

FIT1047 - Introduction to Computer Systems, Networks and Security

Assignment-1 Semester 2, 2020

Submission guidelines

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

Deadline: Assignment 1 due by week-6, Friday 4:00 PM.

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. More details at the end of the assignment specifications.

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 with supporting documentation.
- 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.

Marks: This assignment will be marked out of **100 points**, and count for **20%** 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 and pasting from websites or textbooks.

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. Your work will be submitted and compared with previous students' works and also on the internet for plagiarism using Turnitin software. So please make sure the submission is your work.

1. Boolean Algebra and *Logisim* Task (50 Marks)

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	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	0	0

Assignment Project Exam Help

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

<https://powcoder.com>

1.1 Boolean Algebra Expressions (10 points)

Add WeChat powcoder

Write the Boolean functions as Boolean algebra terms, using either the **Product of Sums (POS)** or **Sum of Products (SOP)** method. Notice that the truth table consists of four inputs and two outputs. Hence you must have two separate Boolean functions representing two outputs to the same input set.

Briefly explain the steps you followed to formulate the two Boolean functions from the given truth table. Please use standard notation for writing Boolean algebra expressions. Boolean “AND” is written without a symbol, e.g. $X1X2$. Boolean OR is written with the ‘+’ symbol, e.g. “ $X1 + X2$ ”. Boolean negation is written using an over-line, e.g. $\overline{X1}$. When writing terms like “NOT $X1$ AND NOT $X2$ ”, there must be a clear gap in the over-lines, e.g. $\overline{X1} \overline{X2}$. **Hint:** Note that $\overline{X1} \overline{X2} \overline{X3} \overline{X4}$ is not equal to “ $\overline{X1} \cdot \overline{X2} \cdot \overline{X3} \cdot \overline{X4}$ ”.

1.2 Logical circuit in *Logisim* (20 points)

Model the resulting Boolean terms from section 1.1 in a single logic circuit in *Logisim*, using only the basic gates AND, OR, NOT. You can use logic gates with more than two inputs. The logic circuit should show all the four inputs, and both the outputs $Z1$ & $Z2$ with appropriate labels. The two outputs should not be split up into two different *Logisim* circuits.

- **Briefly explain** your construction (same as for section 1.1, a short explanation is enough).
- Test your circuit using input values from the truth table and **document at least three test cases** (you may take “Print Screen” of your Logisim logic circuit showing appropriate

inputs and corresponding outputs).

1.3 Optimized circuit (20 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 section 1.1.

1. **Subtask-1:** Optimize the function using Karnaugh map to its minimal form.
2. **Subtask-2:** Optimize the function using Boolean identities and verify its minimal form with that from the subtask-1.

You will need to create **two** Karnaugh maps and work on **two** separate Boolean Identities reduction forms, one for each output Z1 & Z2. Your documentation must show

- a) the K-Maps.
- b) the groups found in the K-Maps.
- c) reduced Boolean functions derived from K-Maps.
- d) The two Boolean identities reduction for Z1 & Z2 with steps.
- e) A comparison of the two methods (K-Map & Boolean Identities methods) showing how they relate to terms in the optimized Boolean function.

Then, use **Logisim** to create a **minimal** circuit, using only AND, OR, and NOT gates. Test your optimized circuit using values from the truth table and **document at least three test cases** (you may take “Print Screen” of your *Logisim* logic circuit showing appropriate inputs and corresponding outputs).

Note: You can use Product-of-Sum or Sum-of-Product to optimise in subtask-2.

2. MARIE (50 Marks)

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 the code and test cases (wherever appropriate or necessary) with accompanying short analysis. You must submit the codes as “**.mas**” files together with the rest of your assignment. The codes must be accompanied by appropriate comments (as a paragraph before any block of code or subroutine or as inline comments wherever appropriate). The test cases should also be working, self-contained MARIE assembly files (without requiring much input from the user). The analysis needs to be submitted as part of the main PDF file you submit for this assignment.

In-class interviews: You may 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.

