Introduction
Emulation is useful when designing computer systems. An “emulator” implements the functionality of various parts of a computer in software, and is often used to make a computer run software developed and compiled for another machine. It may also allow all internal signals to be visualised and recorded; while this is a relatively resource-intensive task, it allows much finer-grained debugging and analysis of a computer’s operation than, for example, adding a stop-point to code and reading memory contents.

In this coursework, you are going to emulate the operation of a very simple computer (including the processor and memory). Simple is the key word here: you need to define, implement and test the capabilities of each of your system’s modules, and it should all connect together to work as a computer. It only needs to implement a very limited instruction set.

Specification
This coursework contributes 15% of the mark for ELEC2204, so should take you around 25 hours to complete. Your emulated processor should (as a minimum):

- Implement the basic functionality of a computer in software.
- Follow the Von Neumann architecture, fetching and executing from memory. It is not expected that you will have time to implement caches or virtual memory.
- Follow a straightforward fetch/decode/execute/write-back cycle.
- Implement a limited instruction set and register topology, which you may define yourself (the lectures will cover instructions necessary to implement the test programs).
- Be written in C or C++ (with comments), with the source code handed in with your report.
- Be tested: you should write a suite of basic test programs in machine code/assembly, that allow you to check that your processor is working as intended; you should also be able to capture the signals used by your system using a ‘debug’ mode defined using compiler directives.

The composition of any initial test programs - for use in developing your system - are up to you. However, included in your report should be details of two specific 'showcase' programs:

1. To calculate (and print out) the squares of all the integers between 0 and 99
2. To calculate (and print out) all the prime numbers between 1 and 1000

A time history of memory accesses (see 'debug mode' above) should be included in your report for these last two programs.

You may discuss this coursework with your course mates, but the code and report you hand in should be entirely your own. You should write a report about your design, functionality and results. This should be around 5 pages, with an absolute maximum of 10 pages (minimum 11pt, single-spaced, A4).
Hand-in and Marking
Hand-in is electronic-only via handin.ecs.soton.ac.uk, deadline is 16.00 on Tuesday 8 May 2018. You will need to submit a ZIP of your source code along with a PDF of your report.

The mark breakdown is:

- A robust and effective design/implementation: 40%
- Basic test program suite and results: 20%
- Showcase test programs and results: 20%
- Quality of technical report: 20%