**Report**

**Introduction Word Count: 624**

The first thing that we did before even started planning for our project was completing research. We did this to make sure we fully understood the journey we were about to begin. We looked at the provided lecture slides to jog our memories and the Manchester Baby Wikipedia page for more guidance before properly starting our project. We all agreed that g++ was the compiler that we were going to use.

**Approach/solution**

When we first began our planning, we decided that a flowchart would be a good way to represent the program as a whole and to get everyone “on the same page” after this we decided that the best idea would be to tackle the store before moving onto the binary functions. We began by making a method that would read in baby code from a text file to store in a vector, we quickly realized that this would take a number of methods and wasn’t as simple as we previously thought as we ended up having to Write new items to new indexes displaying the whole store.

After that we decided we would design the binary operations, this involved incrementing, converting from binary etc. This was fairly straightforward to design and implement. Due to the nature of the baby this meant that we ended up with an Operations.cpp file that had many methods within it, including even a print function.

Due to the operations being complete it meant that we could move onto working on the actual fetch-execute cycle as we had everything in place to do so.

We ended up designing and eventually implementing a “FetchExecute” method in the main.cpp file, this as the name implies handles the fetch execute cycle and uses a while loop to Increment, Fetch , Decode and Execute using a number of functions.

We then moved on to working on extended memory and making the Manchester baby be able to use negative numbers.

After we got these to work and were on the right track it came down to a whole lot of testing, making sure everything we implemented was functioning correctly and no bugs went unnoticed.

Once we completed work on the Manchester baby, we moved onto working on the assembler. Our original plans had to be altered as we realized we had to make use of a symbol table. The first thing we did with the assembler was create a method to read machine code into the program. Then made methods to add data to the symbol table, after the simpler methods were out the way we began working on the actual conversion. Our InstructionConversion() method consisted of a list of if and if else statements which reads the Opcode from the text file and returns a value. Our other conversion methods were far more time consuming and involved converting Integer to Binary and removing white space.

**Problems**

We experienced many problems during our project. One of the most time consuming was probably the Symbol table implementation simply because we didn’t know much about symbol tables. And this meant we had to do research before even attempting to implement it. Another problem we had was with the Converting int to binary in the operations cpp file. This was problematic because it was simply just a very complicated method to implement and involved multiple conditions and even the use of twos complement to work properly. Aside from these big problems and the occasional seg faulting and syntax errors, our implementation went well.

**Conclusion**

In conclusion our project went well and we worked well as a team without any conflict or arguments. We also were very organized and began work early so that we would finish comfortably and in good time.