Design Process Journal

Meeting #1 – First the type of Processor that everyone wanted to build was discussed. The vote was between the accumulator and the stack, with the accumulator winning. The accumulator was chosen with the reason that it could potentially be more user friendly. Git Hub was also agreed to be used as the group repo. The next topic was what everyone wanted to focus on when making the processor. It was agreed that we all wanted to make the processor fast in speed but also user friendly. We then discussed potential types and instructions that we would be implementing.

Meeting #2 – The first item discussed in this meeting was the types and there bit usage breakdowns. I, R, and IR types were agreed upon except we were not able to fully agree upon a bit distribution for the R-type. The I-type has 12 bits for the immediate (which we thought would be more than enough for add’s, and’s, or’s, etc.) and any command with I type will be using the main accumulator register. So for example, the I-type add will add an immediate value to what is currently in the accumulator. The IR-type has access to any of the registers and to a 9 bit immediate value. Our ideas for this IR-type included being able to do comparison to a register (for something like a beq) and then going to an immediate value destination. We then listed out our commands that we wanted and gave them operation codes. We also started working on how some commands would be implemented in assembly code. We also agreed upon using 8 registers including the main accumulator register.

Meeting #3 – The R-type bit distribution has been decided upon. The final R-type has a func code with 2 registers and a tailing 2 unused bits. Our thoughts here, included being able to add, and, or, etc. two registers together and place the value into the accumulator. The func codes were the next item discussed and agreed upon. We have gone through each operation and wrote out how we would like them written in assembly code. Much of the documentation was done during this meeting because we now have everyone agreeing upon the current design of each command and type.

Meeting #4 – The code for Euclid’s algorithm written in our assembly code is almost complete. There has been a key point, raised when trying to do branches using our accumulator. It has been decided that we will create two more operations in the func codes, beqz and bnez. These are branches will compare values against zero. The idea behind these comparisons is that there will be much less swapping of values into registers if we can just check to see if the value in register is zero or not.