FIT1047 S1 2017
Assignment 1

Submission guidelines

This is an individual assignment, **group work is not permitted.**

Deadline: April 13, 2017, 11:55pm

Submission format: PDF for the written tasks, LogiSim circuit files for task 1, MARIE assembly files for task 2. All files must be uploaded electronically via Moodle.

Individualised exercises: Some exercises require you to pick one of several options based on your student ID.

Late submission:

- By submitting a special consideration form, available from `http://www.monash.edu.au/exams/special-consideration.html`
- Or, without special consideration, you lose 5% of your mark per day that you submit late (including weekends). Submissions will not be accepted more than 5 days late.

  This means that if you got x marks, only $0.95^n \times x$ will be counted where n is the number of days you submit late.

In-class interviews: See instructions for Task 2 for details.

Marks: This assignment will be marked out of 70 points, and count for 17.5% of your total unit marks.

Plagiarism: It is an academic requirement that the work you submit be original. **Zero marks** will be awarded for the whole assignment if there is any evidence of copying (including from online sources without proper attribution), collaboration, pasting from websites or textbooks.

The faculty's Plagiarism Policy applies to all assessment:

`http://intranet.monash.edu.au/infotech/resources/students/assignments/policies.html`

Further Note: When you are asked to use internet resources to answer a question, this **does not mean copy-pasting text** from websites. Write answers in your own words such that your understanding of the answer is evident. Acknowledge any sources by citing them.

1 Boolean Algebra and Logisim Task

The following truth table describes a Boolean function with four input values $X1, X2, X3, X4$ and two output values $Z1, Z2$.

X1	X2	X3	X4	Z1	Z2
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	0	1

The main result of this task will be two logical circuits correctly implementing this Boolean function in the logisim simulator. Each step as defined in the following sub-tasks needs to be documented and explained.

1.1 Step 1: Boolean Algebra Expressions (8 points)

Write the Boolean function as Boolean algebra terms. First, think about how to deal with the two outputs. Then, describe each single row in terms of Boolean algebra. Finally, combine the terms for single rows into larger terms. Briefly explain these steps for your particular truth table.

1.2 Step 2: Logical circuit in Logisim (8 points)

Model the resulting Boolean terms from Step 1 in a single Logisim circuit, using the basic gates AND, OR, NOT. You can use gates with more than two inputs.

Explain what you did for each step.

Test your circuit using values from the truth table and document the tests.

1.3 Step 3: Optimized circuit (12 points)

The goal of this task is to find a minimal circuit using only AND, OR, and NOT gates. Based on the truth table and Boolean algebra terms from Step 1, optimize the function using Karnaugh maps.

You will need to create two Karnaugh maps, one for each output. Your documentation should show the maps as well as the groups found in the maps and how they relate to terms in the optimized Boolean function.

Then use Logisim to create a minimal circuit. Dont use any other gates than AND, OR, and NOT. Test your optimized circuit using values from the truth table.

2 Strings in MARIE

In this task you will develop a MARIE application that performs some manipulation of strings. We will break it down into small steps for you.

Most of the tasks require you to write code, test cases and some small analysis. The code must contain comments, and you submit it as `.mas` files together with the rest of your assignment. The test cases should also be working, self-contained MARIE assembly files. The analysis needs to be submitted as part of the main PDF file you submit for this assignment.

In-class interviews: You will be required to demonstrate your code to your tutor after the submission deadline. Failure to demonstrate will lead to **zero marks** being awarded to the entire programming part of this assignment.

2.1 Strings

A *string* is a sequence of characters. It's the basic data structure for storing text in a computer. There are several different ways of representing a string in memory – e.g. you need to decide which character set to use (Unicode or ASCII), and how to deal with strings of arbitrary size.

For this assignment, we will use the following string representation:

- A string is represented in a contiguous block of memory, with each address containing one character.
- The characters are encoded using the ASCII encoding.
- The end of the string is marked by the value 0.

As an example, this is how the string FIT1047 would be represented in memory (written as hexadecimal numbers):

```
046 049 054 031 030 034 037 000
```

Note that for a string with n characters, we need $n + 1$ words of memory in order to store the additional 0 that marks the end of the string.

In MARIE assembly, we can use the `HEX` keyword to put this string into memory:

```
FIT1047,    HEX 046
            HEX 049
            HEX 054
            HEX 031
            HEX 030
            HEX 034
            HEX 037
            HEX 000
```

2.1.1 Your name as a MARIE string (2 points)

Similar to the FIT1047 example above, encode your name using ASCII characters. You should encode at least 10 characters – if your name is longer, you can shorten it if you want, if it's shorter, you need to add some characters (such as !?! or ..., or invent a middle name).

You need to submit a MARIE file that contains the code that stores your name into memory.

2.1.2 Printing a string (4 points)

For this task, you need to write MARIE code that can print *any* string (no matter how long) using the `Output` instruction. Start by using a label `CurrentCharacterAddress` that you initialise with the address of the string. The code should then output the character at the address, increment it by one, and keep doing that until the character at the address is a 0 (which signals the end of the string).

Submit your MARIE code, including a test case printing your name from the previous task.

2.1.3 A subroutine for printing a string (6 points)

Turn your code from the previous task into a subroutine that takes the address of a string as an argument and outputs it. You can start from the following template:

```
PrintString,     HEX 0      / argument: address of string
Print,           HEX 0      / subroutine start
```

Your code needs to start reading the string from the address stored in `PrintString`, stopping when it finds a 0, otherwise printing the character using the `Output` instruction.

Submit your MARIE code, including a test case that calls the subroutine with the address of the string representing your name (you need to find out the concrete address by first assembling your code and then looking up where the string ended up being placed in memory, then adapt your code).

2.1.4 User input (5 points)

The next step is to implement a subroutine that reads a string, character by character, using the `Input` instruction. The subroutine takes an address as its argument which is the location in memory where the string should start. Your code needs to input a character, store it at the given address, then increment the address by 1 and loop back to the input. Once a user enters a 0, the subroutine finishes.

Note that you can switch the input box in the MARIE simulator into different modes: use the ASCII mode to enter the characters, and use Hex or Decimal to enter the final 0 (since the character "0" is different from the integer 0!).

Submit your MARIE code. Document at least two test cases using screen shots of what the memory looks like after entering a string.

2.1.5 Lower case (10 points)

Some people write emails with lots of upper case characters, which is a bit rude and unpleasant to read. So we will implement a subroutine that turns all characters in a string into lower case.

The subroutine takes the address of a string as its argument. For each character in the string, it tests whether it is upper case (i.e., whether it is between the ASCII values for A and Z), and if it is, it turns it into lower case (modifying the string stored in memory). It finishes when it reaches the 0 signalling the end of the string.

Hint: to turn a character from upper case into lower case, just add the difference between the ASCII values for ”a” and ”A”.

Submit your MARIE code and documentation of two test cases.

2.1.6 ROT13 (10 points)

Now we are going to implement a simple *substitution cipher* known as ROT13. The idea is to replace each character in a string by the character 13 places further in the alphabet, wrapping around from z to a. For example, the letter *a* is mapped to *n*, and the letter *p* is mapped to *c*.

Your task is to implement a subroutine that performs ROT13 encoding on a string.

- As a first step, implement a subroutine that can do ROT13 on a string that contains only lower case letters.
- As a second step, extend your code to ignore any characters that are not lower case letters.

Submit your MARIE code and documentation of two test cases.

2.1.7 Complete program (5 points)

As a final step, combine all the previous subroutines into a program that does the following:

- Let a user input a string
- Turn all characters into lower case
- Perform ROT13 on the string
- Output the result