

# CS1Q Workbook

## Student Questions:

## Labs and Tutorials

January 2009.

### Summary

This workbook collects the lab and tutorial material that will be used during the systems component of CS1Q. It is based on exercises that were originally developed by Dr Simon Gay (Dept. Of Computing Science, University of Glasgow). However, all questions about these exercises should be addressed to the tutors in charge of your group.

Prof. Chris Johnson,

Department of Computing Science, University of Glasgow.  
Johnson@dcs.gla.ac.uk. <http://www.dcs.gla.ac.uk/~johnson>

## Worksheet 1 (Semester 2, Week 2 Tutorial)

This worksheet contains exercises on some of the material covered in lecture 2. You should attempt the exercises before your tutorial. During the tutorial, your tutor will work through any exercises which have caused problems. These exercises are not assessed.

### Conversion from Binary to Decimal

Convert the following unsigned binary numbers into decimal.

- (a) 110100
- (b) 01011011
- (c) 10010110
- (d) 111111
- (e) 1111110

### Conversion from Decimal to Binary

Convert the following decimal numbers into binary. How many bits are needed in each case?

- (a) 14
- (b) 127
- (c) 249
- (d) 73
- (e) 257

### Addition in Binary

Work out the following sums in unsigned binary. In each case, check your answer by converting the numbers and the result into decimal.

- (a)  $11011 + 1101$
- (b)  $11010101 + 00111110$
- (c)  $1111 + 1$
- (d)  $111111 + 1$
- (e)  $10101010 + 01010101$

### 2s Complement

Work out the 2s complement binary representation of the following decimal numbers, using 8 bits.

- (a) 97
- (b) -127
- (c) -29
- (d) -76
- (e) -2

Convert the following 2s complement binary numbers to decimal.

- (a) 10000000
- (b) 11111100
- (c) 10101010
- (d) 00001101
- (e) 11010111

**Subtraction**

Work out the following subtractions in 2s complement binary (do this by negating the second number and then adding). Use an 8 bit representation. In each case, check your answer by converting the numbers and the result into decimal.

- (a) 00011011 - 00001101
- (b) 11010101 - 00111110
- (c) 00001111 - 00000001
- (d) 00111111 - 00000001
- (e) 10101010 - 01010101

**Hexadecimal**

Convert the following hexadecimal numbers into binary and decimal.

- (a) 3A
- (b) FF
- (c) A5
- (d) 32
- (e) BD

Convert the following decimal numbers into hexadecimal (an easy way to do this is to convert into binary first).

- (a) 254
- (b) 256
- (c) 64
- (d) 181
- (e) 99

**Text Representation**

The ASCII character set represents each character by a 7 bit binary number, which can also be converted into a 2 digit hexadecimal number. The ASCII codes for the capital letters A–Z are 65–90 (in decimal). A complete table of the ASCII codes (in decimal) can be found on page 64 of Computer Science Illuminated or in many other books. Work out the ASCII representation of your name, in hexadecimal. For example: SIMON is 83,73,77,79,78 in decimal and 53,49,4D,4F,4E in hex.

**Supplementary**

Fractional numbers can be represented in binary in a similar way to decimal: there is a “binary point”, and the columns to the right of the point have value  $\frac{1}{2}$ ,  $\frac{1}{4}$ ,  $\frac{1}{8}$  and so on. For example, 11.012 is  $3 + \frac{1}{4}$ , which is 3.2510.

**Exercise:** convert the following binary fractions to decimal.

- (a) 0.1 (b) 0.11 (c) 0.101 (d) 0.1001 (e) 0.1111

**Exercise:** work out a systematic way of converting decimal fractions to binary.

Some fractions lead to recurring decimal representations: for example,  $\frac{1}{3} = 0.3333\dots$  sometimes written as  $0.3\bullet$ .

The same is true in binary. For example, here is a recurring binary fraction: 0.101010... which could be written as  $0.(10)\bullet$ . To work out the value of this recurring fraction, x say, notice that  $4x = 10.(10)\bullet$  (because multiplying by 4 in binary means shifting the binary point two places to the right), and so

$$\begin{aligned} 4x &= x + 2 \\ 3x &= 2 \end{aligned}$$

and therefore  $x = \frac{2}{3}$ . As it happens,  $\frac{2}{3}$  also has a recurring decimal representation, but this is not always the case.

**Exercise:** Can you find a non-recurring decimal which is recurring in binary?

**Exercise:** Can you see why every non-recurring binary fraction is also non-recurring in decimal?

## Worksheet 2 (Semester 2, Week 3 Lab)

This lab is based on the material on assembly language from lectures 3 and 4. You will use a software emulator for the IT Machine to experiment with some small assembly language programs. The work is not assessed, but you should hand in your program from the section Writing a Program at the end of the lab. Your tutor's comments will help you to understand this section of the course. Appendix 1 of this document contains a guide to the IT Machine.

### Loading and Executing a Program

1. Go into AMS and set up the CS1Q exercise S2Week3.
2. The IT Machine can be found under the start menu, select Programs and IT\_MachineGUI. You should see a display which looks similar to the illustrations of the IT Machine in lectures: there are boxes showing the registers, the memory, the current instruction, etc. All of these boxes will be blank.
3. Use the Programmers' File Editor (look under start/Programs again) to open the file Program1 (it's in the folder S2Week3/programs). Try to arrange your desktop so that you can see the assembly language program and part of the IT Machine display.
4. Select Open from the File menu in the IT Machine, and open the file Program1. The display will not change as a result of opening the file.
5. Select Assemble from the Run menu. This converts the instructions from assembly language to machine language. Select Assembler Listing from the View menu. You will see a window containing the assembly language program and its machine language equivalent, in hexadecimal. In general, any errors in the assembly language program will be shown in the assembler listing. In this case, we know that the program is correct so there are no errors.
6. Select Load from the Run menu. This loads the assembled program into the memory of the IT Machine. The Memory display has two scrollable views of the memory. Each view lists addresses on the left, and contents on the right.
7. Under the menu View select Emulate then One Instruction. Select it to execute the first instruction in the program. The instruction being executed is highlighted in white in the memory display. The instruction is also shown in the Current Instruction display in two forms: machine language and assembly language. The Assembly Statement display shows the corresponding line from the original assembly language program.

The first instruction in this program loads a value into register R6. The new value of R6 is highlighted in red in the register display area.

8. Execute the next instruction. Again you will see the instruction highlighted in the memory display. This instruction stores a value into the memory location labelled x. In the right hand side of the memory display, scroll down so that you can see the new contents of location x (which is actually location 0014) highlighted in red.
9. Step through the rest of the program. Notice that the ADD and MUL instructions show which registers are being used, and cause the ALU display to indicate the operation being carried out.

