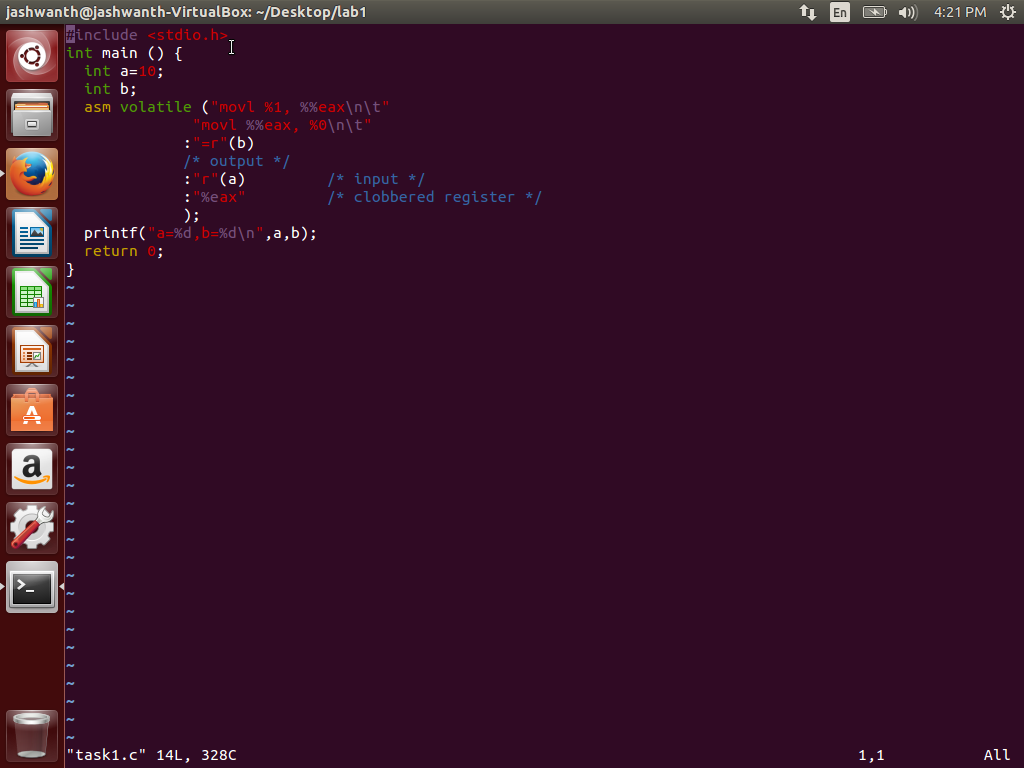
Jashwanth Reddy Gangula

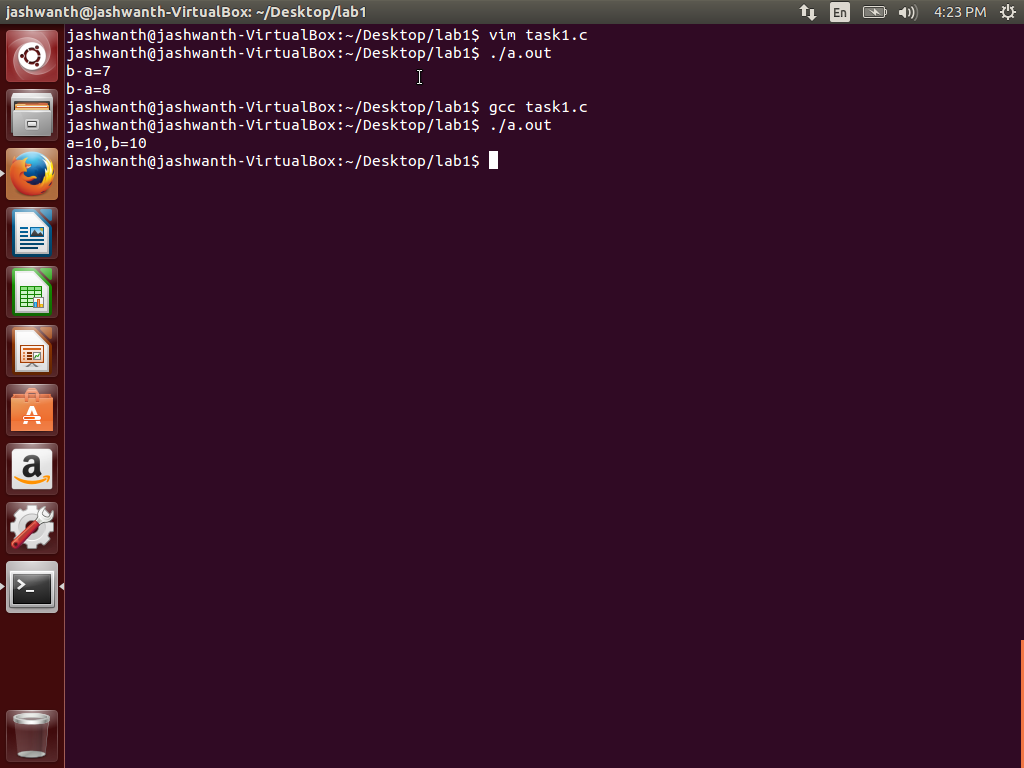
SUID: 646254141

Email: [jgangula@syr.edu](mailto:jgangula@syr.edu)

