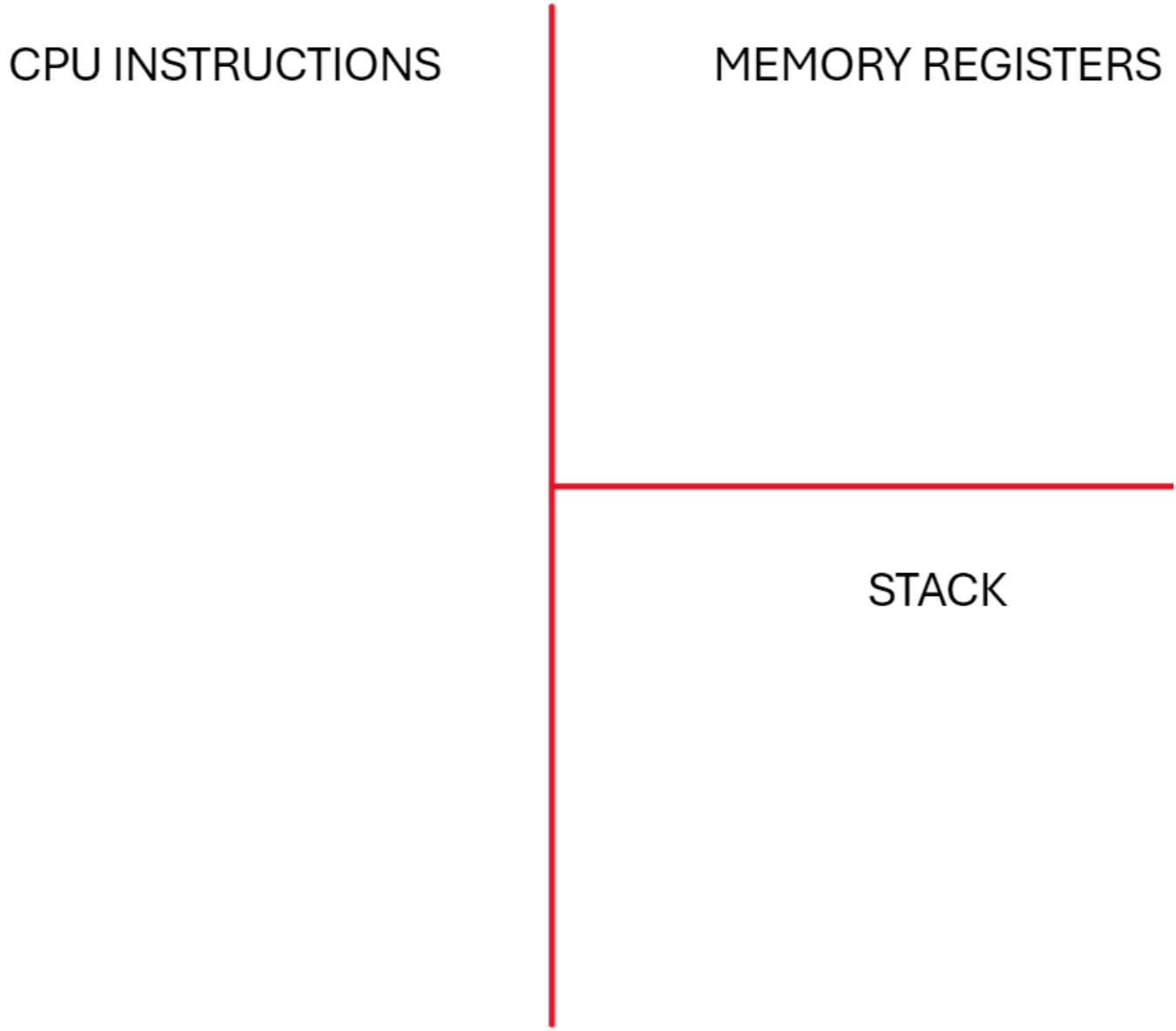
Examining x86 CPU Commands, Memory Registers, and the Stack

Executive Summary: x86 Architecture Fundamentals

Executing binary code in x86 architecture relies on key components: CPU instructions for data manipulation, conditional jumps for program flow control, arithmetic operations, stack management via push/pop instructions, function calls using call/ret instructions, and memory registers for communication between CPU and memory. Understanding these basics is crucial for effective programming and system optimization.

For binary to execute in x86 architecture, we need to observe 3 things.



CPU Instructions:

CPU INSTRUCTIONS

Mov

Jmp

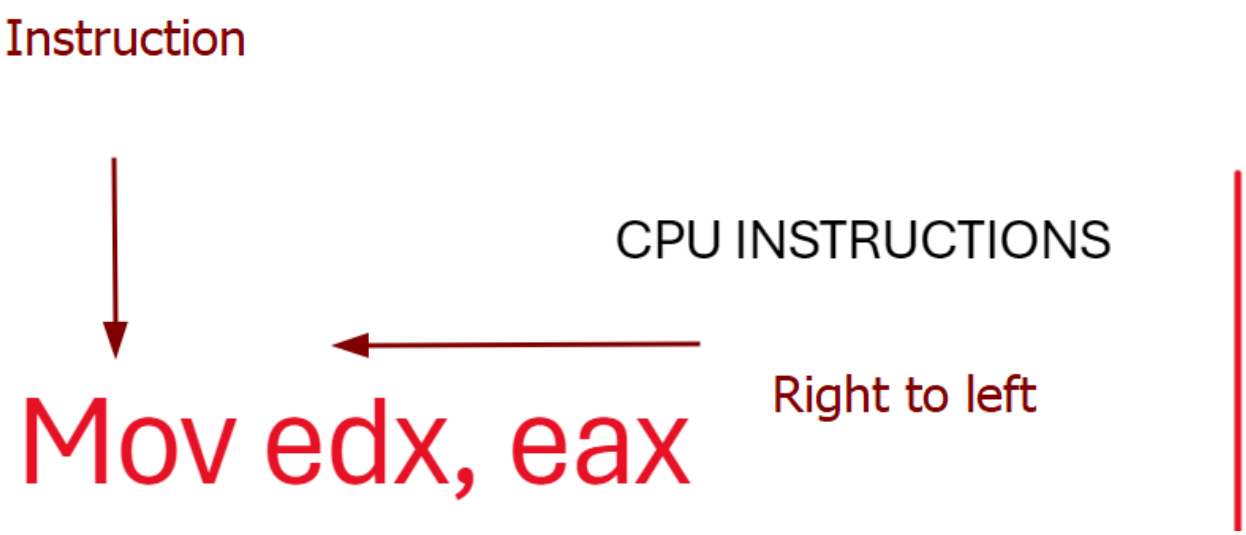
Sub

Push

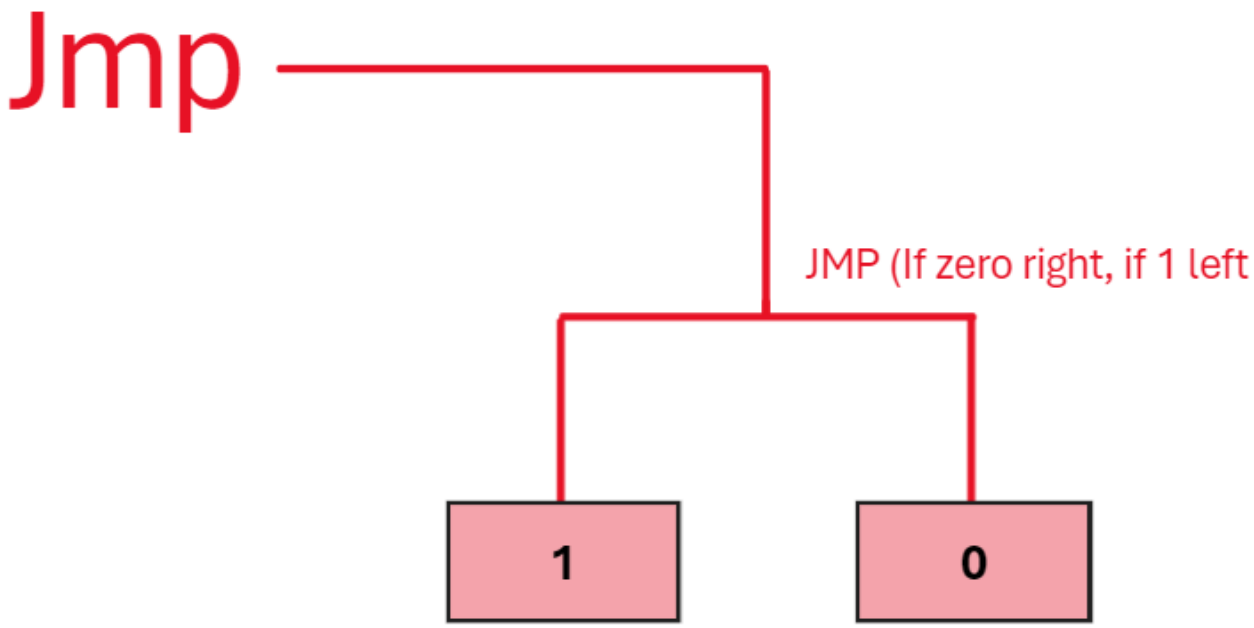
Pop

Data is written from right to left. For instance:
Mov (move), edx (destination), eax (source)

This translates into Move (because the instruction comes first), followed by eax into edx.

