|  |  |  |
| --- | --- | --- |
| Lesson 0 Introduction  This introductory lesson does not contain a practical element, but exists to explain the basic concepts of what is an operating system, what is assembly code and other important basics. If you just want to get straight into practicals, it should be safe to skip this lesson.   |  | | --- | | **Contents**   * [1 Operating Systems](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/introduction.html#os) * [2 Assembly Code](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/introduction.html#ac) |   1 Operating Systems  Throughout these tutorials I will put interesting information in boxes like this one.  Throughout these tutorials I will put information about each command we learn in boxes like this one.  An operating system is just a very complicated program. It has the job of organising other programs on a computer, including sharing the computer's time, memory, hardware and other resources. Some big families of desktop operating systems that you may have heard of include GNU/Linux, Mac OS X and Microsoft Windows. Other devices also need operating systems such as phones, which may use operating systems such as Android, iOS and Windows Phone.[[1]](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/introduction.html" \l "note1" \o "Note 1)  Since the operating system has to interact with the hardware on a computer system, it also has to have specific knowledge of the hardware on a system. To allow operating systems to be used on a variety of computers, the concept of **drivers** was invented. Drivers are small bits of code that can be added and removed from the operating system in order to allow the operating system to talk to a particular piece of hardware. In this course, we do not cover how to create such removable drivers, and instead focus on making specific ones for the Raspberry Pi.  There are all kinds of different designs of operating systems, and this course can only just scratch the surface. In this course we will mainly focus on getting the operating system to interact with a variety of bits of hardware, as this is often the trickiest bit, and the part for which the least documentation and help exists online.  2 Assembly Code  A processor can often perform millions of instructions per second, but they must be simple.  This course will be written almost exclusively in assembly code. Assembly code is code that is *very* close to what the computer understands. How a computer really works is that there is a small device called a processor which is capable of performing simple jobs like adding numbers, and there is a set of one or more microchips called the **RAM** which are capable of storing numbers. When a computer has power, the processor works through a sequence of instructions given to it by the programmer, which cause it to change numbers in the RAM, and interact with connected hardware. Assembly code is a translation into human readable text of those commands.  When programming normally, the programmer writes code in a programming language such as C++, Java, C#, Basic, etc, and then a program called the compiler translates what the programmer wrote into assembly code, which is the further reduced into binary code[[2]](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/introduction.html" \l "note2" \o "Note 2). Binary code is what the computer actually understands, but it is almost impossible for humans to read. Assembly code is much better, but it can be frustrating how few commands are possible. Remember that every command you write in assembly code is something that the processor understands directly, and so the commands are simple by design, as a physical circuit must process each one.  Compiler process  Just like with ordinary programming, there are many different assembly code languages, however unlike ordinary programming, the reason these exist is due to the fact that there exists many different processors, each designed to understand a different language. Thus a program written in assembly code for one machine, will not work on a different one. For most things, this would be a disaster as each program would have to be rewritten for every system it was used on, but for operating systems this isn't so much of a problem, as it would have to be rewritten anyway due to differing hardware. Nevertheless, most operating systems are written in C++ or C, so that they can be converted more easily, and only the sections that absolutely have to be written in assembly are.  You're now ready to move on to the first lesson, [Lesson 1: OK01](http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok01.html) |  |