

Lab 2 – Logic and Machine Code

COMP1200 Perspectives on Computing 2008

original author Eric McCreath
updated Chris Johnson 2005, Brendan McKay 2007, Billy Duckworth 2008

Learning Objectives

This lab has three main learning objectives.

- Firstly, to gain some experience in using the Linux machines. At the end of the lab you should be able to: logon, use a web browser, download files and run java programs.
- Secondly, to gain an understanding of how logic gates can be connected together to construct logic components.
- Thirdly, to gain a basic understanding of how machine code works.

You will need to take notes during this lab. These notes must be shown at the end of the lab to your tutor to demonstrate your completion of the lab.

Downloading and running the programs

The following instructions are for the logic program. A similar approach can be used for the computer simulator. You will need to do the following:

- Start up a Terminal which can be used to compile and run the program from.
- Make a new directory to place the program in. (Use the command “mkdir logic”. You can list your home directory by typing the command “ls”. You should see “logic” as a directory within your home directory.)
- Start a web browser and go to the unit home page (cs.anu.edu.au/student/comp1200). Go to the tutes/labs page and right-click on the “Logic.java” link. Use “Save link as...” to save this file into your newly created “logic” directory.
- From the Terminal again, move into the “logic” directory. (Use the command “cd logic”.) Now when you list this directory you should see the file “Logic.java”.
- To compile this program type the command:
 - javac Logic.java

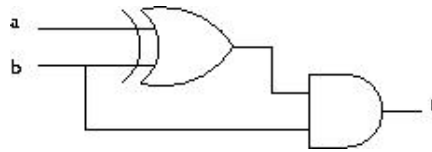
This will create a number of “class” files which are like the machine code for Java. Don't be concerned if you get a warning message from the compiler.

- To run this program type the command:
 - java Logic
- Use the Help menu item in this new program to get basic information on how to use it.

Logic

1. Construct the truth table for the following gates: **and**, **or**, and **not**. Within the simulator connect the inputs of these gates to switches and the outputs to displays. By changing the switches check that the gates are functioning as expected. **Write these truth tables down so you can show your tutor once you have completed this section.**
2. Construct a circuit that will implement the following Boolean expression: $(A \vee B) \wedge \neg C$ What is the truth table for this Boolean expression? Write it down.

3. Construct the truth table for the following logic circuit. Write it down. Is there a simpler design that will produce the same logic component? (Can you design one? Test your answer and write it down.)



4. Within the simulator construct a flip-flop and test its behaviour. (The text book and lecture notes contain designs.)
5. Have this section of the lab marked off by your tutor.
6. (optional extra) A 1-bit adder has three inputs (a_i , b_i , and c_{in}) and two outputs (r_i , and c_{out}). The expressions for the output are as follows:

$$r_i = (a_i \oplus b_i) \oplus c_{in}$$

$$c_{out} = (a_i \wedge b_i) \vee (b_i \wedge c_{in}) \vee (c_{in} \wedge a_i)$$

("A" is the XOR operation.) Using these expression construct a 1-bit adder within the simulator. Combine two 1-bit adders to form a 2 bit adder. Check that it functions as expected.

Machine Code

Use the same approach to download, compile and execute the program called `Simulator.java`.

1. What does the following machine code do? (To try it, load the code at location 05 and set the PC to 05, also place some data in address location A0 and see what it does.) Write down a summary of the effect of the machine code program, and then explain what each of the instructions do.

11
A0
22
FF
93
12
33
A0

2. Translate the instructions below into machine code and run it within the simulator. Explain - what does this program do?

Load R1 <- memory(A0)
 Move R1 -> R2
 Add R3 <- R1, R2
 Store R3 -> memory(A0)

3. Show your result to your tutor and have this lab marked off.

4. (optional extra) Write a program that uses the **Jumpif** instruction (instruction op-code **B**) to create a loop that loops around forever. Try it out in the simulator.