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	Monday 19th May 2025, 09:00am
Marks	20

1 Overview

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.

2 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.

3 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	Function	Description
0x28	postToChat(R0)	Outputs a null terminating string starting at the address contained in R0 to the Minecraft chat.
0x29	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	player.setTilePos(R0, R1, R2)	This function moves the player to the tile (x, y, z) = (R0, R1, R2).
0x2B	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	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	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	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`.
- **For this assignment, you must not modify the provided virtual machine in any way**. The only files in the starter code that you should edit are the *.asm files and README.md.

4 LC-3 Programming Challenges

1. Write an LC-3 assembler program that places a triangle (i.e., isosceles triangle) of grass blocks underneath the player **[5 marks]**.
   a) Place your code in the assembly file "grass_triangle.asm".
   b) The assembly file should contain a predefined constant, `Z_DIST`, that specify the dimensions of the triangle (see Figure 1 for an illustration). Your code should work for all non-negative values of `Z_DIST`, *including* 0.
   c) The triangle should be made of grass blocks (block ID #2).
   d) The triangle should be centred 1 unit under the player tile, so that after the rectangle is created, the player is standing on top of it.

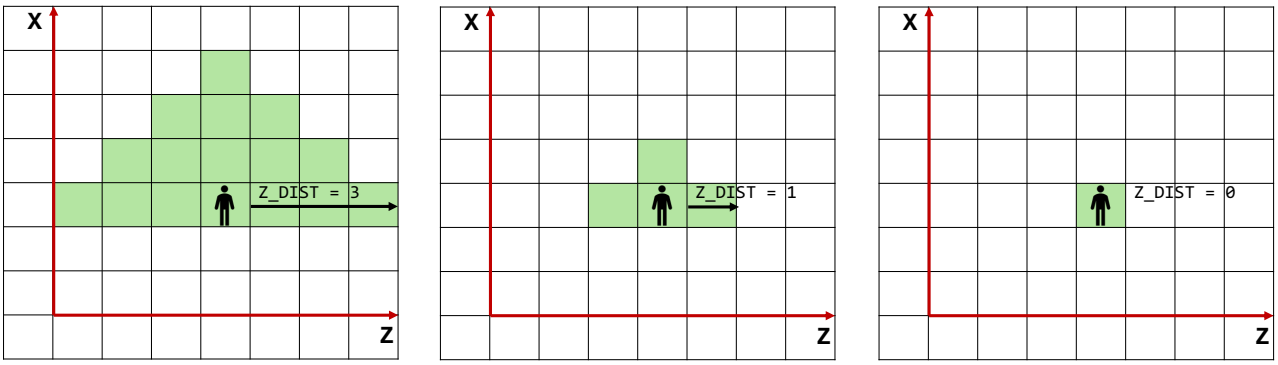


Figure 1: Illustration of the rectangles that should be built for `Z_DIST` = 3 (on the left), `Z_DIST` = 1 (middle) and `Z_DIST` = 0 (on the right).

Marking Rubric:
0 Marks – Program does not compile, **or** no blocks are placed in the Minecraft world.
1 Mark- Program compiles **and** places blocks **and** fulfills at least one test case. e.g., `Z_DIST=0` case.
2.5 Marks – Program compiles **and** places a triangle, but the *dimensions* or *placement* are slightly off (e.g., triangle shifted by 1 block).
5 Marks – Program compiles **and** places blocks to **fulfill all** test cases.

2. Write an LC-3 assembler program that builds a stairway next to the player. You must abide by the following constraints **[5 Marks]**:
   a) Place your code in the assembly file "stairway.asm".
   b) The stairway should be built from stone blocks (block ID #1).
   c) The assembly file should contain three predefined constants, `WIDTH`, `LENGTH` and `HEIGHT`, that specify the dimensions of the stairway.
   d) You can assume that the dimensions are strictly positive and that `LENGTH` >= `HEIGHT`.
   e) One way to visualise the construction of the stairway is to imagine multiple "layers" of blocks stacked on top of each another, as shown in Figure 2. Following this terminology:

      The bottom layer of the stairway should have one corner at
      `(playerPos.x + 1, playerPos.y, playerPos.z + 1)`

and another corner at
(`playerPos.x + WIDTH, playerPos.y, playerPos.z + LENGTH`).

The next layer up should have one corner at
(`playerPos.x + 1, playerPos.y + 1, playerPos.z + 2`)
and another corner at
(`playerPos.x + WIDTH, playerPos.y + 1, playerPos.z + LENGTH`).
and so on…

The top layer should have one corner at
(`playerPos.x + 1, playerPos.y + HEIGHT - 1, playerPos.z + HEIGHT`)
and another corner at
(`playerPos.x + WIDTH, playerPos.y + HEIGHT - 1, playerPos.z + LENGTH`).

