Luke McNeil
Milestone 2

1) Thursday, Friday October 8-9, 2020

[4 hours]
Created a simulator for our stack language written in Chez Scheme. The simulator can take a program such as our relprime example and then run it, simulating the stack as a list.

2) Tuesday, October 13, 2020

[3.5 hours]
Wrote the RTL descriptions for add, sub, or, dup, swap, drop, and over. While doing this I had to figure out how we would refer to the stack of registers in RTL. The way I decided to do it was with Reg.push(value) which pushes onto the stack a value and Reg.pop() which pops off the top thing on the stack.

I also edited our design document in response to Sid's feedback of M1. This includes making places for a title page, table of contents, executive summary, and additional sections. I then combined the table showing instruction description with the table showing opcode and funct into 1 table. I also replaced exit with halt, an instruction which always jumps to itself.

I then wrote an assembler for our stack language in Chez Scheme. I used several procedures from the simulator to read in the program. I then used a hashtable to match opcodes and functs, and then had to do some annoying stuff with converting a number to binary. This seems like it should have been easier since the number is actually being stored in binary, I just don't know how to get to that.

3) Wednesday, October 14, 2020

[3.5 hours]
I recreated the RTL for the ones I had previously done and added it for all of the instructions. I changed from using Reg.pop() and Reg.push() to treating the stack as an array of size 64 where stack[0] is the top of the stack.

I had to think about what would happen when our stack machine runs out of registers on the stack. I decided there would be 64 registers in the stack and 64 registers in the return address stack. For this round of RTL I just assumed that data would be lost when the user tries to add something to an already full stack. This is not ideal, but it will make the implementation easier. This might change in the future.

I then made a factorial program and put it in implementation/example-programs/fact.asm. I used this to see if limit of 64 registers would be enough. In this example it is. This is because of the fact that factorial(8) is already bigger than a 16 bit number. Using updates to the simulator I found that factorial(8) only had a max-stack-size of 11 and a max-return-stack-size of 10. This is plenty less than 64. This might not be true though in other examples where the result does not grow to be more than 16 bits so quickly. A good one to try would be fibonacci.

I also refactored the Design document to contain many different

```
   sections to make it more readable.

For M3 I will
    a) Get up to speed in Xilinx
    b) Start on datapath
    c) Write tests for small components
```