Lecture 6

This lecture was primarily about signed numbers, specifically their representation in binary, how to store / perform other operations on them, and so on.

Immediates

- Some values known ahead of time (literals), can be loaded in earlier
- We call these values immediates
- Immediates are signed 16-bit numbers (*short*s), cause that's what fits into I-Type instructions (seen later)
- ADDI *$target*, *$register*, *Immediate value*
    - Immediate can be in whichever number system, as long as its supported by the assembly language (e.g. hexadecimal, decimal, binary, etc)
- Since immediates are signed values, we don't need SUBI

Storing signed numbers

- Numbers are stored in memory in binary form

RAM

Addr.	Data
0	0000 0000
1	0000 0000
2	0000 0000
3	0000 1111

Each cell holds 1 byte, together cells 0 - 3 hold 1 word

- Big-endian vs Little-endian
    - Do we store the "most significant" (big end) bits first, or the "least significant" (little end) first?
    - "First" in this case means in the earlier memory addresses
    - MIPS is a big endian architecture, so the earlier bits are the most significant values
    - As such, the diagram above represents the decimal number 15 (if it was little endian, that diagram would represent decimal number 251658240)

- How do we store negatives?
  - One idea is to use the leftmost bit as a "sign bit", 1 to represent negative, 0 to represent negative (or vice versa, doesn't matter, as long as you follow the protocol)
  - The issue with this is that it complicates arithmetic operations -- we can't simply add two binary numbers which are represented using this technique, cause we'd have to account for the signed bits with some algorithm.
  - You want basic operations to be as fast as possible, which means finding a way to store binary values that doesn't involve extra steps

  - Solution: Two's Complement
    - **To get the two's complement negative notation of an integer, you write out the number in binary. You then invert the digits, and add one to the result.**
    - All positive numbers start with 0, all negative numbers start with 1
    - Arithmetic operations are unaffected (still work as expected)
      - E.g. 3 + -3 will make 10000 but the 1 overflows and you're left with 0
    - Slight caveat: signed numbers shift our range of representable values down

- Loading bytes, how does MIPS deal with signed numbers?
  - LW - we load the entire 32 bits, so we simply copy over the bits into the target register
  - LH - we load 2 bytes into the bottom, but what about the top 2 bytes?
    - LH automatically makes them all 1s or 0s depending on the sign of the leftmost bit of the bytes you loaded
  - LHU (load half word unsigned) - data in the 2 bytes are not in two's complement notation -- automatically fills remaining 2 bytes with 0s
  - The same applies to LB and LBU (load byte unsigned)