Floating Point Numbers

3-3-2003

Opening Discussion

- What did we talk about last class?
- Have you seen anything interesting in the news?

Fractions in Binary

- Once again, we can explain concepts in binary by looking at their equivalent in decimal. In this case, all the numbers to the right of the point are multiplied by negative powers of the base we are using.
- Note that not all numbers can be equally well represented in different bases. For example, 0.1_{ten} requires an infinite repeating series in base two.

Scientific Notation

- We could do a "fixed point" notation by arbitrarily placing a point after one of the bits in our 32 bit numbers. However, we can do better than that by borrowing from scientific notation.
- In this format the point "floats" because we always multiply by some power of our base. In normalized form, we make it so there is only a 1 left of the point.

Floating Point Format

- A word can be broken up in any number of ways to store a floating point number. The standard division between precision and size range is to have 8 bits for the exponent and 23 for the mantissa. One other bit is used for the sign.
- Double precision numbers use 64 bits. They still have one sign bit, but use 11 bits for the exponent and 52 for the mantissa. They have very good precision.

Gaining a Bit

- The highest bit of the mantissa is implicitly on so it is not stored.
- This causes a problem for zero which shouldn't have a leading 1. An exponent of zero implies that the number has a value of zero. This has problems for numbers between 1 and 2 and your book doesn't explain that sufficiently.

Exponents and Sorting

- The developers of the IEEE 754 standard wanted to enable fast comparison with integer compares. That is part of the reason the sign bit is first.
- For this reason the exponent is stored with biased notation instead of two's complement. For single precision numbers, the real exponent is the binary value, minus 127.
- Integer compares give the right result.

Addition of Floating Point Numbers

- First, align the points by shifting the number with the smaller exponent.
- Second, add the mantissas.
- Third, check if it is normalized and if not, normalize it and check for overflow or underflow of the exponent.
- Fourth, round the result to the proper number of bits.
- Unfortunately this is quite serial.

Multiplication of Floating Point Numbers

- First, add the exponents (we have to subtract out the bias because it is in the sum twice).
- Second, multiply the mantissas.
- Third, normalize the results if they aren't normalized and check for overflow or underflow.
- Fourth, round the result and determine the sign.
- Note that 1 and 2 can be done in parallel.

Subtraction and Division

- Subtraction is done simply with addition after the sign bit has been changed.
- For division, it turns out to be faster to take the multiplicative inverse, then do multiplication so there isn't a separate division algorithm either (other than that required for finding the inverse.

Floating Point in MIPS

- MIPS has a set of instructions for dealing with floating point values.
  - add.s and add.d for single and double adds.
  - sub.s and sub.d
  - mul.s and mul.d
  - div.s and div.d
  - comparisons are done with c.x.s and c.x.d where x can be eq, neq, lt, le, gt, or ge
  - branch with bclt and bclf, for branching on true and false.

More about MIPS

- There are separate floating point registers in MIPS, $f0, $f1, $f2, …, $f31 They use different load an store commands: lwc1 and swc1. Double precision numbers are stored in consecutive registers starting with an even numbered one.
- Note that we don't have as many specialized registers. Nor are there immediate forms. The constants have to be put into global memory.

Minute Essay

- Write the number -58.75 in single precision floating point format. Remember, there is one bit for the sign, 8 for the exponent, and 31 for the mantissa, but the highest bit is just assumed to be on.