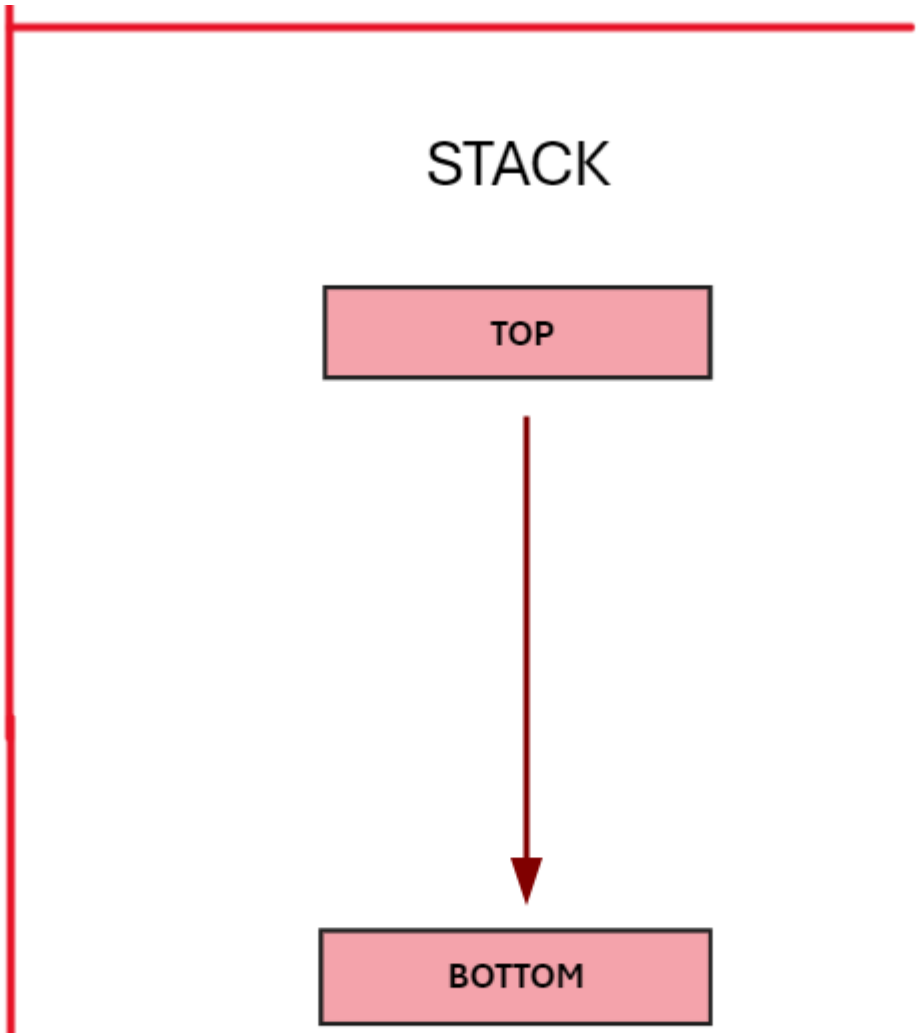


The Jmp instruction is useful if you need the program to jump to one location or another, if a conditional is met. For instance, Jnz would translate to jump if not zero. If the value is true, we would jmp to one part of the program, and if it is false, we would jump somewhere else. If it is zero then go right, if it is 1 then go left.

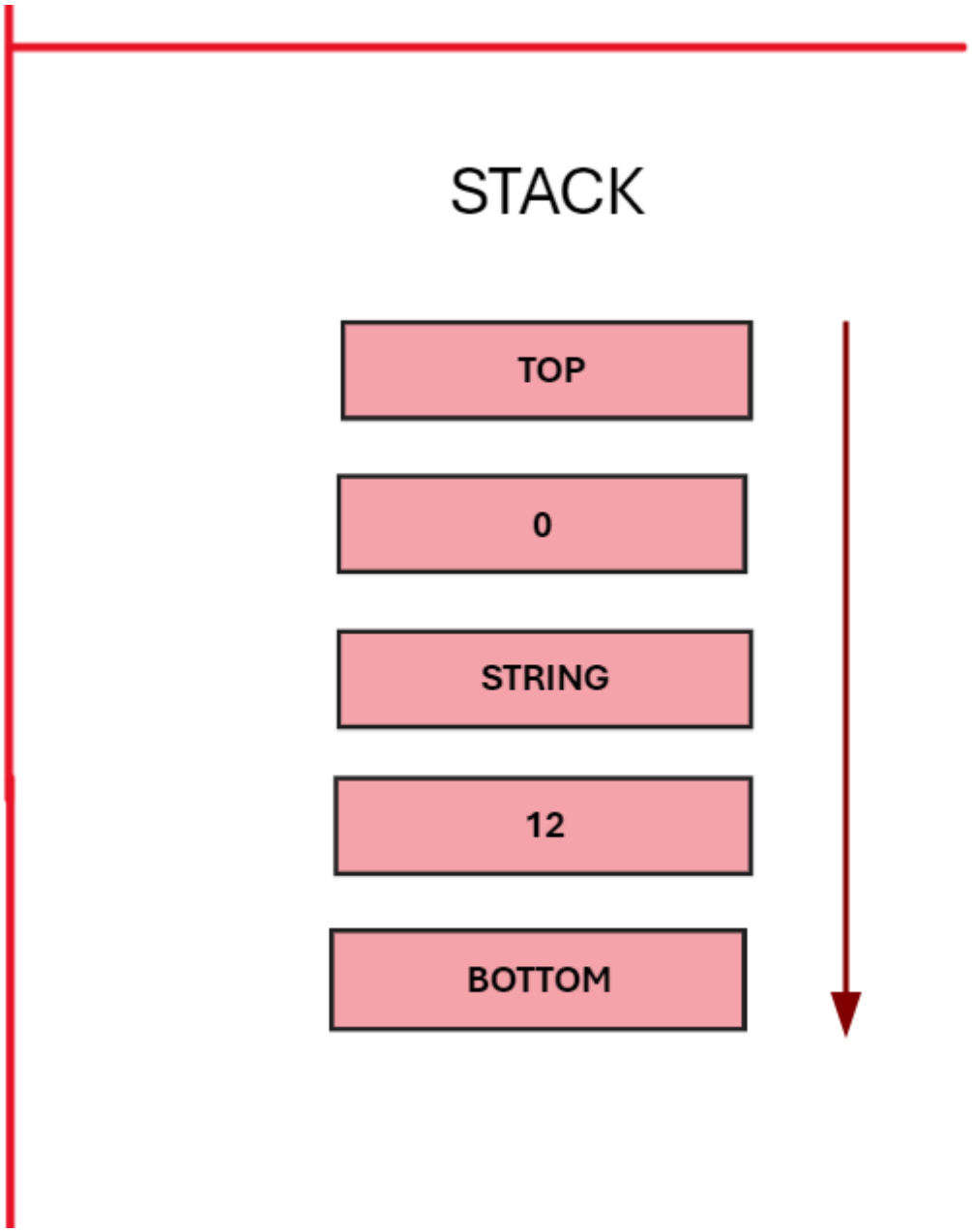


The Arithmetic works exactly how you think they would. You can subtract numbers from other numbers, and even locations from other locations.