[Continued on next page]

## Modifying a Program

1. Open the file Program2 in your editor. This is the program from Lecture 4 which calculates the sum of the integers up to some value n.
2. The program is incomplete because the value of n has not been specified. When designing the program we decided to use register R1 for n. Add an instruction at the beginning of the program which sets R1 to a particular value (try something fairly small). Save the file.
3. Load the file into the IT Machine and assemble it. This time it is important to check for errors by viewing the assembler listing. After correcting any errors (and saving the file), you must open the file again in the IT Machine, and assemble it again.
4. When the program assembles without errors, load it and run it. You can either step through the program one instruction at a time, or click on the double arrow button to run it all the way to the end.
5. When the program terminates, register R2 should contain the sum of the integers up to n. Is the result what you expected? (Remember that the IT Machine uses hexadecimal.)
6. Open the file Program3 in your editor. This is another copy of the same program. Modify the program so that it corresponds to the following pseudo code:

```
s := 1;
while n > 0 loop
    s := s * n;
    n := n - 1;
end loop;
```

Again, you will also need to add an instruction to set n to a particular value.

7. Open, assemble and load this file into the IT Machine as before, remembering to check for errors after assembling. Run the program. What does it calculate?

## Writing a Program

Using the example programs from lectures as a guide (especially the program for finding the largest element of an array, from Lecture 4), write a program which calculates the sum of the elements of an array. The file Program4 is provided for you to put this program in. Assume that the end of the array is indicated by the value -1. You should go through the following steps, on paper, before entering your program into a file and testing it.

1. Write the algorithm in pseudocode.
2. Decide whether to use registers or memory locations for the variables.  
(The array will be in memory, of course.)
3. Translate the algorithm from Ada into assembly language. The overall structure of the program should be similar to the largest element program.
4. Remember to declare memory space for the array and any variables which you are storing in memory. Remember to put some specific values into the array, and end the array with a value of -1.
5. Include comments in your program, so that you can keep track of what each instruction means in relation to the Ada code.

6. Remember to end your program with the instruction:

```
CALL  exit[R0].
```

**Hand in, on paper, your design work and program for this part of the exercise, at the end of the lab.**

### **Supplementary**

1. Try out some of the optimizations mentioned in Lecture 4.
2. Try writing some other programs.

## Worksheet 3 (Semester 2, Week 4 Tutorial)

This worksheet is divided into two parts. The first part contains exercises on the material covered in lectures 5 and 6. You should attempt the exercises before your tutorial. During the tutorial, your tutor will work through any exercises which have caused problems.

The second part is preparation for next week's lab exercise. You will have time to work on this part during the tutorial.

### Before the Tutorial

#### Logic Circuits

In each of the following cases, draw a circuit which calculates the result  $r$  from inputs  $x$ ,  $y$  and  $z$ .

- $r = x \text{ AND } (y \text{ OR } z)$
- $r = (x \text{ AND } (\text{NOT } y)) \text{ AND } (\text{NOT } z)$
- $r = (x \text{ AND } y) \text{ OR } (x \text{ AND } z)$

#### Truth Tables

Work out a truth table for each of the functions in the previous section.

#### Definitions in terms of NOR

Use truth tables to check the following equations.

- $\text{NOT } x = x \text{ NOR } x$
- $X \text{ OR } y = \text{NOT}(x \text{ NOR } y)$
- $X \text{ AND } y = (\text{NOT } x) \text{ NOR } (\text{NOT } y)$

#### Algebraic Notation

Rewrite the logical expressions in the **Logic Circuits** section, using algebraic notation.

#### Multi-input Gates

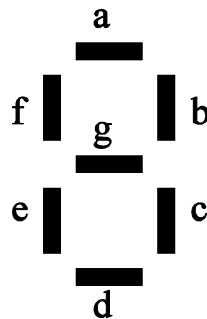
How many 2-input AND gates are needed to synthesize a 4-input AND gate? A 6-input AND-gate? An 8-input AND gate? Can you say anything in general about the number of 2-input gates needed to synthesize an  $n$ -input gate?

#### Algebraic Laws

Use truth tables to verify some of the algebraic laws (Lecture 6, Slides 7, 8 and 9).

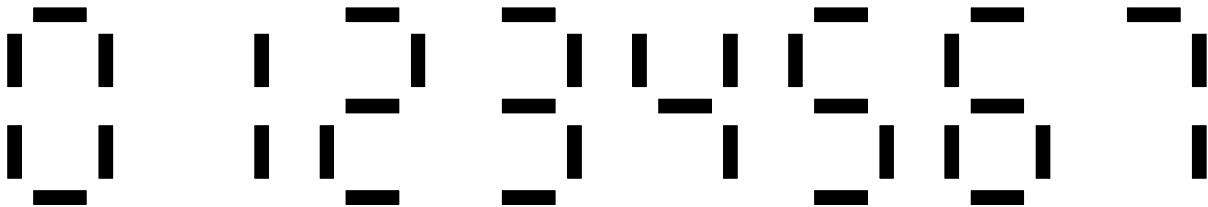
### During the Tutorial

Digital watches and other electronic devices often use a 7 segment display: there are 7 segments (bars) which can be either on or off. The display has 7 inputs, a..g, which are used to control the segments. If an input is 1, the corresponding segment is on (black). If an input is 0, the corresponding segment is off (white).



**Figure 3-1.** Seven Segment Display

Figure 2 presents the combinations of segments which are usually used to display the digits from 0 to 7.



**Figure 3-2.** Numbers from 0 to 7 (for 3 binary inputs)

### Designing a 7 segment display driver

The aim of the exercise, and next week's lab, is to design a circuit which converts a 3 bit binary input, representing a number between 0 and 7, into the correct values for the inputs a..g of the display. We'll call the three inputs x, y, z (so, for example, if we want to display the number 6, which is 110 in binary, the inputs will be x=1, y=1, z=0).

- Work out a truth table showing a..g as functions of x, y and z.
- After Lecture 7 this week, you will be able to use the truth table to work out formulae for a..g. If your tutorial is after the Tuesday lecture then you can do this now.
- After Lecture 8 this week, you will be able to use Karnaugh maps to simplify these formulae. If your tutorial is after the Thursday lecture then you can do this now.

## Worksheet 4 (Semester 2, Week 5 Lab)

In the next few labs, we will be using a software package called LogicWorks to simulate digital circuits. This worksheet contains three exercises. The first two exercises are for you to become familiar with LogicWorks. The third exercise is more substantial. The exercises are unassessed, but you should hand in your design work for **Exercise 3** at the end of the lab. Your tutor's comments will help you to understand circuit design, which is an important part of the course.

### Exercise 1: Getting Started with LogicWorks

In AMS, set up the CS1Q exercise called S2Week5. In your CS1Q folder there should now be a folder called S2Week5. Within this folder is a file called Exercise1 double click on it to start LogicWorks, if this does not work then launch the application from the start/Programs/LogicWorks menu item.

The LogicWorks window is divided into four main areas: at the top, the menus and toolbar; at the bottom, the timing area, which we will not be using; on the right, the component library, and in the center, the design area. The following steps will take you through the construction of a simple circuit which allows an AND gate to be tested.