Lab 1

**Task1**

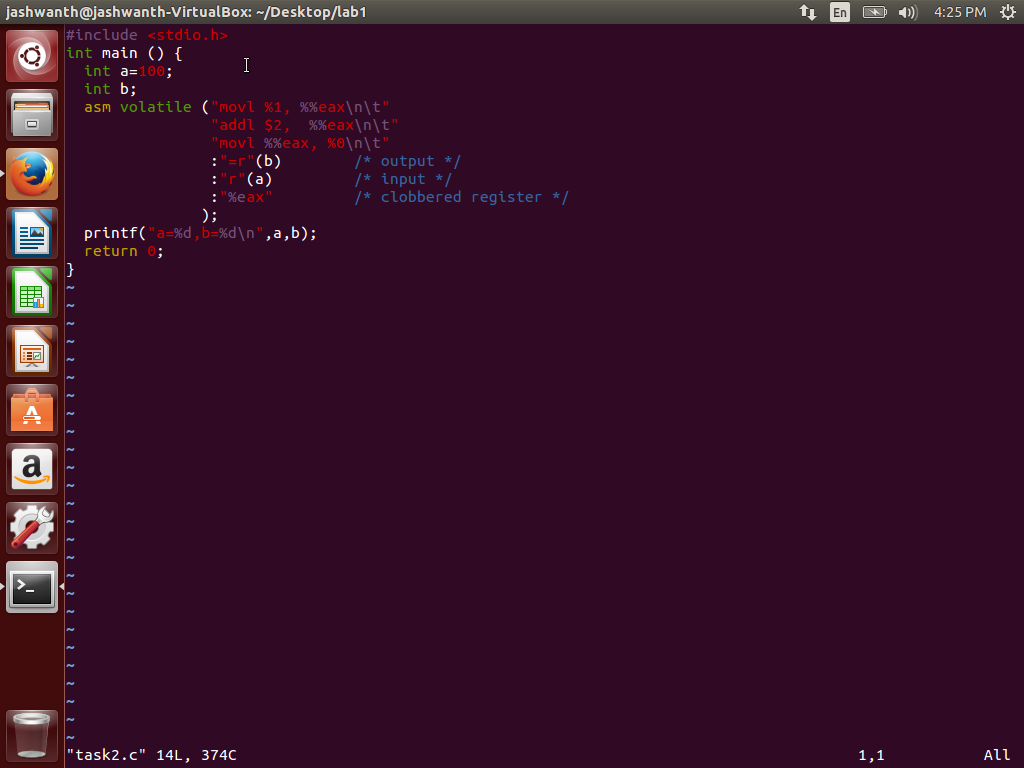
**Output:**



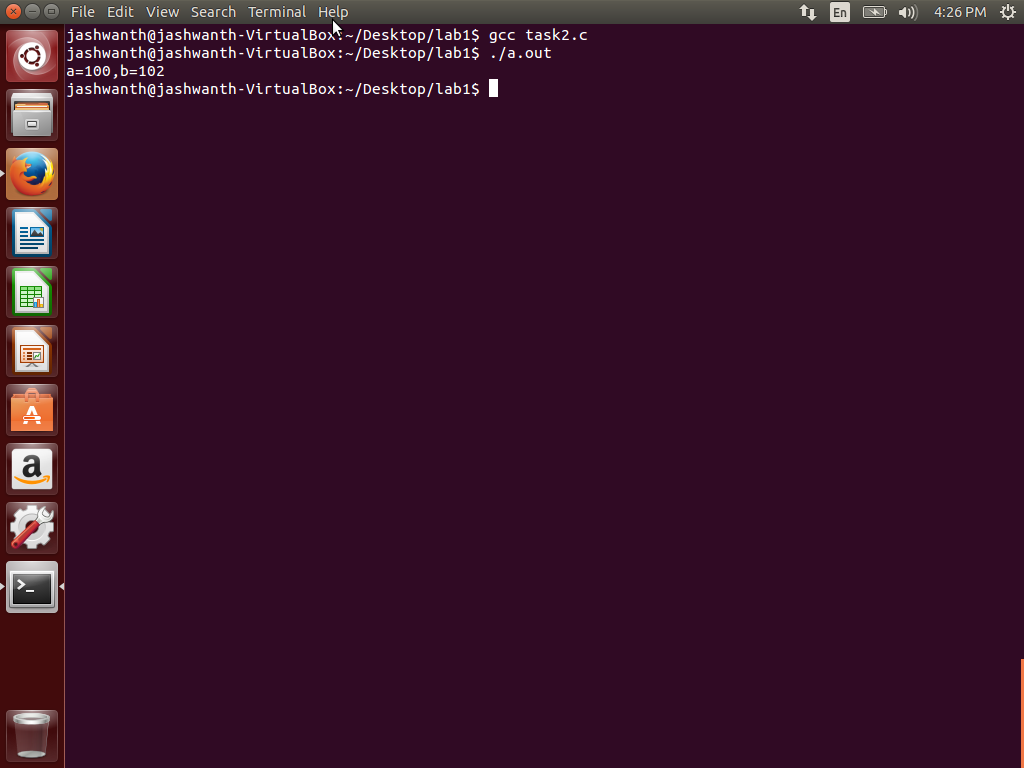
Here we are assigning the value of a to b. The output register is set to operand b, the input contains the register set to a. Hence b is assigned a value of a. Observe the output in the terminal.

