MONASH University

FACULTY OF INFORMATION TECHNOLOGY

FIT1047 Introduction to computer systems, networks and security – OCT 2024

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	Part 1a: Submit your reflections. See details below. Part 1b: Write a short report describing the processes running on your computer. Part 1c: Disassemble and add comments to a MARIE program. Part 1d: Write a MARIE program that can display bitmap numbers. Part 2: Interview for Part 1d.
Value	25% of your total marks for the unit. The assignment is marked out of 60 marks.
Word Limit	See individual instructions.
Due Date	**Part 1a-1d: 11:55 pm Friday 29 November 2024** **Part 2: Interview during Weeks 7-9**
Submission	• This is an individual assignment (group work is not permitted). • Turnitin and MOSS will be used for similarity checking of all submissions. **Ignore the Turnitin warning or error messages for zip files or other non-document files.** • **DRAFT submission is not assessed.** DRAFT upload confirmation email from Turnitin is not a submission. You must press the submit button to accept the terms and conditions in Moodle. • 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 and extend your code** in an interview. (Part 2)
Assessment Criteria	Part 1a is assessed based on relevance of the submission to the unit. Part 1b is assessed based on correctness and completeness of the descriptions. Part 1c is assessed based on correctness of the code and the labels/comments. Part 1d is assessed based on correctness of the code, as well as the documentation/comments. Part 2 is assessed based on the understanding of the code you have written. See instructions for details.

Late Penalties	• 5% 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 five parts. Please ensure you read and follow the instructions carefully.

For Part 1a, you need to submit **one** file through Moodle Assignment activity:

1) A **PDF** file containing Week 4 to Week 6 Reflections.

- Failure to submit this file will result in a maximum of 49% for the entire Assignment 2.

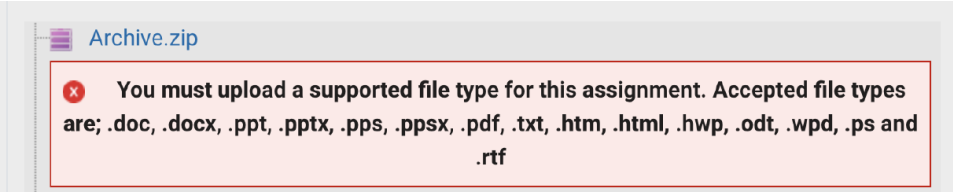
For Part 1b-1d, you need to submit files through the respective Moodle Assignment activities:

1) Part 1b: one **PDF** file (containing answer).
2) Part 1c: one **MARIE** (.mas) file based on the template for Part 1c, and your individualised memory screenshot (.png) downloaded from Moodle.
3) Part 1d: one **MARIE** (.mas) file based on the template for Part 1d.

- Failure to submit the correct file type, or using the correct template, will result in significant marks deduction.

You should clearly indicate the part that the file refers to, e.g. using appropriate file names such as `ID_Part1b_Report.pdf`, `ID_Part1d_CountDown.mas`, etc. The individualised memory screenshot file does not need to be renamed.

- Please check your final upload before submit! You CANNOT revert any submission after it's made.
- If there is a Turnitin warning or error messages for non-document files, you can disregard the error messages.



Part 2 is an online interview scheduled for Weeks 7 to 9. Instructions will be available in Moodle and communicated via an announcement post.

Part 1a: Reflections (Hurdle – you MUST submit it in order to pass this assignment!)

Complete your reflection activities in Week 4 to Week 6 Ed Lesson and copy/paste them into a PDF file. Each week the reflection must have at least 100 words, focusing on relevant and meaningful content specific to each week. Please ensure to add headings for each week.

You can use the template provided on Moodle to organize and write your reflection.

Failure to submit all relevant weekly reflections (whether missing entirely or incomplete) will result in your Assignment 2 having a maximum possible mark of 49%. For example, if the overall combined mark is 61/100, it will be scaled down to 49/100. If the overall combined mark is 44/100, it will remain unchanged at 44/100.