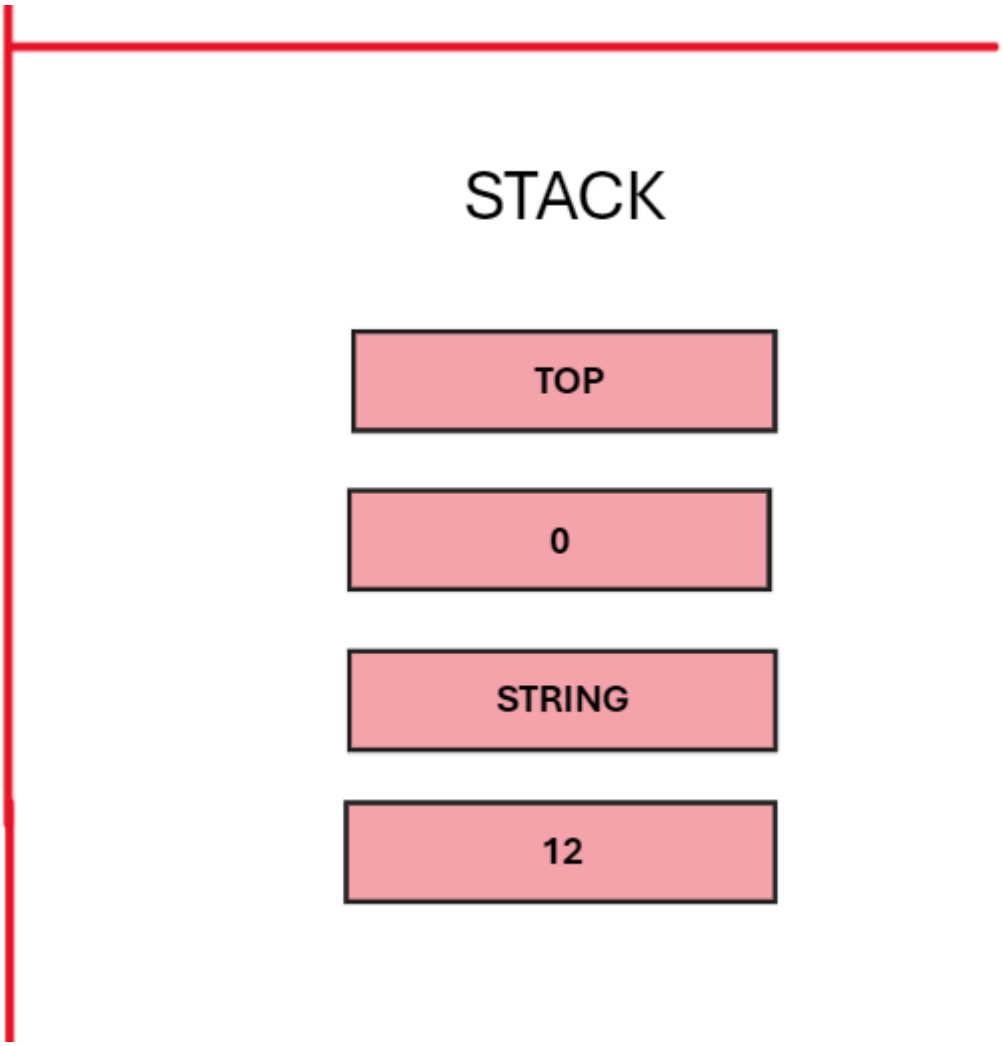
Push and Pop are how we interact with the Stack. The stack is a place with memory where you can store information you need consistent access to. Stacks grow downwards, so the top of the stack is going to be the highest location.



