MONASH University

FACULTY OF INFORMATION TECHNOLOGY

FIT1047 Introduction to computer systems, networks and security - S2 2023

Assignment 2 – 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	25% 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 Friday 21 April 2023**
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). In this assessment, you must not use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task. 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? Include graphs for the comparison. (4 marks)
2. Pick a process you perhaps don't know much about, or which you did not expect to find 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 usage 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 one **template** for the code for tasks 2.1 and 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 Task 2.1-2.2 (based on the template)
- one `.mas` file for Tasks 2.3-2.7 (based on the template)
- one PDF file documenting your test cases

In-class interviews: You will be required to join an interview to demonstrate your code to your tutor during your applied session after the submission deadline. Failure to demonstrate will lead to zero marks being awarded for the entire assignment. In addition, for Task 2.8 you will need to answer further questions about your submitted code (see below for details).

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
  - The first address encodes the length of the string (i.e., the number of characters).
  - The following addresses each contain one character of the string.
- The characters are encoded using the Unicode (UTF-16BE) encoding. Note that 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):

007 046 049 054 031 030 034 037

The first memory location contains the length (7 in this case), the following locations contain the character codes.

Note that for a string with *n* characters, we need *n+1* words of memory in order to store the length plus all the characters.

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

```
FIT1047,    HEX 007
            HEX 046
            HEX 049
            HEX 054
            HEX 031
            HEX 030
            HEX 034
            HEX 037
```

Task 2.1 Your name as a MARIE string (2 points)

Similar to the FIT1047 example, encode your name as a string using the string representation introduced above. 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). If your name contains non-ASCII alphabet characters that can be encoded in Unicode, you can of course use those as well (e.g., combine the ASCII alphabet version of your name with the Unicode version).

You need to submit a MARIE file that contains the code that stores your name into memory. Use the template that you can download from Moodle.

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 (3 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 `PrintCharactersRemaining` that you initialize with the size of the string (which is stored in the first memory location of the string). In addition, the code needs to use a label `CurrentCharacterLocation` that is initialised to the memory location of the first character of the string, i.e., the location after where the size is stored.

The code should then check if there are characters remaining to be printed. If no more characters are available, we have reached the end of the string and can Halt. If more characters are available, the code outputs the character stored at `CurrentCharacterLocation`, increments `CurrentCharacterLocation` by one, decrements `PrintCharactersRemaining` by one and loops back to the test whether any characters are remaining.

Submit your MARIE code that shows the test case of printing your name from the previous task. Use the template that you can download from Moodle (you only have to submit one file that includes your solutions to Tasks 2.1 and 2.2).

Task 2.3: A subroutine for printing a string (3 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 `PrintFrom`, stopping after all characters in the string have been printed, 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.
Use the template for tasks 2.3.-2.7 that you can download from Moodle. Add your code for all remaining tasks to this one template file.

Hint: You can use the `ADR` instruction to get the location of the label where your name starts.

Task 2.4: User input (4 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. The label for that address should be `InputStringStart`.
Your code should initialise a label `CurrentCharacterLocation` to the location of the first character in the string (i.e., leaving one location empty for the size of the string).
Your code then runs in a loop like this:

- Get one character from the user using the Input instruction
- If the input is not 0 (the integer 0, not the ASCII character 0), write the character into

`CurrentCharacterLocation`, increment `CurrentCharacterLocation` by one, and start the loop again.
- If the input is 0, store the size of the string into `InputStringStart` and return from the subroutine.

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 (since the character ‘0’ is different from the integer 0).

Submit your MARIE code (add it to the template file you downloaded from Moodle). Document at least two test cases using screenshots of what the memory looks like after entering a string.

Task 2.5 Upper case (4 points)

Sometimes, you may have written a message to someone, but you want to make it really clear how angry or excited you are, so you decide to rewrite the message using only upper case letters. To make this easier, we will now implement a subroutine that turns all characters in a string into upper case.

The subroutine takes the address of a string as its argument. For each character in the string, it tests whether it is lower case (i.e., whether it is between the ASCII values for a and z), and if it is, it turns it into upper case (modifying the string stored in memory). It finishes when it has processed all characters of the string (as indicated by the string’s size).

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

Submit your MARIE code (add it to the template file you downloaded from Moodle) 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 (10 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 upper case letters. It does not need to work or provide any meaningful output on wrong input, e.g. lower 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 upper case letters.

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

- Let a user input a string (using the subroutine from 2.4)
- Turn lower case into upper case characters (using the subroutine from 2.5)
- Perform ROT13 on the string (using the new subroutine)
- Output the result (using the subroutine from 2.3)

Submit the complete MARIE code (add it to the template file you downloaded from Moodle) and document 3 test cases.

Task 2.7 Extend the ROT13 code to work for all ASCII characters (6 points)

For this task, you need to extend the ROT13 subroutine in your code from Task 2.6 to also work for lower case characters, and to add 2200_{16} to all other characters. Thus, your new program should apply ROT13 to all letters, and shift all other characters into the range of Unicode symbols above 2200_{16}.

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 the previous subroutines into a program that does the following:

- Let a user input a string (using the subroutine from 2.4)
- Perform ROT13 and shifting above 2200_{16} on the string (using the new subroutine)
- Output the result

Submit the complete MARIE code (add it to the template file you downloaded from Moodle) and document 3 test cases.

Task 2.8 In-class interview (8 points)

You need to demonstrate the code you submitted for Task 2.1–2.7 to your tutor in an in-class interview after the submission deadline. Failure to explain how your code works will result in 0 points for the individual tasks that you cannot demonstrate.

In addition, you will be asked to modify the code you submitted in certain ways and explain how the MARIE concepts work that you were required to use for the individual tasks. These additional questions add up to 8 points for this task (Task 2.8).