CprE 381 Homework 1

[Note: This homework covers material on the construction and definition of ISAs and begins to get you introduced to MIPS. It also has you start to run MIPS programs through a simulator (MARS) that you will use going forward both in homework AND to help test your term project. A little extra time spent working with MARS now will pay off later.]

1. ISAs
    a. Identify the following aspects as more RISC-like or more CISC-like
        i. Variable-length instructions
        CISC-like, they operate complex commands that does multiple things in one command. This means it can take a different amount of clock cycles depending on the command.
        ii. Memory-memory operations
        CISC processors use a memory-to-memory framework to execute instructions such as ADD, LOAD, and even STORE.
        iii. Load-store architectures
        RISC-like, RISC usually follow a load-store model where operations can only be performed between registers so memory operations require loading and storing.
        iv. Regular instruction formats
        RISC-like, commands have a fixed size and are executed in 1 clock cycle.
        v. Many instructions
        RISC -like, commands are broken down into similar trivial commands. This means more of the work is delegated for the software and compiler side of things.
        vi. Few instructions
        CISC-like, more of the work is implemented in the hardware. This as a result means that a single instruction can take multiple clock cycles.
    b. Identify if the following items ***impact*** the ISA, ABI (but not ISA), or micro architecture (i.e., a specific implementation of the ISA including which functional units are used and how they are interconnected). Note that some of these may have multiple answers. Provide a brief justification (1 sentence is sufficient).
        i. Number of named registers
        ISA, number of registers is defined by the ISA
        ii. Number of cycles an instruction takes to execute
        Microarchitecture, instruction execution times vary between computers
        iii. Whether or not immediate operands can be used directly in arithmetic instructions
        ISA, the ISA species whether or not there can be immediate values
        iv. Which register number is the stack pointer

ABI, this is up to the choice of the ABI which can determine how different parts of the program will interact.

v. Which, if any, registers numbers produce constants (e.g., 0 or -1)
ISA, this is specified if this can happen in the ISA and computers will comply with it

vi. Which register numbers pass arguments to function calls
ABI, the ABI helps interface between code and is in charge of deciding which registers to use.

vii. Which register numbers are temporary or saved
ABI, it is in charge of how the code will operate with registers

viii. Addressable address range for memory operations
ISA, this is amount of instructions defined by ISA

ix. Type of adder used in the ALU
Microarchitecture, this can differ between different computers and still execute the same code.

x. Which instructions update the PC (remember the PC holds the address of the next instruction to execute)
ISA, this is part of the ISA and the different operations it has.

xi. Which bits of an instruction correspond to the opcode or an operand
ISA, this is defined in the ISA which the computer implements

xii. Number of functional units (e.g., adders or multipliers) in the processor
Microarchitecture, this is a hardware difference between computers that can have the same ISA.

2. MIPS

a. Identify three products not mentioned in lecture videos that use a MIPS-based processor.

Nintendo 64, Playstation 1 & 2, Loongson 3 Series

b. MIPS does not have a true **`mov`** instruction (**`mov`** `dst, src #dest=src`). Why would such a simple, basic instruction not be included in a RISC ISA? List three arithmetic or logical instructions or instruction sequences not mentioned during lecture that produce the same effect as a **`mov`** instruction.

Add with zero, or with zero, lui for upper followed by ori for lower

c. What does the following program do (give a description in English sentences)?

```
xor $1, $1, $2
xor $2, $1, $2
xor $1, $1, $2
```

Swaps the values in register 1 and 2

d. Assume that variables `a`, `b`, `c`, and `d` are mapped to registers `$s0`, `$s1`, `$s2`, and `$s3`, respectively. Write a MIPS program that implements

$$a = (b + c) * 7 - d \,\%\, 4$$

using only **`add`**, **`sub`**, **`addi`**, **`and`**, **`andi`**, **`or`**, **`ori`**, **`sll`**, and **`srl`** instructions. Since `b`, `c`, and `d` must not be overwritten, use as many temporary registers

($t0-$t9) as needed.
add $t0, $s1, $s2 # b + c
sll $t1, $t0, 3 # (b+c) multiply by 8
sub $t0, $t1, $t0 # subtract off one of (b+c) to make *7
andi $t1, $s3, 0x00000003 # dividing by 4 gets rid of last 2 bits meaning that
#will be remainder if you divide by 4
sub $s0, $t0, $t1 # final expression putting in a

