AC21009 - Computer Systems 2A

Coursework 3 - The "Manchester Baby"

You have recently been exploring how computer systems are composed of many different pieces of hardware, for example, CPU, Memory, I/O Devices, all connected together by buses. This assignment focuses on how these pieces of hardware could all be virtualised and represented by software applications which simulate the original hardware. The original hardware that we shall simulate is the Small Scale Experimental Machine (SSEM) built at the University of Manchester in 1948 and better known as the "Manchester Baby".

Your task is to work in **teams of four**:

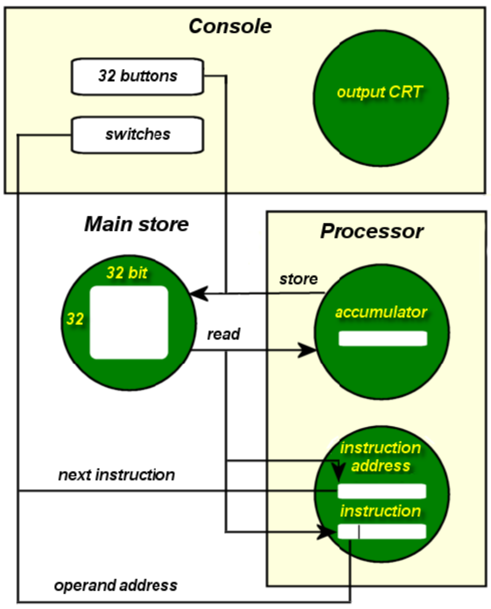
- gain a better understanding of the fetch-execute cycle of a processor by researching the structure of the Manchester Baby

then to use the C or C++ programming language (under Linux) to build:

- a simulation of the Manchester Baby **hardware** able to **demonstrate execution** of a binary machine code program stored in the memory of the computer
- an **assembler** to convert an assembly language program into binary machine code, which can then be loaded and run on your simulator

Background

The broad structure of the Manchester Baby is given in the following diagram:



To achieve this functionality, you will have to consider the kinds of **data structures** that will best provide a software simulation of the **hardware** and the kinds of **methods** required to implement the **functions** that the hardware performs. For example, the Baby has a store (memory) which contains 32 individually addressable lines where each line can store 32 bits of data. You might like to represent each line of the store as an array, and the entire store as an array of such line arrays. You could then write functions to read and write data to and from the store. This is of course only one, quite inefficient, way to approach the simulation and is suggested just to get you started.

More details of the hardware and opcodes used by the Baby can be found in the lecture slides and in two research papers **KAHN2000BABY.PDF** and **SHARP2001GUIDE.PDF**, copies of which can be found on MyDundee. There are also a number of supplementary papers there that will give you more background information on the Manchester Baby, including the lecture notes for lecture 10. A good place to get a general introduction to the Baby is the Wikipedia page for the Manchester Small Scale Experimental Machine:
http://en.wikipedia.org/wiki/Manchester_Small-Scale_Experimental_Machine

Part 1 - Hardware Simulator

Your simulation should take as input a text file containing a binary computer program written in the machine code language of the Baby (essentially 32 lines each of 32 bits of binary data corresponding to the store). When your simulation is running, it should step through each line of the input program, simulating the fetch-decode-execute cycle. For example, after fetching an instruction, it should work out what that instruction is (which operation it corresponds to), what its operand is, then simulate the execution of that operation by moving data around your simulated hardware. The seven operations supported by the Baby are given in the lecture notes.

Your output should be a text based interface (although you can do a GUI if you so wish) which displays, at each step in the fetch-decode-execute cycle, the state of each part of the Baby hardware, for example, displaying register values, memory values, and I/O. Hence the **main** of your program might simply be an endless loop calling functions such as **fetch**, **decode**, **execute** and **printout**).

The representation of the Baby *within your code* is *up to you* (e.g. the store could be held as strings, binaries, integers, vectors) as long as the *output* is a clear representation of the original. Similarly, the internal storage of negative numbers is *up to you* (but see the Number Systems handout on MyDundee for some guidance).

A **sample machine code file** is available on MyDundee, and your simulator should *(at least)* be able to load and execute this program.