1. In the component library, scroll down until you see a component called AND-2. Select it by double-clicking. Click somewhere in the design area to place the component. You will see that the AND gate is still attached to the pointer, so that another one can be placed. You only need one for the moment, so click on the Pointer button in the toolbar to return to a normal pointer.
2. Select a Binary Switch from the component library, and click in the design area to place it somewhere to the left of the AND gate. Click again to place a second switch below the first one.
3. Click on the Draw signal button in the toolbar, which has a light-weight + symbol on it. The pointer will change to a + symbol. Click on the output of one of the switches (the horizontal line on the right), then on one of the inputs of the AND gate. You should see a red line, representing a wire, joining the switch to the input. If this does not work, try clicking again, closer to the output or input.
4. You can correct any mistakes by changing the pointer to an eraser (click on the lightning bolt symbol to the right of the pointer button, in the toolbar) and clicking on any part of the circuit that you want to delete.
5. Connect the output of the second switch to the second input of the AND gate.
6. Select a Binary Probe from the component library and place it to the right of the AND gate. Connect the output of the AND gate to the input of the binary probe.
7. Click on the Run simulator button, at the right of the second row of the toolbar. Notice that the binary probe changes from Z to 0. By clicking on the switches (make sure that you have a normal pointer) you can change the values of the inputs of the AND gate, and see the corresponding output on the binary probe.
8. As with other applications, there is a Save option in the File menu, which you should use to save your work.

[Continued overleaf]

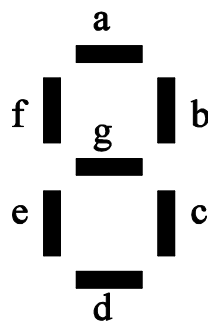
### Exercise 2: The Majority Voting Circuit

1. Close Exercise1 and open Exercise2.
2. Build the majority voting circuit from the lecture notes (Lecture 5 Slide 7). Connect binary switches to the inputs and connect a binary probe to the output.
3. Run the simulator and test the circuit, by setting the input switches and observing the output. Construct a truth table by systematically testing all 8 combinations of input values.
4. Check that this truth table is the same as the one in the lecture notes.

### Exercise 3: Driving a 7 Segment Display

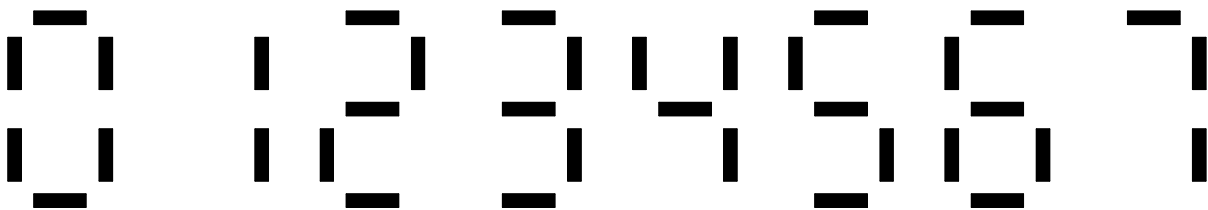
Close Exercise2 and open Exercise3.

In LogicWorks, select the component 7-Seg Disp - Black from the component library and place it in the design area. This is the single digit display from last week's tutorial. It represents a single digit display of the type used in, for example, digital watches. The 7 segments (bars) can each be either lit or unlit. For each segment there is a binary input (labelled a...g) which should be set to 1 in order to light that segment. There is also an input for the decimal point (labelled *dot*), which we won't use; connect it to a GND symbol so that it is fixed at 0. The segments are labelled as follows.



**Figure 4-1.** Seven Segment Display

The aim of this exercise is to build a circuit which converts a 3 bit binary input, representing a number from 0 to 7, into the correct values for the inputs a..g of the display. The usual way of representing the digits is as follows.



**Figure 4-2.** Numbers from 0 to 7 (for 3 binary inputs)

[Continued overleaf]

1. Last week you should have worked out a truth table showing a..g as functions of x, y and z where xyz is the binary representation of the digit to be displayed. If you did not do this last week, do it now.
2. For each of a..g, use a Karnaugh map to work out a minimised formula.
3. Draw a circuit which calculates a...g from xyz. There might be opportunities to reuse subformulae; it might also be possible to achieve further simplifications by factorising parts of the formulae. Draw the circuit on paper before you start building it in LogicWorks. This will help you to see how to lay it out neatly and help you not to make mistakes.
4. Build the circuit, connect binary switches to the inputs, and test it. It is best to organise the circuit neatly, so you can keep track of which components are doing what. It is also a good idea to build and test the parts for a... g one at a time.

**Hand in your design work from this exercise (Questions 1, 2, 3) at the end of the lab.**

### **Supplementary**

Repeat this exercise with a 4 bit input, so that the appropriate hexadecimal digit is displayed.

In a real electronic device, the 7 segment display driver might be better implemented by using the input value to address a small area of ROM (read only memory) containing a 7 bit word for each digit; this word would be fed into the display.

## Worksheet 5 (Semester 2, Week 6 Tutorial)

### Introduction

This worksheet consists of more exercises in the use of Karnaugh maps to design digital circuits. Some of the exercises involve functions of 4 inputs. This means using a Karnaugh map with a 4x4 grid, as discussed and illustrated in lectures, but the idea is exactly the same as in the 3 input case. In exams, you will only be expected to work with Karnaugh maps for functions of 3 inputs.

This worksheet is not assessed, but you should hand in your work at the end of the tutorial so that your tutor can comment on what you have done. A solution will be on the web site next week.

### Background: Binary Coded Decimal

In the past, a system called binary coded decimal (BCD) has sometimes been used to represent decimal numbers in binary. The idea is to consider a decimal number as a sequence of digits, and represent each digit by a 4 bit binary number.

For example, the decimal number 23 (recall that we write  $23_{10}$  to emphasise that we have a base 10 (decimal) number) would be represented by 00100011 because 2 is 0010 in binary and 3 is 0011 in binary.

In BCD, each group of 4 bits is in the range 0000...1001. The values 1010...1111 are not used.

- What is the BCD representation of  $74_{10}$ ?
- Which decimal number is represented in BCD by 10000101?

Although BCD makes it unnecessary to convert between decimal and binary, a serious drawback is that arithmetic becomes more complicated.

- What are the BCD representations of  $19_{10}$  and  $1_{10}$ ? What happens if you add the representations together, using ordinary binary addition? What has gone wrong?

The rest of the exercises on this worksheet are inspired by the problems of working with BCD, but in order to keep the exercises manageable in size, we are going to use binary coded ternary instead.

### Ternary: Base 3 Numbers

Just as binary is base 2, decimal is base 10 and hexadecimal is base 16, ternary is the base 3 number system. Each digit is either 0, 1 or 2, and the column values are the powers of 3: 1, 3, 9, 27 and so on.

Examples:

- $21_3 = 7_{10}$  because  $2 \times 3 + 1 = 7$ .
- $10_3 = 3_{10}$  because  $1 \times 3 + 0 = 3$ .
- $22_3 = 8_{10}$  because  $2 \times 3 + 2 = 8$  (this is the biggest 2-digit ternary number).

