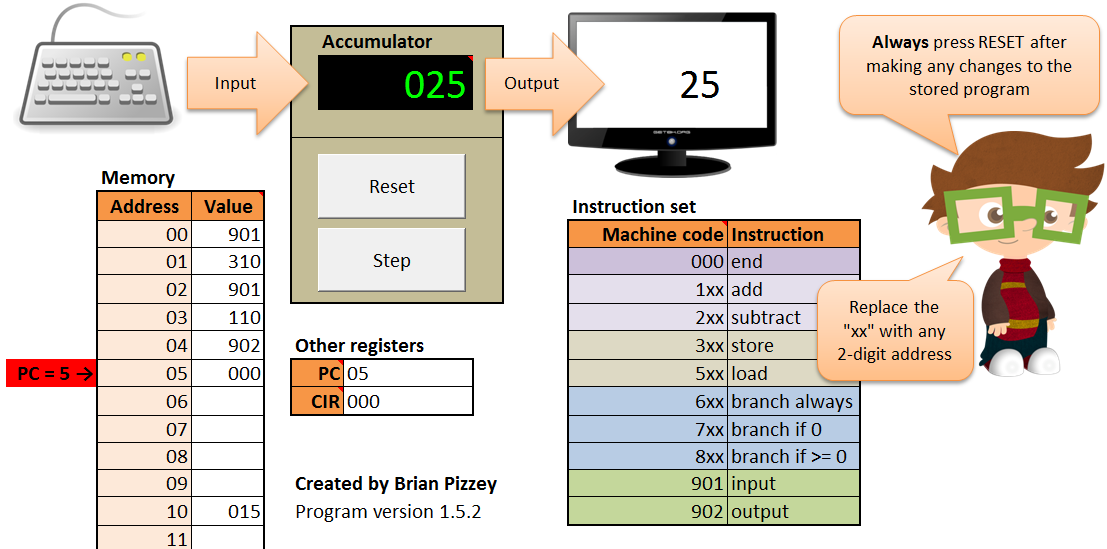
**LMC Machine Code Tutorial**

The Little Man Computer is a tool designed to demonstrate in a simplified way how real computers process instructions from memory to do work.

I provide a few different tools that simulate a LMC. Your primary one should be the Excel workbook (LMC152.xlsm). It looks like the picture below. (If you do not have access to Microsoft Excel, see Alternate Environment below).



The memory area shows the computer program and any data that are stored in memory.

The Accumulator shows the value currently stored in the accumulator (last value the processor's ALU worked on). The Reset and Step buttons allow you to run the program 1 step at a time or reset it back to the start.

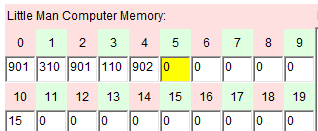
Below the Accumulator, you can see the PC (program counter) and CIR (current instruction register). They always show what memory address has the current instruction and what that instruction is.

Notice that all the memory addresses, and values we are using are in decimal. This is simply to make the program easier for us to use - in a real computer, the values would all be in binary.

**Alternate Environment:**If you do not have access to Excel, you can use this simulator:

<http://www.yorku.ca/sychen/research/LMC/LittleMan.html>

You can type the same programs into it, the memory locations are just arranged in a grid instead of a column. Here is what the first program looks like typed in:



**First Run**

The program when you first open the sheet will ask for two numbers to be input, add them and show the result. Run it step by step and read along to learn more about what is happening:

|  |  |
| --- | --- |
| The PC starts at instruction 0. That instruction is 901. Which stands for INPUT - get a value from input:  Press Step and type in the value 15. |  |
| The 15 is now stored in the accumulator.  The PC has been advanced one step - it shows that the new instruction is 310.  3xx stands for STORE - write the accumulator's value to memory. The xx is the memory location to write the value to.  In this case the location is 10 - we are going to store the value 15 at memory location 10. |  |
| Hit Step again.  Look down in memory to location 10. The value 15 is stored there now.  The PC is now 2 - we are about to do 901 - another INPUT. |  |
| Hit Step and enter the value 12.  The input replaces the value in the accumulator (that is why we needed to do a STORE - to remember the first number we got from input).  The PC is now 3 and the instruction is 110. 1xx means ADD - take the value stored at location xx in memory and add it to the accumulator. |  |
| Press Step.  The 15 we had stored at location 10 of memory is added to the 12 in the accumulator to make 27.  Now the program count is 4, which is instruction 902 - OUTPUT. |  |
| Press Step.  The value in the accumulator is output to the screen.  Now the PC is 5 - the next instruction is 000 - END.  Press Step one more time to see the program terminate. |  |

**Modification 1**

|  |  |
| --- | --- |
| Modify instruction 01 to be 302. Now it says "STORE at location 02". Can you see the problem that will cause?  Step your program, enter a number, then step again… |  |
| Here is where we are. About to run instruction 015. Wait, there is no instruction 15!!! The computer is going to crash and end this program.  Because memory and data are stored in the same memory, we have to be careful not to overwrite the program with data we store as it runs. In this case, our store instruction (302) wrote over the code that was in location 2, breaking the program. |  |

**Branch**

|  |  |
| --- | --- |
| Branch instructions tell the computer to maybe skip to a different part of the instructions.  Type in the program shown to the right (change the memory values to match what I have).  Start stepping and enter the number 10… |  |
| Now we are about to execute 806. 8xx is BRANCH if >= 0. It says go to location xx if the accumulator is greater than or equal to 0.  Since the accumulator is, we will jump to instruction 06… |  |
| We skip right over instructions 3-5 and will now output the value we input and end the program. |  |
| This time, when you get to the branch instruction, the accumulator's value is -10. That is not >= 0 (greater than or equal to).  So we are not going to branch. |  |
| 511 says to LOAD the value at 11. Since nothing is there, the accumulator is now set to 0 (empty).  The next instruction, 210, says to subtract the value we stored in memory location 10 from the accumulator. (2xx is subtract location xx from accumulator). |  |
| Hit step.  The accumulator had 0, then we subtracted -10 (value in location 10).  0 - (-10) = 10… we just made the negative input positive. This is an absolute value program!  The next steps will store the new value… |  |
| Now we are at step 6. From here on out, the program works just like when we entered a positive number. |  |

**Repeat With a Branch**

|  |  |
| --- | --- |
| Branch instructions can send a program back to repeat earlier instructions. This forces the computer to **loop** - to repeatedly execute the same instructions.  Type this program in and try it out - run it through, enter 10 for the input and see what it prints.  As you run it, here are a couple hints:   * Location 15 is always just the number 1 * Location 16 is a counter - watch how it changes *We repeatedly subtract 15 from it - what will that do?* * Location 17 stores the input * Location 18 is where we build up an answer * Right before we check the branch (instruction 8), we load location 16. That is our counter. The counter determines if we repeat the instructions 2-8 again or not.   Then run it again with 20 for the input. **You will have to reset locations 15-18 before rerunning the program.**  Watch it run, then read on…  **TO RERUN THIS PROGRAM, YOU MUST MAKE SURE TO RESET LOCATIONS 15-18 TO THEIR STARTING VALUES.** |  |
| The program multiplies your input by 5. It does this by adding up five copies of the input. Follow along with this English description step by step as it runs.  Here is what our code is doing:   1. Get input 2. Store input in #17 3. Load #18 (starts at 0, will hold our answer) 4. Add #17 to what we loaded from #18 (add one copy of input) 5. Load #16 (our counter) - it starts at 4 6. Subtract #15 (the value 1) from the counter we loaded - decreases counter by 1 7. Store the new counter value back in #16 8. If counter is >= 0 we go back to step 2. Counter will have to go **4,3,2,1,0**,-1 When it hits -1 we will get past this line. Notice that the 4,3,2,1,0 is five times before we leave. That is why we end up adding 5 copies of what we started with. 9. Output our answer from location #18 where we have built it up 10. End | |