**AC21009 - Manchester Baby**

**Group 11**

**Matthew Foulis, Matthew McNeil, Douglas Parkinson**

Word count: 653

To begin this assignment, we first met as a group to discuss the task and identify how best to split the project between us. We discussed the layout of the Manchester Baby registers and store locations to make sure we all understood what it was we were working towards. We then decided that having two members work on the assembler and two work on the hardware simulator was a sensible approach. We then began to work in pairs on these sections however one of the group members had to leave which left only one member, Douglas, working on the assembler. Despite this, he felt able to continue to work on this part and was able to complete it by himself.

Compiler

The compiler was designed to read in an assembly file and translate each line of code.

In order to do this we used bitsets, this created a problem as the code was all converted to big-endian form. This meant that for the machine code to be written in the correct order it would need to be converted into little-endian form, this proved to be extremely complicated and required extensive research in order to understand and resolve the problem.

Hardware simulator

For the hardware simulator we decided to store the binary instructions as an object with the binary number being represented by a string. To create the store, we used a vector of Instructions which allowed us to resize it based on the length of the script. It also allowed us to create member functions which we could use to manipulate each Instruction.

One challenge we faced whilst programming the instructions was being able to manipulate the binary numbers. As they were stored as strings we had to find a way of converting them, so we could then retrieve the opcodes and operands and perform operations on them. To solve this, we decided to use bitsets which allowed us to convert the binary strings to integers. This allowed us to perform calculations on the binary strings the same way we would with normal numbers.

Once we had completed the two separate parts we met to try and combine them together. At this point we ran into many problems which had not been present earlier on. When looping through the program we discovered that the instructions were being assigned wrong numbers which sometimes caused the program to crash. As the cause of the problem was not immediately obvious we had to step through each part of the program to try and locate the source. After extensive testing and debugging we discovered one problem to be an unchecked variable which caused the program to continue even after being told to stop. This caused the program to access areas of memory which were not involved with the program.

Another problem we had was when converting the instruction number to a negative number. When using bitset the function was failing to handle negative values and instead producing massive values. This was proved to be another problem caused by the endian form of the code as the program was storing the accumulator as a big endian but attempting to read it as a little endian.

To avoid these problems next time, we would perform as much testing as we could whilst working on our separate sections. By testing similar scenarios to the final one early on, we could get a better understanding of how the program was reacting to different situations and adjust accordingly. This would also help to identify problems which could then be fixed much earlier and in a lot less time. It would also be useful to research new libraries we are using, such as the bitwise, so that we had a clearer understanding of how they work and how we can use them in the program.

The program was compiled on an Ubuntu System using the g++ compiler in the Queen Mother building labs.