1. **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?**

We are currently using every opcode starting with 1 (mathematical and logical operations).
We are not using any opcodes that start with 01, which is *64 opcodes*.
We are not using any opcodes that start with 001, an additional *32 opcodes*.
Opcodes starting with 00010 are branch operations, we are not using *1 opcode*.
Opcodes starting with 00011 are special operations, we are only two of them. This leaves us an excess of *6 opcodes*.
64+32+1+6 = 103 unused opcodes

2. **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?**

To compare ACC with a constant, we would need to add functionality to the branching operations to allow an input of a constant instead of just comparing ACC with 0. Or, we would need a new register for the comparison operation and then a set of flags that stores the result of the comparison.
For PUSH and PULL, we would need a new register to point to the location of the stack in memory.
For 2's complement, we would need a new mathematical/logical operation that could do "ACC = - ACC".

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

In my program I already divided executeInstruction() into a decode and an execute. In order to do this I had to add a data pointer that stored the data needed for the instruction execution (A.K.A. the operand) and then needed to create additional functions that passed around the pointers to the information I needed.

4. **Make suggestions for ways to further subdivide the executeInstruction() function.**

executeInstruction() can be further divided into:
Decode instruction,
Fetch Operands,
Execute Instruction,
Store Operands.