Part 1b: Processes (10 marks total)

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 processes. Discuss the overall memory usage of all processes and compare it to the total RAM installed on your computer. Use graphs or charts to support your comparison in memory and CPU usage. *(5 marks)*
2. Pick a process you are unfamiliar with or did not expect to find running on your computer. Investigate and provide a brief description of what it does. You may also evaluate its significance or impact on your system. *(5 marks)*

Include a screenshot of your running processes in the report, displaying between 5 and 10 processes, along with CPU and memory usage graphs and/or charts. The combined word limit for Part 1b is 500 words (approximately 1 page, excluding images and tables). Please use APA 7th referencing style for in-text citations and the list of references throughout the report.

Submit your report for this part (Part 1b) as a PDF file (independent of the other parts) in Moodle.

Part 1c: MARIE Disassembly (20 marks total)

Follow the link on Moodle to access your personalised MARIE memory screenshot for this task.

Important: Your memory screenshot is different from the one other students are working on. Only download the file while you are correctly logged into Moodle with your own student account.

Task 1: Disassemble the memory (10 marks)

Based on the memory contents, recreate the MARIE program that corresponds to your personalised memory screenshot. This is called "disassembling" the machine code, since it is the opposite operation of "assembling" the MARIE code into the binary memory contents.

For each memory cell, decode the instruction and (if applicable) the address that the memory cell is encoding. You can make the following assumptions:

- There is exactly one `Halt` instruction in the code
- Every memory location after the `Halt` instruction contains data
- Any memory location that contains the value 0 is not an instruction

Here is an example of a memory screenshot and the corresponding decoded MARIE program:

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	5000	3005	6000	9000	7000	000A	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Disassembled program:

```
Input
Add  005
Output
Jump 000
Halt
DEC  10
```

Note: You need to decode the actual instructions. E.g. for the first memory location, `HEX 5000` would not be a valid answer. The contents of all memory that follows the `Halt` instruction is considered to be data. Therefore, `DEC 10` is the correct decoding of location 5 (instead of `JnS 00A`), and `HEX 00A` would also be correct. You don't need to list all the locations containing zeros starting from address 006 (these will be filled with zeros by the assembler anyway).

Tip: You can verify that your disassembled code is correct by entering it into the MARIE simulator, assembling it, and comparing the memory contents to the screenshot you started with.

Task 2: Add labels (5 marks)

Now update the program you decoded in Task 1. Removing all hard-coded memory addresses by adding labels to replace all memory locations that are used as addresses in the program instructions. Labels should have meaningful names in the context of what the program does (i.e., not just A, B, C). For the example above, this could result in the following program:

```
MainLoop,  Input
           Add  Ten
           Output
           Jump MainLoop
           Halt
Ten,       DEC  10
```

Task 3: Add comments (5 marks)

Comment the code to enhance understanding of its functionality and improve overall readability. Comments should describe the purpose of different sections of the code. For example, if a subroutine is identified, include a comment at the beginning that explains the subroutine's purpose and specifies any arguments it may take. Ensure that the comments are well-organised and concise to facilitate easier comprehension for anyone reviewing the code.

For this part (Part 1c), you need to submit one .mas file based on the provided template, containing ONLY your final code. Please DO NOT submit separate .mas files for each subtask! The assembled code in memory should match your personalised memory screenshot.

Part 1d: MARIE Programming (22 marks)

In this task you will develop a MARIE application that draws numbers on the screen. We will break it down into steps for you.

Each task requires you to write **code** and **documentation**. On Moodle, you will find a **template** for the code. **Your submission must be based on this template**, i.e., you must incorporate your own subroutines implementations into the template. The template already includes the main program that calls the subroutines, but users can modify the code in the main program to execute different tasks.

Your code must contain readable comments and meaningful labels for your tutor / marker to understand the logic flow of your program (e.g. the purpose of a subroutine, `jump` / `skipcond` statement etc.).

