Practical Assignment 5

Due 4 Oct by 23:59 **Points** 80 **Submitting** an external tool

Assessment Overview

Weighting:	80 Points (8% of course grade)
Due date:	Tuesday 4th October 11:59 pm (Start of Week 9) Gradescope open for submissions now, with full Autograder available before end of Week 7.
Task description:	Write Assembly programs to complete the tasks described below and an Assembler to convert those programs to Machine code. Doing so should help you to: • Understand how programs run at a low level. • Understand how programs efficiency can be affected at a low level. Please post your questions on Piazza or ask during your workshop.
Academic Integrity Checklist	Do ✅ Discuss/compare high level approaches ✅ Discuss/compare program output/errors ✅ Regularly submit your work as you progress Be careful ❓ Using online resources to find the solutions rather than understanding them yourself won't help you learn. Do NOT ❌ Submit code not solely authored by you. ❌ Use a public GitHub repository (use a private one instead). ❌ Post/share complete Assembly/Machine code in Piazza/Discord or elsewhere on the Internet etc. ❌ Give/show your code to others

Your Task

You've built the computer, now it's time to start programming it!
Your task for this practical assignment is to write assembly programs for the Hack machine you've built.

1. Download **this zip file** (https://myuni.adelaide.edu.au/courses/72399/files/11562488/download?download_frd=1) containing the template and test files for this assignment.
2. Complete the ASM files and Assembler as described and as outlined below.
   - Submit your work regularly to Gradescope as you progress.
   - Additional resources and help will be available during your workshop sessions.
3. Test your code and write your **own test cases**.

Testing Requirement

Low level code can be especially prone to errors.

To help you develop, understand, and debug your own code **you'll also need to write several test cases for each task**.

- These test cases will be manually reviewed after the assignment due date.
- Marks for each task may be **scaled down as much as 50%** for poor/missing testing.
- The Gradescope autograder will run your test cases and provide *some* basic feedback.
- The additional resources section below includes basic instructions and guides on writing test cases.
- We also recommend asking your workshop supervisors for advice on testing if you're unsure.

Part 1 - Basic Programs (4 points)

In this part you'll be familiarise yourself with Hack assembly by writing a basic arithmetic program.

You'll also need to write your own tests. Take a look at the sample test file provided to see how to write your own test cases.

Task 1.1 - Add and Subtract (4 points)

Write a program in Hack assembly to calculate

Complete the code in `AddSub.asm`

Inputs:

- `R1` contains the value for
- `R2` contains the value for
- `R3` contains the value for

Outputs:

- Write your final answer to `R0`

Test Cases:

- Write at least 2 test cases.
- A sample test case is provided in `AddSub00.tst`
- Each test case should be in a file named `AddSubXX.tst` where `XX` is a number starting at `01`.
- You should also submit any supporting files such as CMP files.
- Your mark for this task may be **scaled down** for poor/missing testing.

Part 2 - Conditionals & Loops (24 points)

In this part you'll be writing more complex programs that involve jumps.

Task 2.1 - Absolute Value (8 points)

Write a program in Hack assembly to calculate the **absolute value (https://en.wikipedia.org/wiki/Absolute_value)** of a given number.

Complete the code in `Abs.asm`

Inputs:

- `R1` contains the number

Outputs:

- Write your final answer to `R0`

Test Cases:

- Write at least 3 test cases.
- A sample test case is provided in `Abs00.tst`
- Each test case should be in a file named `AbsXX.tst` where `XX` is a number starting at `01`.
- You should also submit any supporting files such as CMP files.
- Your mark for this task may be **scaled down** for poor/missing testing.

Task 2.2 - Multiply (16 points)

Write a program in Hack assembly to multiply 2 numbers.

Complete the code in `Mult.asm`

Inputs:

- `R1` contains the first number
- `R2` contains the second number

Outputs:

- Write your final answer to `R0`

Test Cases:

- Write at least 5 test cases.
- A sample test case is provided in `Mult00.tst`
- Each test case should be in a file named `MultXX.tst` where `XX` is a number starting at `01`.
- You should also submit any supporting files such as CMP files.
- Your mark for this task may be **scaled down** for poor/missing testing.

Part 3 - Arrays (28 points)

It's time to apply your knowledge of Memory to work with array data structures.

Your solutions to this part will also be evaluated on efficiency; number of instructions used, with bonus points available.

Task 3.1 - Array Largest (12 points)

Write a program in Hack assembly to calculate the largest value in a given array.

Complete the code in `ArrMax.asm`

Inputs:

- `R1` contains the RAM address of the first element in the array
- `R2` contains the length of the array

Outputs:

- Write your final answer to `R0`

Test Cases:

- Write at least 5 test cases.
- A sample test case is provided in `ArrMax00.tst`
- Each test case should be in a file named `ArrMaxXX.tst` where `XX` is a number starting at `01`.
- You should also submit any supporting files such as CMP files.
- Your mark for this task may be **scaled down** for poor/missing testing.