The counting sequence for 2-digit ternary numbers is 00, 01, 02, 10, 11, 12, 20, 21, 22.

- Check that this sequence of 2-digit ternary numbers corresponds to the numbers 0... 8 in decimal.

**BCT: Binary Coded Ternary**

Imagine that life has been discovered on Mars, and that Martians use ternary numbers (perhaps because they have three legs, three tentacles, and three fingers on each tentacle). Martian computers use binary, of course, but they represent numbers by means of a binary coded ternary (BCT) scheme. In BCT, each ternary digit is represented by a 2-bit binary number. For example, the ternary number  $12_3$  is represented by 0110 because 1 is 01 in binary and 2 is 10 in binary. The 2-bit binary value 11 is not used in BCT.

- What is the BCT representation of  $20_3$ ?
- Which ternary number is represented in BCT by 1001? What is its decimal equivalent?

**Converting Binary to BCT**

A certain Martian electronic device calculates a 3-bit binary value, and this value is going to be displayed on a 2-digit BCT display device. This display device has 4 inputs, called a, b, c, d. The 2-bit value ab represents the first BCT digit and the 2-bit value cd represents the second BCT digit. For example, if the inputs to the display are 0101 (a=0, b=1, c=0, d=1) then it will display 11 (and the Martian user interprets this as the ternary number  $11_3$ , which is equivalent to  $4_{10}$ ).

We therefore need a circuit which can be given a 3-bit binary number xyz and calculates the values abcd to be input to the BCT display.

- Design this circuit. You should go through the following sequence of step:
  - (a) For each 3-bit binary value, work out the corresponding 2-digit ternary value. Then work out the BCT representation of each of these ternary values. You now have, for each 3-bit binary value xyz, a 4-bit binary value abcd.
  - (b) Make a truth table showing x, y, z as inputs and a, b, c, d as outputs.
  - (c) Considering a, b, c, d as separate functions of x, y, z, work out a Karnaugh map for each one.
  - (d) Use the Karnaugh maps to work out a formula for each of a, b, c, d.
  - (e) Draw a circuit diagram which calculates a, b, c, d from x, y, z.

## Worksheet 6 (Semester 2, Week 7 Lab)

The exercises on this worksheet use LogicWorks. If necessary, refer to Worksheet 4 (from the previous lab session) for basic instructions on the use of LogicWorks. This worksheet is unassessed, but you should hand in your design work, on paper, for **Exercise 1** (see below) at the end of the lab.

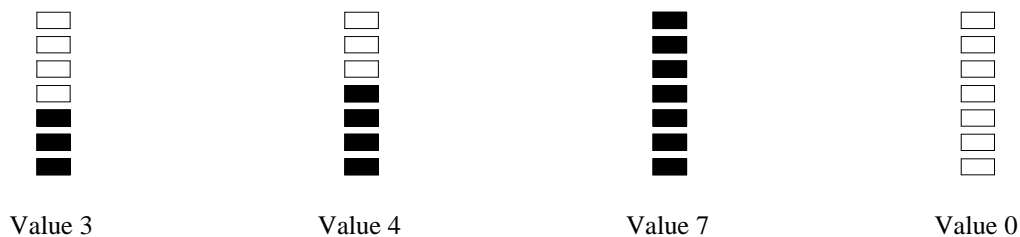
### LogicWorks Tips

Sometimes it is useful to label the inputs and outputs. Text labels can be added to the diagram by clicking on the button labelled A in the toolbar. Then click on the diagram at the point where you want to add text, and type.

If you want to rotate components (for example, so that the output of an AND gate points downwards instead of to the right), select Orientation from the Schematic menu to change the orientation of the next component which you place; or, press the arrow keys before placing the component. This is very helpful if you want to produce a tidy layout (recommended). If you want to move components, just click and drag them with the mouse, first making sure that you have an arrow pointer.

### Bargraph Driver

Some electronic devices present numerical information in the form of a bargraph. Typically there is a column of lights, and at any time the number of consecutive lights which are on, starting from the bottom, represents a numerical value. Music systems often use this scheme to show the overall volume setting, or the strength of the signal in a particular frequency range. For example, if the bargraph can represent a value between 0 and 7 (in this diagram, black represents a light being on, and white represents a light being off):



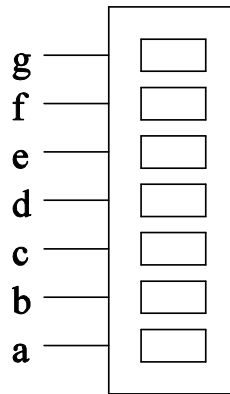
**Figure 6-1.** Example Values for Bargraph Displays

A bargraph driver is a circuit which inputs a binary number (for this exercise it will be a 3 or 4 bit number) and outputs a,b,c... which are the values of the lights, from bottom to top. These values are 1 for on (black in the diagram) and 0 for off (white in the diagram).

1. Work out the truth table for a bargraph driver which inputs a 3 bit number xyz (i.e. x, y, z are the digits of a 3 bit binary number, from left to right) and outputs a,b,c,d,e,f,g (so a is the bottom light and g is the top light).
2. Work out a Karnaugh map for each output a,..., g.
3. From the Karnaugh maps, work out formulae for a,...,g. Try to work out the simplest formulae (i.e. use the largest possible rectangles).

[Continued Overleaf]

- Draw a diagram of a circuit which will calculate a,...,g from x, y, z. When you build this circuit in LogicWorks, the outputs a,...,g will be connected to a component which looks like this:

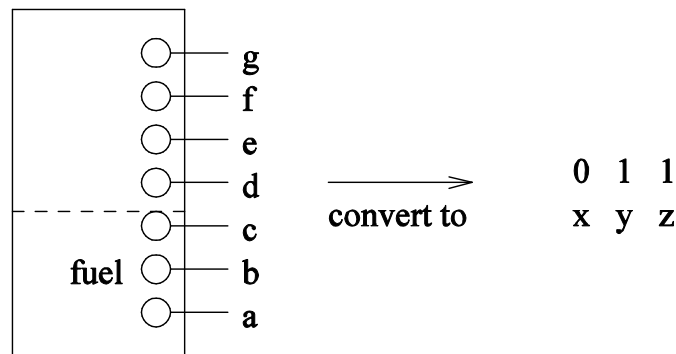


**Figure 7-2:** Output component for Bargraph Display

- Go into AMS and set up the CS1Q exercise S2Week7. Start LogicWorks on the file Exercise 9. The file already contains a bargraph component, with inputs a...g. Build the circuit from Question 4, connecting the a,...,g outputs to the bargraph. Connect the x,y,z inputs to binary switches. Test your circuit by setting each 3 bit value in turn on the x,y,z switches and checking that the bargraph displays the value correctly.

**Inverse of Bargraph Driver**

Now consider the inverse problem: designing a circuit which inputs a,...,g and outputs x,y,z. For example, imagine that a,...,g are the outputs of a series of sensors at different heights in a fuel tank, and have value 1 if the sensor is submerged, 0 otherwise.



