Assignment 3 – Processes and MARIE Programming

Purpose	Processes and programs are what makes computers do what we want them to do.In the first part of this assignment, students will investigate the processes running on their computers. The second part is about programming in MARIE assembly language. This will allow students to demonstrate their comprehension of the fundamental way a processor works. The assignment relates to Unit Learning Outcomes 2, 3 and 4.
Your task	For part 1, you will write a short report describing the processes that are running on your computer. For part 2, you will implement a simple game in the MARIE assembly language.
Value	30% of your total marks for the unit The assignment is marked out of 50 marks.
Word Limit	See individual instructions
Due Date	**11:55 pm Thursday 14 September 2022**
Submission	Via Moodle Assignment Submission. Turnitin and MOSS will be used for similarity checking of all submissions. This is an individual assignment (group work is not permitted). You will need to **explain your code** in an interview.
Assessment Criteria	Part 1 is assessed based on correctness and completeness of the descriptions. Part 2 is assessed based on correctness of the code, documentation/comments, and test cases. See instructions for details.
Late Penalties	10% deduction per calendar day or part thereof for up to one week Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page
Feedback	Feedback will be provided on student work via: general cohort performance specific student feedback ten working days post submission

INSTRUCTIONS

This assignment has two parts. Make sure you read the instructions carefully.

Part 1: Processes (10 marks)

For this task, write a brief report about processes that you observe running on your computer. You can use one of the following tools (depending on your operating system):

On Windows, use the Task Manager
On macOS, use the Activity Monitor
On Linux, use a command line tool like htop, top, or the ps command

Answer the following questions:

1. Briefly describe the columns displayed by the tool you use that relate to a) memory usage and b) CPU usage of a process. What can you say about the overall memory usage of all processes, compared to the RAM installed in your computer? (4 marks)
2. Pick a process you perhaps don't know much about, or which you did not expect tofind running on your computer. Try to find out and describe briefly what it does. (4 marks)
3. Briefly explain why it is important that the operating system manages the files stored on your computer (rather than each application having to manage those files itself). (2 marks)

Include a screenshot of your processes in the report along with utilization graphs. The word limit for this part (all three questions together) is 600 words (about 1 page, not including images and tables).

Part 2: MARIE Programming (40 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. You need to submit a working MARIE program for each individual task and you will get marks for each separate program.

Each task requires you to write **code** and **test cases**. On Moodle, you will find a **template** for the code for tasks 2.2 and one template for the remaining tasks. **Your submission must be based on these templates**, i.e., you must add implementations of your own subroutines into these templates.
The code must contain comments, and you need to submit one zip file containing one `.mas` file for each task together with the rest of your assignment.

In-class interviews: You will be required to demonstrate your code to your tutor during yourapplied session after the submission deadline. Failure to demonstrate will lead to zero marks being awarded for the entire assignment.

Code similarity: We use tools to check for collaboration and copying between students. If you copy parts of your code from other students, or you let them copy parts of your code, you will receive 0 marks for the entire assignment.

Rubric: The marking rubric on Moodle provides details for the marking. Each task below is worth a certain number of marks. A correctly working MARIE program of each task that is well documented and contains the required test cases will receive full marks.

Missing/incomplete documentation will result in a loss of up to ¼ of the task's marks.
Missing or undocumented test cases result in the loss of 1 mark per test case.

Introduction: 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. usually you would need to decide which character set to use, 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, this means you need to use the hexadecimal numbers for the ASCII characters. Note that MARIE provides Unicode (UTF-16BE) as an option, but the first 128 characters are the same as the ASCII coded characters,
> 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
```


Task 2.1 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 including space).

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

Note that this code will not actually do anything else than storing your name in memory and you will not need to use the template for this.

Task 2.2 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 initialize with the address of the string (for example with StringtobePrinted in the template). 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 that shows the test case of printing your name from the previous task.

Task 2.3: A subroutine for printing a string (4 points)

Turn your code from the previous task into a subroutine that takes the address of a string as an argument and outputs it. Use the template for this task.
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. Document your test with a screen shot showing MARIE memory after executing the code.
Hint: 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).

Task 2.4: User input (5 points)

The next step is to implement a subroutine that reads a string, character by character, usingthe 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 a given address (use the last two digits of your student ID in HEX with prefix '1', for example, if last two digits of student ID is 01 then 101), 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 UNICODE mode to enter the characters, and use Hex or Decimal to enter the final 0 (sincethe character '0' is different from the integer 0).

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

Task 2.5 Lower case (5 points)

Some people write emails with lots of uppercase 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 signaling 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 (i.e. document your Input into the program and the output you observe, so that it can be repeated during the interview.

Task 2.6 ROT13 (12 points)

Now we combine the previous subroutines and add another subroutine that implements 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 with lowercase letters. It does not need to work or provide any meaningful output on wrong input, e.g. upper case or other characters.

For example:

Input: abp
Output:
noc

Input: abp?A!
Output: noc?A!

Thus, for this task you need to do the following:

Implement a subroutine that can do ROT13 on a string that contains only lower case letters.

Combine the new subroutine and all the previous subroutines into a program that does the following:

- Let a user input a string (input one character at a time)
- Perform ROT13 on the string
- Output the result

Submit the complete MARIE code and document 3 test cases.

Task 2.7 Extend the ROT13 code to work for all ASCII characters (8 points)

For this task, you need to extend the ROT13 subroutine in your code from Task 2.6 to ignore all characters that are not lower case letters. Thus, your new program should apply ROT13 to all letters and keep other characters as they are (remember that upper case letters are turned into lower case before applying ROT13).

For example:

Input: abp?A!
Output: noc?n!

Thus, for this task you need to do the following:

Implement a subroutine that can do ROT13 on a string that contains any ASCII characters.

Combine the new subroutine and all the previous subroutines into a program that does the following:

- Let a user input a string (input one character at a time)
- Turn all letters into lower case (keep other characters as they are)
- Perform ROT13 on the string
- Output the result.

Submit the complete MARIE code and document 3 test cases.