CS1100 - Lecture 1

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

1 Basic Concepts

Very briefly, a computer is a device for performing computations. When compared to a calculator, it has the additional capacity to store a a sequence of instructions and execute it.

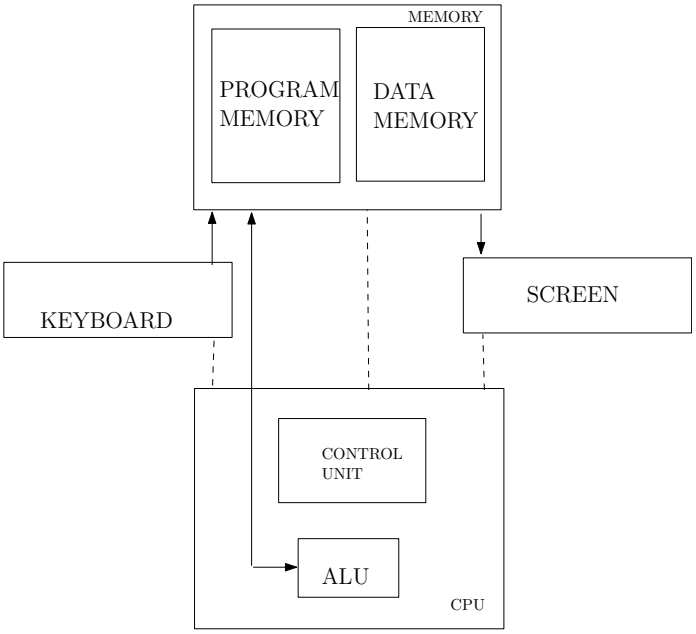
A *program* is a sequence of instructions, written in a suitable format.

Example 1. A program to compute the sum of two numbers

```
int x,y,z
input x
input y
z <-- x + y
output z
```

In the above program, x,y and z are known as *variables*.

Basic units of a computer are shown in the figure below:



Memory refers to the storage unit in a computer. For easy understanding, one may imagine that memory has two units: one for storing programs and the other for storing data. Programs are stored in the program memory while variables are stored in the data memory. Keyboard is the commonly used *input device* and monitor/screen is the commonly used *ouput device*. **Central Processing Unit** (CPU) is the unit which co-ordinates all other units. It has a *control unit* which controls its operations and an ALU to perform various arithmetic and logical operations.

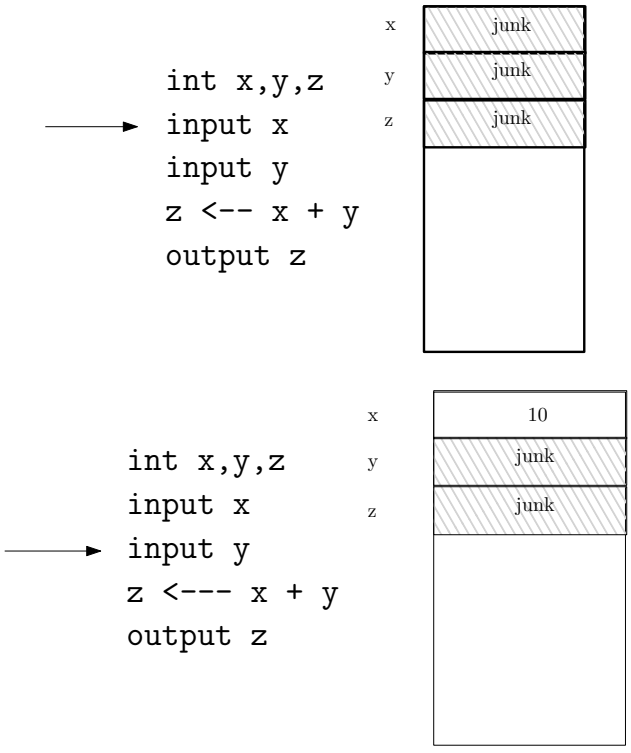
Associated with each variable, there is an address for it in the data memory. Similar to the way a pincode is internally used by the postal department to refer to a place, the computer internally uses the address of a variable to refer to that variable. But a programmer usually thinks in terms of variables. For each variable, there is some space given in the data memory to store the value of the variable. Value of a variable changes as the program execution progresses.