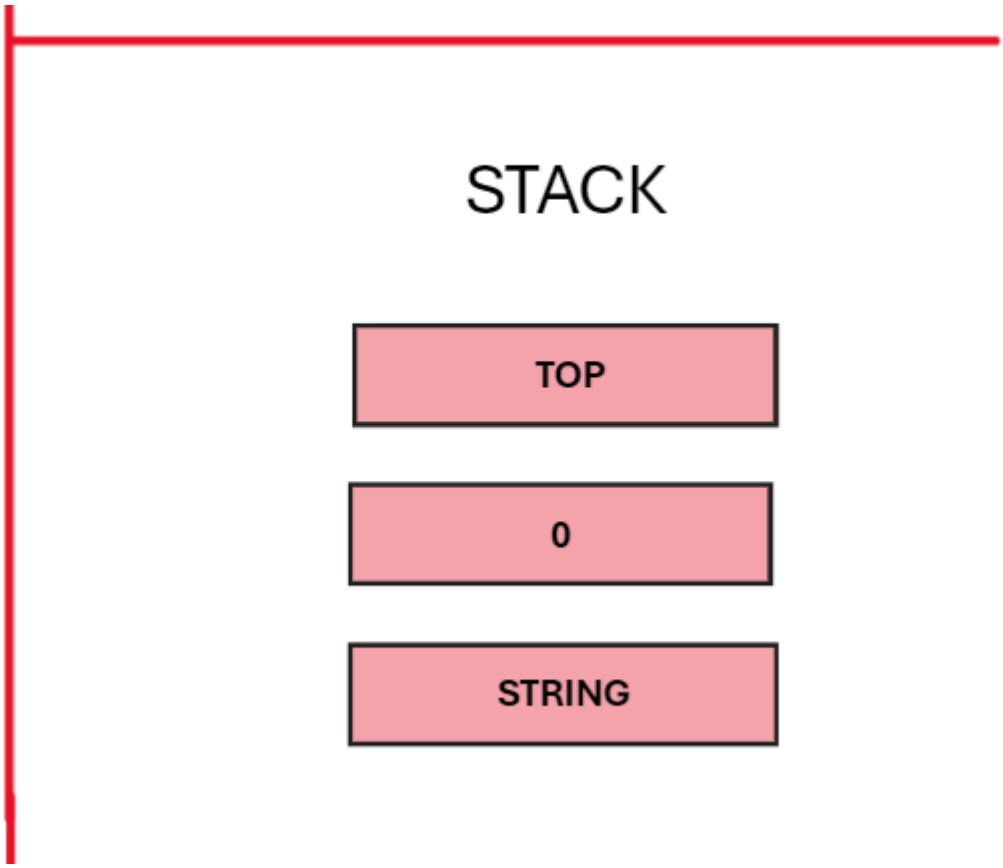
Let's say we want to call a function and set up the variables for said function. We will push those variables into the stack.
Let's push 0, string, 12.



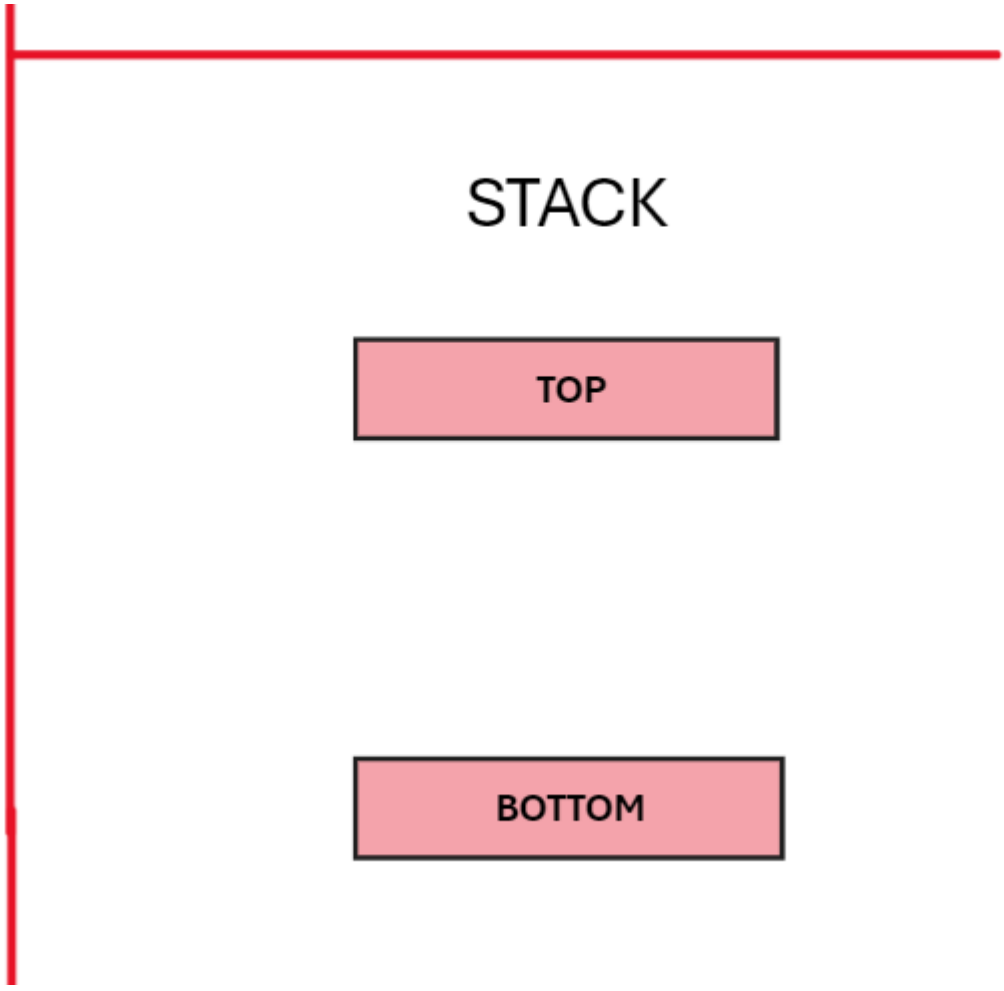
When we are done with these variables, we can POP them off of the stack. What this does is remove the lowest part of the stack that has information on it.