**Figure 7-2.** Overview of Inverse Bargraph (Demultiplexer)

When the fuel is at a certain depth, a number of sensors will be submerged, from the bottom upwards. We want to convert the sensor values into a 3 bit binary value xyz which gives a measure of the fuel level.

- One way of designing the circuit would be to think in terms of a 7 input truth table, and express x, y, z in sum of products form using minterms built from a...g. This would lead to quite complicated formulae, which would be difficult to simplify because the Karnaugh maps would be unmanagable. There is a simpler way which uses adders. Work out how to do this and draw a circuit diagram.

Now suppose that we have a bargraph component which uses 10 lights (inputs a...j) to represent a value between 0 and 10. This might be a more realistic situation, given that we use decimal in everyday life. The

driver now inputs a 4 bit binary number  $wxyz$  (the binary representations of numbers larger than ten will not be used) and outputs  $a, \dots, j$ .

7. Work out the truth table for the new bargraph driver. You don't need to include the rows for inputs 1011..., 1111.
8. Use your answer to Question 6 to work out definitions of  $a, \dots, j$  without using Karnaugh maps. Explain why each definition is correct, similarly to Question 6.

# Worksheet 7 (Semester 2, Week 8 Tutorial)

## Introduction to the Assessed Exercise

The work that is to be included in this year’s Assessed Exercise is covered in Worksheets 7 and 8.

The exercise involves the design and implementation (in LogicWorks) of digital circuits. There are two stages:

1. Design work, to be done during this tutorial.
2. Building and testing the circuits, to be done during next week's lab.

## Submission requirements from this tutorial

You should submit the answers to questions 1 to 8 from this tutorial on paper, at the beginning of your tutorial in Week 10

## Introduction

The assessed exercise will take place during this week’s tutorial and next week’s lab. The exercise involves the design and implementation (in LogicWorks) of digital circuits. There are four stages:

1. Design work, to be done during this week’s tutorial.
2. Building and testing the circuits, to be done during next week’s lab.
3. Submitting the LogicWorks circuits; the deadline is two days after your lab session next week.
4. Submitting a written report; the deadline is the next tutorial.

## Braille

The Braille system represents letters by means of patterns of raised bumps, enabling blind and partially sighted people to read (by touch) and write (by using a braille writing machine). Each letter consists of a 2 by 3 grid of positions, each of which may contain a raised bump. In print we will indicate a raised bump by ● and the absence of a bump by ‘.’ The basic system represents the alphabet as follows.

●. . .. ● .. ..	●. ● .. .. .. ..	●● ● .. ● .. ..	●● ● .. ● .. ..	●. ● .. ● .. ..	●● ● .. ● .. ..	●● ● .. ● .. ..	●. ● .. ● .. ..	●. ● .. ● .. ..	●. ● .. ● .. ..	●. ● .. ● .. ..	●. ● .. ● .. ..	●● ● .. ● .. ..	●● ● .. ● .. ..	●. ● .. ● .. ..
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

●● ● ●. ● ●. ●	●● ● ●. ● ●. ●	●. ● ●● ● ●. ●	●. ● ●● ● ●. ●	●. ● ●● ● ●. ●	●. ● .. ● ●● ●	●. ● .. ● ●● ●	●. ● ●● ● .. ●	●● ● .. ● ●● ●	●● ● .. ● ●● ●	●. ● .. ● ●● ●
P	Q	R	S	T	U	V	W	X	Y	Z

Figure 7-1: Braille encoding of Alphabet

If you look carefully you might spot a pattern, spoilt by the inclusion of W (Louis Braille, the inventor of the braille system, was French, and W does not occur in the French language). A numeric digit is represented by the following special prefix:



followed by a letter in the range A (representing 1) to J (representing 0):

•.	•.	••	••	•.	••	••	•.	•.	•.
..	•.	..	•.	•.	•.	••	••	•.	••
..	..	..	..	..	..	..	..	..	..
1	2	3	4	5	6	7	8	9	0

Figure 7-2: Braille encoding of Digits

Other bump patterns are used for other prefixes (e.g. upper case) and a system of abbreviations of common words and parts of words. For more information, if you are interested, visit [www.hotbraille.com](http://www.hotbraille.com).

A braille character can be viewed as a 6 bit binary word, one bit for each position in the grid, in which 1 represents a bump and 0 represents no bump. For the rest of this exercise we will think of a braille character as a 6 bit word abcdef where the bits a..f correspond to the grid like this:

a b

c d

e f

For example, the letter M, as we have seen, is denoted by the following Braille symbol:

••  
..  
•.

This corresponds to the binary code 110010 (a=1, b=1, c=0, d=0, e=1, f=0). We will be using a braille display device, similar to the 7 segment display from Worksheet 4, which has 6 inputs (a..f) and produces a display similar to our printed form of braille: a grid in which each position is black for a bump or white for no bump.

**Design work for the tutorial: single digit braille display**

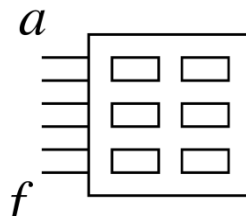
Design a circuit which calculates the appropriate values a, b, c, d (to be used as inputs to the braille display device) for each digit in the range 0 . . . 7. You only need to calculate a . . . d because e and f are always 0. Your circuit should have 3 inputs x, y, z, so that xyz is a 3 bit binary number representing a number in the range 0 . . . 7, and 4 outputs a, b, c, d. The outputs should specify the appropriate braille letter in the range J . . G (ignore the number prefix symbol).

For example, if the input is 101 (i.e., x=1, y=0, z=1), representing the number 5, then the output should be 1001 (a=1, b=0, c=0, d=1), which when combined with e=0, f=0 yields 100100, representing the braille letter E which stands for 5:

•.  
•.  
..

You should go through the following steps, as usual:

1. Work out a truth table with inputs x, y, z and outputs a, b, c, d.
2. Work out a Karnaugh map for each output a, b, c, d.
3. From the Karnaugh maps work out the simplest possible formulae for a, b, c, d.
4. Draw a diagram of a circuit which will calculate a, b, c, d from x, y, z. When you build this circuit in LogicWorks, the outputs a, b, c, d will be connected to a component which looks like the following:



**Figure 7-3:** LogicWorks Output Component for Braille

Instructions for finding this component are mentioned below in Exercise 2 but you should lay out your initial circuit sketch with this in mind.

**This design work will be used in the lab next week, and will form part of your written submission.**

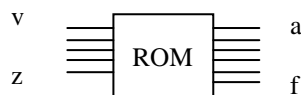
**Design work for the tutorial: single letter braille display**

Suppose that we want to design a circuit which displays braille letters. To represent the 26 letters A . . . Z we need to use 5 bit words. A natural way to do this, which is compatible with the standard ASCII character set, is to use the words 00001 . . . 11010 (decimal 1 . . . 26). (In the ASCII character set, which uses 7 bit words, these representations are prefixed by 10 so that the letters A . . . Z have ASCII codes 65 . . . 90.)