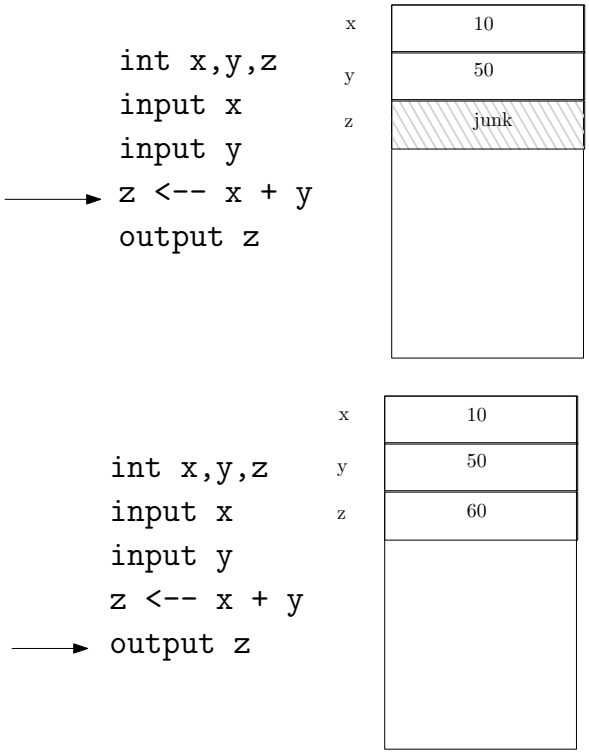
2 Memory state during program execution

A *Program Counter (PC)* indicates the position of the next instruction to be executed. When a program as in Example 1 executes, instructions are executed one by one, from the beginning. During this time, the program is in the program memory. While executing an input instruction for getting

the value a variable, the value is accepted from the keyboard and copied to the space given for that variable, in the data memory. Similarly, when an output instruction is executed for a variable, the value of that variable is taken from the space given for that variable and it is displayed on the screen. When an arithmetic/logic assignment operation is performed, the values of variables after the arrow in the assignment instruction are taken to the ALU, the operations are performed and the result is written back to the space given for the variable on left hand side of the arrow.

The memory state diagram after executing each instruction in Example 1 is shown below (assuming that the user enters 10 and 50 through keyboard, as values for x and y):





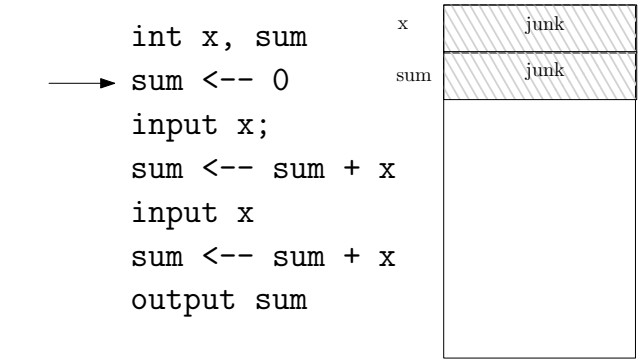
After executing the last step, the output will be displayed on the screen.

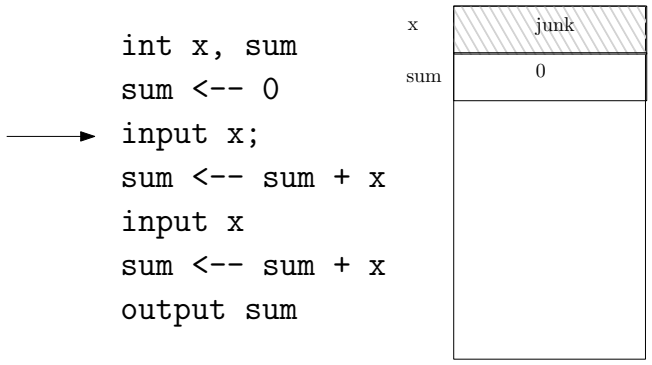
The previous example of adding two numbers can also be written in a different way.