Interview (Part 2): You will be required to participate in an interview between Week 7 and Week 9 (after the submission deadline), where you will demonstrate your code to your tutor. **Failure to demonstrate your code will result in a score of zero for the entire Part 1d, regardless of your submission on Moodle.** Additionally, during the interview, you will need to answer further questions about your submitted code (see below for details).

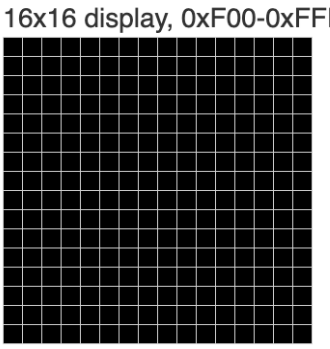
Code similarity: We use tools such as MOSS and Turnitin to check for collaboration and copying between students. If you copy parts of your code from other students, or you let them copy parts of your code, this will result in a report to the Academic Integrity team. As a result, **you may receive a penalty such as 0 marks for the entire assignment, 0 marks for the whole unit, or in severe cases (such as contract cheating), suspension or expulsion from Monash University.**

Rubric: The marking rubric on Moodle provides details for the marking. A correctly working MARIE program that covers all tasks and is well documented will receive full marks. Missing/incomplete documentation will result in a loss of up to ¼ of the task's marks.

Introduction: Bit-mapped displays

So far, the only output capability we have seen in the MARIE system is using the `Output` instruction, which will print a single 16-bit value. Many computers of course are capable of displaying arbitrary graphics, often in high resolution and great colour depth.
In the lectures on input/output systems, we have seen that one way to implement this is to ***map*** a certain location of the memory to an output device. I.e., writing to that memory location (e.g. using a `Store` instruction) causes the output to happen.
In the simplest form of graphics hardware, we can dedicate part of the RAM to be ***graphics memory***. Each memory cell corresponds to a ***pixel*** on screen, and the value in the memory cell encodes the ***colour*** of the pixel. That way, we can create arbitrary graphics by simply writing values into the memory.

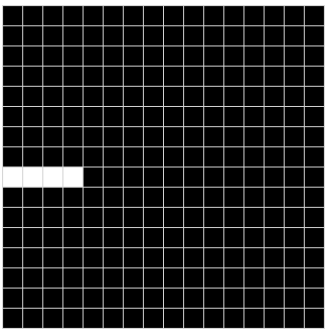
The MARIE simulator has a feature called ***Display***, which you access from the list of tabs that also shows the output log, RTL log etc:



The display shows the memory from address F00 to address FFF as a 16x16 pixel screen. The value in the memory locations represents the colour of the pixels. We will only use the colours black, represented as 0, and white, represented as FFFF. When you start the MARIE simulator and assemble your code, the memory starting from location F00 is (usually) filled with zeroes, which means that the display is black. Let's now change the contents of the memory using some `Store` instructions:

```
        Load  White
        Store 0F80
        Store 0F81
        Store 0F82
        Store 0F83
        Halt
White,  HEX   FFFF
```

After running this program, the display will look like this:



You can see that the first four pixels in the 9th row have now turned white.

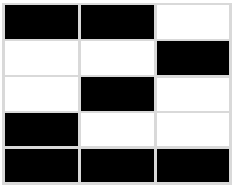
Task 1: Reset the display (4 marks)

Write a subroutine `SubResetDisplay` that turns all pixels in the graphics memory white. Remember that the graphics memory ranges from address 0F00 to address 0FFF, and that white pixels are represented by the value FFFF. Document your subroutine with comments.

Task 2: Paint a number (10 marks)

The template for this task contains data for **bitmaps** of the digits 0-9, stored at the label `DigitPixels`. Each digit consists of 3x5 pixels of data. The first 3 words are the first row of pixels, the next 3 words are the second row, and so on. For example, the digit 2 is represented as

```
0000    0000    FFFF
FFFF    FFFF    0000
FFFF    0000    FFFF
0000    FFFF    FFFF
0000    0000    0000
```

→ 

You can see the pattern here, the zeros "paint" the shape of the character 2 in black, with the background in white (FFFF).