Efficiency:

- Your code runs, but how efficient is it? Your code will be tested on a large data set to measure its performance compared to a basic solution.
- You will gain/lose as much as 2 points depending on the efficiency of your code.
- Make sure you have a working solution before trying to optimise!

Task 3.2 - Array Sort (16 points)

Write a program in Hack assembly to sort a given array **in-place** in **descending order** (largest to smallest).

You may implement any sorting algorithm but should aim for a complexity of $O(n^2)$ or better.

Complete the code in `ArrSort.asm`

Inputs:

- `R1` contains the RAM address of the first element in the array
- `R2` contains the length of the array

Outputs:

- Write your `True` (-1) to `R0` when your program finishes.
- The correctly sorted array should replace the original array in its location.

Test Cases:

- Write at least 5 test cases.
- A sample test case is provided in `ArrSort00.tst`
- Each test case should be in a file named `ArrSortXX.tst` where `XX` is a number starting at `01`.
- You should also submit any supporting files such as CMP files.
- Your mark for this task may be **scaled down** for poor/missing testing.

Efficiency:

- Your code runs, but how efficient is it? Your code will be tested on a large data set to measure its performance compared to a basic solution.
- You will gain/lose as much as 3 points depending on the efficiency of your code.
- Make sure you have a working solution before trying to optimise!

Part 4 - Assembler (24 points)

We've written programs in assembly, but do we understand how to convert those to machine code?

Using your preferred programming language (Python, C++ or Java) implement an assembler as described below.

- Template files are provided for each of these programming languages.
  - Download the Python version **HERE** (https://myuni.adelaide.edu.au/courses/72399/files/11562489/download?download_frd=1) .
  - Download the Java version **HERE** (https://myuni.adelaide.edu.au/courses/72399/files/11562490/download?download_frd=1) .
  - Download the C++ version **HERE** (https://myuni.adelaide.edu.au/courses/72399/files/11562491/download?download_frd=1) .
- You will need to complete the methods provided.
- Submit your completed source and test files in the same directory as your files from Parts 1-3.
- Only submit files for 1 programming language.

Task 4.1 - Basic Machine Code Translator (12 points)

Write code to implement the basic parsing and translation of A and C instructions.

- Use the provided template files in your preferred programming language
- Complete the following methods:
  - `doSecondPass/generateMachineCode`
  - `parseInstructionType`
  - `parseInstructionDest`
  - `parseInstructionJump`
  - `parseInstructionComp`
  - `parseSymbol`
  - `translateDest`
  - `translateJump`
  - `translateComp`
  - `translateSymbol`
- You may add methods, but do not modify the provided method signatures
- You do not need to implement the Symbol Table or L-instructions
- Submit your completed source and test files in the same directory as your files from Parts 1-3.
- Only submit files for 1 programming language.

Test Cases:

- Write at least 3 test programs.
- A sample test program is provided in `Translator100.asm`
- Each test case should be in a file named `Translator1XX.asm` where `XX` is a number starting at `01`.
- Your mark for this task may be **scaled down** for poor/missing testing.

Task 4.2 - Full 2-Pass Assembler (12 points)

Write code to implement the first pass and Symbol Table.

- Use the provided template files in your preferred programming language
- Complete the `SymbolTable` class and `doFirstPass` methods, as well as updating the `translateSymbol` method.
- You may add methods, but do not modify the provided method signatures
- Submit your completed source and test files in the same directory as your files from Parts 1-3.
- Only submit files for 1 programming language.

Test Cases:

- Write at least 3 test programs.

- A sample test program is provided in `Translator200.asm`
- Each test case should be in a file named `Translator2XX.asm` where `XX` is a number starting at `01`.
- Your mark for this task may be **scaled down** for poor/missing testing.

You're done!

Submit your work to Gradescope using the button below.

- You may submit via file upload or GitHub.
  - If using GitHub, ensure your repository is private.
- Your ASM files should either be:
  - In the root of your submission (i.e. no subdirectory)
    ~ or ~
  - In a directory named `prac5`
- Your Assembler implementation source files should be:
  - Alongside your ASM files
  - Only contain the files for 1 programming language

Be sure to submit all files with each submission.

Additional Resources

The following resources may help you complete this assignment:

- **Chapter 4 of the Text Book** for basics of programming in Hack Assembly
- **Week 7 Workshop** on Hack Assembly
- **Chapter 6 of the Text Book** for basics of how an Assembler works
- Guide to Testing and Writing Test Cases (COMING SOON)
- **Appendix 3 of the Text Book** for specification of the test language used in test cases.

This tool needs to be loaded in a new browser window

Load Practical Assignment 5 in a new window