

CSCE-312 | Spring 2021 | Project 6

Assembler and Disassembler

Project Submission: 100 points

Due Date: Wednesday, April 14th, 11:59 PM

Grading

Project Submission [100%]:

You will be graded for the correctness of your implementation of the assembler and disassembler. Your program will be tested on a number of Hack programs by comparing its outputs against the expected outputs. Partial credit will be allocated based on the number of passing test cases. See rubrics at the end of this document for further details on grading.

Deliverables & Submission

1. **Submit your code on HackerRank** (setup instructions appear at the end of this document).
If you are having trouble with HackerRank you may instead submit your code on eCampus in a zip file as specified below.
2. **Submit a compressed file FirstName-LastName-UIN.zip to e-campus containing:**
 - a. Signed Honor Code Document,
 - b. A **text file FirstName-LastName-UIN.txt** with your **HackerRank ID**.
 - c. **(ONLY if also submitting code to e-campus)** Your code files AND a readme.txt which includes any relevant instructions required to run your program.

Late Submission Policy: Refer to the Syllabus

*******PLAGIARISING CODE WILL RESULT IN SEVERE PENALTIES *******

Background

Low-level machine programs are rarely written by humans. Typically, they are generated by compilers. Yet humans can inspect the translated code and learn important lessons about how to write their high-level programs better, in a way that avoids low-level pitfalls and exploits the underlying hardware better. One of the key players in this translation process is the **assembler** -- a program designed to translate (source) code written in a *symbolic* machine language into (target) code written in *binary* machine language. This project marks an exciting landmark in our odyssey: it deals with building the first rung up the software hierarchy.

Objective

Write an **Assembler** program that translates programs written in the symbolic Hack assembly language into HACK machine code that can execute on the Hack hardware platform which you built-in Project 5.

Additionally, you will write a **disassembler** program that will reverse the above process (i.e. translates HACK machine code into HACK assembly code).

You will submit and test your code for both assembler and disassembler.

Contract

Your assembler must read an **arbitrary number of lines of HACK assembly code from stdin** (*update April 13, 2021: can also read from a file input if you are turning to e-campus*), translate them into HACK machine code, and write the translated machine code to stdout (*update April 13, 2021: can also write to a file input if you are turning to e-campus*). All provided test assembly code will be valid HACK symbolic code composed of correctly formed A-Instructions and C-Instructions; it will assemble on the built-in assembler, but it may include empty lines, whitespace, and comments.

Your disassembler must function similarly to the assembler, but in reverse order. Here you need not handle empty lines, whitespace, and comments because those are not permitted in the Hack machine code specification. Obviously, there are also no variables and labels in machine code.

Hack Assembly Language Instruction Encoding

Symbolic syntax: *dest = comp ; jump*

Binary syntax: 1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3

<i>comp</i>		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D A	D M	0	1	0	1	0	1
a=0	a=1						

<i>dest</i>	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

<i>jump</i>	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

dest = comp ; jump (both *dest* and *jump* are optional)

Use of Comments: The Hack assembly language supports **comments** in the style of C++ single-line comments. They are signified by two forward slashes ("//") and the assembler must ignore

anything after those slashes to the end of the line. For example, “@123” and “@123 // hello” are the same instruction.

Use of Whitespace: Spaces and tabs are not semantically important. Removing all spaces and tabs from a line should not change the meaning of the program. Empty lines must be omitted from the assembled output.

Symbols: The Hack assembler supports the following symbols, apart from variables and labels.

<i>Label</i>	<i>RAM address</i>	<i>(hexa)</i>
SP	0	0x0000
LCL	1	0x0001
ARG	2	0x0002
THIS	3	0x0003
THAT	4	0x0004
R0-R15	0-15	0x0000-f
SCREEN	16384	0x4000
KBD	24576	0x6000

Test Programs

UPDATE (4/13/2021):

Due to unforeseen issues with HackerRank we’ve published all the test cases on eCampus under Project 6/project_6_test_cases.zip

A brief overview of the test cases is as follows:

Assembler:

1. single A instruction
2. trivial C instructions
3. A, C instructions with empty lines and comments
4. Every C instruction
5. Built-in symbols
6. Variables
7. Variables, labels

Disassembler:

1. single A instruction
2. trivial C instructions
3. A, C instructions with empty lines and comments
4. Every C instruction

Proposed Implementation

Labs 20 and 21 discuss the implementation of an assembler and you should go read the notes. At a high level, you have three phases: Parsing the provided file, construction of a symbol table (labels, variables, etc), and translation into assembly.

