

## If statements

Stuck? Join our Discord for help! <https://discord.gg/wvfe3XJ>

The general format of an if statement is

if(condition)

{

do-stuff-here

}

else if(condition) //this is an optional condition

{

do-stuff-here

}

Else

{

do-stuff-here

}

If statements use 3 important instructions in assembly:

* *cmpq source2, source1:* it is like computing a-b without setting destination
* *testq source2, source1*: it is like computing a&b without setting destination

Jump instructions are used to transfer control to different instructions, and there are different types of jumps:

|  |  |
| --- | --- |
| Jump Type | Description |
| jmp | Unconditional |
| je | Equal/Zero |
| jne | Not Equal/Not Zero |
| js | Negative |
| jns | Nonnegative |
| jg | Greater |
| jge | Greater or Equal |
| jl | Less |
| jle | Less or Equal |
| ja | Above(unsigned) |
| jb | Below(unsigned) |
|  |  |

The last 2 values of the table refer to unsigned integers. Unsigned integers cannot be negative while signed integers represent both positive and negative values. SInce the computer needs to differentiate between them, it uses different methods to interpret these values. For signed integers, it uses something called the two’s complement representation and for unsigned integers it uses normal binary calculations.

Start r2 with

*r2 -d if1*

**Remember to run**

***e asm.syntax=att***

And run the following commands

*aaa*

*afl*

*pdf @main*

This analyses the program, lists the functions and disassembles the main function.



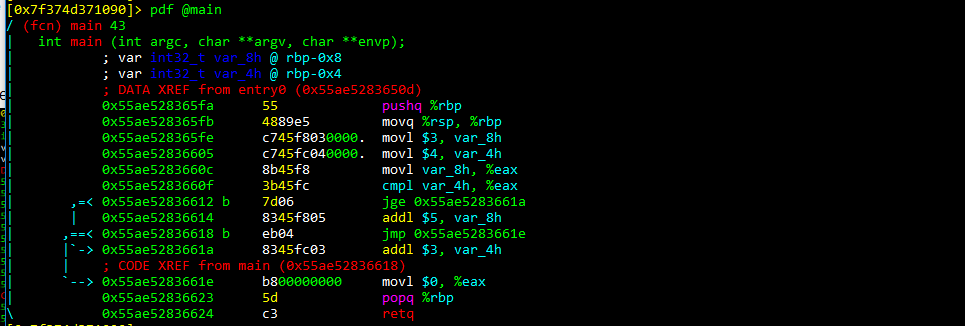
We’ll then start by setting a break point on the jge and the jmp instruction by using the command:

*db 0x55ae52836612*(which is the hex address of the jge instruction)

*db 0x55ae52836618*(which is the hex address of the jmp instruction)

We’ve added breakpoints to stop the execution of the program at those points so we can see the state of the program

Doing so will show the following:

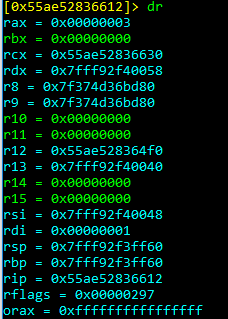


We now run *dc* to start execution of the program and the program will start execution and stop at the break point. Let’s examine what has happened before hitting the breakpoint:

* The first 2 lines are about pushing the frame pointer onto the stacker and saving it(this is about how functions are called, and will be examined later)
* The next 3 lines are about assigning values 3 and 4 to the local arguments/variables var\_8h and var\_4h. It then stores the value in var\_8h in the %eax register.
* The cmpl instruction compares the value of eax with that of the var\_8h argument

To view the value of the registers, type in

*dr*

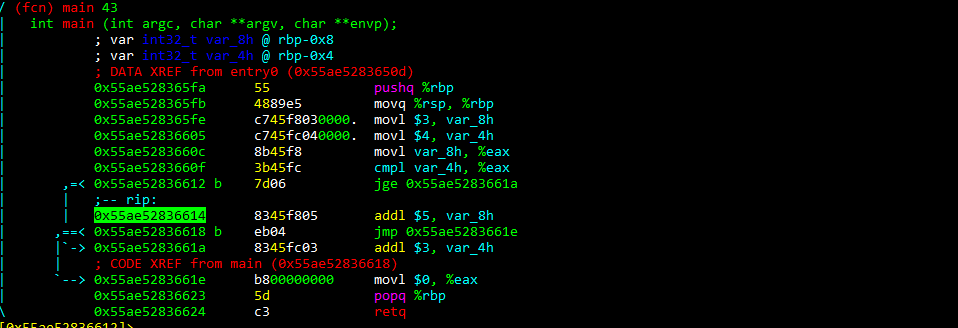


We can see that the value of rax, which is the 64 bit version of eax contains 3. We saw that the jge instruction is jumping based on whether value of eax is greater than var\_4h. To see what’s in var\_4h, we can see that at the top of the main function, it tells us the position of var\_4h. Run the command:

*px @rbp-04x*

And that shows the value of 4.

We know that eax contains 3, and 3 is not greater than 4, so the jump will not execute. Instead it will move to the next instruction. To check this, run the *ds* command which seeks/moves onto the next instruction.



The rip(which is the current instruction pointer) shows that it moves onto the next instruction - which shows we are correct. The current instruction then adds 5 to var\_8h which is a local argument. To see that this actually happens, first check the value of var\_8h, run *ds* and check the value again. This will show it increments by 5.



Note that because we are checking the exact address, we only need to check to 0 offset. The value stored in memory is stored as hex.

The next instruction is an unconditional jump and it just jumps to clearing the eax register. The popq instruction involves popping a value of the stack and reading it, and the return instruction sets this popped value to the current instruction pointer. In this case, it shows the execution of the program has been completed. To understand better about how an if statement work, you can check the corresponding C file in the same folder.