In a nutshell: A critically important aspect of building a new computer system is designing the low-level *machine language* , or *instruction set* , with which the computer can be instructed to do various things. As it turns out, this can be done before the computer itself is actually built. For example, we can write a Java program that emulates the yet-to-be-built computer, and then use it to emulate the execution of programs written in the new machine language. Such experiments can give us a good appreciation of the bare bone "look and feel" of the new computer, and lead to decisions that may well change and improve both the hardware and the language designs. Taking a similar approach, in this module we assume that the Hack computer and machine language have been built, and write some low-level programs using the Hack machine language. We will then use a supplied CPU Emulator (a computer program) to test and execute our programs. This experience will give you a taste of low-level programming, as well as a solid hands-on overview of the Hack computer platform.

Key concepts: op codes, mnemonics, binary machine language, symbolic machine language, assembly, low-level arithmetic, logical, addressing, branching, and I/O commands, CPU emulation, low-level programming.

WATCH:

- Unit 4.1: Machine Languages: Overview
- Unit 4.2: Machine Languages: Elements
- Unit 4.3: The Hack Computer and Machine Language
- Unit 4.4: Hack Language Specification
- Unit 4.5: Input / Output
- Unit 4.6: Hack Programming, Part 1
- Unit 4.7: Hack Programming, Part 2
- Unit 4.8: Hack Programming, Part 3
- Unit 4.9: Project 4 Overview
- Unit 4.10: Perspectives

DO:

- Project 4: Machine Language

- **Submission instructions:** note that the project 4 files are located in two folders named 'mult' and 'fill'. You should put both files (Mult.asm and Fill.asm) together inside a zip file, and not inside any folders.

- The test provided for Fill on the software suite is a visual one - you see whether the screen turns black when a key is pressed, and turns back to white when it is unpressed. When you submit your file we run an automatic test to check that when changing the value of the keyboard register your program changes the values of the screen memory. We provide the automatic test and compare file on the project 4 submission page . You should run it the same way you did with the tests provided in the software suite: use the supplied Assembler to assemble your Fill.asm into Fill.hack. Then load Fill.hack in the CPU emulator, this time with the FillAutomatic.tst test.

- If you are taking the course as an auditor, you can check your work yourself, using the tests described here . If you are taking the certificate option, submit your project zip file here .

GET HELP:

Module 4 Discussion Forum