CEC 470 Project 2 Questions:   
As part of your project, answer the following questions. You may discuss the concepts with others in the class, but each of you must submit your answers in your own words.

1. What opcode will blank memory initialized to 0x00 look like to the processor?

The opcode will take the form of a memory store in the accumulator followed by a 16 bit operand to be used as a memory address.

2. Of the 256 possible opcodes we can get from and 8-bit opcode, how many are not being used in our instruction set, i.e., how many instructions could we add for future expansions of our processor?

256 possible – 128 mathematical ops – 12 memory ops – 7 branch ops – 2 special ops = 107 remaining opcodes. This is further restricted, as 00010111 is an unspecified operation that would need to be a “branch” op, and 0000xx11 would all be memory operations. If we consider these opcodes “specified”, that would take away 5 more opcodes, leaving us 102 entirely unspecified opcodes.

3. What would we need to add to our simulator to be able to include the following instructions: compare ACC with a constant, PUSH to or PULL from the stack, and take the 2's complement of ACC?

We’d want to add a branch option of type 111 to represent the need to look for a constant after the memory address and compare to ACC (00010111). To push or pull, we’d want to change the options for register in load and store; we could accomplish this by assuming that ending the command with 11 would signify using the stack instead of whichever register was otherwise specified (0000xx11 -> stack command, limited to assuming we’d always use a particular method). To take 2’s compliment of ACC, we’d most likely need to use a special opcode, as the mathematical ops cover the entire opcode (there are no available opcodes that fall under math in our system). This could specifically be 00011000 or any other variant opcode that wouldn’t overlap with existing special opcodes.

Because comparisons can already be handled by the system, I assume there would be no added complexity beyond creating a designated opcode to enable us to compare ACC with a constant; PUSH and PULL ops would behave the same as loading and storing, so assuming it’d be sufficient to use one method for every PUSH / PULL op, we could accomplish this by adding a stack command variant to our load/store ops. Finally, to achieve 2’s compliment, we’d need to invert and increment ACC, which we have the ability to do, we’d just need to perform that using a special opcode.

4. If executeInstruction() were divided into two parts, decode and execute, what additional global resources would be needed for your simulator?

If we divided executeInstruction() into two parts and didn’t allow parameters to be passed, we would need to consider enumerating the opcodes or creating functions for each operation to be called from decode; basically, by separating decode, we need some way to guarantee that we execute the correct function in execute().