Lecture 24, Nov 3, 2022

More RISC-V Instructions

- 4 types of shift instructions:
  - Shift left logical `sll`
    - Fill the LSBs with zero
  - Shift right logical `srl`
    - Fill the MSBs with zero
  - Shift right arithmetic `sra`
    - Fill the MSBs with the sign bit
    - This keeps the sign of a two's complement negative integer
  - Also have immediate versions `slli`, `srli`, `srai`
    - The immediates are 5 bits, since we only ever need to shift something by a maximum of 32 bits
  - Left-shifting by N is equal to multiplying by 2^N
  - Right shifting by N is equal to dividing by 2^N (and truncating the remainder)
- Using shifts we can extract or assemble bit fields
  - If `s6 = 0x1234ABCD`:
  - `srli s6, s7, 8 # s6 = 0x001234AB`
  - `andi s6, s6, 0xFF # s6 = 0x000000AB`
- Accessing bytes or half-words
  - `lb` and `lbu` – load byte and load byte unsigned
    - Since we're taking 8-bits from memory and putting it into a 32-bit register we need to know what to do with the rest of the bits
    - `lb` fills the rest with the sign bit (sign extension), `lbu` fills with zero
    - Does not have to be aligned
  - `lh` and `lhu` – load half word and load half word unsigned
    - Has to be half-word aligned (address divisible by 2)
  - `sb` and `sh` – store byte and store half word
    - Both store the least significant byte and half-word of the register
    - This keeps the other bytes in the word the same, so there is no question of sign extension

Program Flow

- Like data, the program itself is also stored in memory
  - Each instruction is 32-bits so each consecutive instruction is a difference in memory address by 4
- The *program counter* (PC) holds the address of the current instruction the processor is executing
  - When an instruction completes, it's automatically incremented by 4
  - By changing the program counter, we can make the program jump around, thus accomplishing nonlinear program flow
- We have conditional branch instructions that modify the program counter based on some condition, so we can accomplish structures such as conditionals and loops
  - There are many flavours of conditional branch instructions, but they al compare 2 source registers
    - `beq s1, s2, LABEL` – branch if equal, will branch if `s1` and `s2` are equal and set the program counter to `LABEL`
    - `bne` – branch if not equal
    - `blt`, `bge` – branch less than, branch greater than or equal to
      - Unsigned versions `bltu, bgeu`
    - Pseudo-instructions:
      - `beqz`, `bnez` – branch if equal to zero/not equal to zero
      - `bnez`, `blez`, `bgtz`, `bltz`, etc
      - `bgt`, `ble`