RMIT UNIVERSITY

C++ Programming Studio COSC2804

Assignment 2

Assessment Type	Individual assignment. Submit online via Canvas. Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums.
Due Date	Sunday 5th October 2025, 23:59am
Marks	20

Overview & Instructions

This assignment will assess learning goals in computer architecture, focusing on assembly code programming. The tasks include:

a) Writing programs in LC-3 assembly language to demonstrate your understanding of loops, branches, subroutines, two's complement, and traps.
b) Converting between hexadecimal, binary and assembly representations of an LC-3 program.
c) Simulating a computer architecture via an existing Little Computer 3 (LC-3) virtual machine, written in C++.
d) Configuring all the required development tools (Minecraft Java Edition, Visual Studio Code + LC-3 extension, laser, Git, etc.)

The assignment is to be completed individually.

1 Learning Outcomes

This assessment relates to the following learning outcomes:

- **[CLO2]:** Apply fundamentals of computer architecture, operating systems, and system deployment to the design and development of medium-sized software applications.
- **[CLO4]:** Demonstrate skills for self-directed learning, reflection, and evaluation of your own and your peers work to improve professional practice.
- **[CLO5]:** Demonstrate adherence to appropriate standards and practice of Professionalism and Ethics.

2 Preliminaries

The aim of this assignment is to hone your LC-3 programming skills, using the game *Minecraft* as a testbed. This section will get you started with creating an LC-3 development environment and understanding how to write LC-3 programs that interact with Minecraft.

Setting up the tools required for carrying out this assignment.

Setup instructions can be found in the course Canvas shell. See the module Getting Started with LC-3, which is linked on the front page.

Communicating with Minecraft via LC-3.

LC-3 is a very simple language that offers no native way of communicating with Minecraft. To get around this, we have provided a modified LC-3 virtual machine that contains additional TRAP routines for this purpose. The additional TRAPs are summarised in the table below.

Trap Vector	Alias	Function	Description
0x28	chat/CHAT	postToChat(R0)	Outputs a null terminating string starting at the address contained in R0 to the Minecraft chat.
0x29	getp/GETP	player.getTilePos() --> R0, R1, R2	Gets the position of the "tile" that the player is currently on. The x, y and z coordinates are output in registers R0, R1 and R2 respectively.
0x2A	setp/SETP	player.setTilePos(R0, R1, R2)	This function moves the player to the tile (x, y, z) = (R0, R1, R2).
0x2B	getb/GETB	getBlock(R0, R1, R2) --> R3	This function retrieves the ID of the block at tile (x, y, z) = (R0, R1, R2) and returns it to R3.
0x2C	setb/SETB	setBlock(R0, R1, R2, R3)	This function changes the ID of the block at tile (x, y, z) = (R0, R1, R2) to the value stored in R3.
0x2D	geth/GETH	getHeight(R0, R2) --> R1	This function calculates the y-position of the highest non-air block at (x, z) = (R0, R2) and returns the value to R1.
0x27	reg/REG	printRegisters()	Outputs the current register values to the console.

Notes:

- `TRAP 0x27 (printRegisters)` is provided for debugging purposes, since unlike the LC-3 web simulator shown in class, the virtual machine included with the starter code does not provide an easy way of inspecting the register values.
- Function arguments and return values are passed via the registers. For example, `TRAP 0x2D (getHeight)` assumes that the *x* and *z* arguments are passed via registers `R0` and `R2` respectively and outputs the return value to `R1`.
- You should edit and submit *.asm files as instructed below.

3 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source.
- Copyright material from the internet or databases.
- Collusion between students.

For further information on our policies and procedures, please refer to the following: https://www.rmit.edu.au/students/student-essentials/rights-and-responsibilities/academic-integrity.

We will run code similarity checks.