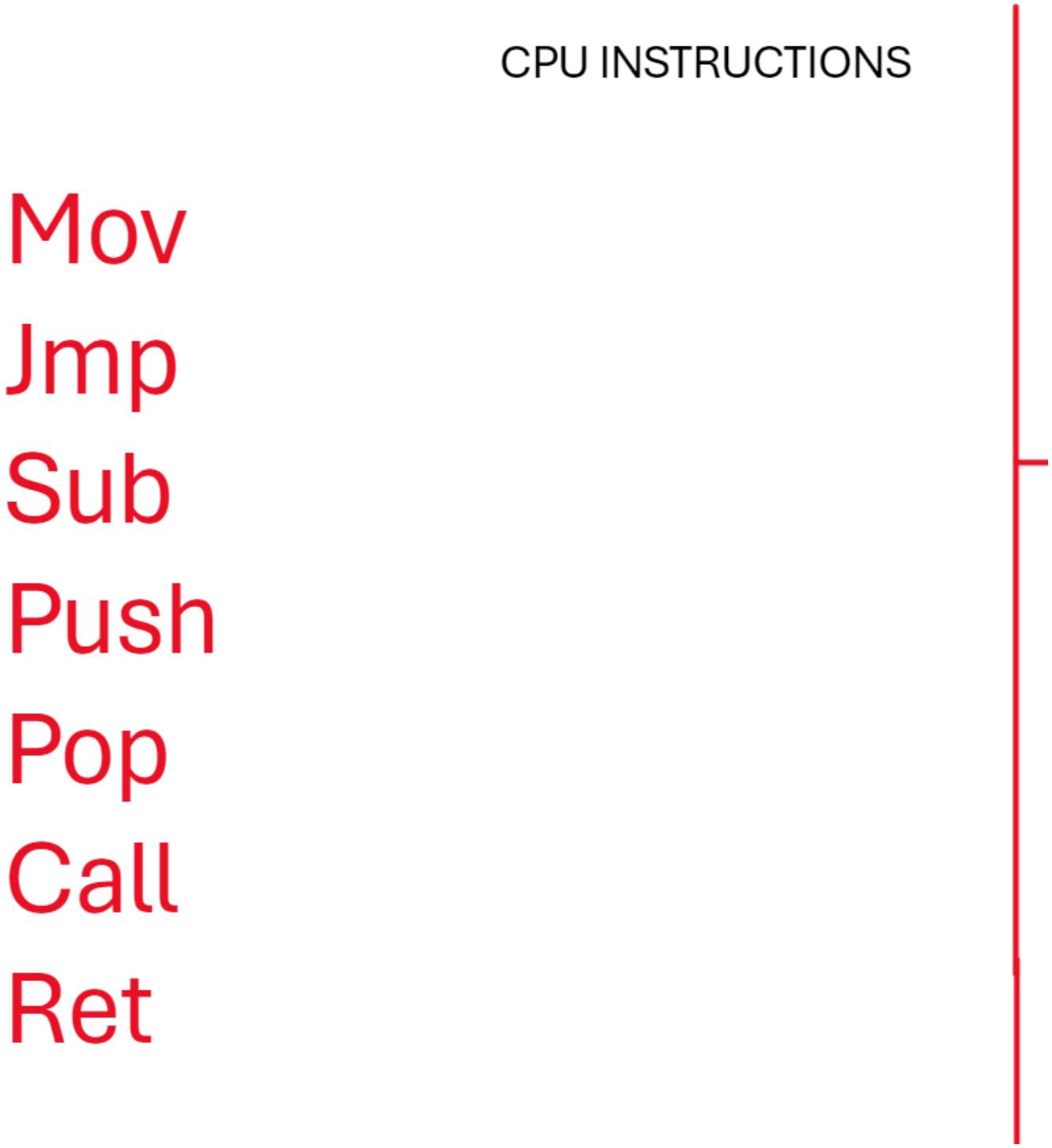
Pop!



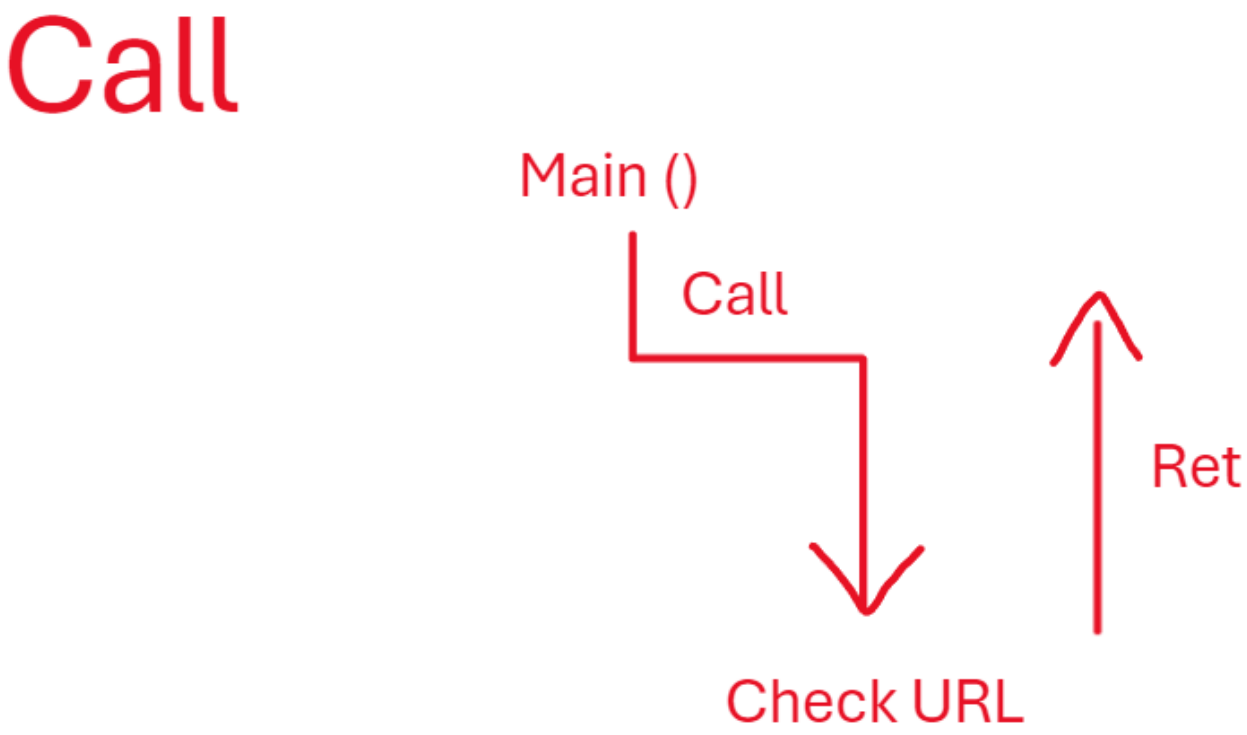
and so forth.



