# Project Report

## Task List

**Nasir Ratnani**: Reading in instructions/operands from a user provided .asm file and parsing instructions/operands so that they can be sent for comparison. He did his role effectively when making sure that the file was read correctly as well detecting any new line changes. He wrote his own helper methods that were necessary in detecting the end of file as well.

**Raviteja Lingineni**: Parsing Input and Processing the Operations as well as handling labels. String Comparing instructions/operands with those supported, ordering the supported instructions/operands into binary. He did his role effectively in making sure these tasks were complete. He researched how to do space delimitation as well as handle the ordering of the elements for the instruction.

**Sriram Bhat**: Transforming supported instructions/operands into binary and outputting into binary file. He did his role effectively in making sure the tasks assigned to him were complete. He researched how to output into a file as well as creating the necessary sub-classes.

## Project Description

This project uses MIPS Assembly Language in order to assemble MIPS instructions into their proper binary format so that it can be read by the MIPS processor/simulator and execute the instructions. The project runs through the MARS 4.5 MIPS Simulator. It assembles with a given file named “test.asm” within the home folder of the user’s computer that contains the sample input to be run. Altering the value of the file\_location label within the InputReader.asm file can easily change this.

Here are just a couple we support, more are listed in the readme pdf:

* add R[rd] = R[rs] + R[rt]
* mult {Hi,Lo} = R[rs] \* R[rt]

## Program Implementation

The entire program itself consists of many subfiles and procedures. At the core, the idea is quite simple, we read in a file, we parse it and then we process what we parsed and then print out the output. We wrote program in a way that it mimics the same thing we as humans do when trying to convert to binary. We created a file that could take in the operands and order them immediately as they come in. In order to determine the right operation type we used a string equals method against our lookup table that consists of multiple parallel arrays. The beauty of our implementation is that is it is very easily scalable and efficient. The methods that we used in implementing the R-Type become useful in the

## Project Process

In order to do this project we initially set out tasks for each member to research, as specified in our project checkpoint. During the middle of the semester one of our team members, Ozair, had to leave due to personal-issues, thus increasing the workload on the remaining three. However, we were still able to redistribute the research tasks. We knew that having a member short, we didn’t want to task ourselves by branching out too far, so we set attainable goals. Our primary goal was to get the R-Type instruction working, and once we were able to achieve that, we decided to move on. We used a methodology of divide and conquer to make our program work.  Once we were able to tackle the issue of reading in files effectively, we worked on splitting the files at the end of each line. As this goal was accomplished we further broke down the individual lines into separate words, which would then be analyzed through comparisons with supported instructions/operands and handled independently depending on the instruction they symbolize. Finally each instruction/operand translated and reassembled together as a binary string which is then output to a text file as the final result.

## Challenges Met

* Reading Input:

Because there is no such thing as a universal readline method within MIPS we had to come up with our own method of reading in information from files using the syscall process, and then handling end of line and end of file events. Our approach to this was to take in file input character by character and proceed from there. Once at space it hit we know that the instruction has been taken in. This leaves the operands, which are separated by commas. Because we can already split at spaces, we replace commas that are taken in with spaces. This allows our split method to be reused, thus saving us any unnecessary code. After this point we proceed to parsing words.

* Parsing Words:

One of the most challenging parts of the project was parsing the words. Each instruction in MIPS comes as a different argument and argument lengths. We need to account for spaces and commas as well as any comments that might follow.

* Word Lookup:

In order to get the opcodes and certain operations we needed to create the operation codes as well as function codes and types in a structure. We were able to this using asciiz words, and then we put them in arrays as words to keep them aligned regardless of character length. We also made all of the arrays parallel, that way one lookup can give us the indexes of all the data we might need.

* Handling I and J  types:

The interesting thing about I and J types is that they deal with labels. Messing with labels means that  we now have to mess with the program counter and log a value for that somewhere. We made an attempt to make this happen, our program would read the labels and store them in array, however, we were unable to resolve an error where elements would be overridden in the array thus making it hard for us to receive the loop counter variable.

## Experience Gained

* Using the Stack

It is safe to say that we are no longer novices in the MIPS programming language. The biggest thing we had to accomplish in this project was doing multiple actions, and in order to do so, we needed to become comfortable with the stack. We realized many times, often after many hours, that we kept failing to push the return address on to the stack pointer and our programming kept stopping. We also realized that variables that weren’t on the stack were manipulated and more than from class, we learned this from firsthand experience.

* Using Git

Since the project was done as a group project, code sharing is an interesting question. We all knew how to use github, so we ended up collaborating and committing our changes. We all got more familiar with the usage of git commands to pull and push as well as using force pulls to make changes and keep track. We also began write our documentation on Github in the event anyone in the future might be interested in continuing the project. Our code is commented well.

* Data Management

My particular role in the project had to deal with creating structures that can be quickly searched as well as easy to add data to. It was troublesome at first to mess with arrays in MIPS because there is a lot of jargon between bits, bytes and words. I think I gained a lot valuable experience and expertise working with arrays. Also using spaces as buffers was new to me, so I took that as a learning challenge and moving data to the buffer using string copy methods was interesting.

\*We have made a collaborative effort on this document, but we have made our own modifications as well as additions to make it personal as well.

Here is the link to our official Github Repository:  
<https://github.com/NeoScript/3340project/wiki>

Here is a link to our Youtube Video that demonstrates the project in action:

https://youtu.be/C4KgjyG9hcs