4 Getting Help

There are multiple venues for getting help. The first places to look are the Canvas modules and discussion forum. You are also encouraged to discuss any issues you have in class with your tutors. Please refrain from posting solutions (full or partial) to the discussion forum.

5 Marking Guide

The rubric that will be used to grade the assignment is viewable on Canvas. (See the bottom of the Assignment 2 page.)

Important note: Please do not rename any of the predefined constants in the *.asm files. Most components of the assignment will be autograded, and changing the names of the constants will mess up the autograding scripts!

6 LC3 - Programming Challenges

1) Write an LC-3 assembler program that checks whether the player is within a certain *Manhattan distance* of a specified "goal" point. **[5 Marks]**

a) Place your code in the assembly file "`manhattan_dist.asm`".
b) The assembly file should contain predefined constants, `G_X`, `G_Y` and `G_Z`, that specify the position of the goal point.
c) An additional predefined constant, `GOAL_DIST`, should specify the distance bound to be checked. You may assume that `GOAL_DIST` > 0.
d) The Manhattan distance between the player and the goal point is given by:

$$d_{manhattan} = |\text{playerPos.x} - \text{G\_X}| + |\text{playerPos.y} - \text{G\_Y}| + |\text{playerPos.z} - \text{G\_Z}|$$

e) If the following inequality is met, the program should output "The player is within distance of the goal" to Minecraft chat. Otherwise, it should output "The player is outside the goal bounds" to Minecraft chat.

$$d_{manhattan} \leq GOAL\_DIST$$

Marking Rubric:

0 Marks - Program does not compile, no outputs.
1 Mark - Program compiles **and** outputs Minecraft chat Massage. However, the answer may not be correct in some cases.
3 Marks - Program compiles **and** outputs Minecraft chat Massage. However, the answer may not be correct in at most one edge case but passes all other tests.
5 Marks - Program compiles **and** the output **fulfill all** test cases **and** use appropriate control structures (e.g., loop or branching).

2) Write an LC-3 assembler program that converts a number stored in memory into its binary representation in the Minecraft world. **[5 Marks]**

a) Place your code in the assembly file "`write_binary.asm`".
b) The number to convert is specified in the LC-3 starter file as `NUMBER_TO_CONVERT`.
c) You can assume that the number to convert will always be *non-negative*.
d) Use air (block ID #0) to represent zeroes and stone blocks (block ID #1) to represent 1s.
   i) The **least** significant bit should be written to (playerPos.x, playerPos.y, playerPos.z + 1).
   ii) The next bit should be written to (playerPos.x, playerPos.y, playerPos.z + 2) and so on (see Figure 1 for a visual explanation).
   iii) Since the word size in LC-3 is 16 bits, your program should always output 16 blocks, writing extra zeroes as air blocks if necessary.

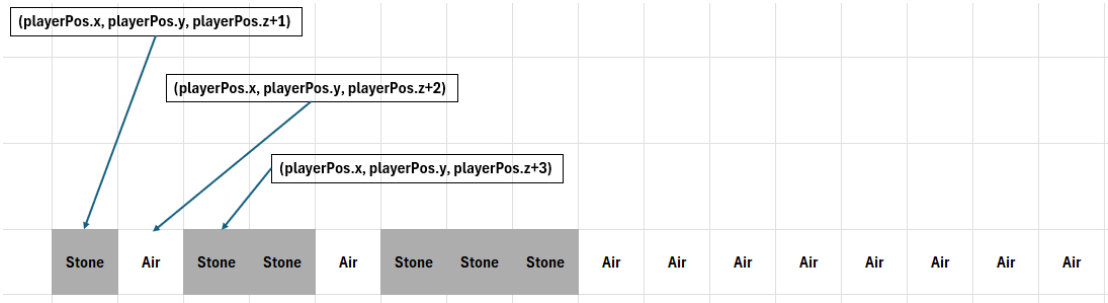

Figure 1: Illustration of how to output the binary representation in Minecraft. The number represented here is 237 = 0000000011101101. The least significant bit is closest to the player.

