Computer Architecture

Lab 7

In this lab, we'll look at the floating point coprocessor.

You should turn in the source file for the program you're asked to write. It should be submitted via the tool available on Sakai for this lab. It is due by the start of the next class period.

Floating Point Operations in MIPS

We know that the hardware for floating point arithmetic is different than the hardware for integer arithmetic. Since instructions directly determine which physical path the machine uses for its computation, we need different instructions for floating point operations.

Coproc 1 has floating point registers f0 – f31. The floating point instructions use these registers, not the ones from the normal MIPS register file.

The floating point instructions end in .s or .d (single or double precision.) Single precision uses one 32 bit register; double precision use two 32-bit registers (f0-f1, f2-f3, etc.). So, to add you use add.s or add.d and then reference the f registers in coprocessor 1. The other arithemetic operations are similar. Remember you can only use the f registers with the floating point instructions.

We can load the floating point registers using a load pseudoinstruction: l.s fd, addr

We can store as well: s.s fd, addr

We can encode in the .data field using .double or .float: x: .float 4.8

You can also load immediate using li.s fd, val or li.d fd, val

For console printing, we can use:

syscall 2 is print float, but you have to put the float in $f12
syscall 3 is print double, you put the doubles in $f12, $f13

Problem 1 – Int 64

Perform 64 bit integer arithmetic. This is hard because there isn't specific support for it. So, you need to put the high and low words of the number in memory individually using .word. Then use integer operations to add the low words, track whether a carry occurs (there is no status register so you have to figure out how to do that) and then add the high words (plus 1 if carry.) Then store back in memory in the right places. There is a lot of bookkeeping.

Don't worry about console in/out for this. There is not direct support for that, and doing so properly would take more time than is intended for this lab (though it is still worthwhile to think about doing.) Just put some values in memory and add them together and store them back in memory.

Problem 2—Single Precision Floating Point

Store three floats and output their sum. Using the floating point operations, this should be straightforward.

Problem 3 –Double Precision Floating Point

Store three doubles and output their sum. The hardest part will be not cut-and-pasting your code from Problem 2. But, compare to how hard 64 bit integers were to work with in Problem 1.

Problem 4—Templates

Think about writing a template function that can add single or double precision based on a user switch. Yes, just think about it. You don't have to write anything for this problem, but you should be able to do such a thing in C++.

The big takeaway from doing all this is to drive home how computer arithmetic works and to remain aware of how number formats are really different. If you work in JavaScript and everything is just var this and var that, or in Python when things are who knows what, you can forget what the machine is doing. For some programming tasks, maybe that's fine, but the point of this course is to deepen your understanding of programming in general and to be capable of harnessing the metal of the machine directly when that power would be useful.