Name as Strings

This section introduces the concepts you need for the rest of the assignment. 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 and how to deal with strings of arbitrary length. For this assignment, we will use the following string representation:

- A string is represented in a contiguous memory location, with each address containing a single character.
- The characters are encoded using the ASCII encoding.
- A name will be composed of two strings, namely, **First Name** and **Last Name**.
- End of each string (First Name or Last Name) will be marked by an ASCII character '0'.
- End of a name is marked by another ASCII character '0'.

As an example, this is how, a name “John Noah” would be represented in memory (written as hexadecimal numbers):

04A	06F	068	06E	030	04E	06F	061	068	030	030
J	o	h	n	0	N	o	a	h	0	0

Note that for a name with ‘n’ characters, we need “n + 3” words of memory in order to store all the characters belonging to a name and the additional ‘0’s that marks the end of each of the two strings (First Name and Last Name), and end of the name.

In MARIE assembly language programming, we can make use of the new instruction “**ADR**”, HEX keyword and a label “myName” to put this string into memory:

```
myNameAdd, ADR myName
myName,    HEX 04A //J
           HEX 06F //o
           HEX 068 //h
           HEX 06E //n
           HEX 030 //0
           HEX 04E //N
           HEX 06F //o
           HEX 061 //a
           HEX 068 //h
           HEX 030 //0
           HEX 030 //0
```

2.1 Write a MARIE program to store multiple user entered names in consecutive memory rows and print them at the end of data entry. (25 marks)

Prepare a MARIE program to enter names (First Name and Last Name) using ASCII (Unicode) characters. You should enter a maximum of 5 characters for each part of a name, and that would require a full name to occupy at most 13 characters to be stored in MARIE memory. One full name is to be stored in a row of MARIE memory. Each memory row in MARIE is 16 words long and can store a maximum of 16 ASCII (Unicode) characters. Your program should store names in specific locations of MARIE memory and print them at the end of the name entry process.

The input subroutine must be named **subInputNames**. Your subroutine should take ASCII (Unicode) '0' terminated "First Name" and ASCII (Unicode) '0' terminated "Last Name" as described in the previous section. After that, a second ASCII (Unicode) '0' will be stored to mark the end of the full name, and the program should proceed to take the next name. Store the first name starting from the memory address 300_H in such a way that a single name is stored in a single memory row. That means, your second name should be stored starting from memory address 310_H, your third name should be stored starting from 320_H memory address, etc. Your program must be designed to take names endlessly until a user wants to end the name entry process. The program should exit the **subInputNames** subroutine (return control to the calling program) only when the user has entered ASCII (Unicode) '\$'. You need to store the '\$' entered at the beginning of a new row in MARIE memory to mark the end of the name database. You should be able to verify the correct working of your program by viewing the memory content starting from #300_H.

In the second part of this task, you need to prepare a MARIE subroutine called **printStringName** that can print out the full names entered earlier by the **subInputName** subroutine. Start by using a label **PrintName** that holds the start address of the first name (like, myNameAdd in the example above). When printing the names, the "First Name" and the "Last Name" should be separated by an ASCII (Unicode) 'Space' character and the end of a full name should be marked by an ASCII (Unicode) 'New Line' character. You will use the ASCII '\$' (stored at the end of the names) as the termination point for the subroutine and return control back to the main program.

To receive full marks, you need to write a MARIE main program to call the subroutines that you have created, making use of the appropriate MARIE instructions (e.g. JnS, JumpI). Save your code as "**2_1_Enter_and_PrintingNames.mas**" and provide appropriate test cases. (Hint: For testing your code, you may use three names: "your" **First Name and Last Name**, "your" friend's **First Name and Last Name**, and "your" tutor's **First Name and Last Name**. The names should be entered from the keyboard and printed to the output window).

2.2 Convert Names to UpperCase in MARIE and Print (25 marks)

Write a subroutine to change and print all characters of the user entered name strings to UPPER CASE characters. Name the subroutine as **convertToUpperCase**. This subroutine modifies all characters of a string in memory to upper case. The program should then print the original name string and the modified upper case string. Submit your MARIE code and documentation of test cases converting at least three names. To receive full marks, the main program from section 2.1 must be extended to print the original name string and the modified upper-case name string by calling the respective subroutines. Name your MARIE code file as "**2_2_UpperCase.mas**".

