1. Building a computer from parts
    a. You know enough at this point to build your own computer
        i. Add two numbers
            1. Use an adder to do so
        ii. Implement other operations like subtract, AND, OR, XOR, so on
            1. Use those components, combine all the above into one unit (an ALU)
            2. Need to pick between them, so MUX / switch to tell us what to do
        iii. Calculate a running sum of numbers
            1. Need storage and state logic
            2. Otherwise, use ALU to calculate values
        iv. Add based on values stored somewhere
            1. Need storage, registers (from sequential design) to hold stored values, and state logic
            2. Otherwise, use ALU to calculate values

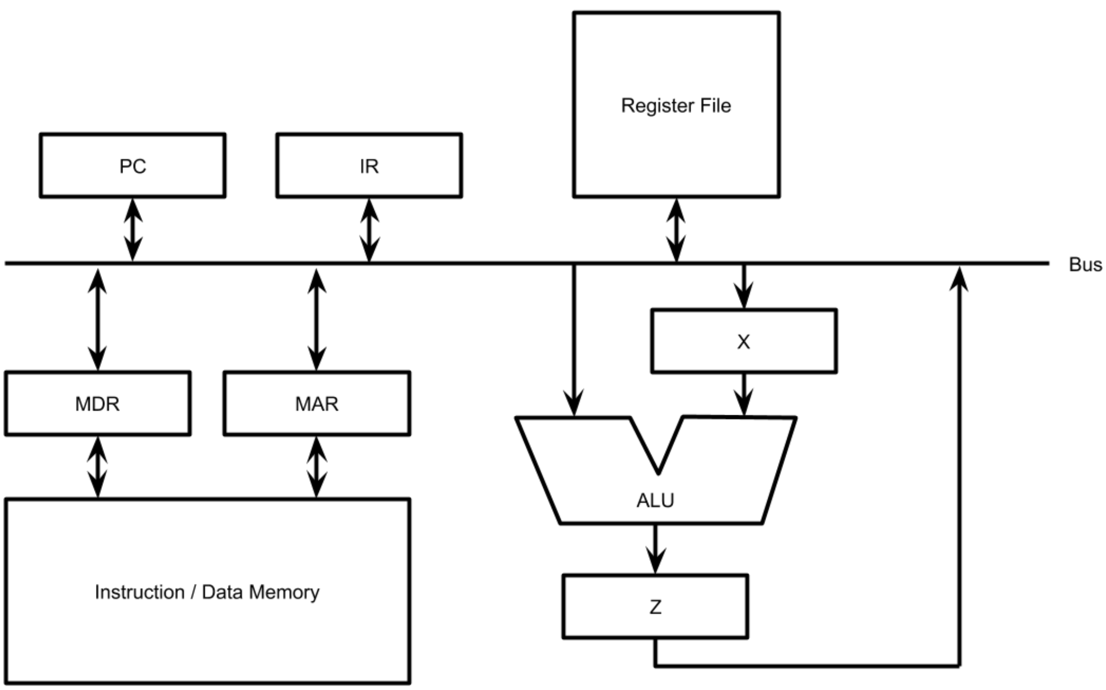
2. Von Neumann architecture
    a. Almost all current machine designs based on concepts developed by John von Neumann
        i. Will follow the same when building our own CPU in Lab 3
    b. Architecture based on following three key concepts
        i. Data and instructions stored in single read/write memory
        ii. Contents of said memory addressable by location
            1. Doesn't matter what type of data is there
        iii. Execution of a program occurs sequentially
            1. To change this, order must be explicitly modified

3. Tasks of a computer (from before)
    a. Move data – in and out of the machine, using input registers
    b. Process data – ALU does this
    c. Store data – bunch of different places
        i. Registers in the CPU
        ii. RAM
        iii. Other ones we'll talk about later
            1. CPU cache, which is a subset of RAM
            2. Backing storage, like hard drives (HDDs)
    d. Control – switches and MUXes

4. Putting together a basic CPU
    a. Need to add registers to hook up to our ALU
        i. ALU must store values somewhere
        ii. Could have direct paths from ALU to registers to store values
            1. This is expensive, though
        iii. Alternative: a *bus*
            1. Collection of low-resistance wires used to transfer information from one place to another (or multiple places)
            2. Analogy: streets
                a. Each house can have a separate path to Trader Joe's
                b. More efficient to share the path, which are streets
            3. Bus often has lines for data, lines for addresses, and lines for control
                a. Inside a CPU, bus only contains data lines
                b. Control, address lines routed separately

b. Let's have registers connected to a bus
    i. Registers will load from the bus
        1. Either from memory, ALU, or external I/O
        2. Need control lines to choose which to load from
    ii. Also need addresses to determine which register to load into
    iii. Register file – collection of registers, each one with different address
c. Keeping track of state
    i. Need to add some extra parts to keep track of where we are in program
        1. PC – program counter
            a. Points to next bit pattern that will be put into the IR
        2. IR – instruction register
            a. Holds the bit pattern that is to be decoded (the current instruction)
        3. X, Z – extra registers
            a. Hold values so multiple things don't have to write to the bus at once
        4. MAR, MDR – memory address and data registers
            a. Interfaces to our memory, which can be considered RAM

5. Single bus and executing instructions
    a. Simplistic single bus CPU below



    b. Sequence of actions
        i. Fetch – need to get instruction at memory location specified by PC into IR
            1. PC places its value on bus, MAR takes in value
            2. Memory returns desired value at location MAR to MDR
            3. MDR places its value on bus, IR takes value in
        ii. Decode – CPU determines what actions to take
            1. Values stored in IR tell CPU exactly what to do
            2. "Decode" the IR to determine what steps to take next
        iii. Execute – run the instruction and generate a value or other action

1. Example – let's add two memory locations and place result in register file
    a. Address of first operand placed on bus from IR
    b. MAR takes in value, memory returns desired value to MDR
    c. MDR places its value on bus, X takes in value for temporary storage
    d. Address of second operand placed on bus from IR
    e. MAR takes in value, memory returns desired value to MDR
    f. MDR places its value on bus
    g. ALU takes in current value on bus and X, places it output in Z
    h. Z places its value on bus, register file takes in value
    i. IR places register address on bus, register file takes in address