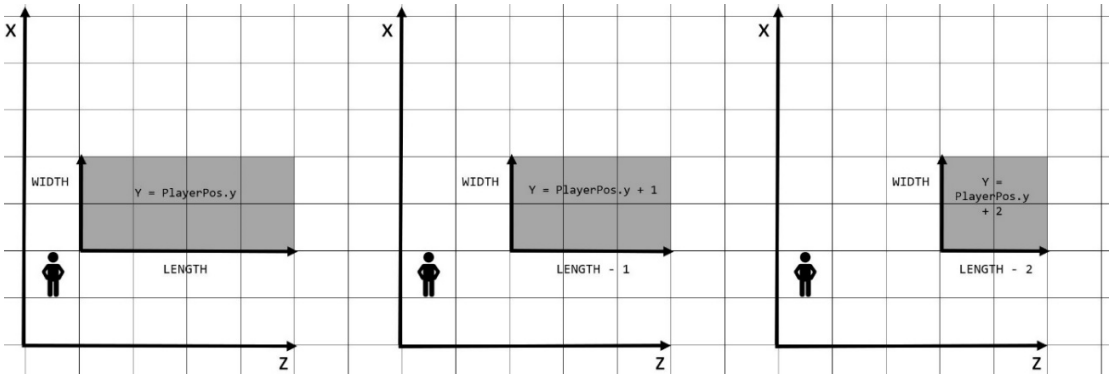


Figure 2: Illustration of how a stairway with `WIDTH` = 2, `LENGTH` = 4 and `HEIGHT` = 3, can be built from three layers of blocks stacked on top of each other. The bottom layer is shown to the left; the top layer is shown to the right.

f) See Figure 3 for an illustration of how the stairway should appear in-game.

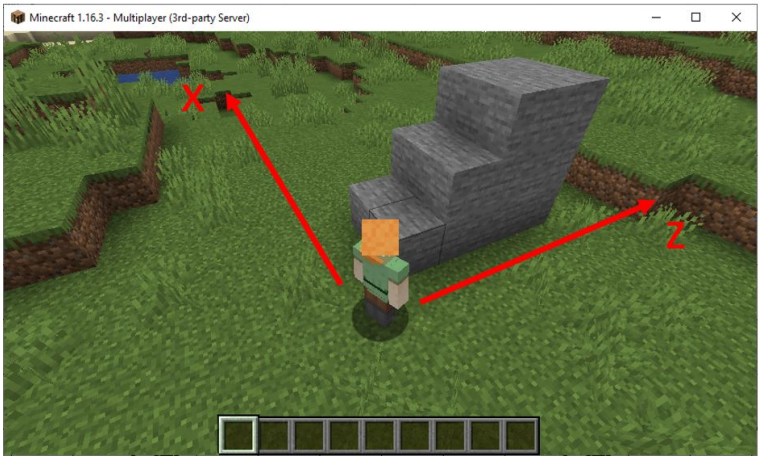


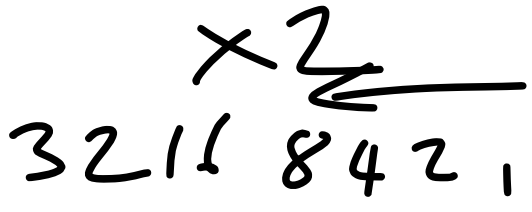
Figure 3: In-game illustration of a stairway with `WIDTH` = 2, `LENGTH` = 4 and `HEIGHT` = 3.

Marking Rubric:

0 Marks – Program does not compile, **or** no blocks are placed in the Minecraft world.
1 Mark- Program compiles **and** places blocks **and** fulfills at least one test case. e.g., `HEIGTH = 2` case.
2.5 Marks – Program compiles **and** places a staircase, but the *placement* is slightly off (e.g., staircase shifted by 1 block, or staircase shorter by 1 unit).
5 Marks – Program compiles **and** places blocks to **fulfill all** test cases.

3. Write an LC-3 assembler program that performs the following actions: retrieve two 16-bit binary words from the Minecraft world, store values in registers R5/R6, compute their arithmetic sum, and then represent and place the resulting sum as a binary word back in the Minecraft world. **[10 marks]**
   a) Place your code in the assembly file "binary_addition.asm".
   b) In the Minecraft world, 16-bit binary words are represented as follows: air (block ID #0) represents zeroes and stone blocks (block ID #1) represent 1s. The words are always placed continuously along the x direction. It is expected that the binary wors are *manually* placed in Minecraft world before the program is executed.
   c) The first number should be read from a line of 16 blocks starting at:
      `(playerPos.x + 1, playerPos.y, playerPos.z)`
      and extending to:
      `(playerPos.x + 16, playerPos.y, playerPos.z)`
   d) Store the first number in R5. Ensure that you use this specific register, since your solution will be **autograded**!
   e) The second number should be read from a line of 16 blocks starting at:
      `(playerPos.x + 1, playerPos.y, playerPos.z + 1)`
      and extending to:
      `(playerPos.x + 16, playerPos.y, playerPos.z + 1)`
   f) Store the first number in R6.
   g) Store the summation results in R7 and `printRegisters()`.
      h) The result should be written to a line of 16 blocks in Minecraft world starting at:
         `(playerPos.x + 1, playerPos.y, playerPos.z + 2)`
         and extending to:
         `(playerPos.x + 16, playerPos.y, playerPos.z + 2)`

   See Figure 4 for a visual explanation.