Hint: you can easily identify the lowercase ASCII value ranges through the ASCII table and notice the numeric difference between the uppercase and lowercase characters.

2.3 Reversing a String in MARIE (25 marks)

Implement a simple “encryption scheme” by reversing the name string stored in memory in step 2.1. Your task is to write a subroutine that takes the address of the first name as its argument and stores the characters in the reverse order starting at the address stored at label **ReverseString**. When the subroutine ends, the memory at that address is stored in **ReverseString** must contain the string but in reverse order. To receive full marks, the main program from section 2.2 must be extended to print the original name string and the reversed string by calling the respective subroutines. Save your code as “**2_3_ReverseString.mas**”. Submit your MARIE code and documentation of test cases reversing at least three names. Name your MARIE code file as “**2_3_ReverseString.mas**”. An illustrative example is given below that explains the concept of the task in this section.

Original String

Address																
300 _H	m	y	0	n	a	m	e	0	0							
310 _H	y	o	u	r	0	n	a	m	e	0	0					
320 _H	t	u	t	o	r	0	n	a	m	e	0	0				
330 _H	\$															

Reverse String

Address																
350 _H	e	m	a	n	0	y	m	0	0							
360 _H	e	m	a	n	0	r	u	o	y	0	0					
370 _H	e	m	a	n	0	r	o	t	u	t	0	0				
380 _H	\$															

Files to be submitted:

Assignment submission files:

One zipped **folder** named “YourFirstName_LastName_StudentID.zip” containing the following files.

1. Report for the written tasks

(One PDF file called YourFirstName_LastName_StudentID.pdf).

- a. The report should include your Full name, your student ID, your class number and your tutor's name.
 - b. Include the original truth table in the pdf.
 - c. The pdf report should show the workings for the following:
 - i. **Subtask-1:** Optimize the function using Karnaugh maps to its minimal form
 - ii. **Subtask-2:** Optimize the function using Boolean identities and verify its minimal form from the subtask-1.
 - iii. You will need to include two Karnaugh maps worked on two separate Boolean Identities reduction forms, one for each output Z1 & Z2. Your documentation must show
 1. the K-maps.
 2. the groups found in the maps
 3. The Boolean identities reduction steps
 4. and how they relate to terms in the optimized Boolean function.
 - d. Documentation related to Task 2, if there is any.
2. Task 1.2 – Logisim File. Name the file as “LogicCircuit.circ”.
 3. Task 1.3 – Logisim File. Name the file as “OptimizedCircuit.circ”
 4. Task 2.1 – MARIE File. Name the file as “2_1_Enter_and_PrintingNames.mas”
 5. Task 2.2 – MARIE File. Name the file as “2_2_UpperCase.mas”
 6. Task 2.3 – MARIE File. Name the file as “2_3_ReverseString.mas”

Zip the folder with this name (YourFirstName_LastName_StudentID.zip) and submit it to Moodle. You need to make sure there are no spaces in any of the filenames.

Tutorial Week-1 to Week-5 (inclusive) Reflective diary submission:

- Weekly laboratory and problem exercises (**Weight: 5% Unit Marks**)
- Maximum 500 words.

Hand in the individual reflective diary/journal of lab work submitted for each week from week 1 to 5 and their analysis and reflection on what you have learnt in these five weeks and submit it along with the Assignment 1. A reflective journal is a document you write down your each week's lab reflective entries. It can be something positive and constructive learning experience or difficulties you faced during the learning process you can self-reflect on and learn from past experiences. A reflective diary/journal can help you to identify important learning concepts in the unit. The experiences also include your relationships, with tutor and colleague you may have interacted with. By writing a reflective diary, you can find all your thoughts, concepts and any other learning experience you have come across. A reflective journal also provides a better understanding of your thought process.

End of Assignment 1