

Getting Started with LC-3

Assignment 1&2 focuses on assembly programming using the educational language LC-3. This page contains links to valuable resources and explains the setup steps required to write LC-3 programs that communicate with Minecraft.

Reference manuals & textbooks

The key references that we'll use in class are:

- [LC-3 Assembly Language - A Manual](#)

This booklet contains many lab exercises that we will study in class. If you can solve all these exercises, you will be well set up to succeed in Assignment 2. The manual does not contain any solutions, but we will periodically release solutions sometime after the problems are shown in class (to ensure everyone makes a genuine attempt first!)

- [The LC-3 ISA](#) ↓

This book excerpt explains precisely how the LC-3 instructions are encoded as 16-bit binary words. The design of those instructions is what we define as an ISA ([Instruction Set Architecture](#) ↗).

- [Introduction to Computing Systems : from Bits and Gates to C and Beyond](#) ↗

This book provides additional, in-depth information about LC-3 and computer architecture.

Web simulator

When starting to learn LC-3, we recommend using the following online simulator to develop your code:

<https://wchargin.com/lc3web> ↗

We will use it first in [Week 1 - Studio Class 2](#), providing some exercises to help you familiarise yourself with it and throughout our investigation of LC-3 in the weeks to follow. Some of the main advantages of this tool are:

- It allows you to step through programs one instruction at a time.
- It is easy to see what the registers and memory locations currently hold, and it is easy to modify these values by clicking on them.
- You can hover over hexadecimal values to see their decimal equivalent.

Communicating with Minecraft

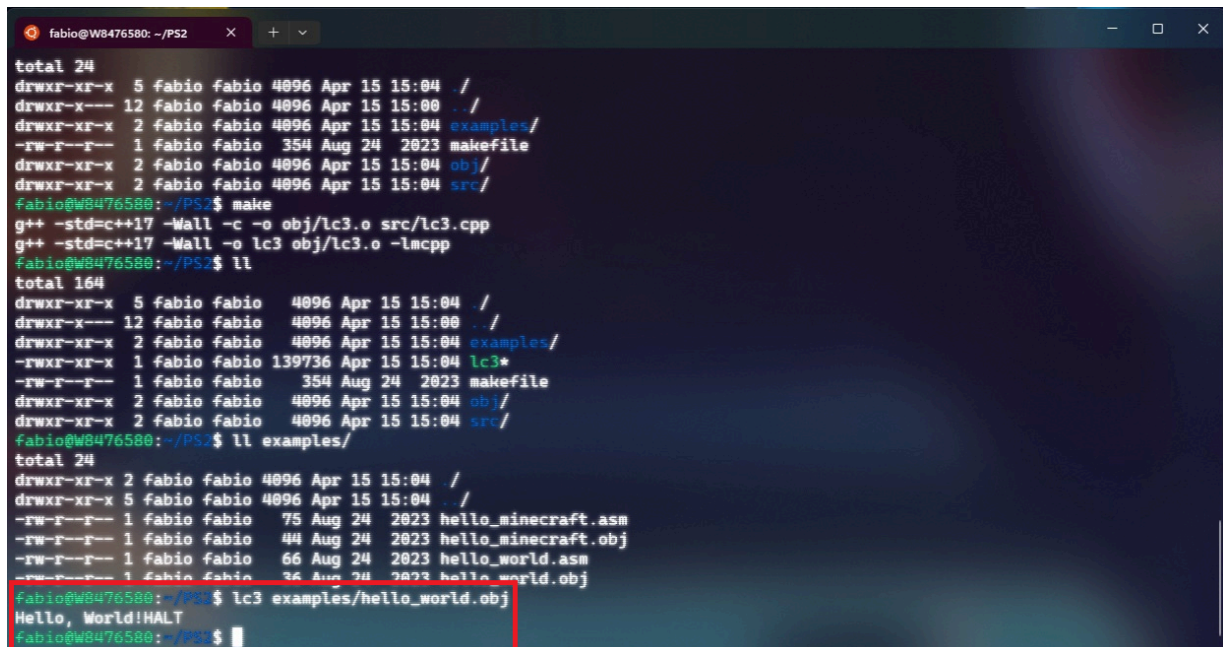
Unfortunately, the above web simulator won't allow you to interact with Minecraft. To do this, you'll need to use an LC-3 "virtual machine" (VM) that we have modified for this purpose.

To get started with the VM:

- Open a terminal window (a WSL terminal if you are a Windows user) and run the below commands.

