MONASH University

FACULTY OF INFORMATION TECHNOLOGY

FIT1047 Introduction to computer systems, networks and security – Oct 2022

Assignment 2 – Processes and MARIE Programming

Purpose	Processes and programs are what make 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 program 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 16 December 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). • DRAFT submission is not assessed. • You will need to **explain your code** in a video 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.

You need to submit three files through the Moodle Assignment activity:

1) **A .pdf document with the answers to the questions, and your documentation of the test cases for Part 2**
2) **A separate .zip archive with the MARIE files**
3) **A video recording demonstrating your MARIE programs**

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 of a process
    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 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 utilization graphs. The word limit for this part (all three questions together) is 600 words (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 two working MARIE programs and a video recording demonstrating your MARIE programs. You will get marks for each individual task.

Each task requires you to write **code** and **test cases**. On Moodle, you will find a **template** for the code for task 2.1 and a **template** for the remaining tasks. **Your submission must be based on these templates**, i.e., you must add implementations of your own codes and subroutines into these templates.

The code must contain comments, and you need to submit one zip file containing two `.mas` files, and a video recording together with the rest of your assignment.

Video interviews: You are required to make a short video to demonstrate your code. Failure to demonstrate will lead to zero marks being awarded for the entire assignment.

1. Record a video presentation (using Panopto, Zoom, Teams or any software of your choice) demonstrating **Task 2.1** and **Task 2.6** (length maximum 5 minutes).
2. If you could not demonstrate the required task, briefly explain the reason, and replace it with another task.
3. At the start of the video, introduce yourself and show your ID (Monash or others) while introducing yourself (excluded from 5 minutes limit).
4. Make sure your camera and microphone are available during the recording. The video needs to be in a common format (AVI, MOV, MP4, M4V, etc) and should be of high enough quality to be clearly understood and viewed. The video should be no more than 100 MB in size.

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 ASCII encoding, this means you need to use 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: Printing your name as a MARIE string (6 marks)

1. 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). *(2 marks)*

2. Next, you need to write MARIE code that can print any string (no matter how long) using the Output instruction. Start by using a label **CurrentCharacterADR** that you initialize with the address of the string (for example with **NametobePrinted** 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). *(4 marks)*

Use the template for this task. Submit your MARIE code that prints your name.
Document a test case with a screenshot showing MARIE Output log after executing the code.

Task 2.2: A subroutine for printing a string (4 marks)

Turn your code from the previous task into a subroutine, **subPrintString**, that takes the address of a string as an argument and outputs it.
Your code needs to start reading the string from the address stored in **CurrentCharacterADR**, stopping when it finds a 0, otherwise printing the character using the Output instruction.

Insert the subroutine in the template, including a test case in the main routine that calls the subroutine with the address of the string representing your name.
Document your test case with a screenshot showing MARIE Output log after executing the code.

Task 2.3: A subroutine for string input (5 marks)

The next step is to implement a subroutine, **subInputString**, that accepts 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 a given address, then increment the address by 1 and loop back to the input. Once a user enters a 0, the subroutine finishes.
Start by using the label **StringAddress** in the template to hold the start address of the string. The string must be stored starting from the memory address **$2XX_H$** (use the last two digits of your student ID in HEX with prefix '2', for example, if the last two digits of student ID is 01 then 201).

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).

Insert the subroutine in the template, including codes in the main routine that calls the subroutine to accept a string and store it in the memory.
Document at least two test cases using screenshots of how the entered string in MARIE Memory looks like after executing the code.

Task 2.4: A subroutine to convert string into lowercase (7 marks)

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 lowercase.

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

For example:
Original String: `UPPER-2-lower!`
Converted String: `upper-2-lower!`

For this task you need to do the following:

- Implement a subroutine that can convert a string to lowercase and store it back into memory.
- Combine the new subroutine and all the **previous subroutines** into a program that does the following:
  1. Let a user input a string and store it in memory (input one character at a time)
  2. Read the string from memory, convert it to lowercase, and store it back in memory
  3. Read from memory and output the converted string

Hint: to turn a character from uppercase into lowercase, just add the difference between the ASCII values for "a" and "A".

Insert the subroutine in the template, including codes in the main routine that calls the subroutines to convert the entered string and output it.
Document at least two test cases using screenshots of MARIE output logs and how the modified string in MARIE Memory looks like after executing the code.

Task 2.5: A subroutine to perform ROT13 encoding on a string (10 marks)

Encryption is an important way to protect the confidentiality of sensitive information. Here we will implement a simple substitution cipher known as ROT13. The idea is to replace each character in a string with 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, **subROT13**, that takes the address of a string as its argument. It performs ROT13 encoding on a string with lowercase letters and stores it back in memory. If it is not a lowercase character (e.g. upper case or other characters), it will skip and continue to check the next character until it reaches 0, signalling the end of the string.

For example:
Original String: `encode-ROT13!`
Encoded String: `rapbqr-ROT13!`

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

- Implement a subroutine that can do ROT13 encoding to lowercase characters in a string and store them back in memory.
- Combine the new subroutine and the **previous subroutines** into a program that does the following:
  1. Let a user input a string and store it in memory (input one character at a time)
  2. Read the string from memory, perform ROT13 on the string, and store it back in memory
  3. Read from memory and output the encoded string

Insert the subroutine in the template, including codes in the main routine that calls the subroutines to encode the entered string and output it.
Document at least three test cases using screenshots of MARIE output logs and how the modified string in MARIE Memory looks like after executing the code.

Task 2.6: A complete program with nested subroutines (8 marks)

Subroutine nesting is a common programming practice in which one subroutine calls another subroutine to make complex programming easier. For this task, we will complete the program by implementing a nested subroutine that calls all previous subroutines from Task 2.2 to Task 2.5.

You need to implement a nested subroutine, **Encode_ROT13**, that can converts a string to lowercase, and encodes it with ROT13, by utilising all previous subroutines. It will also output the original string, the converted string in lowercase, and the final encoded string.

For example:

```
Input:    [A2]MARIE?Program!
Output:   [A2]MARIE?Program!
          [a2]marie?program!
          [n2]znevr?cebtenz!
```

Thus, for this task you need to implement a nested subroutine that calls all the **previous subroutines** to do the following:

1. Let a user input a string and store it in memory (input one character at a time)
2. Read from memory and output the original string
3. Read the string from memory, convert it to lowercase, and store it back in memory
4. Read from memory and output the converted string
5. Read the string from memory, perform ROT13 on the string, and store it back in memory
6. Read from memory and output the encoded string

Use the template for this task. Submit the complete program and document at least three test cases using screenshots of MARIE output logs showing the strings after executing the code.