Your task is to write a subroutine called `SubDrawDigit` that paints a digit into the graphics memory. The beginning of the subroutine should be structured as follows:

```
DigitDataAddress, HEX 0
DigitDisplayAddress, HEX 0
SubDrawDigit, HEX 0
```

In the `DigitDataAddress` argument, you will pass the address of the first pixel data for the digit you want to paint, as defined in the `DigitPixels`. In the `DigitDisplayAddress` argument, you will provide the address of the top-left corner in the graphic memory where you wish to start painting the digit. For example, to paint the digit 0, starting from the second pixel in the second row, we could use the following code:

```
            Load  DigitPixelsAddr
            Store DigitDataAddress
            Load  Display11
            Store DigitDisplayAddress
            JnS   SubDrawDigit
            Halt
Display11,  HEX   0F11
```

Note that the address 0F11 (label `Display11`) lies exactly 17 words after the start of the graphics memory. This means we're skipping the first row (16 words) and the first pixel in the second row (1 word).

Here we simply use `DigitPixelsAddr` to refer to the first character (for the digit 0). For the other characters, we would have to add a corresponding offset into the `DigitPixels` memory.

To paint a digit in your subroutine, you can follow this "recipe":

- Each digit consists of 15 pixels (3 columns x 5 rows). You need to loop through these 15 pixels, loading each one from the `DigitPixels` definition and storing it in the graphics memory.
- Your subroutine should contain two nested loops:
  - Outer Loop (Rows): Loop through each row of the digit.
  - Inner Loop (Columns): Within the outer loop, loop through each column of the current row.
    - Retrieve the corresponding pixel data from `DigitDataAddress`
    - Store the pixel data at the correct graphics memory address (e.g. `DigitDisplayAddress` plus the current row offset and current column offset).
- Once you have "copied" all 15 pixels from the `DigitPixels` definition into the graphics memory, you can exit the subroutine.

Your subroutine must include sufficient comments to help others (such as the person marking your assignment) in understanding the purpose of your code.

Task 3: Create a countdown timer (8 marks)

Your final task is to implement a subroutine `SubDigitTimer` that clears the screen and then counts down from 9 to 0, drawing those digits on the bit-mapped display using the subroutines developed in the previous tasks.

To get full marks, your code should utilise a loop that decrements a counter, calling `SubDrawDigit` based on the counter's value, rather than using a series of instructions to calls `SubDrawDigit` with each digit's address. Consider using additional subroutines to enhance the structure your code.

Additionally, it would be beneficial for the countdown to pause for a fraction of a second between digits. Users may adjust a counter to control the duration of this pause. Think of a method to achieve this, so that the countdown takes approximately 10 seconds to complete a computer. Document you approach in the code comments.

A sample output video for this task is provided for your reference:
https://drive.google.com/file/d/1VBZCMVfn-bCQsd66aIlhkFP4FTAq-2jV/view?usp=sharing

For this part (Part 1d), you need to submit one .mas file, based on the template, containing the code for all subroutines. Do not submit one .mas file *per each subtask*!

Part 2: Interview (8 marks)

You are required to demonstrate the code you submitted for Part 1d (Tasks 1–3) during an interview with your tutor (**to be conducted between Week 7 and Week 9**) after the submission deadline. Failure to explain how your code works will result in receiving 0 marks, as well as 0 marks for any individual tasks that you cannot demonstrate.

In addition, you will also be asked to modify your submitted code you in specific ways and explain the MARIE concepts you applied in the individual tasks. These additional questions add up to 8 marks for the interview.

Failure to attend the interview will result in 0 marks for the entire Part 1d and Part 2, regardless of your submission in Moodle.