**Task2:**

**Program to fil the question marks in the assignment**



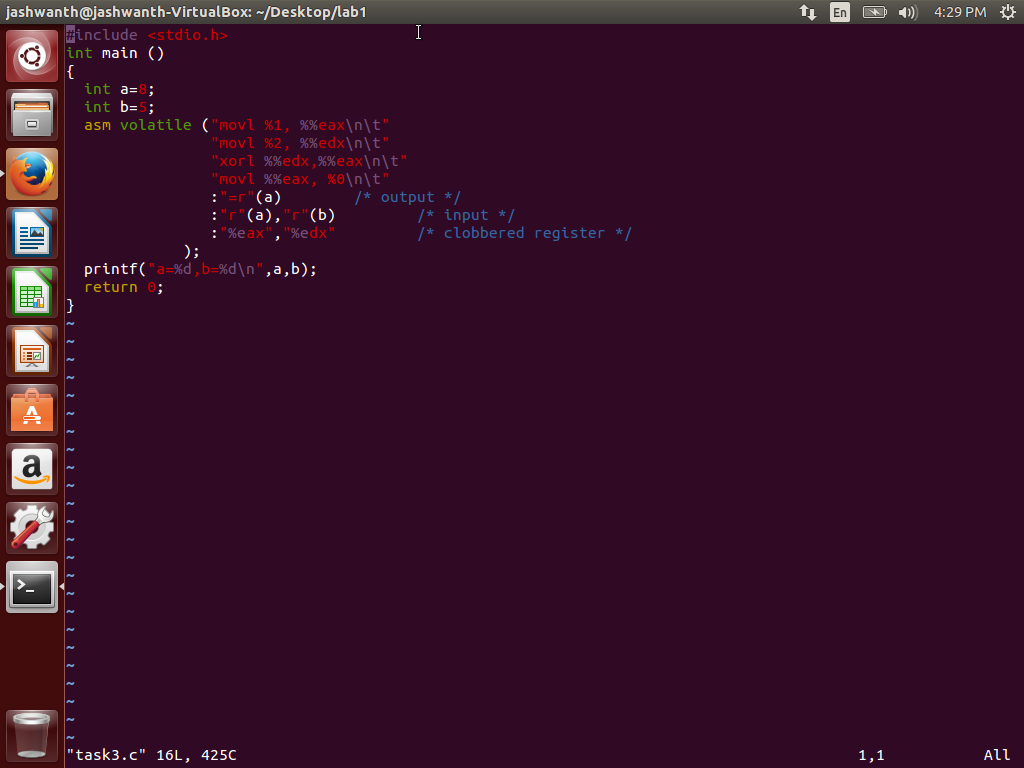
**Output:**



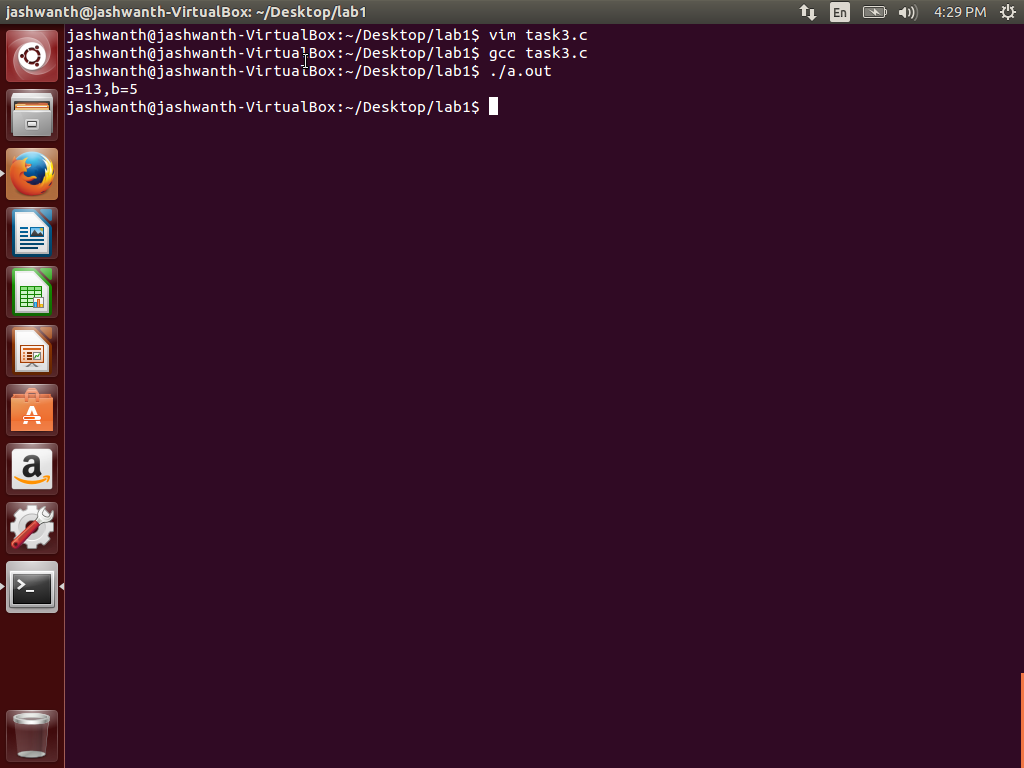
Here we get the value of b = a+2 = 102. The addl instruction adds an offset of 2 to the

eax register holding the value of a. Observe the output in the picture above.