There are many extensions that you could implement if you have sufficient time (these should be *in addition to* your standard Baby simulator). For example, you may like to:

- add an option to extend the hardware of the Baby so that there is more memory space to work with
- extend the instruction set to increase the number of instructions that the Baby can perform
- extend the memory addressing modes available by adding e.g. immediate addressing - how will you encode the addressing mode?
- produce a high-resolution GUI displaying the content of registers and memory during program execution

Part 2 - Assembler

Your assembler should take as input a text file containing source code for an assembly language program, and convert this into the equivalent Baby binary machine code, which it should write to a text file - this text file should be in the same format as the input file for your hardware simulator, hence your simulator should be able to run programs which have been assembled with your assembler. The implementation should include support for a **symbol**

table, so that the source code can include **variable names** (which are converted to memory locations) and **labels** (so that program jumps can be coded by referencing the labels, rather than the actual memory address).
Possible extensions to your assembler include:

- detecting a variety of typical errors in the assembly language code
- adding meaningful error messages to display when errors occur
- reporting progress during the assembly process (generally, an assembler need not be verbose if no errors are encountered, but for this simulator, verbosity is useful)
- creating more assembly language programs.

Submission & Assessment: This coursework is due for submission via MyDundee (under AC21009 Coursework Assignments) at 12 noon on Monday 20th November 2017 (Monday of Week 11) and is worth 14% of your total grade for this module. Keep a copy of what you submit in case there are problems with your submission.

You should submit via MyDundee a ZIP (***not*** *RAR, GZ or other*) file containing:

- the **source code** for your Baby Simulator, ready to be compiled under Linux
- the **source code** for your Assembler, ready to be compiled under Linux
- a **report** of *600-800 words* (with word count given) which explains how you approached the assignment, the problems that you faced, and the solutions that you have created. **Your report should also clearly state which Linux C/C++ compiler you used for your project.**

All elements should include your team number and members' names. One submission per team is sufficient.

You should also submit **peer review sheets** as hard copy to the AC21009 coursework box in QM Lab 0. If your whole team **agrees** the distribution of marks (even if this is not an even split) you may submit one sheet for your whole team, otherwise each student should submit their own sheet. Where no sheet is submitted, an even distribution of marks will be assumed. Peer reviews can also be included in your ZIP file submission if you prefer.

Your software should run on the QM Lab PCs, as these will be used when marking, and should *not* require the use of any special code libraries which are not normally available on these machines. During the lab session on **Tuesday 21st November**, your team will demonstrate your simulator and assembler to Iain and Jianguo - this is your opportunity to draw our attention to the best parts of your solution. During the demonstration, you will be expected to:

- **run your simulator** and show it loading and executing the sample program (and any others that you create), and
- use your assembler to **assemble your source code files** and run them on your simulator; you will also be asked to **modify one of your source code files**, re-assemble it and demonstrate that its performance has changed accordingly.

The demonstrations will constitute a part of the marking process for this coursework so attendance is compulsory, and you should **check in advance that your solution functions correctly on the lab computers**. Not every member of your team need be present at the demonstration, but contribution to the delivery of the demonstration should be considered as part of your peer review.

Marking Guidance:
Simulator: A complete solution which loads the machine code sample and outputs a clear text-based representation of the Baby (store, registers and decoding process) for each instruction in the machine code and which demonstrates correct operation of the Baby's instruction cycle will achieve 35 marks (out of 50). Marks will be lost for a poor or incomplete display, incomplete functionality, poor simulation of the fetch-execute process and poor or incomplete coding. Extra marks will be given for outstanding implementation and for including any of the extensions suggested.
Assembler: A complete solution which correctly identifies all possible opcodes and correctly converts these to machine code with commentary to the user, and saves the code to a text file which is compatible with the simulator, plus two machine code samples, will achieve 22 marks (out of 30). Marks will be lost for incomplete or incorrect functionality or incomplete coding. Extra marks will be given for outstanding implementation and for including any of the extensions suggested.
Report/Demonstration: A well-written report and good demonstration which meets all of the given guidelines will achieve 16 marks (out of 20). Extra marks will be given for enhanced document layout and an outstanding demonstration.
A complete solution to all parts will thus achieve 72% (A5). Extended solutions will achieve higher marks.