# Introduction

For the third assignment in Computer Systems 2a, we were given the task of creating a simulation of the Manchester Baby hardware able to demonstrate execution of a binary machine code program stored in the memory of the computer., as well as an assemble to convert an assembly language program into binary machine code, which can then be loaded and ran on our simulator.

# Approach

We began this assignment by doing research on the original structure of the Manchester baby, through the use of the internet and previous lecture notes. we did this as it allowed us to gain a better understanding on what we were to create, as we learned more about the fetch-execute cycle. After we had gained a thorough understanding of what was required from us, we began to work upon our code in the labs.

We first began working on the hardware simulator. We planned for the simulator to have the different functions of the baby implemented. Implementing the ability of our program reading a text file stored on the computer, that contains a binary computer program as an input file, into our Simulator. The binary file would have been translated by the Assembler from the machine code language of the Baby into binary code that the Simulator could pull into the store in order to translate and run the code.

Andrew worked on the assembler, which would be taking another text file containing source code for an assembly language program as an input file, and then it would convert the code into the binary machine code that the Baby is using.

# Problems

When doing the assignment, we discovered a problem that repeatedly occurred with the binary to decimal conversion, and vice versa. This proved to be a problem as we initially had created a binary to decimal conversion, but using strings. Some of the numbers we needed to translate from elsewhere in the program needed to be stored in strings in order to remain functional to be used later and some were needed as integers. Answer tables different points during the program we needed to convert different data types to and from binary whilst receiving the result in a completely different format than the original. This meant that there were a lot of different scenarios we needed to cover for. As a team we discovered that a partial solution to this was by using many different versions of converting int types, character types and string types. Another difficulty that we faced was that of opcodes. Opcode is an operation code that tells the processor the job that needs to be done. It was difficult to implement this as we had never come across code like this before.

# Solutions

We were able to overcome the binary to decimal conversion (and vice versa), by using examples, and loops. This allowed us to created instances where the code knew what to do for certain numbers, and thus was able to implement this into other numbers that came about. With the initial conditions of the program we were not initially able to created a robust conversion method. In order to overcome this problem, we decided to plan out all of the different scenarios that could happen, including small and high numbers, negative numbers and also numbers that would make the data type storing them different sizes which caused problems originally with our code. We worked together discussing the steps we were taking to convert the numbers logically and invested in researching this problem in detail until we were able to succeed in our tests of an exclusive set of testing numbers and strings to convert.

The problem with the opcodes was solved by using a string, which contained the opcode and a binary to decimal conversion, as this was able to change the binary into what we knew it was, and thus link it with the appropriate operation.

# Linux C/C++ compiler

Our makefile is included with the program files. We use c++11 version of the compiler for both our Assembler and Simulator programs. g++ -std=c++11 -o *Name* -Wall -pedantic -Wextra *Simulator.cpp baby.h / Assembler.cpp.*

# Conclusion

In conclusion, despite finding the task initially daunting and difficult to approach, we believe that over the course of the assignment that we became well versed in C++, have improved our ability to work as a team of programmers designing and implementing code, and have gained a great insight into what it took to create the original Manchester baby, allowing us to complete our task to a high standard.