Manchester Baby Report

# How we approached the assignment

For this project, we started by deciding to use C++ instead of C because we felt more comfortable as a group in using C++ and additionally we all had to be in agreement because otherwise it would be too tricky to put together all branches of the work and not have conflicts in the code.

As the project has two separate programs to create, we decided to split our efforts from the beginning on the Manchester Baby simulator and the assembler. Three members of the team would work on the simulator while the other two would work on the assembler, and the idea was that whichever section was completed first, then the group working on that would go on to help on the other part of the project.

When first meeting, we found it useful to go through the appropriate literature provided regarding the Manchester Baby, especially “Burton (1997) – Programmer’s Reference Manual for the Baby” which provided useful insight into how exactly we should go about designing the simulator in particular – for example understanding that the first 5 bits were the operand, the next 8 bits meant nothing, then there was the 3 bit op-code, and finally everything after that was meaningless too.

Experimenting with the TOM (thoroughly obedient moron) software during the lab hours was also a beneficial tool in learning about assembly language and was useful because it allowed you to write your own programs, and through this increased our understanding of how the computer deals with instructions and how the registers within the processor work to compute the value(s) set out by the instructions.

# The problems we faced

1. An issue faced when working on the simulator was finding the best way to store and convert the data taken from the machine code, whether that be keeping the data as binary or converting to decimal and then back and forth as needed.
2. Another issue was with when converting the binary value from Big Endian to Little Endian (which the Manchester Baby deals in) by using pointers the original array containing the binary value would be altered, meaning that it could not be used again for other purposes.
3. In the simulator coding process, we mistakenly went for immediate addressing in which a constant value is passed to the accumulator instead of register addressing where the register in the specified operand is fetched and then the instruction is executed.

# The solutions we created

1. We decided that it was best to keep the data as binary so that it could be more easily manipulated and so we thought that the best way to store the binary was in a char array. Also, storing as binary keeps with the nature of the project which heavily leans on the binary format.
2. To combat the issue of the original array being altered, we ended up using temporary arrays to store the Little Endian binary value.
3. Instead of a constant value, the location of the register to be fetched was passed instead which is in accordance to how the Manchester Baby operated. Therefore SUB 12 would fetch the value stored at that location and then subtract that binary value from the binary value stored in the accumulator.

# What we would do differently

1. We would look to make it so that the assembler could turn machine code into assembly language code, which debatably is actually easier to do than assembler code to machine code due to the fact that it is easier to manipulate the binary values of the machine code than it is to manipulate plain text.
2. Plan the design for the simulator in advance before coding so that we run into fewer problems during the coding process. It felt like for a considerable amount of the time during coding, we did not know exactly how we wanted to store and manipulate values (what sort of arrays, how to output the values etc). So more time spent diagramming and reading up on the source material would have been useful.

Word count = 659 (not including headers)