Objective: The objective of this lab was to create a program that would take in multiple RISCV Instructions and convert them into a 8 digit hex value. This was done over the course of 3 weeks with help from Jack Burgess, Ryan Clausius and Zach Millsap

Overview: In general our program is supposed to take in three different RISCV programs. Once we take in these programs we have a file prashing function that will read in these files. At which point we call our function Code To Hex. Which then sorts these based on there op code type. For example it sorts it as R, I,S,B,U type. From here it will then sort into the different instructions within those different types. For example the R instructions has an ADD SUB XOR OR AND, which our code will sort these into. We will then hard code the Op code based on whatever type the instruction is. After that we will hard code in the function 3 and function 7 if it requires it.

The last step is to then encode the register or immediate values. So our code to hex program takes in register 1 , 2 and RDX values. With these values we are able to convert these binary arrays into actual hex values to then store into our hex char string. For example our file function might pass in ADD X8, X25, X15. At which point the registers will be passed as int values. These int values will then be converted into binary arrays. So since each of these registers is 5 bits we create a new 5 bit array for each register and then assign each element to its corresponding binary number. This is done with a simple decimal to binary converter that will determine if that element is high enough to have a 1 in that place.

These arrays are then passed back to the Code to Hex function. And Hard coded into the longer 32 element Binary array. Once we have the 32 binary array completed with all of the different sorting that is done we will then call a function called Binary to Hex. This function will take in an int array with size 32 and then convert it to hex. It does this by breaking the array into 8 separate parts. Each part is 4 bytes long and represents one hex value. We then sort it based on the 16 different hex values something can be. The result is stored in an 8 element char array. Then we move to the next part of the binary array and this result is stored in the next element of the char array. Once we are done we inverse the char array and return it back to the main function.

We added the following instructions to our lab 1 to test and see if what we had coded was correct. These being SW, LW, ADDI and BEQ. These were the only instructions that we found that we did not include in our original lab because they were not required at that time.

Struggles and Challenges: We ran into multiple problems during this lab. This first issue that we ran into was trying to read from the file. To solve this problem we had the reading function break down each thing into separate parts. This includes sorting it into different types. For example if a command is an R type then we need to store its name, and the 3 registers. If the command is an I type then we need to store 2 registers and the immediate. This was done for each of the types. Another issue that we had is that we could not determine how to show the Loops or the exits. This is because these do not have any hex value that we could convert them into. As such whenever you actually run the program the instructions will not include loops and it will be on the programmer to know how many loops to include. This is because we could get our function to jump to the create thing.

Another issue we had was with sorting the different types of instructions into there different parts. To solve this issue we mostly did it by hard coding in different addresses because many of the addresses would be constant. Mainly because of the Hard set Func3 Func7 or the op code. These would remain similar to every single instruction so we could just hard code them in based on the name of the instruction.

The last major issue we had to overcome was trying to change the actual instruction code into hex. It became clear that we knew how to sort them into 32 element binary arrays. By knowing this we just had to create a binary to hex Function. The binary to hex function solved most of our issues as know we were able to sort the instructions into the different parts. These parts were then once again sorted into 16 different switch cases meant to represent the 16 different hex values which is then returned as a char array.

Milestones and Implementation: Here are some key moments and milestones taht we thought were worth sharing. These are in order in which they happened.

- Group was created and the original code was reduced to a compilable form
- Op code deconstruction was learned and the input file was changed from MIPS to RISCV 5
- The work was assigned to each person and work was started
- Zach and Jack created the story board for the different sorting functions
- Ryan started creating the binary array and its struct
- The File reading function was completed

- The R type sorting was completed with different hardcoded addresses for different Func3 and Func7
- The I type sorting was completed with different hardcoded addresses for different func3 values.
- The S type sorting was completed with different hardcoded addresses for different func3 and immediate handling.
- The finally B type sorting was completed
- We were able to store what the file read into a binary array
- The Binary array process was completed and now we could begin the hex
- The BinaryToHex function was completed and the correct hex array was returned for each function
- The different bonus assembly programs were wrote by Jack and Ryan
- Some more specific instructions were created in CodeToBinary
- These cases would now work
- The lab report was completed and check by the group
- The lab report and code was submitted by the group

I will now be talking about some of our Implementation in a more board way as some of the more specific things have already been covered. One thing that we did is call and pass all of our array variables by reference. This is mainly because these values are already pointers so it was easy to just return them as in. We also declared them as Global variables so we can reference them outside of the different functions.

Another thing we did is to represent each of the binary and hex values as char arrays. This was so that whenever we are sorting them we could more easily assign a certain bit or section of bits to an actual hex value by creating temp variables and comparing them with hard coded switch cases. Although this is slower than a division or a mod algorithm it is much easier to write and much easier to understand and debug.

One area in which we were not able to successfully complete was the Loop instructions and the exit commands. This is because these are stack pointer values and so we left them as zero as we did not know how to read them in or decode them. Another thing that we did not successfully complete was reading in the SW instruction. This was because the file reading function could not correctly sort these out. Then our hardcoding would be wrong because of these sorting errors.

One last thing that we did that was somewhat unique is that we created specific variables for each register. The thinking behind that is we could be able to assign each of these register to a certain block of the binary array. This way the 32 element array could hold the right registers in the correct spots.

Work done by group member:

Zach: Created the File read functions using pointers and sorting the different parts of the instruction into certain variables. He also worked on the story boarding and help debug some of the errors

Ryan: Created the code to Hex function. He also helped create the Code to Binary function. He also helped with the bit sorting and the hard coding. Ryan also wrote the first question in assembly regarding the bubble sort. He also created the different switch cases for the I S B types.

Jack: Created the Binary to hex function. He also wrote the lab report. Also story boarded with zach. He wrote the code to binary function with Ryan and created the R instruction type. Helped divide work and manage the group. Lastly Jack did the 2nd assembly question regarding the fibonacci numbers. He also mapped out all of the different 32 bits in the array and did some of the hard coding with ryan.