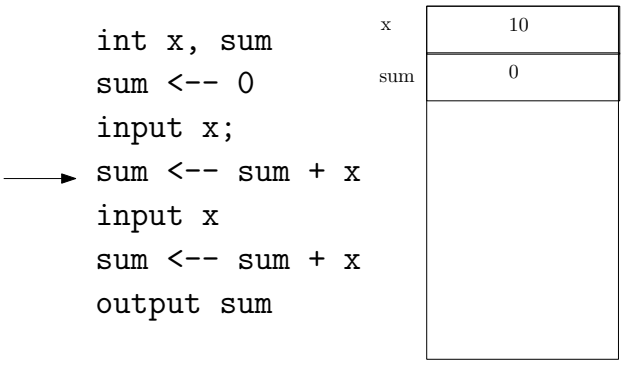
Example 2. A second program to compute the sum of two numbers

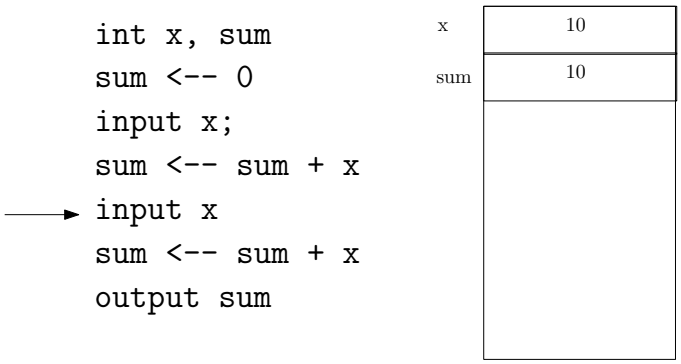
```
int x, sum
sum <-- 0
input x;
sum <-- sum + x
input x
sum <-- sum + x
output sum
```

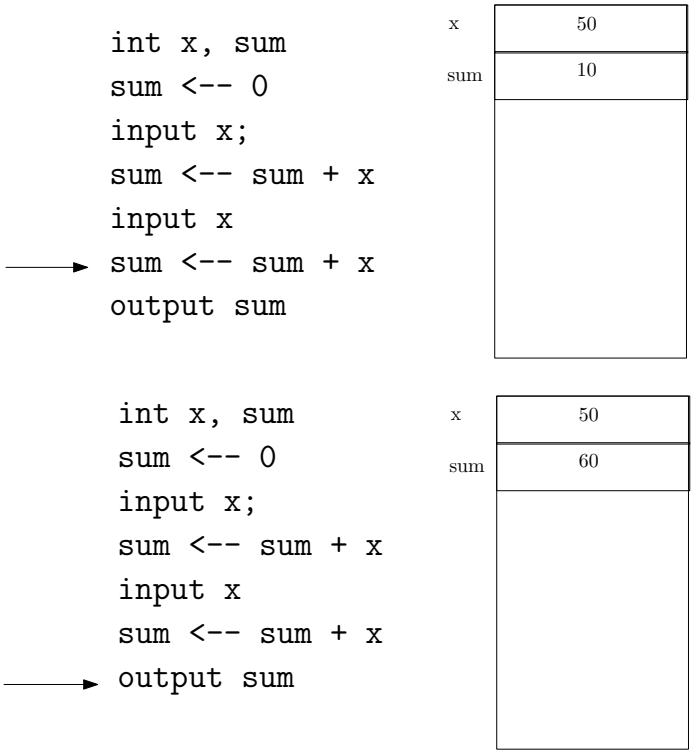
The memory state diagram for Example 2 is shown below:


int x, sum
sum <-- 0
input x;
sum <-- sum + x
input x
sum <-- sum + x
output sum
x
junk
sum
junk


int x, sum
sum <-- 0
input x;
sum <-- sum + x
input x
sum <-- sum + x
output sum
x
junk
sum
0


int x, sum
sum <-- 0
input x;
sum <-- sum + x
input x
sum <-- sum + x
output sum
x
10
sum
0


int x, sum
sum <-- 0
input x;
sum <-- sum + x
input x
sum <-- sum + x
output sum
x
10
sum
10

