CS 250 Midterm 1 - Fall 2022

schari

September 2022

1 Sections of textbook to review

- Chapter 3
- Chapter 4
- Chapter 2.17
- Chapter 5.1-5.10

2 What is Computer Architecture?

3 Information Representation

- Computers use a representation defined by a pair of symbols to represent all kinds of information
  - This is known as a bit, where a bit is defined as either a 0 or a 1
  - A bit string is an ordered sequence of bits
  - A byte is an 8-bit bit string
  - Ex: 01101001 is a byte and a bit string
- The reason computers use a 2-symbol representation is because it is easy to do so by controlling voltage; on is a 1 and off is a 0
- How do you represent a bit string electrically?
  - To represent a k-bit string electrically, you'll want k wires to each hold one bit of the string
  - A bunch of k wires carrying k bits for a k-bit string is a k-bit **bus**.
  - On a diagram, you typically see a bus represented as a single line.
  - A k-bit string can represent 2^k unique sequences
- Hexadecimal Notation

Table 1: Table between hexadecimal, binary, and decimal values

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1000	10
B	1011	11
C	1100	12
D	1101	13
E	1100	14
F	1111	15

  - To shorten the binary data that computers read, we usually use hexadecimal notation to view binary data
  - 4 bits map to one hexadecimal digit

- Prefixes for 2^k
  - Kibi is $2^{10} \approx 10^3$ (which is kilo)
  - Mebi is $2^{20} \approx 10^6$ (which is mega)
  - Gibi is $2^{30} \approx 10^9$ (which is giga)
  - Tebi is $2^{40} \approx 10^{12}$ (which is tera)
  - Note: you drop the last two letters of the 10^k prefix and add bi (for binary) to approximate the 2^k prefix
  - $2^{10} = 1024$ and $10^3 = 1000$

4 Computer Memory

- **Memory** is computer hardware that functions can write data to and read data from
- Memory contains locations where data is stored, as well as unique addresses that point to those locations
  - How do we define 2^k unique "points" in physical memory with k bits?

  - We delegate $2^{k/2}$ wires of the bit string as "horizontal" wires and the other $2^{k/2}$ wires of the bit string as "vertical" wires
  - This creates a grid with 2^k individual locations defined by k bits
- With this, we can define a pointer-mapping circuit that takes a k-bit pointer that maps to 2^k locations in memory
- What actually goes at each of these "locations"?
  - You would use a piece of circuitry called a **register**.
  - The register is made up of 4 parts: k 1-bit latches (1 latch for each bit), an enable line, an input bus, and an output bus
  - The output bus returns the contents of the latches (which each store one bit in the bit string)
  - The enable line tells the latch when to accept new values for each bit through the input bus (the latches won't change in value until we tell it to change)
- Pointing
  - To actually receive data from memory, we use a circuit called a **decoder**
  - The decoder has k wires as an input and 2^k wires as an output
  - Based on the input wires, the decoder will have one of the output wires carrying voltage, while the rest of them have no voltage
  - One of the applications of a decoder is to use the decoder as a pointer-mapping circuit that points a k-bit address to one of 2^k locations in memory
- The multiplexer (mux)
  - The multiplexer is a circuit that takes in an address and returns the contents of the location in memory that address points to
  - It is used to read data from memory
  - It has 2 inputs: an n-bit bus that represents the address of the register you want the data of, and 2^n k-bit buses that represent the wires connecting from the memory to the mux
  - The mux is given the address as input, then the mux retrieves the data from the corresponding bus
  - The mux then outputs the data through its k-bit bus
- The demultiplexer (demux)
  - The demultiplexer is a circuit that takes in an address and some data and writes that data to the location in memory that address points to

  - It has 2 inputs: an n-bit bus that represents the address of the register you want to write to, and a k-bit bus that represents the new data you want to store the value of
  - The demux points to one of the corresponding 2^n k-bit output buses and outputs the k-bit string to write the string to the corresponding location in memory
  - Essentially the inverse of the mux function; mux function reads information, while the demux function writes information

- When a bit string is transported from memory to the processor, it's called a fetch.

5 Processors

- The Harvard and Von Neumann architectures
  - The major difference between the two architectures is that the Harvard architecture has separate memory areas for holding the instructions and the data, whereas the Von Neumann architecture contains it all in one memory
  - Pros vs Cons of each:
    - Harvard
      - Pros: Simultaneous access of instructions and data, can optimize the memory design specifically for instructions and data
      - Cons: Two potential memory bottlenecks; may run out of memory for instructions or data
    - Von Neumann
      - Pros: Requires less memory; both of them occupy the same memory
      - Cons: Less secure because a given address could either point to an address or a piece of data
  - Nowadays, most processors use the Von Neumann architecture
- What are processors?
  - Many people use processor the same way as they use CPU, but they are not the same thing; processors are just a chip that can perform a multistep computation
- A general-purpose processor
  - Why would you want a processor that can do a lot of things vs one that does a few things very efficiently?

  - More cost-effective to manufacture a lot of a few processors than lots of different processors
  - The blueprint of a general-purpose processor contains a few things: a store of memory that stores data, a circuit that conducts computation (called an **ALU**, or Arithmetic and Logic Unit), and MUXes and DeMUXes to read data to the ALU and write the results of the computation back to memory
  - The ALU contains MUXes that point the operands passed in as the input data to the corresponding engines to conduct the computation (ex, an arithmetic engine, a graphics engine, etc)

- Representing machine code execution with a general-purpose processor
  - To execute a computation, you need an address pointing to the instruction that you wish to execute
  - This means that you have to store computation in memory
  - Some may choose to store their instructions separately from the data (which is Harvard architecture)

- Executing instructions
  - A simplistic model for how computers execute machine instructions is the fetch-execute model
  - Essentially, the computer fetches the next instruction, then executes it, and idles while there is nothing to execute

- Clock Rate and Processor Speed
  - Many circuits have a clock that paces the circuit's execution (how often should the processor execute an instruction?)
  - The registers and muxes/demuxes have a fixed delay between operations
  - The different functions in an ALU can have differing amounts of delay
  - Benchmarking a processor's execution
    - Benchmarking a processor's performance when executing a program requires 3 values:
      - $\frac{Instructions}{Program}$ depends on the algorithm being executed (Software)
      - $\frac{Clockcycles}{Instruction}$ depends on the compiler and circuit design (SW + Hardware)
      - $\frac{Seconds}{Clockcycle}$ depends the worst-case delay (HW)
      - The final fraction is $\frac{Seconds}{Program}$

6 Machine Instructions

7 Why Assembly?