**Task3:**

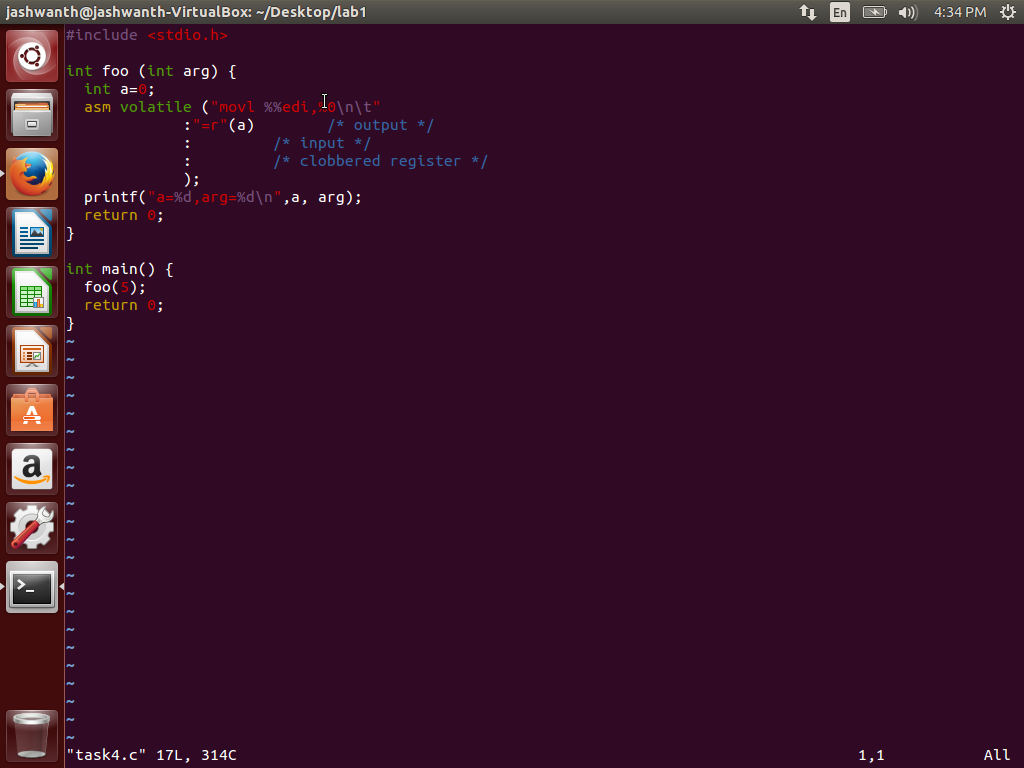


Output:

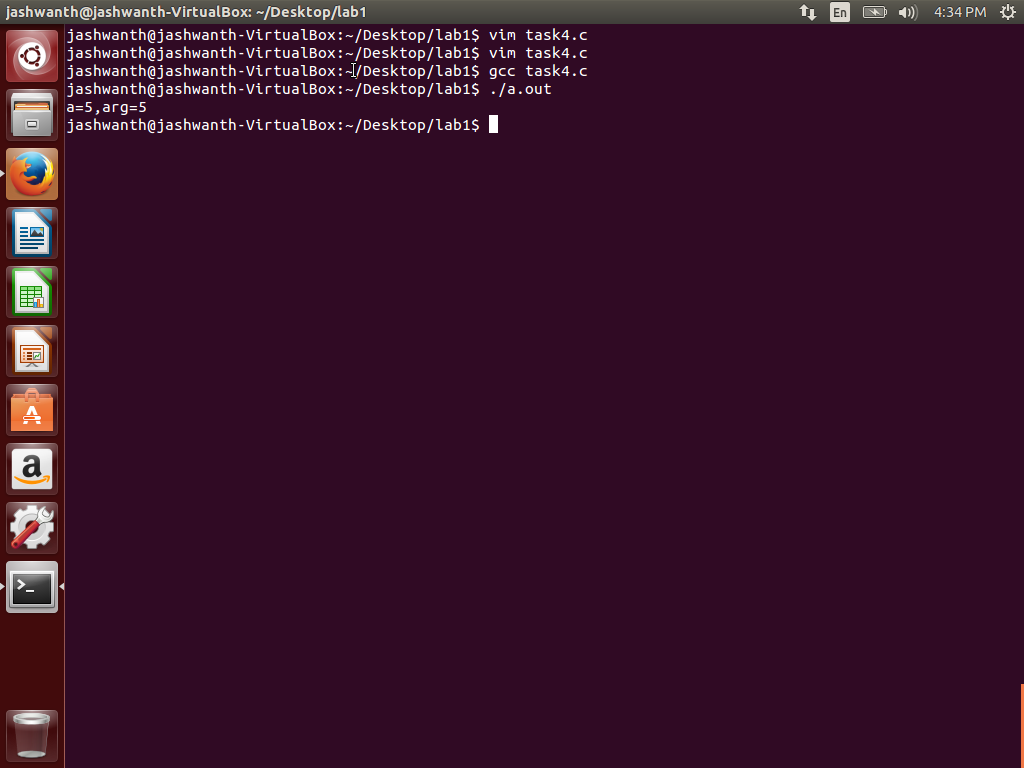


Here the exor operation is performed as a = a^b = 8^5 = 13. We have used the xorl instruction to perform the operation on the two registers holding the values of a and b.

