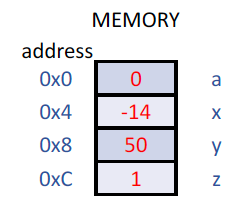
# Introduction

You are now familiar with C programming. Next, you will learn about what your programs look like when they are translated to assembly language, the native language of the computer.

# Model 1: Assembly language

Consider the diagram of memory below. The memory is byte-addressed, but we are just showing the address of the first byte of each 4-byte ***word***. The program's name for those memory locations are the variable names on the right. The machine's name for those memory locations are the addresses on the left.



1. How much memory is required for storing the variables a,x,y, and z?

16 bytes

1. ~~What is the value (in binary) of byte at address 0x9?~~
2. Consider the following program written in a high-level language.

a = x + y - z

To execute this program, which memory addresses have to be read? Which have to be written? Assume you can read or write a whole word using the address of its first byte (e.g., 0x0 for a)

|  |  |
| --- | --- |
| **Address** | **Read / Written?** |
| 0x4 | Read |
| 0x8 | Read |
| 0xC | Read |
| 0x0 | Written |

For a computer to perform an add or subtract, it requires the operands to be in a temporary storage called a ***register***. Let's suppose our architecture's registers can each hold one 4-byte word.

Our architecture supports 5 kinds of operations. These make up the ***assembly language*** of the computer.

|  |  |
| --- | --- |
| **Description** | **Notation for the instruction** |
| perform an operation (could be +, -, or bitwise operation) on the values in two registers and store the result in a register | ra <- rb + rc |
| perform an operation (could be +, -, or bitwise operation) on the value in one register and a constant and store the result in a register | ra <- rb + constant |
| move a value from memory into a register | ra <- M[address] |
| move a value from a register into memory | M[address] <- ra |
| determine which instruction to execute next  (will get to this one later!) | if ra == rb  PC <- address  else  PC <- PC+4 |

1. Re-write the program in #3 using ***only*** the assembly language instructions in the above table. You may assume you have three registers called r0, r1, r2.

R0 <- M[0x4]

R1 <- M[0x8]

R2 <- R0 + R1

R0 <- M[0xC]

R1 <- R2 – R0

M[0x0] <- R1

1. Here is the high-level language program that would have resulted in the original state of memory in our memory diagram.

int a = 0;

int x = -14;

int y = 50;

int z = -1;

Re-write this program in assembly language.