If we followed our normal design process, we would work out a truth table with 5 input columns (v,w, x, y, z say) and 6 output columns (a . . . f as before). This would lead to complex definitions of a . . . f with no prospect of simplification because we can't easily use Karnaugh maps for more than 4 inputs.

Instead we will use a different approach, which is supported by LogicWorks and is more likely to be used in practice for complicated conversions between representations. We will program a look-up table into a read-only memory (ROM). The idea is to build a memory device which stores the entire truth table. This memory will store 32 words, each of 6 bits.

It is read-only memory, so its contents can't be changed once it has been built: all we can do is look up the contents of each location. The memory has 5 inputs, corresponding to the 5 input columns of the truth table, and 6 outputs, corresponding to the 6 output columns. It looks like this:



A particular row of the truth table is specified by particular values of v . . . z. When used as inputs to the ROM, these values specify a particular location within the memory. The ROM will be programmed so that the contents of this location are the output columns a . . . f for that row of the truth table.

In order to program this ROM in LogicWorks, we need to convert the output columns of the truth table into 2-digit hex numbers.

5. Work out a truth table with the following columns. There will be 32 rows, so use a big enough piece of paper.
  - Input columns v,w, x, y, z which run from 00000 to 11111.
  - A column showing which letter corresponds to each input combination. Remember that A is 00001. The inputs 00000, and 11011 onwards, do not correspond to letters; in these cases leave this column blank.
  - Output columns a, b, c, d, e, f which represent the pattern of bumps in the Braille grid for each letter. In the cases which do not correspond to letters, put 0 in all of the output columns.
  - A column which shows the 6 bit binary number abcdef as a 2 digit hex number. Hint: convert ab to the first hex digit and convert cdef to the second hex digit.

This design work will be used in the lab next week, and will form part of your written submission.

## Worksheet 8 (Semester 2, Week 9 Lab)

### Submission requirements from this lab

The design work from the previous tutorial and the implementation work from this week's lab together form the core of the open assessed exercise for the CS1Q Systems component.

You should submit the LogicWorks files "Exercise 2" and "Exercise 3", and the file "BraillePROMdata", through AMS. The deadline is 10pm on the second working day after your CS1Q lab session in Week 9. For example, if your lab is on Tuesday, the deadline is 10pm on Thursday; if your lab is on Friday, the deadline is 10pm on the following Tuesday.

### Marking

This assessed exercise is worth 30% of the total contribution of assessed coursework to CS1Q. As assessed coursework contributes 20% of the module mark, this exercise is worth 6% of the final module mark. The following marking scheme will be used to produce a mark out of 60 for this exercise.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Total
Marks	4	4	4	3	4	2	2	7	30

### Assessed Work in the Lab

These questions are intended to be answered during the lab session in week 9, but you can start earlier if you want to. First of all, go into AMS and set up the CS1Q exercise "S2Week9"

6. Start LogicWorks on the file "Exercise 2". The file already contains a braille display component, with inputs a . . . f. Build the circuit from Question 4, connecting the a, b, c, d outputs to the braille display. Connect the e and f inputs to 0 (use GND or switches set to 0). Connect the x, y, z inputs to binary switches. Test your circuit by setting each 3 bit value in turn on the x, y, z switches and checking that the correct braille character is displayed.
7. Start LogicWorks on the file "Exercise 3". This file also contains a braille display component. The following steps will enable you to program a ROM with the data from Question 5.
  - a) In the "S2Week9" folder, open the file "BraillePROMdata" by double-clicking. Enter the 32 2-digit hex numbers from Question 5 (in last tutorial), separated by spaces or newlines (whichever you prefer). Save the file and close it.
  - b) Back in LogicWorks (on the file "Exercise 3"), select "PROM/RAM/PLA Wizard" from the "Simulation" menu. Select "Programmable Read Only Memory" in the resulting dialogue box, and click on "Next".
  - c) In the next dialogue box specify 5 address lines, 6 bits per word, select "Read data from a raw hex file", and click on "Next".
  - d) In the next dialogue box, click on "Select Raw Hex File", then browse to find the file "BraillePROMdata". You will have to select "All Files" in the "Files of type" box. When you find the file, click on "Open". Then click on "Next".
  - e) In the next dialogue box, enter the name "Braille" for your ROM, then click on "New Lib" to create a new component library in which to put your ROM.
  - f) In the next dialogue box, enter "S2Week9" as the filename for your library, and click "Save". The previous dialogue box will return; click "Finish".

You should now see a component called "Braille" in the component library. Select this component and place it in the design area. Connect the outputs (they are called Out5 . . . Out0 but they correspond to a . . . f) to the inputs of the braille display. Connect the inputs (called In4 . . . In0 and corresponding to v . . . z) to binary switches. Test the circuit by checking, for each combination of the switches, that the correct Braille pattern is displayed.

(Continued on next page).

**Additional question to be answered in the report**

8. Go back to the 7 segment display exercise (Worksheet 3 (Tutorial Week 4) and Worksheet 4 (Lab Week 5)) and find the truth table for the 7 segment display driver. Write an explanation of how to use a ROM to implement the display driver. You should include an explanation of the number of inputs and outputs which the ROM must have, and specify the contents of each memory location.

## Worksheet 9 (Semester 2, Week 10 Tutorial)

### Sample Exam Questions : Introduction

The exam will contain one question on HCI, one question on mathematics and information management, one question on the systems material, and one question that links systems issues to those in HCI and information management.

On this worksheet, Question 1 is an example of the first systems question, and Question 2 is an example of the second systems question. Each question is intended to be answered in 30 minutes. During this week's tutorial you can attempt Question 1, and your tutor will either go through or hand out a model solution. You can attempt Question 2 in your own time, and a model solution will be available later.

### Question 1

- (a)
- (i) Explain the difference between a high-level language and a low-level language, and name one language of each type. [2]
  - (ii) Explain the function of a D flipflop [1]
  - (iii) Give an example of the use of flipflops in the design of a CPU. [1]
  - (iv) Explain what is meant by a sequential circuit, and say what the alternative is. [2]
  - (v) Give an example of a sequential circuit. [1]
  - (vi) Name three of the main components of a CPU and briefly describe their function. [3]
- (b) You are required to design a circuit which, given an input  $xyz$  representing a 3 bit binary number  $n$ , produces an output  $abc$  representing  $n+1$ . For example, if the input is 011 ( $x=0, y=1, z=1$ ), representing  $n=3$ , then the output is 100, representing 4. If the input is 111 then the output is 000.
- (i) Draw a truth table which shows  $a, b, c$  as functions of  $x, y, z$ . [3]
  - (ii) Draw a Karnaugh map for each of  $a, b, c$ . [3]
  - (iii) Use the Karnaugh maps to work out formulae for  $a, b$  and  $c$  in terms of  $x, y$  and  $z$ . [3]
  - (iv) Draw a diagram of the circuit which calculates  $a, b$  and  $c$  from  $x, y$  and  $z$ . [3]
  - (v) Compare the number of components in your circuit with the number of components in a standard 3 bit adder. [3]

[Continued overleaf]

**Question 2**

- (a) Explain the difference between circuit switching and packet switching. Give an example of a communications network which uses each style. [4]
- (b) Explain why routes in the Internet are not chosen by consulting a central database of all possible host-to-host routes. [5]
- (c) Explain in general terms how Internet routing is actually carried out. [6]
- (d) Most airline web sites now allow customers to buy tickets online. An alternative way of supporting online ticket sales would be for customers to install specialised client software which would communicate with a server at the airline, either using the Internet or by means of direct telephone connections. What do you think would be the advantages or disadvantages of this alternative system, both technically and in relation to HCI issues? [10]

## Worksheet 10 (Semester 2, Week 11 Lab)

The exercises on this worksheet will illustrate points from recent lectures and improve your understanding. The last exercise will be useful preparation for the exam. You should also take advantage of this final opportunity to ask your tutor any questions about topics from the whole module.