Having implemented the assembler, the next step is to write a disassembler. The process is similar to what you are doing in the assembler, just in reverse. For example: instead of translating JMP into 111, you would translate 111 into JMP. You do not need to handle comments or extra whitespace (because a correct assembler will not produce them) or symbols (because assembly is a lossy process -- it is impossible to recover the symbols). The test cases are case-sensitive and you must output capital letters in your disassembled code.

Resources

The relevant reading for this project is [Chapter 6](#), lecture slides, and the Week12 lab notes (**Labs 20 and 21**). Your assembler implementation can be written in any one of the following three high-level programming languages (C++, Java, or Python). If you would like to use another language supported on HackRank you may do so, but without support from the teaching staff.

We will also curate a live FAQ document in the coming days and make it available on eCampus in Project 6 folder.

Rubric (100 points):

- A. **(60 points)** Assembler runs correctly on valid assembly programs and produces correct machine code. Test cases may include comments, empty lines, spaces, etc but all will assemble correctly on the built-in Assembler.
- B. **(40 points)** Correctly disassembles well-formed Hack machine code into valid Hack assembly code.

Submission Details:

Submission of code may be performed using HackerRank to allow for immediate grade feedback. The HackerRank contest may be found at [this link](#) or linked on eCampus. You are also required to **submit a text file on eCampus with your HackerRank ID**.

(Update: April 13 2021) If you are having difficulty with HackerRank you may submit your code in your eCampus zip submission. Please include a file **readme.txt** which includes any relevant instructions required to run your program.

HackerRank Primer:

[HackerRank](#) is a competitive programming website that allows for submission and testing in a large number of programming languages. If you don't have one already you should create an account (Sign Up on the homepage)

Once you have an account, you need to register for the [Project 6 contest](#). See additional tips on the following pages.

The banner features a dark background with a blurred image of a person and some code snippets on the right side. The text is centered and white.

CSCE 312 - Spring 2021 - Project 6

Apr 3 2021, 12:00 am CDT to Apr 15 2021, 12:00 am CDT

[Sign Up](#)


Ends in

9	2	33	43
days	hrs	mins	secs


About

Project 6 for Dr. Tyagi's CSCE 312 class of Spring 2021


Once you've registered for the contest, you can select a challenge and start working. There are two challenges that correspond with the two parts of this project -- assembler and disassembler.

 **CSCE 312 - Hack Assembler**



Success Rate: 0.00% Max Score: 60 Difficulty: Medium

[Try Again](#)

 **CSCE 312 - Hack Disassembler**

Success Rate: 0.00% Max Score: 40 Difficulty: Medium

[Try Again](#)

The important parts of the challenge page are the description (parts redacted so they fit in a screenshot) and the submission box. You are recommended to write your code on a local text editor and then upload it when you are ready. The submission box is supposed to save your code, but it can be a little finicky. You can hit “Run Code” to try your code on the sample test cases or custom input, and “Submit Code” to test your code against all of the test cases.

CSCE 312 - Hack Assembler

Problem

Submissions

Leaderboard

Discussions

Your assembler must implement the translation specification provided in the project documentation and used in Project 5.

Input Format

Variable number of lines representing a program which is valid Hack assembly (valid, in this case, meaning that it is accepted by the CPU Emulator -- not that has no comments, whitespace, etc)

Constraints

N/A

Output Format

Write ASCII 0s and 1s representing the produced machine code to stdout

Sample Input 0

```
@123
```

Sample Output 0

```
000000001111011
```

Sample Output 2

```
0000000000000001
0000000000000010
1110000010010000
```

Contest ends in 9 days

Submissions: 0

Max Score: 60

Difficulty: Medium

Rate This Challenge:

☆☆☆☆☆

More

Admin Options

[Edit Challenge](#)

[View Submissions](#)

You can see past submissions here

Select your programming language here

Current Buffer (saved locally, editable) Rust

```
1 // Enter your code here
```

You can submit code for testing here

Line: 1 Col: 1

[Upload Code as File](#) ☐ Test against custom input

Run Code

Submit Code