In project 1, we translated an assembly code into a machine language (i.e. hex representation of binary numbers). The basic idea was to write an assembler that checked for syntax of the assembly code and using the architecture specification, we translated the opcode, registers (source & target), etc. into its numerical representation. This process was done in 2 passes, first pass went through the assembly code to look for symbolic representation of labels and created a label table to store the location of each label. The second pass was used to finally translate the assembly language or syntax to its binary representation. For example, instruction opcode “ADD” was translated to “000” in binary and any operands following the opcodes were also translated into binary and using the label table to fill in the labels into its corresponding index. It’ll be assumed that the error checking is done appropriately within the assembler (e.g. register value is not out of bound, symbol exists, etc.).

 Once the assembly code has been successfully translated, the machine code is then fed to a processor (in project 1, a sequential version of RiSC processor). The program is loaded to a memory register and upon power up (and initialization), the program counter (PC) fetches its first instruction, decodes, executes, and stores (to the memory and/or register file). This process is repeated sequentially (i.e. at each clock increment) until the program terminates (no more instructions to execute).