### Investigating Processes

Press Ctrl-Alt-Del and select Task Manager. This will start the Task Manager, an application which displays information about the tasks or processes which are being executed, and the resources they are using.

Click on the Applications tab to see a list of the applications which are running. See how this changes when you start a new application, or close a running application.

Click on the Processes tab to see a list of all the processes which are running. These are the tasks which are taking it in turns to be executed by the CPU: this is where multi-tasking is working. Notice that there are far more processes than the number of running applications: the operating system uses several processes for its own purposes. For example, the process called EXPLORER.EXE is the user interface of the operating system itself: it controls the desktop.

Click on the Performance tab to see a graph of CPU usage. This shows the proportion of time the CPU is spending on executing applications, rather than managing multi-tasking. Start a new application (for example, Microsoft Word) and observe the corresponding peak in CPU activity.

Within the Applications view, it is possible to terminate any of the currently running applications by clicking on End Task. Try this out, but beware that terminating an application may result in loss of data because it may not be given a chance to tidy itself up. Sometimes it is necessary to terminate applications in this way, if they get themselves into error states from which they are unable to recover.

It is also possible to terminate internal operating system processes in the same way, within the Processes view, but this is not a good idea because they are necessary for the normal operation of the OS. For example, by terminating the EXPLORER.EXE process you can remove the user interface of the operating system; the system will continue to run without the Start menu, the task bar or the normal desktop view. It would then be necessary to log out, and log in again to restore normal service.

### Investigating the Internet: DNS Lookup

Start Internet Explorer and go to the site [www.kloth.net/services/nslookup.php](http://www.kloth.net/services/nslookup.php) which provides an interface to the Domain Name Service. You can enter any domain name and find out the corresponding IP address.

In the Domain box, enter a domain name (for example [news.bbc.co.uk](http://news.bbc.co.uk)). In the Query box, select A (IP address). Click on the button Look it up. (What is happening is that a program called nslookup is executed on the machine which is running the web server for [www.kloth.net](http://www.kloth.net)). You will see an IP address: four decimal numbers separated by dots (e.g., 212.58.226.8). Type the IP number into the address field of Internet Explorer and see what happens.

It is possible for a domain name to resolve to several IP addresses. Typically this means that an organisation is using several servers to provide the same service, probably because a high volume of requests is expected. Try looking up [www.nascar.com](http://www.nascar.com) to see an example of this.

[Continued overleaf]

### **Investigating the Internet: Route Tracing**

In Internet Explorer, go to the site [www.traceroute.org](http://www.traceroute.org) which is a catalogue of sites providing route tracing services. A good one to try is University of California, Berkeley (scroll down or follow the link to USA, then click on the Berkeley link). Type a domain name (or IP address, if you like) into the box and press Enter. For example, try [www.dcs.gla.ac.uk](http://www.dcs.gla.ac.uk) which is the department web server. What happens is that a program called `tracert` is executed on the machine running the route tracing service --- this machine is presumably located in California. This program sends a series of packets to the chosen address (and measures how long they take to arrive), producing a list of intermediate points which the packets pass through. How many network hops does it take for packets to reach Glasgow from California? You will see all sorts of domain names and IP addresses in the listing. IP addresses beginning with 130.209 are in the University of Glasgow. Domain names ending with `.ja.net` are part of JANET, the network used by UK universities to access the internet.

Try some of the other route tracing servers on the original list at [www.traceroute.org](http://www.traceroute.org). Each one will trace routes starting from itself, so you can see routes to Glasgow from various parts of the world. Can you see any differences in route lengths from different countries or different continents?

### **Investigating the Internet: HTTP Requests and Responses**

In Internet Explorer, go to the site [www.rexswain.com/httpview.html](http://www.rexswain.com/httpview.html) which allows you to see the exact content of the HTTP request and response when the browser accesses a website. Enter the name of a website into the URL field, for example [www.dcs.gla.ac.uk](http://www.dcs.gla.ac.uk), click SUBMIT and see what comes back. The header contains some information about the server, and specifies what kind of information follows (in this case, HTML code). The content is the HTML code that the browser has to convert into the visual representation of a web page.

### **Creating an HTML Document**

In AMS, set up the exercise S2Week11. Go into the S2Week11 folder in your CS1Q workspace and double-click on the file `lab5`. Because it is an HTML document (as you can see by selecting Details from the View menu), Internet Explorer will start, and you will see a very simple page with some text and a link. Select Source from the View menu to see the HTML code which produces this page.

In your S2Week11 folder, double-click on the file `lab5word`. This is also an HTML document, which was produced by Save as web page from Microsoft Word. View the source and compare it with the previous one; you will see that Word inserts a large amount of formatting information into the HTML file.

The file `lab5` can be edited with Programmer's File Editor or Notepad. The web page produced by `lab5word` can be edited directly with Word (right-click and use 'open with'). Using Word is an easy way to produce web pages, although more specialised HTML editors give much more control over the structure of the page.

The department does not allow first year students to publish web pages, so these HTML files can only be viewed locally; you can't make them visible from the department's web site. However, if you have internet access at home, or if you sign up with a commercial provider of web space, you could create and publish your own web site.

### **Useful Preparation for the Exam**

The university provides you with an email service in the form of a web application; you use it by connecting to the appropriate web site with your web browser (for example, Internet Explorer). Web-based email services are also popular outside the university; you might have used an email service such as Hotmail or Gmail.

An alternative way of providing email is through a special-purpose software application which would run on your computer, storing your messages locally and using the internet when necessary to send and receive messages. You might have used an email application of this kind at home, for example Outlook or Eudora.

What do you think are the advantages and disadvantages of each kind of email service? There are all sorts of issues, including HCI, security, technical implementation details, and others. Think about this question, and discuss it with your friends, but you do not have to write anything down or hand anything in. You will find that thinking about the issues raised will be useful preparation for the exam.

# Appendix 1: A User Guide for the ITM, A Simple Computer Architecture

**Cordelia Hall and John O'Donnell**  
**University of Glasgow**

**October 2002**

The ITM (Information Technology Machine) is a simple computer architecture designed specifically for learning the fundamentals of computer systems and machine language. It omits many of the features that make real machines complicated, while retaining most of the essential characteristics of computers. The ITM is a modern RISC style architecture, with 16-bit words and 16 registers and a Load/Store style instruction set. It lacks facilities for Input/Output; the only way to find out what is happening inside the machine is to use the emulator.

