3. 0000|00 00|000 1|0000| 1000|0 001|01 00|0011
   0x00108143

   addiu $s1, $s1, 1
1. Type I
2. 9 opcode, 17 for $s1 rs register, 17 for $s1 rt register, 1
   for the immediate value
3. 0010|01 10|001 1|0001| 0000|0000|0000|0001
   0x26310001

   slti $t0, $s0, -1:
1. Type I
2. 10 opcode, 16 for $s0 rs register, 8 for $t0 rt register,
   -1 for immediate field
3. 0010|10 10|000 0|1000 |1111|1111|1111|1111
   0x2a08FFFF

   bne $t0, $zero, loop
1. Type I
2. 5 for opcode, 8 for $t0 rs register, 0 for $zero rt
   register, -4 for immediate (want to jump back 3
   instructions and your current instruction is considered a
   instruction so you need to subtract off an additional
   instruction, each instruction is 4 bytes but byte
   addressable)
3. 0001|01 01|000 0|0000| 1111| 1111| 1111| 1100|
   0x1500FFFC

   j begin:
1. Type J
2. 2 for opcode, 1048576 for address(2^22 / 4)
3. 0000|10 00|1000|0000|0000|0000|0000|0000
   0x08100000
```

c. We've discussed in class that you cannot load an arbitrary 32 bit integer (e.g., `0xFEED3210`) 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 `0xFEED3210` into `$t0`. Then, assuming that **`lui`** is not supported by the ISA, provide a valid three-instruction sequence that loads `0xFEED3210` into `$t0`. Translate these into MIPS machine code providing the same steps as part 1.b.

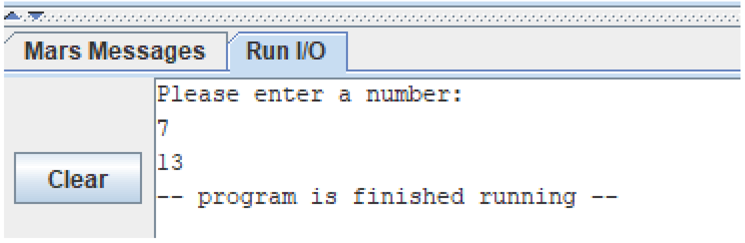
```
#with lui
lui $t0, $t0, 0xFEED
ori $t0, $t0, 0x3210

#lui not available
ori $t0, $t0, 0xFEED
sll $t0, $t0, 16
ori $t0, $t0, 0x3210
```

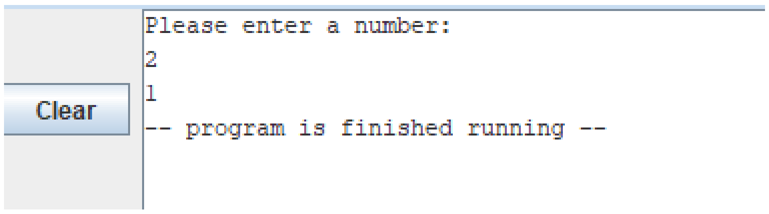
2. MIPS Programming with procedures *[You should actually simulate this program using the provided version of MARS to confirm that they work. Do not simply copy these from*

online examples or from the result of a compiler. ***YOU MUST COMMENT WHAT YOU ARE DOING****.]*

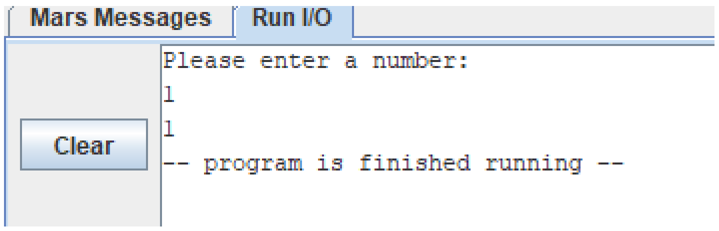
a. Write a MIPS program that iteratively (i.e., using a for loop) calculates the Fibonacci number, F_N, for an inputted number, N. Have N be an integer entered by a user and print F_N to the console. *[See MARS lecture companion files for an example of how to read an integer in MARS and print an output.]*