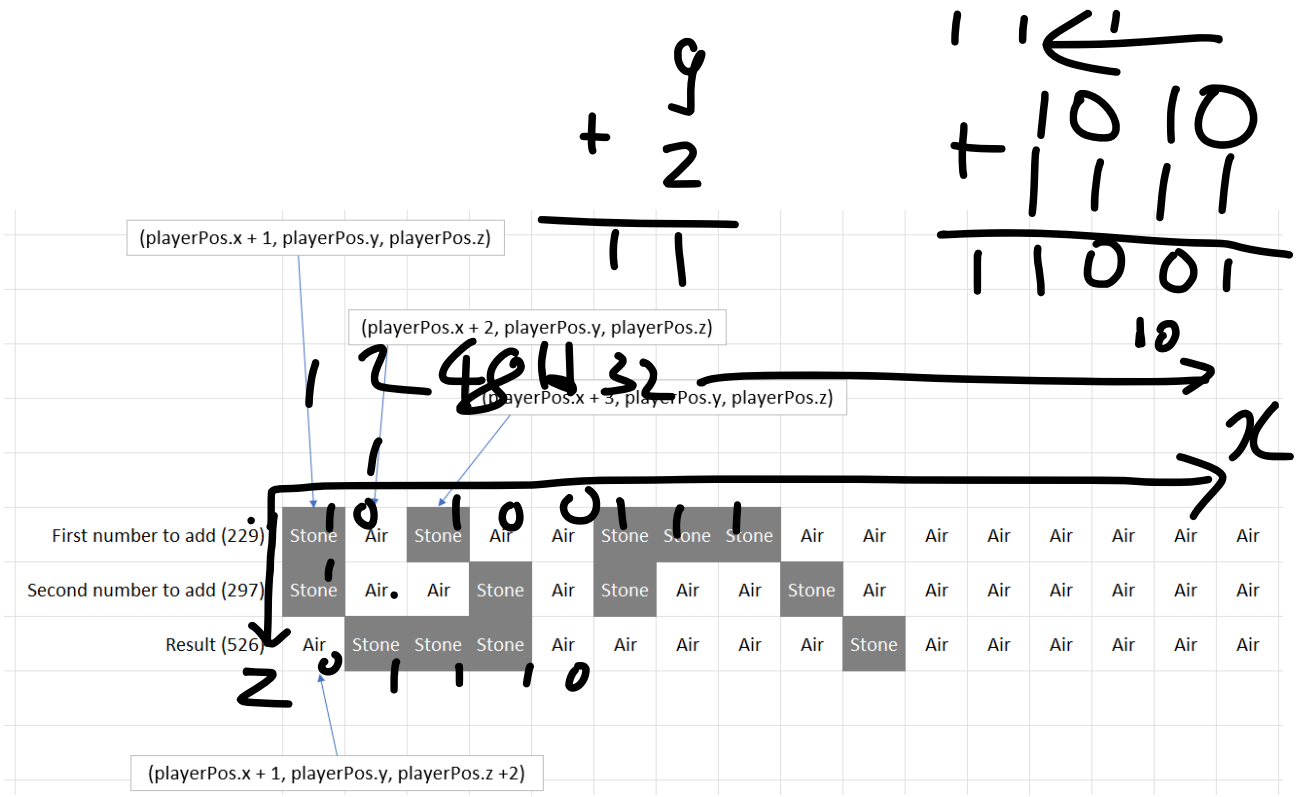


Figure 4: Illustration of how the sum 229 + 297 = 526 should appear in the Minecraft world for Problem 12.

Marking Rubric:

0 Marks – Program does not compile, **or** no block placement in the Minecraft world.

3 Marks- Program compiles **and** reading binary words are correct (i.e., correct number in registers R5/R6).

6 Marks – Program compiles **and** reading binary word is correct **and** summation is correct (i.e., correct sum in R7).

8 Marks – Program compiles **and** places blocks to **fulfill all** test cases, but register values are not correct.

10 Marks – Program compiles **and** places blocks to **fulfill all** test cases.

5 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.

6 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.

7 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!