Programming is supported by a software package that contains an assembler and an emulator that gives both interactive feedback and post-mortem trace files. These tools can be controlled from a graphical user interface.

---

## ***Architecture of the ITM***

---

To keep the ITM simple, it doesn't have bytes, short words, long words, and so on, unlike most real computers. Instead, it has a word size of 16 bits, and each register and each memory location contains a 16-bit word.

The central processing unit (CPU) contains a register file with 16 registers named R0, R1, R2, ..., R9, Ra, Rb, Rc, Rd, Re, and Rf. It also has several registers concerned with program execution, including:

- The PC (program counter) register contains the address of the next instruction to be executed
- The IR (instruction register) contains the instruction currently being executed

---

## ***The Instruction Set***

---

The ITM has a simple instruction format with a 4-bit operation code (opcode) field. This limits it to 16 instructions, and actually it doesn't even have quite that many!

The table below summarises the instructions. The Opcode column gives the hexadecimal digit operation code, which is needed in order to figure out the machine language representation of the instruction. The Mnemonic column gives the symbolic name of the instruction used in assembly language programming. The Operand column shows the typical form of operands for the instruction. Finally, the Meaning column describes the behaviour of the instruction using Java-like notation.

For clarity, the table contains specific examples—for example, it mentions register R1 and a label. In practice, any register or label may be used instead, with one exception: it is an error to execute an

instruction that modifies R0. The value of R0 is always 0, and a program should never attempt to change that.

Addresses are specified in the form `label[R2]`, where `label` is a name that appears in the label field of some assembly language statement. The assembler figures out what address corresponds to `label`, and it will place that into the instruction. The effective address used in an instruction is the value of the address word (the address of the label) plus the contents of the index register. For example, suppose that `result` is a label that refers to memory location `$002c`, and suppose that R4 contains `$0012`. Then `LOAD R1,result[R4]` will fetch the word from memory location `$003e` (the sum of `$002c` and `$0012`), and that value will be placed into register R1.

Opcode	Mnemonic	Operand	Meaning
1	LOAD	R1,label[R2]	R1 := Mem [label+R2]
2	LDVAL	R1,\$num	R1 := \$num
3	ADD	R1,R2,R3	R1 := R2 + R3
4	SUB	R1,R2,R3	R1 := R2 – R3
5	NEG	R1,R2	R1 := -R2
6	MUL	R1,R2,R3	R1 := R2 * R3
7	STORE	R1,label[R2]	Mem [label+R2] := R1
8	CMPEQ	R1,R2,R3	R1 := (R2 == R3)
9	CMPLT	R1,R2,R3	R1 := (R2 < R3)
A	CMPGT	R1,R2,R3	R1 := (R2 > R3)
B	JUMPT	R1,label[R2]	if (R1==1) then PC := label+R2
C	JUMPF	R1,label[R2]	if (R1==0) then PC := label+R2
D	JUMP	label[R1]	PC := label+R1
E	CALL	label[R1]	push(PC); PC := label+R1
F	RETRN		PC := pop

---

## **Assembly Language**

---

### **Format of Assembly Statements**

A program written in ITM assembler language is a text file; that is, it contains plain ASCII characters with no formatting. The program consists of a sequence of lines, and a line may be one of the following:

- An instruction
- A DATA statement
- A comment
- A blank line

Everything that appears after a % is a comment, and has no effect on the program's meaning. Blank lines and comments have no effect on the program's meaning, but they are used to make it more readable.

Instruction statements consist of the three fields. Each field must contain no spaces, and the fields are separated by one or more spaces (use spacing to line up the fields vertically and make the program look neat). After the last field, there should be a comment beginning with %. The fields of an instruction statement are:

- *Label*: The label field optional; if the statement has a label, it must begin in the first character of a line, and it may contain only letters.
- *Operation mnemonic*: This is the symbolic name for an instruction, and must be one of the mnemonics listed in the Instruction Set table above.
- *Operands*: This field specifies the registers, numbers and addresses that are required by the instruction. Its syntax should follow the form given in the Instruction Set table. No spaces are permitted within the operand field.

DATA statements are used to define a word containing a number rather than an instruction. It looks syntactically like an instruction statement, where DATA appears in the mnemonic field, and a hexadecimal constant appears in the operand field. When you use a DATA statement to define an ordinary integer variable, it should have a label—the variable name—but the label is not required.

---

## ***Running the ITM Software***

---

The ITM software contains an assembler, an emulator, and a graphical user interface. You can launch the system by clicking on the IT\_Machine icon. In general, you write a program, prepare the assembly language source file, assemble it, load it, and execute it. We'll go through each of these steps in detail.

### **Prepare the Assembly Language Source Program**

The assembly language program needs to be a text file (i.e. a file containing only visible ASCII characters, with no binary formatting characters), and it needs to have an extension of `.ap` for assembly program. This means that the file must have a name like `program.ap`. The file name must *not* end with `.txt`.

It can be surprisingly difficult to get an editor on Windows to produce a text file with the right name. Don't use Word or WordPad. Edit the source program with NotePad, and when you save the file, you must (1) enter the file name as `program.ap`, and then (2) select type **All Files** in the **Save As Type** combo box (do not leave the default type **Text Documents (\*.txt)**). If you don't follow these instructions, you're likely to end up with a file with a name like `program.ap.txt` and/or containing binary formatting codes, and it just won't work.

## Launch the ITM System

Click on the IT\_Machine jar file, and the system will launch.

## Navigate to the Source Program

Select File: Open, and a dialogue box will appear that lets you navigate to your source program. By default it will open in a folder containing some example programs, and you can develop your program in that same folder.

## Assemble the Source

Once you have edited and saved your source program using NotePad, select Run: Assemble, and the system will translate your program to machine language. Now select View: Assembler Listing. Look at the result; if there are any error messages you must fix the mistakes in the source program, save it, and do Run: Assemble again. Don't go on to the next step until all the error messages have been resolved!

## Load the Object

Now select Run: Load, which tells the operating system to load your machine language file into the ITM computer's memory. You'll see a lot of hexadecimal numbers appearing in the memory section of the window—you'll see that this is the same as the machine language code in the assembler listing

## Execute the Machine Language Program

Now run the program. You can either run it one instruction at a time (recommended for debugging) by clicking on the single right triangle icon, or you can run it at full speed by clicking on the double triangle. These operations are also available through the run menu. You can also experiment by setting breakpoints; these enable you to tell the system to stop the execution at a certain instruction, which is helpful in debugging.

## Examine the Post-mortem Trace

When the program has finished, look at the state of the registers and memory to determine whether it worked correctly.

### Acknowledgements.

The ITM architecture was designed by David Gillespie, Cordelia Hall and John O'Donnell. The assembler and emulator were implemented by Cordelia Hall, and the first version of the system documentation was also written by Cordelia Hall. The graphical user interface was implemented by Grant McGarry, supervised by John O'Donnell.