3. MARS Introduction
   a. Read the introduction to MARS found at http://courses.missouristate.edu/KenVollmar/mars/Help/MarsHelpIntro.html and Part 1 of the MARS tutorial found at:http://courses.missouristate.edu/KenVollmar/mars/tutorial.htm. Some of the specifics of MIPS that are referenced you may just be learning about (or haven't learned about at all in the case of syscalls) However, you should be able to answer: does MARS simulate the ABI, ISA, and/or microarchitecture of MIPS? Explain. *[We'll use MARS since a modified version of it is used for testing the MIPS processors you will design for your term project.]*
   MARS simulates the microarchitecture that implements the MIPS ISA. This is because MARS has memory and you can see the memory locations of the microarchitecture and you can use MIPS commands to alter that data stored.
   b. Download the MARS simulator from your Canvas course. Load prob3.s into your simulator. Run three times with different inputs and report the result (and corresponding inputs). Looking at the code, what does this program do? (write some C code that *could* compile to this)

It takes in two user inputs. The first input, input1, is multiplied by 32. The second input, input2, acts as a integer pointer. Whatever value you specify in input2 it will go to that number in the vals string because they are integers and integers are 4 bytes. The once it grabs that value it will add it by input1 which has been multiplied by 32. It will then finally output it.

```
void main(){
int nums[] = {25, 1, 4, 10, 381, 42, 100, 60, 0, 12, 25};
int* pointer = 0x10010018; //base address for string
int input1;
int input2;
int outputNum;

scanf("%d", &input1);
```

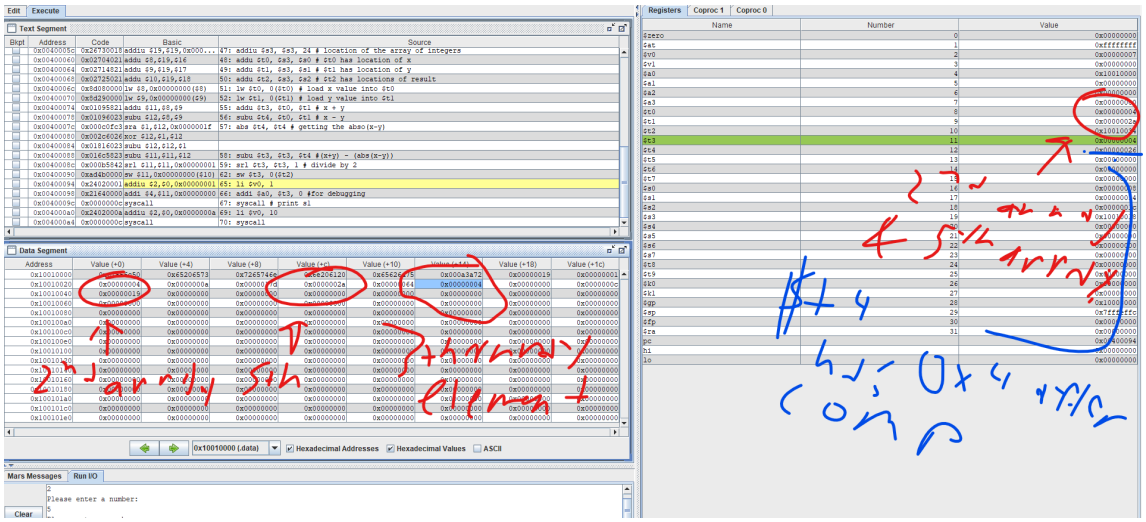
```
input1 *= 32;

scanf("%d", &input2);

pointer += input2;
outputNum = *pointer + input1;

printf("%d", outputNum);
}
```

c. Modify prob3.s to find the minimum of two array values and write the value back to a third array location. The minimum of two numbers, x and y, can be given by:

$$MIN(x, y) = (x + y - abs(x - y))/2$$

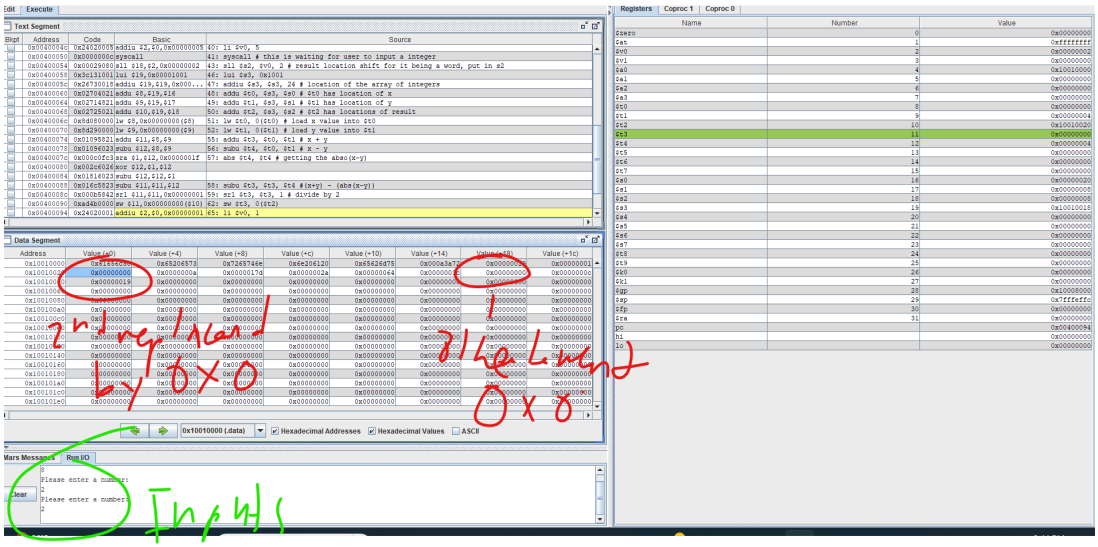
To find the *abs* in MIPS, you may use the **abs $t0, $t1** *pseudo-instruction* (sets $t0 to abs($t1)).

All three indices should be entered by the user. For this assignment you may assume that the user inputs correct values. Provide three test cases (inputs and observed output) showing your code works and describe how you verified correctness. Submit your new program as prob3_<NetID>.s.



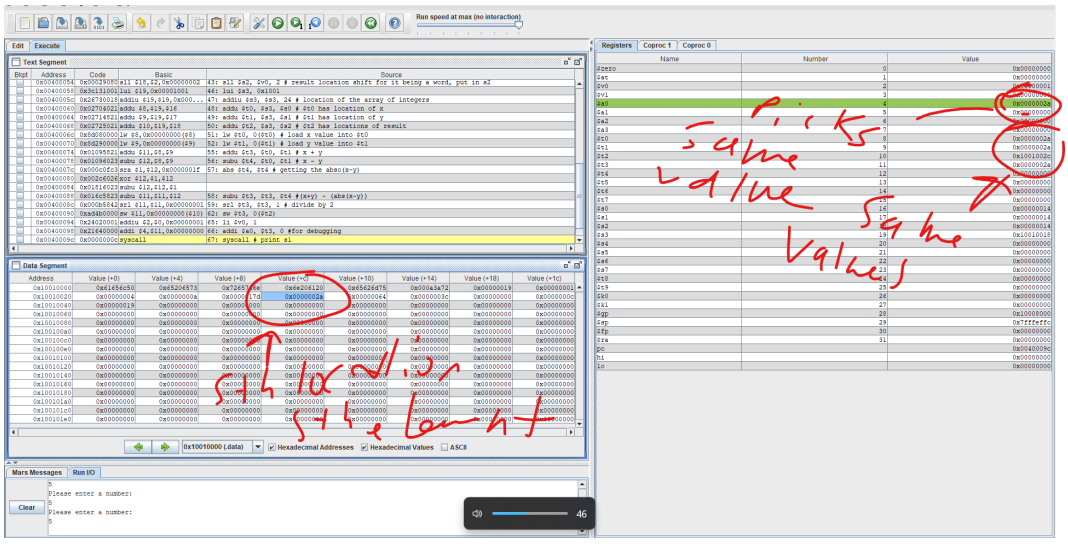
1. Inputs: 2 5 7

   Results: It compared 2 which was 4 and 5 which was 42. It got both of those and correctly chose 4. After it then put 4 into the 7th array location.



2. Inputs : 8, 2, 2

   Result: The 2nd location in the array was replaced by 0 as location 8 was 0 and location 2 was 4. This means it works as intended.



3. Inputs: 5, 5, 5

   Result: Put 42 into the 5th spot where it already was. This means it works as intended as the min between 42 and 42 is 42 and it put it in its same spot.