**Task4:**

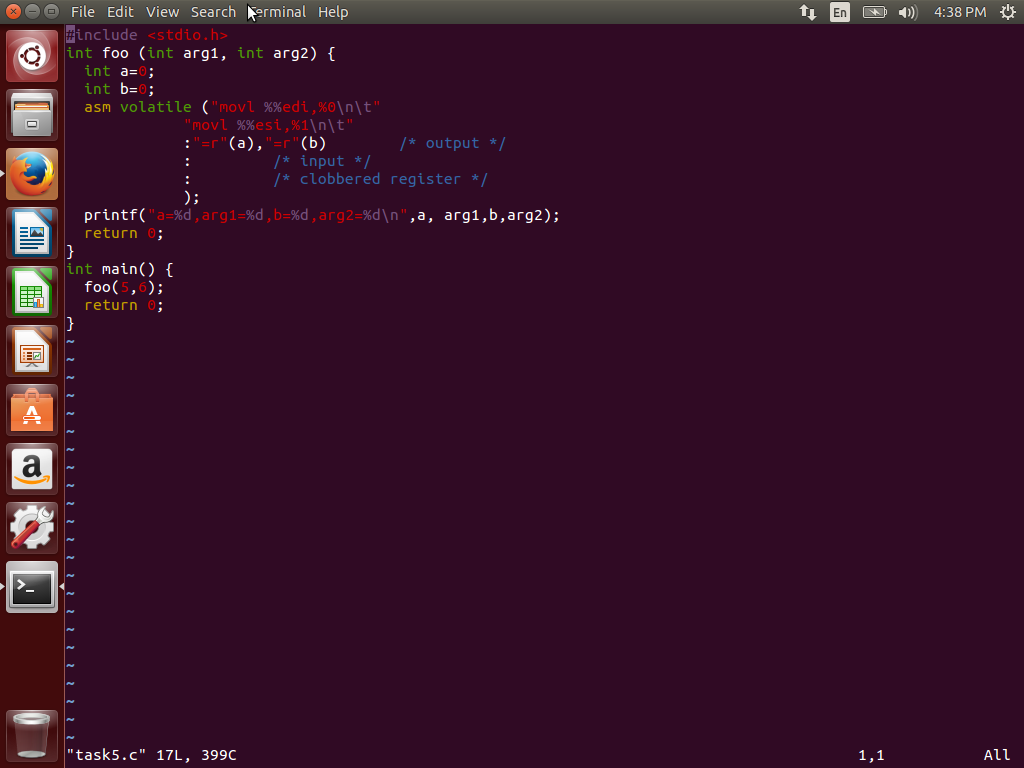
****

**Output:**

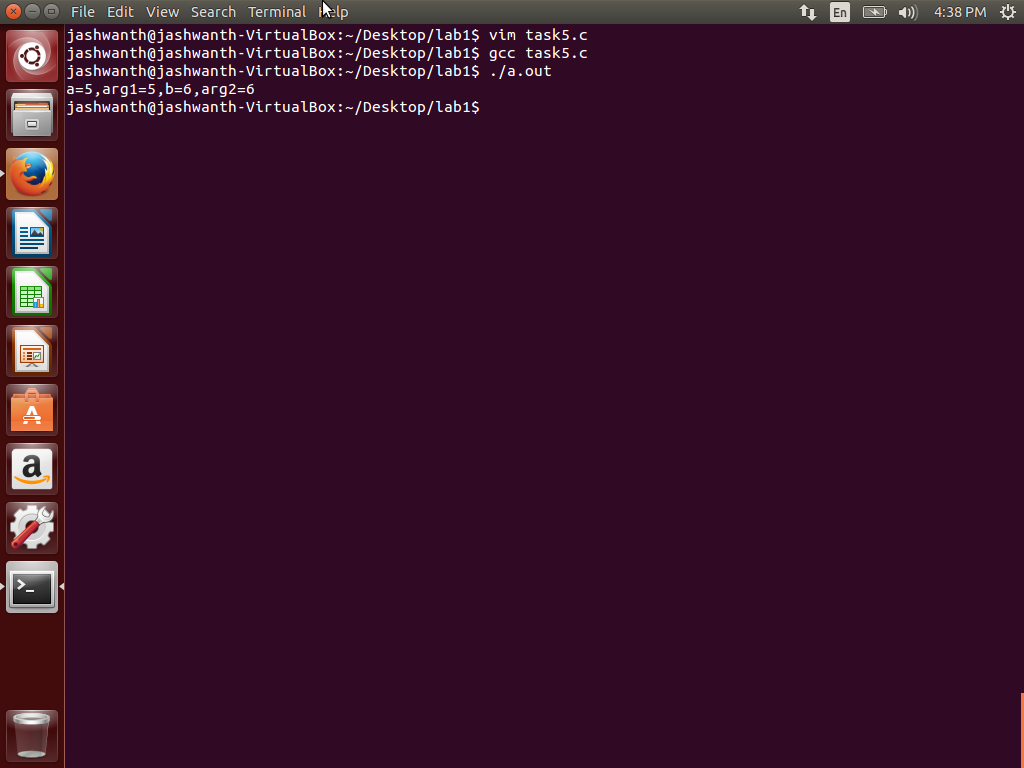
****

The function foo uses **edi** register to move the value of first argument to a. %%edi is used in the mov instruction to indicate that we have the copy value present in the memory location of edi to variable a. Observe the value of argument(5) set to a in the figure above.