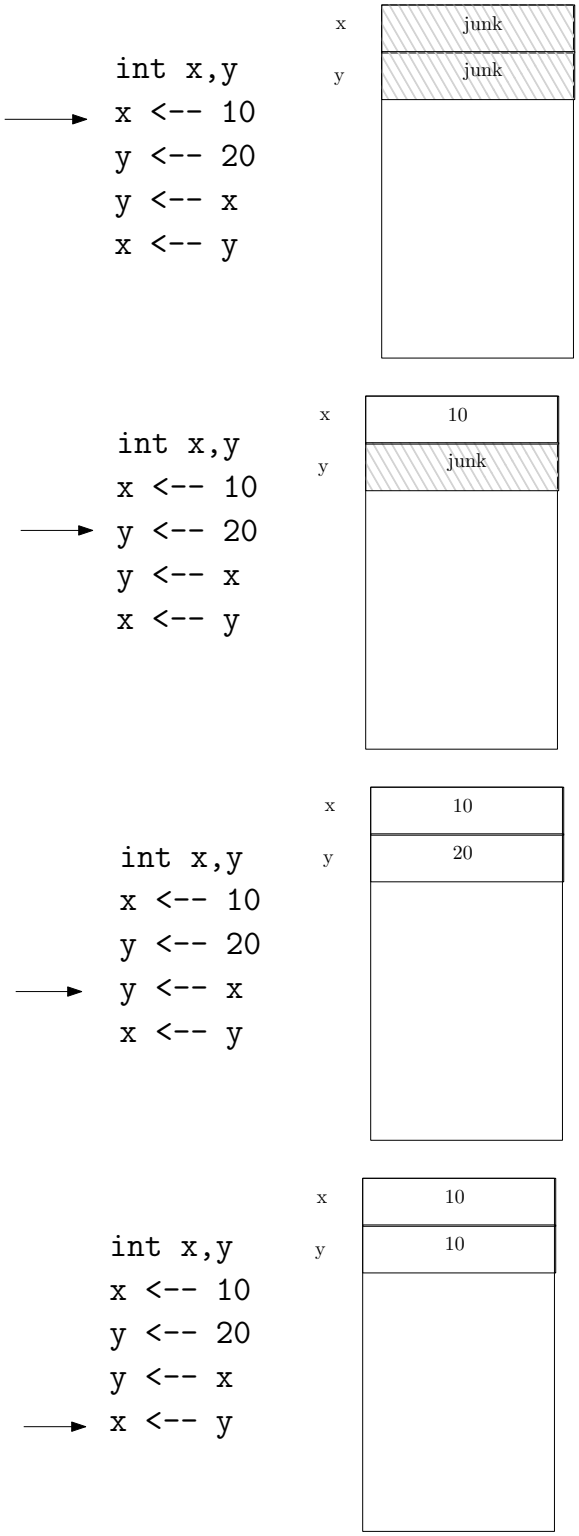


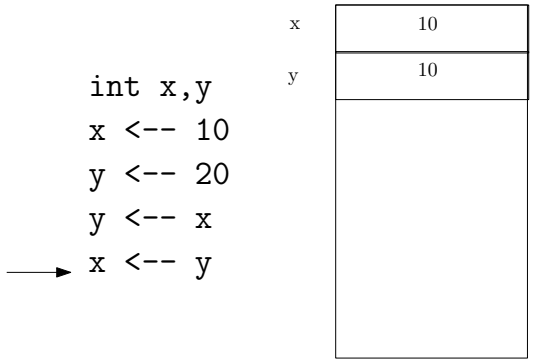
After executing the last step, the final sum is printed on screen.

Now, we consider the problem of exchanging the values of two variables. As a first attempt, consider the program below:
Example 3.

```
int x,y
x <-- 10
y <-- 20
y <-- x
x <-- y
```

The above program will not perform the exchange of the values of the two variables, as explained in the figure below:

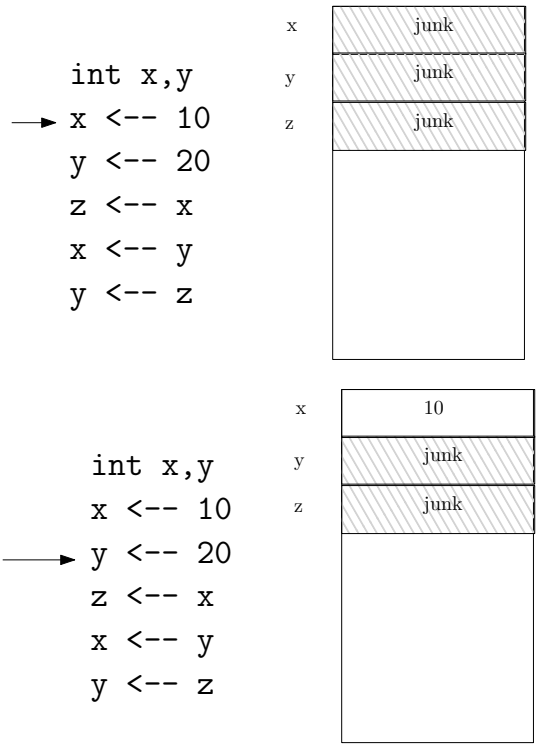

int x,y
x <-- 10
y <-- 20
y <-- x
x <-- y
x
y
junk
junk
int x,y
x <-- 10
y <-- 20
y <-- x
x <-- y
x
y
10
junk
int x,y
x <-- 10
y <-- 20
y <-- x
x <-- y
x
y
10
20
int x,y
x <-- 10
y <-- 20
y <-- x
x <-- y
x
y
10
10

