CPRE 381 Homework 1 (Fall 2024) – ETHAN ROEPKE

[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.] You should NOT use any AI software to answer questions. However, you can check online resources to learn more about the topics.

1. ISAs

a. Identify the following aspects as more **RISC-like** or more **CISC-like**. Please give an explanation for each one; otherwise, you will not get any points. (5pt)
   i. Fixed instruction size (e.g., all instructions are 32 bits)
      i More RISC like, they have fixed-length instructions(32 bits) to simplify decoding and execution. Overall it allows for faster processing.
   ii. Complex addressing modes (e.g., based on displacement, indexed)
      i More CISC like, they are known for supporting a variety of complex address modes which to access memory in flexible ways.
   iii. Instructions that perform multiple operations (e.g., load and increment)
      i More CISC like, they feature instructions that perform multiple operations within a single instruction. This simplifies the programming but makes the executions slower.
   iv. Instructions with many operands
      i More CISC like, they support instructions with multiple operands, which involves multiple registers and memory access.
   v. A number of complex instructions that perform multiple operations in a single instruction.
      i More CISC like, they can perform complex tasks, such as loading a value from memory, performing arithmetic, and storing the result, all in one instruction.

b. Identify if the following items impact the ISA, ABI (but not ISA), or microarchitecture (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). (16pt)

i. Support for floating-point arithmetic operations
   i The ISA defines the available instructions for floating-point arithmetic and specifies how these operations are supported.
ii. The convention for passing function return values
   i The ABI, but independent of ISA. This specifies conventions such as which registers will be used to pass a return value.
iii. Cache size and associativity
   i The microarchitecture design defines details the cache size
iv. Instruction pipelining and hazard handling
   i The microarchitecture and not part of the ISA. This is the pipelining and hazard handling that implement features that optimize how instructions are executed.
v. The number of cores in a processor
   i The number of processor cores is part of the microarchitecture, determining how the processor is physically implemented.
vi. Size of the program counter register
   i The ISA defines the size of the program counter, as this determines how large an address space the architecture can support.
vii. Which bits of instruction determine the immediate value
   i The ISA specifies the format of instructions, including which bits are allocated to immediate values in instruction encoding.
viii. Instruction encoding format (e.g., R-type, I-type)

   i The ISA defines the encoding format of instructions and how instructions are structured and decoded by the processor.

2. MIPS (Practice questions for Exam 1)

a. RISC architectures typically do not include a dedicated '**INC'** (increment) instruction (e.g., *inc dst,* which adds 1 to the destination register). (1) Why might such an instruction be excluded from a RISC ISA? (2) List three alternative instructions or instruction sequences that achieve the same effect as an **INC** instruction. (10pt)
   a RISC architectures try to simplify instruction sets by only including instructions that perform basic operations. A dedicated increment

instruction is not included because it is not considered a basic operation. Instead, they can be easily replaced with more general instructions. RISC architectures emphasize having a small and simple instruction set, which reduces the complexity of decoding instructions and allows for more efficient pipelining and execution.
ADD dst, dst, 1
ADDI dst, dst, 1
SUBI dst, dst, -1

b. MIPS does not have a true '**DIV**' (division) instruction (**DIV** `dst, src1, src2 #dest=src1/src2`). List three alternative sequences of instructions that can achieve the same result as a DIV instruction, assuming division by a power of 2 (e.g., 2, 4, 8…). (10pt)
   a srl dst, src1, 1 # dst = src1 / 2
   b srl dst, src1, 2 # dst = src1 / 4
   c srl dst, src1, 3 # dst = src1 / 8

c. What does the following program do (give a description in English sentences)? (5pt)

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

   a This sequence of xor statements swaps the values of the registers $1 and $2. At the end of the execution, $1 and $2 values will be swapped without using a place holder.

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 the following expression **using only** add**,** sub**,** addi**,** and**,** andi**,** or**,** ori**,** sll**, and** srl **instructions:** a = (b + 8) × 4 + (c / 8) + (d % 2)
Since `b`, `c`, and `d` must not be overwritten, use as many temporary registers (`$t0-$t9`) as needed. **Please write the explanation for each single line.** (10pt)

   b
```
addi $t0, $s1, 8        # $t0 = b + 8
sll $t0, $t0, 2         # $t0 = (b + 8) * 4
srl $t1, $s2, 3         # $t1 = c / 8
andi $t2, $s3, 1        # $t2 = d % 2
add $t3, $t0, $t1       # $t3 = (b + 8) * 4 + (c / 8)
add $s0, $t3, $t2       # a = (b + 8) * 4 + (c / 8) + (d % 2)
```

3. MARS Simulation

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.] (4pt)*

MARS simulate mainly using ISA. This allows users to write and execute MIPS assembly code. It implements the instruction formats and provides registers, memory operations, and basic input/output operations. Thus, MARS is focused on the ISA by simulating how MIPS instructions operate at a high level.

Download the MARS simulator from http://courses.missouristate.edu/KenVollmar/mars/download.htm. 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) (10pt)

Numbers inserted: 5 and 8 Result: 160

Numbers inserted: 11 and 2 Result: 356

Numbers inserted: 10 and 10 Result: 320

This code shifts the first value inputted by 5 so multiply by 32. Then shift the second value inputted by 2 and multiply by 4. The second index is used to access a value from a array. The final value is the firspt input number multiplied by 32.

b. Modify prob3.s to average two array values and write the value back to a third array location. All three indices should be entered by the user. For this assignment, you may assume that the user inputs the 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. **Please include comments for each line of code.** (25pt)

Numbers inserted: 12, 2, and 8

Numbers inserted: 2, 3, and 4

Numbers inserted: 1, 1 and 1