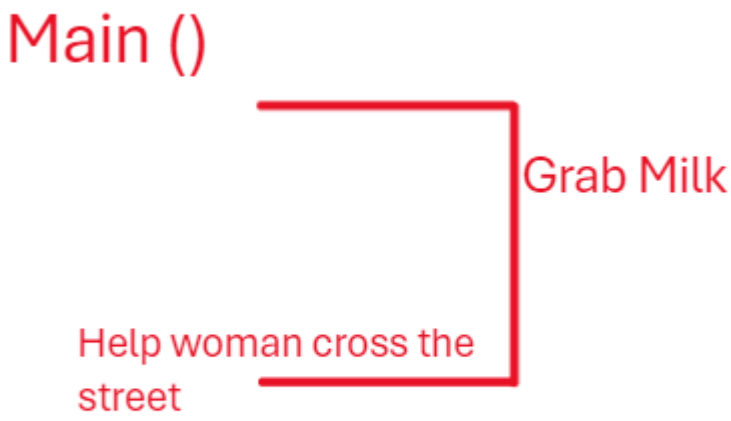
Let's take a look at two more additional instructions called Call and Ret.



Every function in the program is going to be called from the main method. For instance, when the main function starts, a bunch of variables will be introduced, then a call function will do something, and return back to the main method.

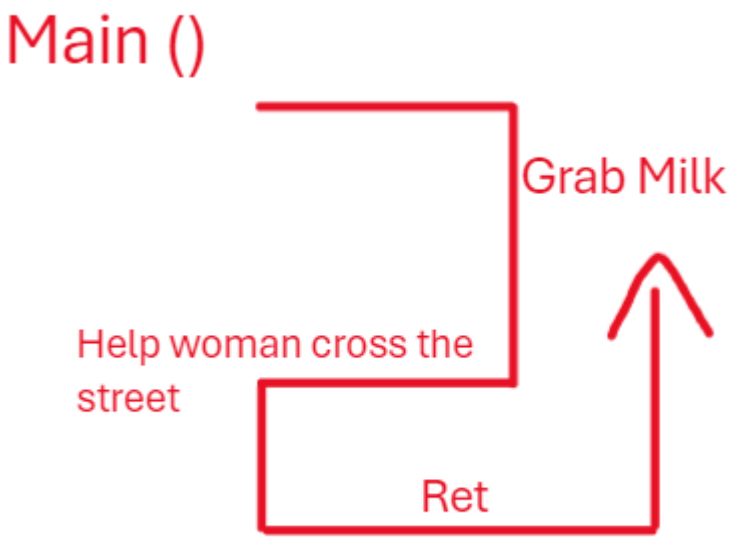


What if you need to make a call within a call?

