| Homework 6-1: Vic Programming Background This exercise practices what was learned in [lecture 6-1](https://drive.google.com/open?id=134UlvR8aydmzc8IYNmUBQoPqiNWS6JtN&usp=drive_fs) (*Computer Fundamentals, Part* I). The purpose of Lecture 6-1, along with Homework 6-1, is to get a hands-on understanding of how computers work, and practice programming using a low-level language.  **This is a self-study exercise**. You don’t have to submit it, and it won’t be graded. That said, you are advised to complete this exercise, for several reasons. First, understanding how computers work is an important part of your CS education. Second, in a later homework in the course you will implement this simple computer in Java, and therefore you must become familiar with its architecture and operations. Finally, all the material presented in Lecture 6-1 and in this homework will be covered in the final examination. The Visual Computer (Vic) Vic is implemented as a web application, meaning that you run it from a browser. It allows you to play with exactly the same computer platform that was presented in [Lecture 6-1](https://drive.google.com/open?id=134UlvR8aydmzc8IYNmUBQoPqiNWS6JtN&usp=drive_fs). You can load this computer with instructions written in the Vic machine language (like 800, 490, 191,...), as well with some input values, and then sit back and watch how the loaded code executes on the machine. The web application also features an **Assembler**, which is a program that allows translating symbolic Vic instructions (like read, store x, add y) into executable instructions (800, 490, 191, …). The assembler is an essential tool: Writing programs in executable code is a pain, and writing programs in symbolic code is much easier.  **Getting started, Part I:** Go to the [Visual Computer](https://faculty.runi.ac.il/vic/software/computer/), and start playing with the simulated computer. Start by clicking HELP, and enter a few numbers into the *Input*. Next, enter some executable instructions into the *Instruction Register*, and execute them (by hitting ENTER). For example, you can read a number or two, store the read values to some memory cells, load / add them into the *Data Register*, and so on. Next, enter some instructions into the *Main Memory*, and execute them by using the “play” controls at the bottom left of the Vic window. Next, use the “Load” button to load either the add2.vic program or the max2.vic program, which you can find in the supplied “vic code.zip” file. Finally, execute these programs (one at a time…), and make sure that you try all the available animation and program flow options. The names of the “Animation” menu items are a bit confusing, but their operations are very clear. Note that the execution speed can be controlled using the speed slider.  **Getting started, Part II:** Once you feel comfortable with the computer, you are ready to start writing programs using the Vic symbolic language. Start by invoking (“מילה יפה ל- ״התחל להריץ) the *Vic Assembler*. To do so, click the “Vic Assembler” field at the bottom right of the Vic window (if you don’t see this button, you may have to scroll or resize the Vic window). At this point you will have two windows open: The Vic computer window and the Vic Assembler window. It is convenient, although not required, to put them side by side, as shown below.    Once you have the two windows open, start playing with them. Write some symbolic instructions in the Assembler’s Source Code window, and then translate, load, and execute them. Writing symbolic programs You can write symbolic code directly into the Assembler’s Source Code window, but use this option only for getting started. You can easily lose the written code (if the window is closed), and the “save” button at the bottom left of the Assembler window does not work (sorry about this). So, write your symbolic programs using a text editor like VS Code, Notepad, Sublime, etc. (but not Word!), and save them on your PC using file names like *progName*.asm. Then load the *progName*.asm file into the Assembler window (or, simply, copy-paste your symbolic code into the window), and continue with the Translate-Load-Execute routine. If you have to debug your code, do it in the text editor (not in the Assembler window!), then save, load or copy-paste, etc. Programs 1. Write a Vic program (readWritePos.asm) that reads a sequence of numbers that ends with a 0. For each number in the series, if the number is greater than 0, the program writes the number. For example, When applied to the input -3, 2, 5, -­1, 7, -­2, 8, 0, the program outputs 2, 5, 7, 8.  2. Write a Vic program (find.asm) that reads a non-zero number (say *x*), followed by a sequence of one or more non-zero numbers that ends with a zero. If *x* appears in the sequence (once or more), the program writes its first location in the sequence; otherwise, the program writes ­-1. For example, if the input is 7, 2, 1, 7, 2, 5, 0, the program outputs 3 (the first element in the sequence is considered 1). If the input is 5, 3, 1, ­3, 15, 7, 0, the program outputs ­-1.  3. Write a program (odd.asm) that reads a single positive number from the input. If the number is odd, the program writes 1; if the number is even, the program writes 0. For example, if the number is 5, the program outputs 1; if the number is 8, the program outputs 0.  4. Write a program (multiply.asm) that reads two non-­negative numbers, and writes their product. For example, if the input is 3, 12, the program outputs 36.  5. Write a program (getMem.vic) that reads two numbers: the constant 300, followed by a number (say *x*) between 0 and 99. The program writes the value of the memory register whose address is *x*. For example, if *x* = 90, the program writes the value of M[90], i.e. the value of the memory cell whose address is 90. Tips: (i) this is a tricky program; (ii) unlike the other programs in this homework, this program must be written using executable instructions, *not symbolic instructions*: each command must be written as a 3-­digit number; (iii) the first input, the constant 300, is needed in order to write this program.  **(No) Submission:** This is a self-study exercise. You don’t have to submit it, and it won’t be graded. It’s for your own entertainment and education. |
| --- |