# Assignment 3 Project Report for Team 16

Jason Khan, Isaac Lowry, Paul Ritch, Patrick Turton-Smith & Michael Wamarema

# Our Approach

First of all, we agreed to split into two separate groups, with three of us working on the simulator itself, and the other two working on the code assembler for the simulator. This was decided on which part of the project each member of the team felt most comfortable working on. Within those groups we delegated tasks to each other to help speed up the project.

Whilst both groups typically worked independently of each other during the project, at the beginning we all contributed to creating the basic structure of each program with a focus on agreeing aspects that would affect both programs. This was done to ensure maximum compatibility between the two programs. We met up regularly to discuss progress and issues we had encountered. Everyone uploaded their code to a shared repository on GitHub so everyone in the team had access to each other’s code and could make their own programs compatible with the rest of the team’s.

# Difficulties encountered

We encountered several difficulties during development of the simulator & its assembler. Most of these difficulties were based around initially misunderstanding the structure of the Baby and its instruction set, however this was quickly resolved after some further research into the Manchester Baby itself. It was fairly challenging to be able to relate to and comprehend how the Manchester Baby was run, as many of us feel quite distant from the old age of computing. After further reading we were able to understand the concept a lot better and this in turn helped us in creating the program.

While working on the assembler, we had to make two passes over the assembly source code and translate it into binary. The main problem encountered was retrieving the operand, as the line number had to be retrieved so the code knew where the variable operand was. This was solved by iterating through a vector with the source code inside it and obtaining the index of the string element where the variable definition was stored. Another issue was implementing syntax error checking into the assembler. The solution we came up with was to sort each word during the second pass and find out if there is a variable definition for it, by searching through the source code vector. If the word was not a comment, instruction or defined error, a syntax error message would be output.

# Changes we would make

We wanted to give the user the option to run the simulator with a larger instruction set to allow for more complex programs in addition to the option to increase the memory size (which was implemented). Unfortunately, we ran into problems adding this whilst retaining backwards compatibility with the Baby’s existing instruction set, without which would’ve rendered any program made for the original Baby’s instruction set un-executable on our simulator. Additionally, we wanted to provide support for multiple memory addressing modes, but we ran out of time to implement this.

# Description of the final solution

The final solution submitted authentically simulates the operation of the original Small Scale Experimental Machine – the “Manchester Baby”, and also contains an assembler to compile assembly language code into machine code ready for the Baby to execute. Both the simulator and assembler were written using C++ and compiled using g++ on Linux.

Our project has the following key features:

* ability to compile assembly language code into machine code compatible with the Baby,
* ability to catch syntax error during compilation,
* ability to run machine code programs compiled for the Manchester Baby, with the state of each component of the Machine displayed after each instruction is executed,
* ability to extend the memory space of the Manchester Baby to increase the length of the programs that can be run on the Machine.

Word count: 625