Marking Rubric:

0 Marks - Program does not compile, **or** no blocks are placed in the Minecraft world.
1 Marks - Program compiles **and** places blocks **and** fulfills at least one test case. e.g., `NUMBER_TO_CONVERT = 237` case.
3 Marks - Program compiles **and** places blocks correctly to pass a majority of test cases.
5 Marks - Program compiles **and** places blocks to **fulfill all** test cases.

3) Write an LC-3 program that flattens a 3x3 area centred underneath the player. **[10 Marks]**

a) Place your code in the assembly file "`terraform.asm`".
b) Let *h* denote the height of the land at the (x, z) location of the player. The height of the 3x3 area centred underneath the player should be levelled so that it all has a height of *h*.
c) Locations with a land height of less than *h* should be built up to the required height by placing dirt blocks (block ID #3). Be careful not to replace any blocks unnecessarily; only air blocks should be replaced with dirt.
d) Locations with a land height of more than *h* should be cut down to the required height by replacing non-air blocks with air blocks (block ID #0).
e) For the purposes of this problem, land height is defined as the y-position of the highest non-air block at the given x,z coordinates.
f) See Figure 2 for a visual explanation.
g) Note: under all circumstances the 3x3 area around the player should have blocks that can be walked on.
h) You can assume there will be no trees or other "overhanging" blocks.

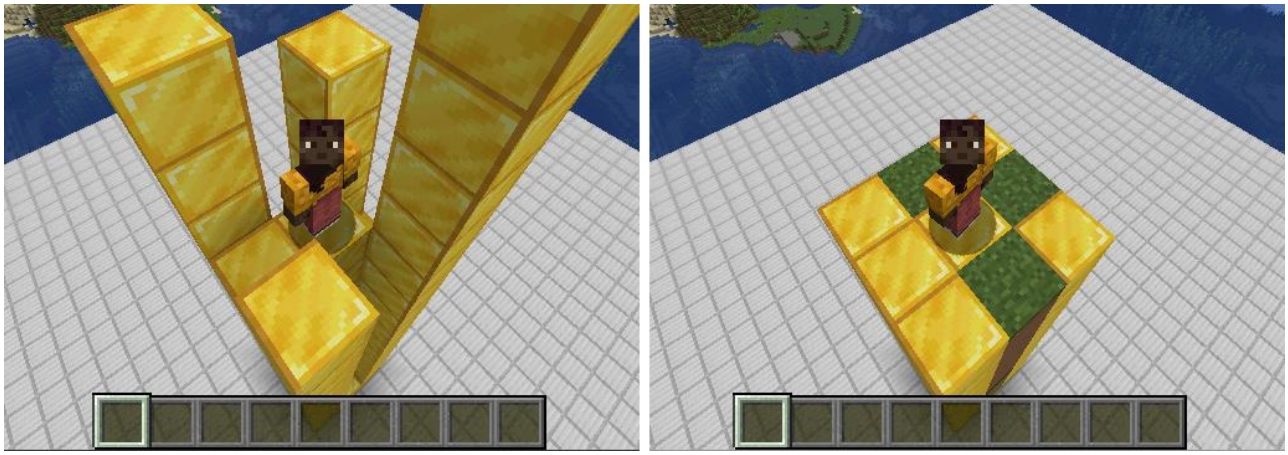

Figure 2: An example that shows how the terraform program for Problem 3 should behave.

Marking Rubric:

0 Marks - Program does not compile, and/or no output to the Minecraft world.
3 Mark - Program compiles **and** some terrain is terraformed **and** passes at least 1 test case.
5 Marks - Program compiles **and** terrain is terraformed **but** only works in some specific circumstances.
8 Marks - Program compiles **and** terrain is terraformed **but** output is slightly off.
10 Marks - Program compiles **and** the output **fulfills all** test cases.