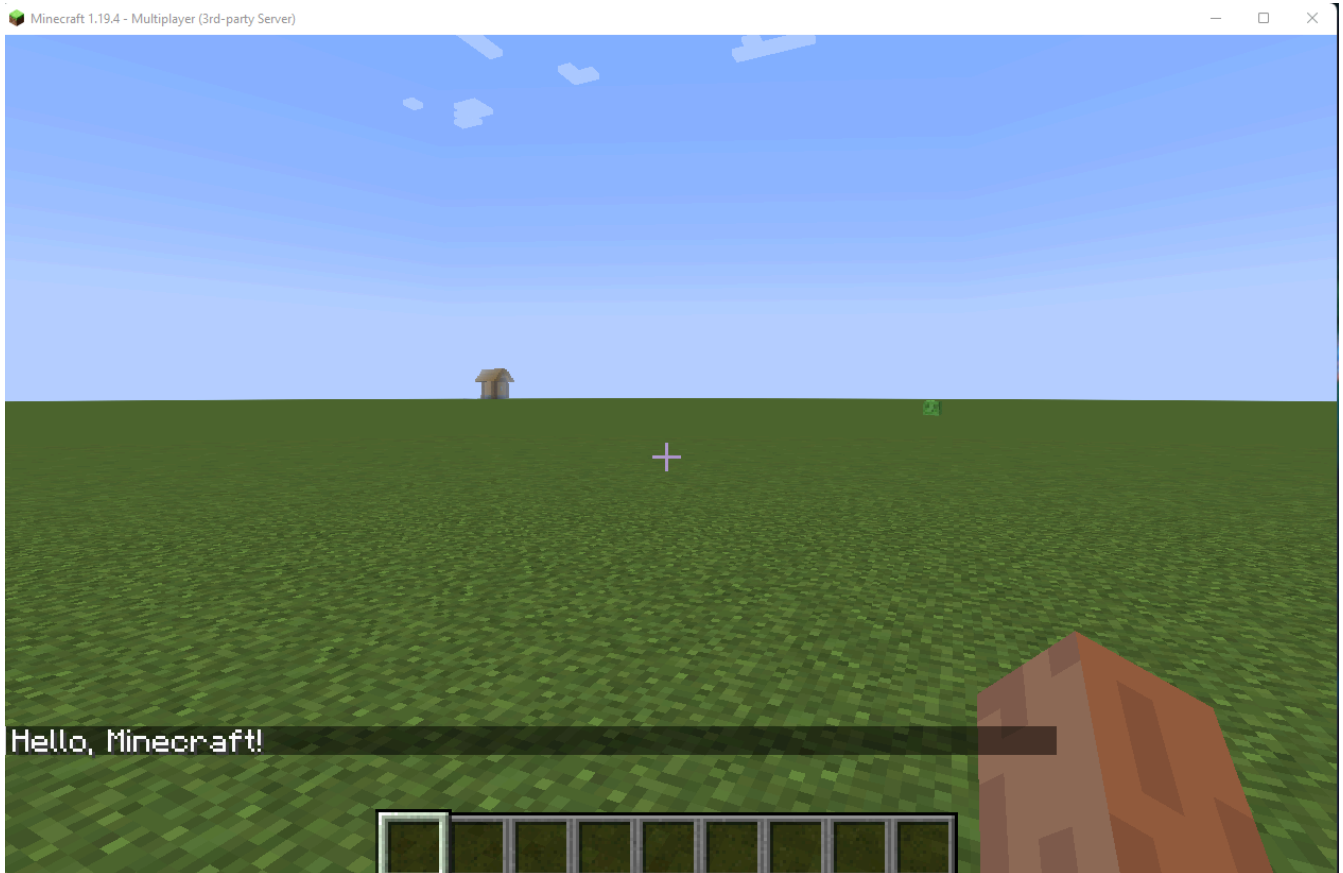
```
cd ~ && mkdir PS2
cd PS2
git clone https://github.com/rozukey/lc3-vm-mcpp.git
cd lc3-vm-mcpp && make
sudo make install
```

- An assembled "Hello, World!" program is included with the download. To run it, use ``lc3 examples/hello_world.obj``.
- The program should output "Hello, World!HALT":



```
fabio@W8476580: ~/PS2
total 24
drwxr-xr-x  5 fabio fabio 4096 Apr 15 15:04 ./
drwxr-xr-x 12 fabio fabio 4096 Apr 15 15:00 ../
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 examples/
-rw-r--r--  1 fabio fabio 354 Aug 24 2023 makefile
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 obj/
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 src/
fabio@W8476580:~/PS2$ make
g++ -std=c++17 -Wall -c -o obj/lc3.o src/lc3.cpp
g++ -std=c++17 -Wall -o lc3 obj/lc3.o -lmcpp
fabio@W8476580:~/PS2$ ll
total 164
drwxr-xr-x  5 fabio fabio 4096 Apr 15 15:04 ./
drwxr-xr-x 12 fabio fabio 4096 Apr 15 15:00 ../
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 examples/
-rwxr-xr-x  1 fabio fabio 139736 Apr 15 15:04 lc3*
-rw-r--r--  1 fabio fabio 354 Aug 24 2023 makefile
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 obj/
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 src/
fabio@W8476580:~/PS2$ ll examples/
total 24
drwxr-xr-x  2 fabio fabio 4096 Apr 15 15:04 ./
drwxr-xr-x  5 fabio fabio 4096 Apr 15 15:04 ../
-rw-r--r--  1 fabio fabio 75 Aug 24 2023 hello_minecraft.asm
-rw-r--r--  1 fabio fabio 44 Aug 24 2023 hello_minecraft.obj
-rw-r--r--  1 fabio fabio 66 Aug 24 2023 hello_world.asm
-rw-r--r--  1 fabio fabio 36 Aug 24 2023 hello_world.obj
fabio@W8476580:~/PS2$ lc3 examples/hello_world.obj
Hello, World!HALT
fabio@W8476580:~/PS2$
```

