Our simulator is a program that takes in files of written machine code and prints the output to the command line. Similar to the assembler, we read a file filled with machine code and store it in a buffer. Unlike the assembler, we only have to split the machine code by new lines rather than all whitespace. Each line of the machine code is a string of bits, which we pull apart to get opcode bits, register bits, and offset bits or destination bits. To get the opcode bits we get the bits that are from bits 24 to 22 by anding the passed in line with 29360128 which is equal to 2^24 + 2^23 + 2^22. We get the first register location which is in bits 21 to 19 by anding the passed in line with 3670016, which is 2^19 + 2^20 + 2^21. The second register is bits 18 to 16, so we and the passed line with 458752, which is 2^16 + 2^17 + 2^18. Finally the offset bit or destination bit is in bits 15 to 0, so the same additions as previous calculations but 2^15 to 2^0. We also keep track of the current program counter. We store each line as a temporary value and bit shift it to get what opcode it is equal to in order to determine what process to run on that line.

We then have if statements to match the opcode and run the code inside the respective if statement to get our output for that opcode. Then the if statement code follows the guidelines of each respective machine code instruction. Then we error check in jump and link register, load word, store word, and branch on equals to see if we are going outside of possible memory locations. When calculating registers in the if statements, we need to and the current instruction with the passed in register, then shift the A output >> 19 to get the correct register number, and register B >> 16. This is so we can call state->reg[tempA] for example to get what is currently stored in register A. At the beginning of each instruction, we print out the state of the current registers and memory addresses.

The way our program decides when it is time to halt is when it hits the halt case. Our while loop runs while our end case is equal to 0, but when we hit halt we set it to 1, breaking us from our while loop. Once done, we print out the amount of instructions completed.

One of the things we struggled with was our jump and link register seems to be not working. For some reason it doesn't appear that the program counter changes according to the value passed from register B. This causes one of our test cases to be an infinite loop that we are unable to break out of.

We struggled a lot with the multiplication part. We figured out how to use the nand to isolate one bit, so that we could determine if this bit was a 1 or a 0 by branch on equaling it with a bit with the same power of 2. We struggled determining how to use this method in order to shift our mcand.