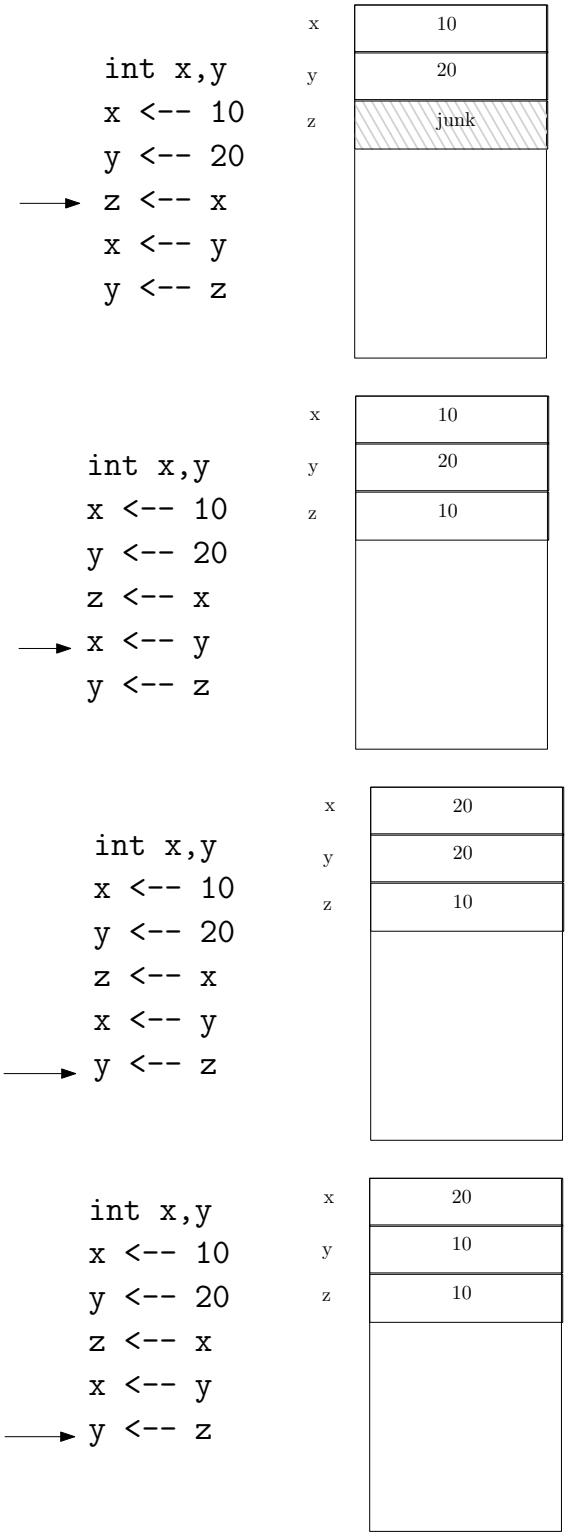


The program can be corrected as follows.

```
int x,y
x <-- 10
y <-- 20
z <-- x
x <-- y
y <-- z
```

The memory state diagram for the corrected version after each step of execution is as follows:




int x,y
x <-- 10
y <-- 20
z <-- x
x <-- y
y <-- z
x
y
z
10
20
junk
int x,y
x <-- 10
y <-- 20
z <-- x
x <-- y
y <-- z
x
y
z
10
20
10
int x,y
x <-- 10
y <-- 20
z <-- x
x <-- y
y <-- z
x
y
z
20
20
10
int x,y
x <-- 10
y <-- 20
z <-- x
x <-- y
y <-- z
x
y
z
20
10
10

Homework 1. Draw the memory state diagram of the given program after each step of execution and get the values of variables x and y after the last step of execution.

```
int x,y
x <-- 10
y <-- 20
x <-- x + y
y <-- x - y
x <-- x - y
```