**Task5:**

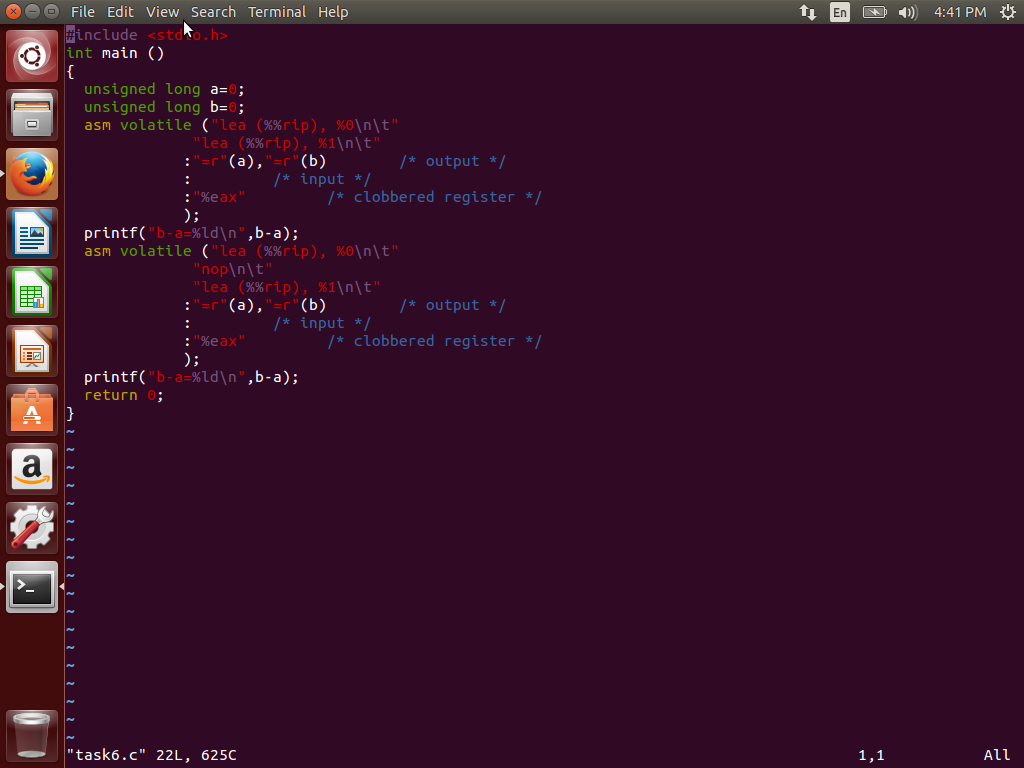
****

**Output:**

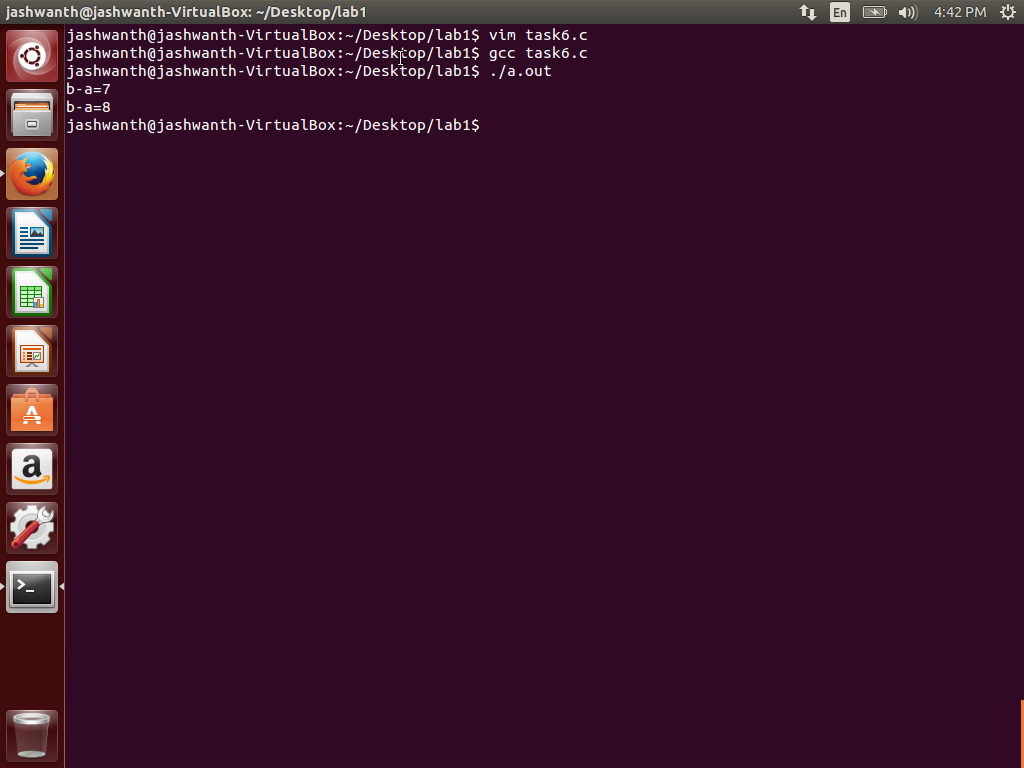
****

Register ESI is typically used to pass the second argument just like EDI in the previous task. Here we are passing two arguments to foo . Hence the first argument is copied to a and second argument is copied to b.

**Task6:**

****

**Output:**

****

**Explanation:**

The LEA instruction is used to load effective address to the destination operand. By using “lea (%%rip), %0”, we can store the current value of PC to an output argument. In the first printf both the assembly instructions are next to each other. Both a and b are of type unsigned long which are 8 bytes long. Hence PC address is loaded to a, PC+7 is loaded to b. Here the assumption is that each address is byte length accessible. Hence the difference of 7 is printed. In the second case **NOP** instruction sits between the two instructions. Hence b contains PC+8 value and the difference is shown as 8 in the output. We can infer the width of NOP instruction as 1 byte long.