The importance of environments

This lab and the coursework use `iverilog`, which is the simulation tool discussed in the first section 0-tools. However, in order to develop and test complex digital circuits we need more than just a verilog compiler, as there are often multiple moving parts to co-ordinate. So we may need:

- Compilers and assemblers for our target ISA (in this case MIPS)
- Multiple input and output test-cases (small MIPS programs)
- Custom programs in C++ to generate test-cases or generate reference outputs
- Utilities to check whether the output of a processor matches the expected output

You need to make sure that when you modify your solution that it still behaves in the same way when your team-mates run it on their computer. As a team you also want to make sure that your team's overall solution works in the assessment environment.

In 1st year programming people used a combination of `repl.it` and their own local environment. So some people used native OSX, some people used MinGW, others used WSL, others used a VM, some may have used Visual Studio, and so on. From a C++ point of view these are all equivalent, as C++ is effectively its own platform when you combine it with the C++ standard libraries: in principle, the same C++ code should compile and work the same everywhere, whether it is Windows or Linux or OSX or Android. However, in practise there are often incompatibilities that arise (you may have already found some), and these become substantial one you are trying to co-ordinate multiple programs that need to be compiled and interact with the file-system.

To resolve this problem, the standard platform assumed here is Ubuntu (specifically 20.04), as this is one of the easiest environments to run within all the other platforms. It is easy to get running in OSX and Windows, or other Unix-like operating systems through a Virtual Machine.

The environment is all about testing and deployment

Just because the target system is Ubuntu, that doesn't mean you have to install it as your main operating system. If you like OSX or Windows, then stick with it - personally I run Windows 10 despite doing almost all research programming in Ubuntu via WSL.

You don't even need to do all of your learning and development in Ubuntu. It is possible to set up iverilog and the other tools in OSX and MinGW, and to do lot's of productive work there. However, any deployed code (i.e. things you will submit for assessment) should be tested in Ubuntu by at least one team-member, as that is what the assessment environment will be.

How to get hold of Ubuntu

Most people will want to work on their local laptops, so it is useful to have a local version of Ubuntu (or a very Ubuntu-like environment), regardless of your laptop's operating system. Following are some suggested approaches for this, though ultimately it is your own decision - you should go for the setup that seems most productive for you.

Windows

WSL (Recommended)

The recommended route in Windows 10 is to use WSL (Windows Subsystem for Linux). This is a Microsoft-supported method for installing Ubuntu (and other Linux distributions) in Windows 10, and is resource efficient and reliable.

If you are used to use Visual Studio Code, you may also find the VSCode Remote Development Extensions useful, though they are not required.

Ubuntu in a Virtual Machine

You may prefer to install a full virtual machine. This can be fun, is a good learning experience, and makes it easier to run some graphical user-interfaces. Generally it is also reliable, but you may encounter more issues and errors when getting started with this method.

If you take this approach, then you need to install a Hypervisor, which is a piece of software that lets you install a guest Operating System (e.g. Ubuntu) within an existing host Operating System (e.g. whatever your laptop is running)). I would recommend using VirtualBox as the easiest to use Hypervisor. There is another Hypervisor called Hyper-V that is installed as part of Windows, but it is more difficult to configure and work with (Hyper-V is actually what WSL runs on, but WSL adds lots of features to make it simpler).

MinGW or Cygwin (Not recommended)

While you *can* make MinGW very Ubuntu like, it can be more difficult to maintain compatibility with the Ubuntu environment. Things like working with the files in the file-system are not always consistent, and installing and compiling libraries and programs can be more complex. That said, if you really love either environment, then as long as you can get iverilog installed then it can be used for development. You just should make sure that someone else on the team is verifying it against Ubuntu occasionally.

OSX

OSX is much more Linux-like than Windows, which makes it a lot closer to Ubuntu (which is also a Linux). So it already includes a terminal and many of the same tools as Ubuntu for development (e.g. `g++` and `bash`). However, it also has a number of incompatibilities with Linux, as Apple tends to customise the commands, tools and paths, and many of the libraries are out of date. There is also no trivial route to start a Virtual Machine, as seen with WSL in Windows.

So there are two routes you can take, either or both of which you might find useful.

Installing tools via Homebrew

Homebrew is a package manager for OSX which makes it easy to install libraries and tools from the command-line. It is widely used by developers who prefer OSX, as it allows them to more closely replicate the server environments in which most back-end and computational code is deployed. Note that it does *not* install Ubuntu, it will simply install a lot of programs which mirror standard Linux tools more closely.

For the purposes of development you may want to skip the compilation of `iverilog` and simply install it directly via brew, as they have quite a recent version of `iverilog`. For this the process is:

1. Install Homebrew
2. Install the `icarus-verilog` package: `$ brew install icarus-verilog`

From this point on you may want to install other tools, and you can install them using the `brew` command. If you see us talking about `apt install` in the context of Ubuntu, the brew equivalent is *usually* `brew install`.

For examples of the kinds of incompatibilities that are found:

- The scripts in build_utils.sh compiles some programs using `g++`, and assumes that the compiler will select at least the C++11 revision of the language from 2011. However, Apple use the `clang` compiler (a fine compiler), which pretends to be `g++`, and does not choose c++11 by default. So to get it to run in OSX it was necessary to add the `-std=c++11` flag to the compilation statement.
- The version of Icarus verilog that comes with brew by default is version 11 (at the time of writing), which is one release behind the main version of Icarus. As a consequence it is more restricted in the order of parameters, and *requires* that all source files appear after parameters. Later versions allow them to be mixed (in the same that `g++` does).

Installing Ubuntu in a Virtual Machine

The VirtualBox hypervisor works in OSX as well as Windows, and is a good method for running virtual machines in OSX.

We have previously done a walk-through of installing a Virtual Machine in OSX, which shows the steps for installing Ubuntu in OSX. This was for a slightly different version of OSX, but the overall steps should still be the same.