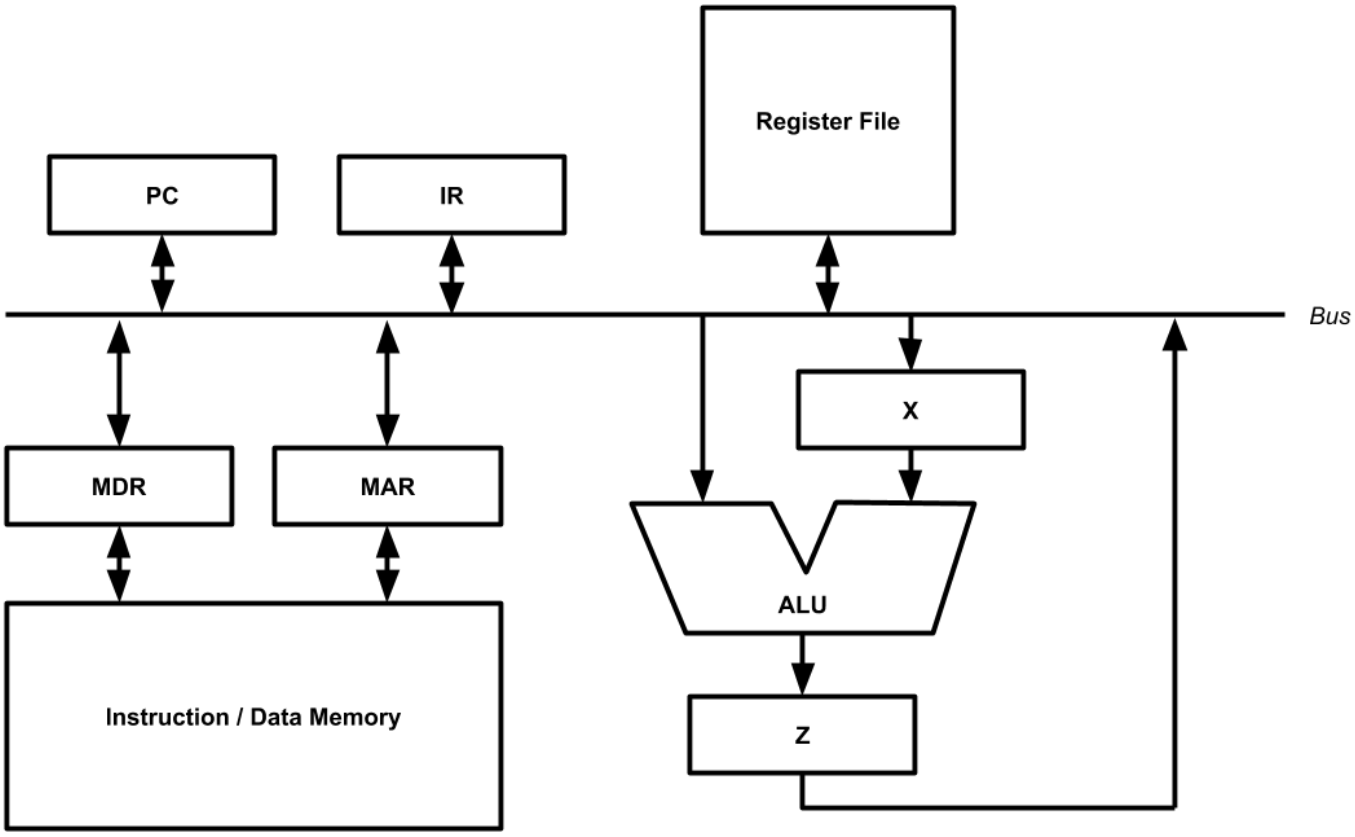
1. From last time
    a. Went over the single bus machine below




    b. Example – let's add two memory locations and place result in register file
        i. Times for each step in parentheses
        ii. Get value at first memory location (2 clock cycles + time for memory read)
            1. Address of first operand placed on bus from IR
                a. IR → Bus
            2. MAR takes in value, memory returns desired value to MDR
                a. Bus → MAR
                b. Tell memory to read from first operand
                c. Memory → MDR
            3. MDR places its value on bus, X takes in value for temporary storage
                a. MDR → Bus
                b. Bus → X
        iii. Get value at second memory location (1 clock cycle + time for memory read)
            1. Address of second operand placed on bus from IR
                a. IR → Bus
            2. MAR takes in value, memory returns desired value to MDR
                a. Bus → MAR
                b. Tell memory to read from second operand
                c. Memory → MDR

iv. Add things together, place in register file (3 clock cycles + ALU add time + register file write time)
    1. MDR places its value on bus
        a. MDR → Bus
    2. ALU takes in current value on bus and X, places it output in Z
        a. X → ALU
        b. Bus → ALU
        c. Tell ALU to add
        d. ALU → Z
    3. Z places its value on bus, register file takes in value
        a. Z → Bus
        b. Bus → Register File
    4. IR places register address on bus, register file takes in address
        a. IR → Bus
        b. Bus → Register File
        c. Tell register file to write

c. Can see that one bus is a huge limiting factor
    i. Lots of contention for the bus, many things want to use it
        1. In particular, fetching instructions versus fetching data from memory
            a. CPU can't do both at once because of the single bus
            b. Throughput is lower than the CPU can handle, could go much faster
            c. Known as the von Neumann bottleneck
    ii. Every time we use the bus for something else, need another clock cycle
        1. Cannot place two things at same time on bus in one clock cycle
    iii. Idea: another bus
        1. Works better, but not perfect
        2. Good for A = A + B
        3. Adds more cost to machine to support another bus
    iv. Solution: third bus
        1. Now we can handle A = B + C
        2. Costs even more
        3. Three bus CPU takes fewer cycles than one or two buses to execute an instruction
            a. Still takes more than 1 to do so, though

d. Couple of reasons why we can't reduce to 1 cycle
    i. Must increment PC to reach next instruction
        1. Requires using the ALU to do so
        2. Solution: place an adder by the PC to do only that
    ii. Complex addressing modes require multiple trips to memory
        1. Solve by limiting the instruction set to a load/store architecture
        2. ALU instructions can only use registers
        3. With all the above, instructions can execute in a single cycle
        4. Idea behind RISC

2. RISC versus CISC machines
    a. Review of material from ECS 50
    b. Back in the 1970s, had dozens of machines and instruction sets
        i. CISC – complex instruction set computer
        ii. Lots of powerful instructions that did a lot at once
    c. Different teams at Stanford and Berkeley looked over this
        i. Looked at making instruction sets simpler
        ii. Keep only a few instructions that can be done very fast
        iii. Found that if you implement a program in this method, can be done faster than CISC
    d. RISC – reduced instruction set computer
        i. Less powerful instructions, and will need more of them to run same program versus CISC
        ii. However, can make the clock *much* faster to compensate