# AC21009 Assignment 4 – The Manchester Baby Report

**Group:** 11

**Participants:** David Topping, Christian Zlatanov, Mathew Gallahcher, Stacy Onyango, Conor Kelly.

**Word Count:**

**Compile Command:**

g++ -std=c++17 -Wall -Werror -pedantic -g -o main menu.cpp assemblerFiles/assembler.cpp assemblerFiles/bufferLine.cpp assemblerFiles/instruction.cpp assemblerFiles/instructionSet.cpp assemblerFiles/outputBuffer.cpp assemblerFiles/symbol.cpp assemblerFiles/symbolTable.cpp assemblerFiles/utility.cpp manchesterBabyFiles/manchesterBaby.cpp manchesterBabyFiles/memoryLocations.cpp

**Run Command:**

./main

## How we approached the assignment

Firstly, we talked on a Microsoft Teams chat and agreed as a group to research and have a full understanding of the tasks required before we met. During the initial call we discussed how we would manage the code; we decided on using GitHub to host our code. We then split our group into two, one half would work on the Manchester itself and the other half would work on the assembler.

For the Manchester baby we split things down into a few main categories: data structures, reading in the machine code, incrementing the control instruction register, fetching the program instructions, decoding the program instruction, and executing the instruction. The aim was to try and split these tasks up as evenly as possible based on the time it would take to implement them and difficulty. This was achieved as well as we possibly could.

For the Assembler it was a much easier task to split it into 4 main classes: The Main Assembler, The Instruction Set, The Symbol Table & The Output Buffer. These classes helped us implement the assembler since they were 1:1 representations of the major parts of an assembler.

## Problems that we faced

When making the Manchester Baby Simulator, initially we didn't have a full understand of the brief. This meant when we went straight into programming the simulator, we were making silly logical mistakes. We realised this was not a sensible way to face the problem; so, we stopped and planned each class with their functions and members. This worked a lot better and meant we had a clear understanding of the tasks.

It was assumed that everyone was confident using GitHub for code sharing. However, this was incorrect and let to mistakes with the GitHub repository that were very time consuming to fix. We then made sure everyone was up to speed and knew how to use GitHub. Spending this extra time saved us a lot of time overall.

It was a significant challenge working with binary numbers within the program and performing operations on the binary numbers. It took a lot of time to work out how to increment a binary number and have them in a suitable form for our data structures. We used bitsets to help us deal with this. A function was also found online that increments a bitset by one, this was extremely useful for incrementing the CI register.

One of our group members took sick during the assignment. As they were part of the assembler team they had to meet and discuss how they were going to approach the problem by reallocating the other members tasks.

When implementing the assembler a few problems came up. Examples of these problems are: Deciding how to implement the Symbol Table & How to decode each line and translate it. It was decided that the implementation of the symbol table would be a linear, open-addressed hash table. This allowed us to dynamically resize it depending on the number of symbols needed and easily find symbols when they would be needed later. It took a while on how to decide to determine the various parts of the assembler lines. We eventually decided that something was a label if it ended with a ‘:’, anything with a ‘;’ in it was a comment, if the exact word was in the instruction set it was an instruction and anything else was an operand.

## The solutions that you have created

We have created a Manchester Baby simulator. This has a main menu that allows you to run the default simulator. This reads in the machine code from a file, line by line and stores all the data in a data structure called the store (an array of 32 registers). The CI is then incremented, the instruction is fetched from the store, this instruction is decoded and then executed. The main menu also allows you to run the simulator but with a custom store size, the custom store size allows you to have bigger programs etc. As an extension to the Manchester baby there is another menu option that gives the option of running the Manchester baby but with a custom store size (over 8).

The assembler can also be interacted with via the menu. The menu allows you to choose the paths of both the input and output files. When the user chooses to run the program, it will read each line in the input file and output messages to show the user what it is doing. It will also end when the program reaches an error.