- To test the Minecraft functionality, launch a local Minecraft server and connect to it, as explained in the module [Getting Started with Minecraft++ and ELCI](#).
- Once you've loaded into the game world, go back to the terminal window at the VM location and run ``lc3 examples/hello_minecraft.obj``.
- If everything is successful, you should see "Hello, Minecraft!" appear in the game chat.



Adding the LC3 Assembly extension to Visual Studio Code

We will use Visual Studio code to write LC-3 programs communicating with Minecraft. Most of you will already have it installed, but you can download it using this link if you don't.

For LC-3 syntax highlighting, we recommend installing the "LC3 Assembly" extension, which you can get [here](#) ➞.

NOTE: Windows users should install VS Code locally but develop in WSL (see [these instructions](#) ➞).

Installing laser (for assembling LC-3 programs)

To compile our assembly code into executable .obj files, like the hello_world and hello_minecraft programs tested above, we will use a tool called "laser".


To install laser, run the below commands in your Linux/Mac terminal:

```
cd ~ && mkdir PS2
cd PS2
git clone https://github.com/rozukke/laser-mcpp.git
cd laser-mcpp/src
make
sudo make install
```

To try out laser, create a new file called "hello_world.asm" and open it in VS Code.

Now paste in the following:

```
.orig x3000
lea r0, HW
puts
halt
HW .stringz "Hello, Minecraft!"
.end
```

When you build it ([instructions](#) ) the output in the terminal should look something like this:

```
laser -a hello_world.asm
```

```
Assembling "/lc3_vm/examples/hello_world.asm"...
0 error(s) and 0 warning(s) in pass one
0 error(s) and 0 warning(s) in pass two
Done!
```

```
1 job processed in 00:00:001
1 successful, 0 failed
```

The directory containing the `hello_world.asm` file should also contain other files (`hello_world.bin`, `hello_world.hex`, etc.). The most important one for our purposes is the `.obj` file, the binary executable file the VM can run.

An explanation of the Minecraft TRAPs

The virtual machine you downloaded above has been augmented with additional "TRAP" subroutines, allowing LC-3 programs to communicate with Minecraft. To see this in action, modify the `"hello_world.asm"` file that you created above so that it reads as follows:

```
.orig x3000
lea r0, HW
trap 0x28
halt
HW .stringz "Hello, Minecraft!"
.end
```

Note that the `PUTS` command from earlier has been replaced with `TRAP 0x28`, which calls the `postToChat()` function in the `mcpp` C++ API.

Any time you assemble an LC-3 program that contains a raw trap call, you'll get a warning from laser:

```
Assembling "/lc3_vm/examples/hello_minecraft.asm"...
0 error(s) and 0 warning(s) in pass one
Warning (hello_minecraft.asm line 3): '0x28' is not a predefined trap routine
```

0 error(s) and 1 warning(s) in pass two
Done!

The above is fine -- it's just a warning that TRAP 0x28 is not a "standard" TRAP routine.

IMPORTANT: PLEASE DO NOT USE RAW TRAP CODES! There are aliases available for all of the Minecraft related traps so that you don't have to look at the below table too often! This program works equally well and is much more readable:

```
.orig x3000
lea r0, HW
chat
halt
HW .stringz "Hello, Minecraft!"
.end
```

HANDY TIP: There are small help sections available for both **laser** and the **lc3 VM**. They contain a few very useful features, such as debug files for **laser** and different output formats (signed, binary, hex) for the VM. Check them out with *--help* or *-h*.

Lastly, note that to run LC-3 programs that interact with Minecraft, you may need to launch and join a Spigot server. Some commands will work without joining the server, but anything player-related will not.

The table below shows the full set of Minecraft TRAP routines that are built into the VM, along with their aliases:

Trap Vector (DO NOT USE)	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 block's ID 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()	This function is provided purely for debugging since, unlike the LC-3 web simulator, the VM included with the assignments does not provide an easy way to inspect the current register values.