```
Please enter a number:
7
13
-- program is finished running --
```



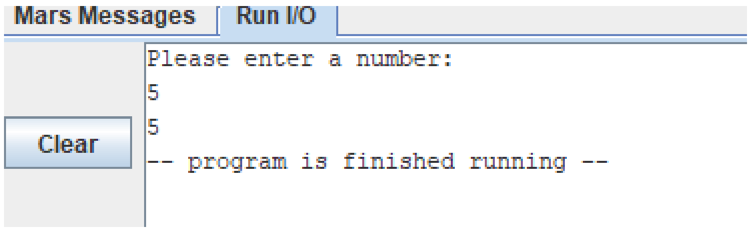
```
Please enter a number:
2
1
-- program is finished running --
```



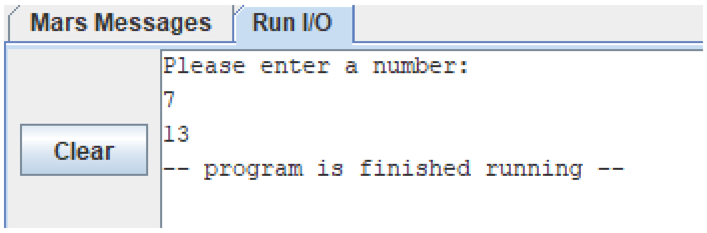
```
Please enter a number:
1
1
-- program is finished running --
```

**Note I did not add any negative argument checking, will output 1 if input is negative or 0*

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.



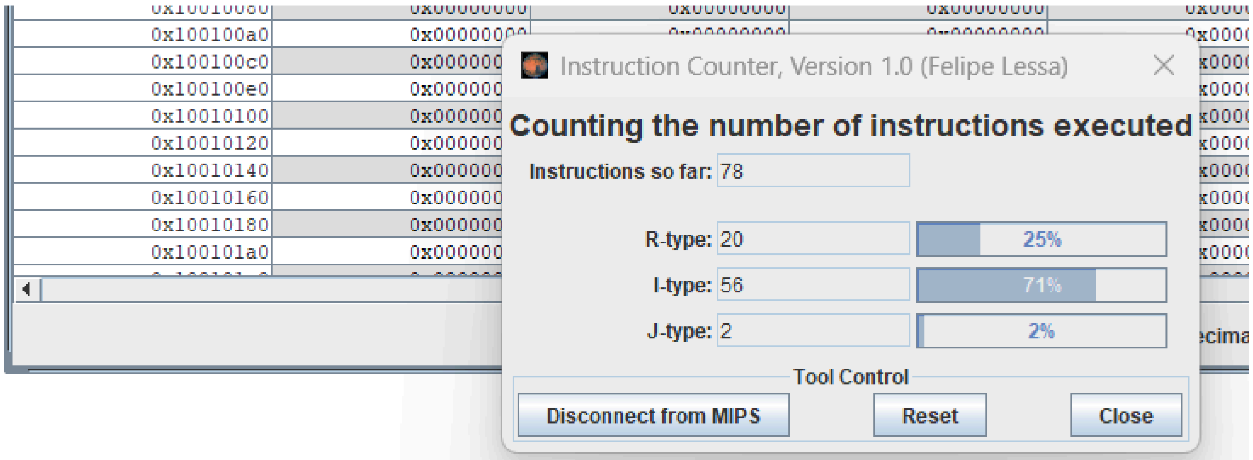
```
Please enter a number:
5
5
-- program is finished running --
```



```
Please enter a number:
7
13
-- program is finished running --
```

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.]*

6 +6(n-1), exits when n == 1, for non-recursive

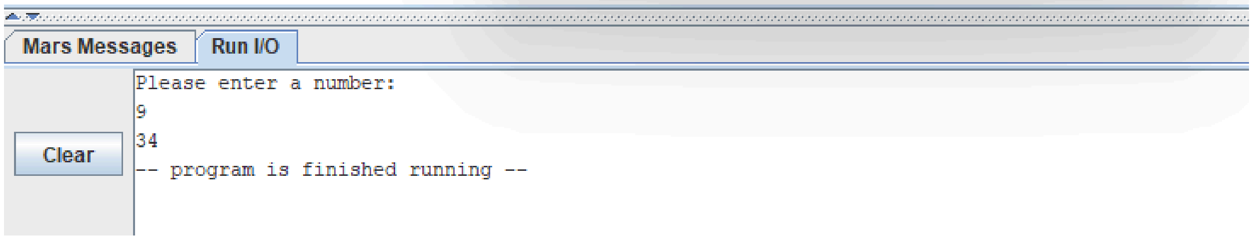


