**CS281 – Homework #2**

1. Go back to the course slide and revisit the code for the `swap()` function we discussed in class. Recall to execute swap we execute the `jal swap` instruction, and then when swap() finishes we returned to main using `jr ra`. Both instructions manipulate the program counter (`pc`).

   For example, `jal swap` first puts `pc+4` into the `ra` register and then sets the `pc` to the address of swap. On the way back, `jr ra` sets the pc register to the value of `ra`. Can you think of any good reasons why the RISC-V designers chose this model rather than letting the programmer just set and manipulate the `pc` register directly. Note that the `pc` register is an internal register used by the processor, it is not exposed to the programmer to manipulate.

2. RISC-V, like many processor architectures allocates a fixed number of bits in each instruction to indicate the opcode. Specifically, RISC-V allocates 7 bits for the opcode resulting in a maximum of $2^7$=128 instructions. While this is sufficient to support software on RISC-V, the limited instruction set can impact program readability and understanding, leading to bugs. Describe one way the assembler tooling can help with this problem.

3. Suppose the program counter (`pc`) is set to 0x200000000. What range of addresses can be reached using the jump-and-link `jal` instruction? (aka, what is the set of possible values for the `pc` after the jump instruction executes?)

4. Repeat question 3, but instead of the `jal` instruction, consider the `beq` instruction. What is the set of possible values for the `pc` after the `beq` instruction executes?