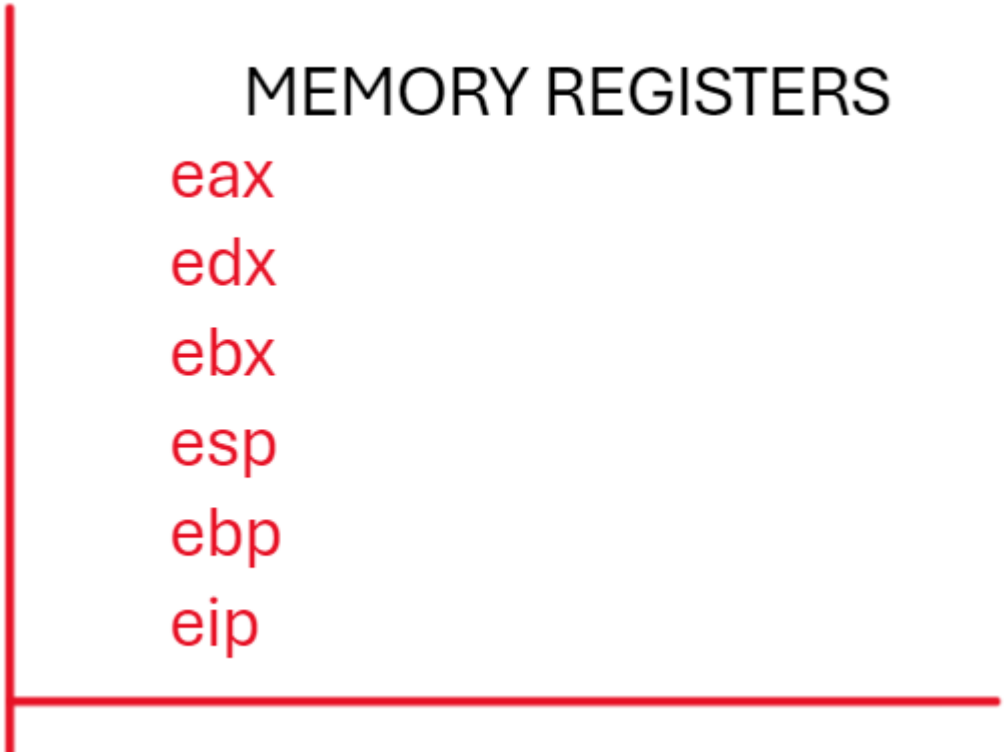


And I need to be able to go back to where the pickup milk function is once I am done helping the woman cross the street. Well, the location grab milk is saved in the extended base pointer

(EBP).

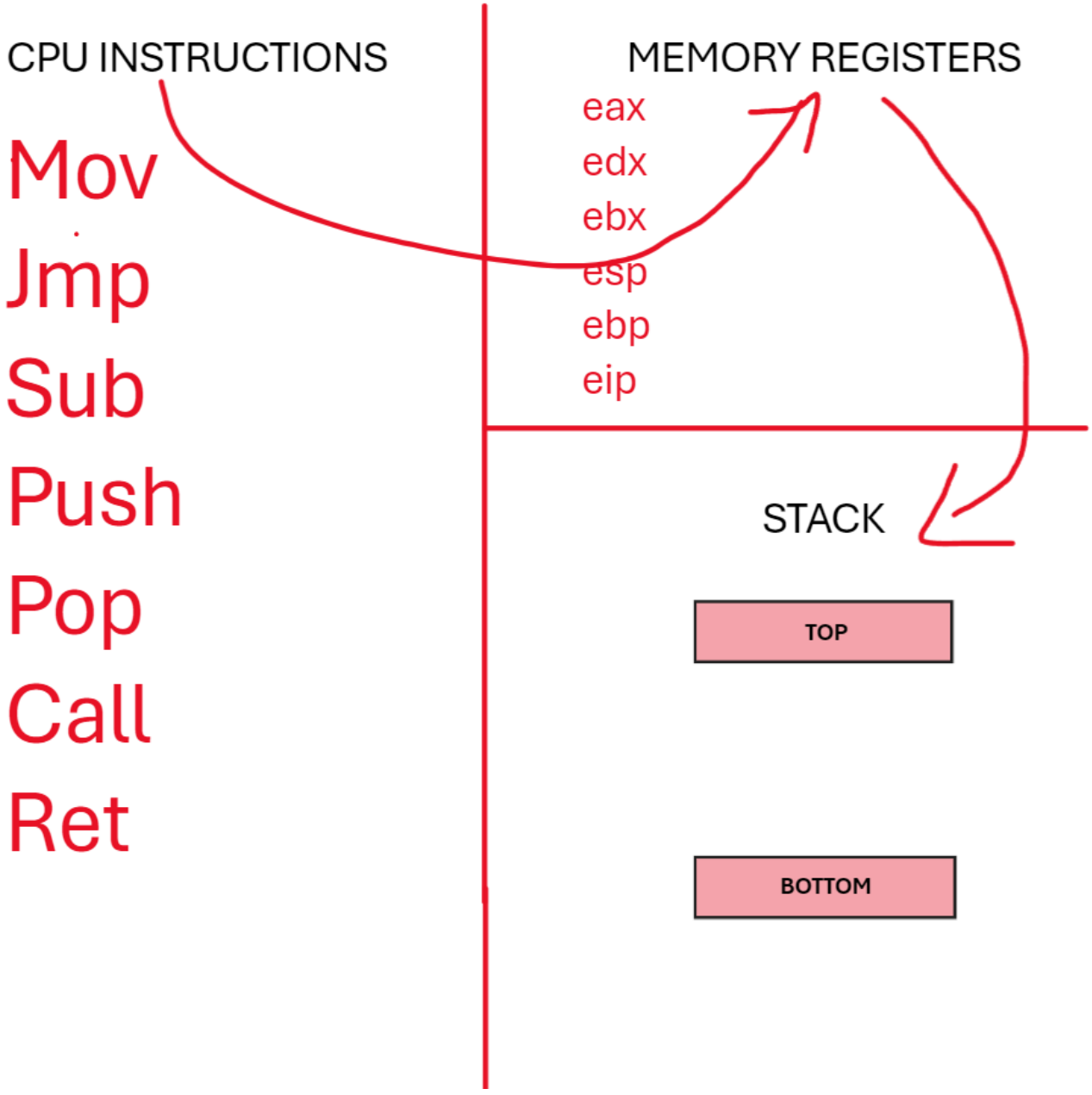


Let's take a look at our final section and learn about memory registers.



- EAX : Accumulator Register
- EDX: Data Register
- EBX: Base Register
- ESP: Extended Stack Pointer

- EBP: Extended Base Pointer
- EIP: Extended Instruction Pointer
  The communication from the CPU to the Stack is facilitated by those memory registers.



Thank you for taking the time to read. Appreciate the support.