```
0x100100a0    0x00000000    0x00000000    0x0000
0x100100c0    0x000000                    x0000
0x100100e0    0x000000                    x0000
0x10010100    0x000000                    x0000
0x10010120    0x000000                    x0000
0x10010140    0x000000                    x0000
0x10010160    0x000000                    x0000
0x10010180    0x000000                    x0000
0x100101a0    0x000000                    x0000
```

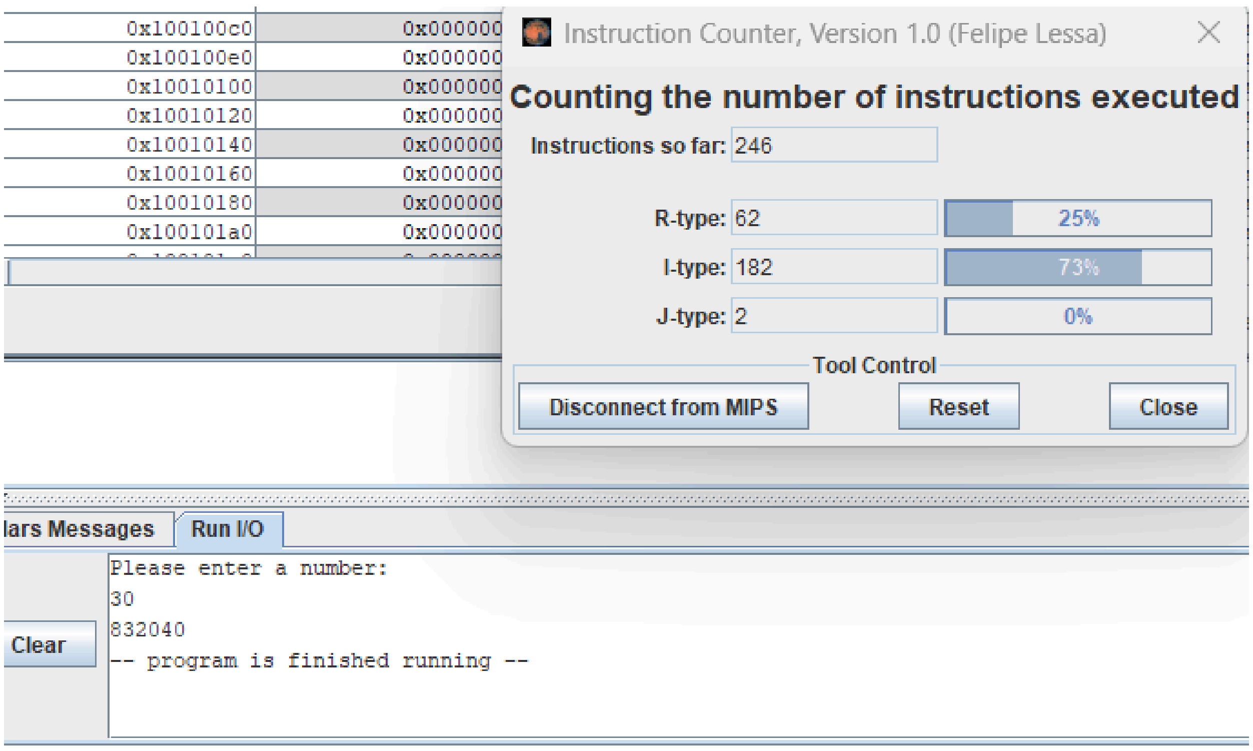
Instruction Counter, Version 1.0 (Felipe Lessa)

Counting the number of instructions executed

Instructions so far:	78	
R-type:	20	25%
I-type:	56	71%
J-type:	2	2%



```
Please enter a number:
9
34
-- program is finished running --
```



Instruction Counter, Version 1.0 (Felipe Lessa)

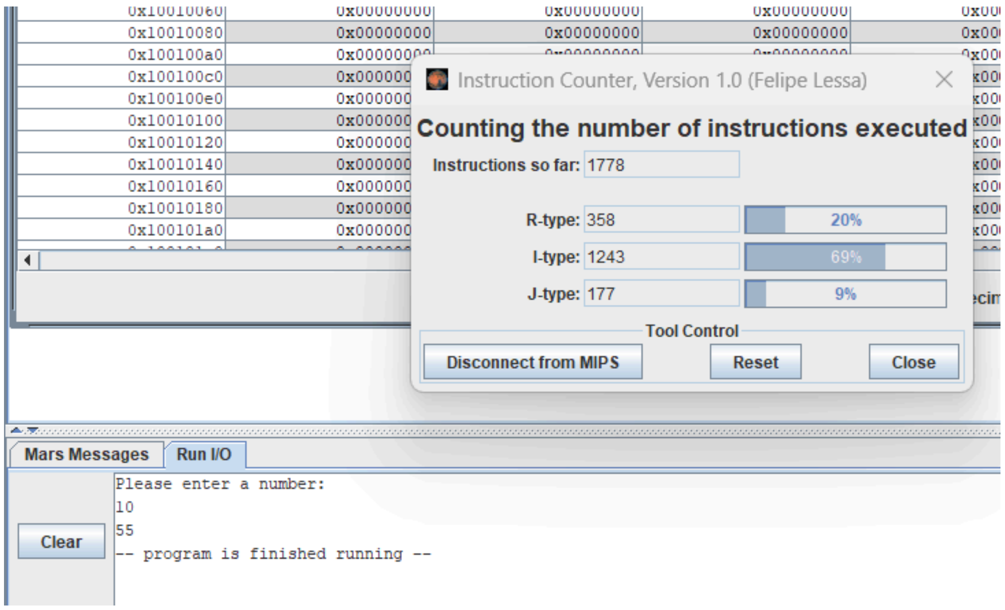
Counting the number of instructions executed

Instructions so far:	246	
R-type:	62	25%
I-type:	182	73%
J-type:	2	0%

```
Please enter a number:
30
832040
-- program is finished running --
```

Recursive
13 in else and 4 in if base case

*(13+ 4 constants) * n^2 times*



Instruction Counter, Version 1.0 (Felipe Lessa)

Counting the number of instructions executed

Instructions so far:	1778	
R-type:	358	20%
I-type:	1243	69%
J-type:	177	9%

```
Please enter a number:
10
55
-- program is finished running --
```