Lecture 21, Oct 27, 2022

Introduction to Assembly Language

- A human-readable form of the processor's native language
- Many different flavours, e.g. x86, ARM, RISC-V, etc
- Assembly is translated into binary machine code by an assembler
  - High-level languages are translated into assembly and then compiled into machine code
  - While high level languages such as C don't care about the underlying processor, assembly is targeted for a specific architecture
- Each instruction specifies two things: the operation and operands
  - Operands can come from registers, memory, or constants in the instruction itself
- Assembly instructions are encoded as a word and stored in memory
  - In RISC-V these are 32-bit words

Introduction to Computer Organization

- Processor communicates to memory via an address bus and data bus (bidirectional or two unidirectional buses)
- Control signals such as read and write are on another bus
- I/O devices are connected to the same data bus, control signal bus, etc
- The memory and I/O ports are each assigned a range of addresses called *memory maps*
  - This way we can identify whether a read/write is to memory or I/O or something else
  - Referred to as *memory-mapped I/O*
  - e.g. Memory can be mapped to addresses `0x0 - 0x3FFF'FFFF`, LED can be mapped to addresses `0xFF20'0000 - 0xFF20000F`
    * In this case an address of e.g. `0x10000000` is in memory

Memory Architecture

- Registers are small, so memory is used to store large amounts of data
- Memory can be thought of as a 2D array that you can index into
  - e.g. at address 0 is word 0, at address 4 is word 1, etc
    * This is because words are 4 bytes but memory is byte-addressable
- With a k bit address (k address lines or wires), we can address $A = 2^k$ bytes or 2^{k-2} words
  - The first $k - 2$ bits select the "row", or the word, and the last 2 bits select the "column", or the byte within the word

Notes on Lab 6

- Most non-trivial circuits are separated into 2 functions
  - The datapath (where the data moves), with e.g. ALUs, registers, etc
  - The control path (manipulates the signals in the datapath), with e.g. mux select signals, register enables, etc
- Given a datapath that computes $A^2 + B$, compute $Ax^2 + Bx + C$
  - Registers holding values for A and B; enables on the datapath
    * Inputs are muxed, with both register inputs coming either from the data input or from the ALU output
  - ALU
    * Inputs are muxed, allowing either A or B to go to both inputs
    * 2 operations: 0 adds, 1 multiplies
  - Result register for the ALU
  - To do the operation, we need to first compute A^2, store it somewhere, and then add B to it