CMPEN 331 Exam 1 Review

Will Bochnowicz

February 18, 2023

Contents

1 MIPS Instructions

Assembly is the language of the computer. These are basic operations that are easily converted (by the assembler) into machine code: the 1s and 0s that computers actually use to compute numbers. It is one of a number of Instruction Set Architectures, formal definitions of what a processor can and can't do, which have slightly varying Instruction Sets per processor model. For example, ARMv7-m is an Instruction Set Architecture, upon which the Cortex-M processors have conforming Instruction Sets despite having slightly different implementations.

MIPS is a family of Instruction Set Architectures. They are RISC based, which means that they are relatively restricted in size and can only do operations on the contents of registers. The contents of memory must be moved into a register before it can be acted upon. There are a number of ways that MIPS instructions, which are actual commands to the processor, are represented in binary.

A register is a small storage medium that is located directly in the CPU. It is the fastest form of memory, but has extremely limited size due to its location within the CPU. MIPS has 32 general purpose registers, as well as some others (LO, HI, PC, floating point registers) that serve specific purposes.

Some MIPS instructions are so-called "pseudoinstructions", as they are not actual instructions in the MIPS ISA. Rather, they are shorthand for ease of programming and are expanded into actual instructions by the assembler. For example, the `blt $s1, $s2, Label` instruction actually expands to two instructions: `slt $at, $s1, $s2` and `bne $at, $zero, Label`, where `$at` is a register that is reserved for the assembler to do operations of this sort (side-note: `$at` is one of the 32 general-purpose registers, despite being reserved by the assembler).

1.1 R-Type

R-Type instructions represent instructions that act on three registers. These registers are labeled as the Source, Target, and Destination

Examples of R-Type instructions: `add`, `sub`, `mul`

op	rs	rt	rd	shamt	func
6	5	5	5	5	6

op: “Operation” – The basic operation of the instruction, also called “opcode”. 6 bits (just happened to be leftover from dividing 32 bits between 4 5-bit fields and 2 other fields).

rs: “Resister Source” – the first operand of the instruction. 5 bits, as there are 32 general registers.

rt: “Register Target” – the second operand of the instruction. This can occasionally serve as the destination in other types of operations. 5 bits, as there are 32 general registers.

rd: “Register Destination” – the third operand of the instruction. This is where the result of the operation is stored. 5 bits, as there are 32 general registers.

shamt: “Shift Amount” – The amount that the target register is shifted, used for sll and srl operations (where the source register is unused). This field is zero for most arithmetic operations. 5 bits, in order to access all 32 bits in a word.

func: “Function Code” – Specifies the variant of the Operation that is used. For most R-Type, this contains the entire operation (“op” is zero for add, sub, mult, div, etc). 6 bits.

1.1.1 Problems

Q :Write the binary representation of the `add` command.

Note that `add`’s `op` and `func` fields are 000000 and 100000, respectively. (1)

Also assume that for $a = b + c$, a is in register 1, b is in register 2, and c is in register 3.

1.2 I-Type

1.3 J-Type

2 Addressing Modes

3 Solutions

1: The binary for this operation is 00000000010000110000100000100000.
OP = 000000
RS = 00010
RT = 00011
RD = 00001
SHAMT = 00000
FUNC = 100000

Index