DAY17_advent_of_the_cyber

---

2. Introduction to x86-64 Assembly

Computers execute machine code, which is encoded as bytes, to carry out tasks on a computer. Since different computers have different processors, the machine code executed on these computers is specific to the processor. In this case, we'll be looking at the Intel x86-64 instruction set architecture which is most commonly found today. Machine code is usually represented by a more readable form of the code called assembly code. This machine is code is usually produced by a compiler, which takes the source code of a file, and after going through some intermediate stages, produces machine code that can be executed by a computer.

Without going into too much detail, Intel first started out by building a 16-bit instruction set, followed by 32 bit, after which they finally created 64 bit. All these instruction sets have been created for backward compatibility, so code compiled for 32-bit architecture will run on 64-bit machines. As mentioned earlier, before an executable file is produced, the source code is first compiled into assembly(.s files), after which the assembler converts it into an object program(.o files), and operations with a linker finally make it an executable.

The best way to actually start explaining assembly is by diving in. We'll be using `radare2` to do this - radare2 is a framework for reverse engineering and analysing binaries. It can be used to disassemble binaries(translate machine code to assembly, which is actually readable) and debug said binaries(by allowing a user to step through the execution and view the state of the program).

Luckily for us, everything we need has been provided to you via an Instance that you can deploy and log into:

1. Press the "Deploy" button on the top-right of this task
2. Wait for the IP address of the target Instance to display
3. Log into your Instance using the following information:

IP Address: `10.10.204.229`

Username: elfmceager

Password: adventofcyber

Let's proceed to run through how Radare2 works exactly. Although you shouldn't do this if the program is unknown, it is safe for us to execute to see what *should* be happening like so:

```
ashu@ashu-Inspiron-5379 ~/D/t/c/christmas-re> ./file1
the value of a is 4, the value of b is 5 and the value of c is 9
```

The above program shows that there are 3 variables(a, b, c) where c is the sum of a and b.

Time to see what's happening under the hood! **Run** the command `r2 -d ./file1`

This will open the binary in debugging mode. Once the binary is open, one of the first things to do is ask r2 to analyze the program, and this can be done by typing in: `aa`

Note, when using the `aa` command in radare2, this may take between 5-10 minutes depending on your system.

Which is the most common analysis command. It analyses all symbols and entry points in the executable. The analysis, in this case, involves extracting function names, flow control information, and much more! r2 instructions are usually based on a single character, so it is easy to get more information about the commands.

I.e. For general help, we can run: `?` or if we wish to understand more about a specific feature, we could provide `a?`

3. Computer says...Done?!

Once the analysis is complete, you would want to know where to start analysing from - most programs have an entry point defined as main. To find a list of the functions run: `afl`

```
[0x00400a30]> afl | grep main

0x00400b4d    1 68           sym.main
0x00400e10  114 1657         sym.__libc_start_main
0x00403870  346 6038 -> 5941 sym._nl_find_domain
0x00415fe0    1 43           sym._IO_switch_to_main_get_area
0x0044cf00    1 8            sym._dl_get_dl_main_map
0x00470520    1 49           sym._IO_switch_to_main_wget_area
0x0048fae0    7 73   -> 69   sym._nl_finddomain_subfreeres
0x0048fb30   16 247  -> 237  sym._nl_unload_domain
```

Note that memory addresses may be different on your computer.

As seen here, there actually is a function at main. Let's examine the assembly code at main by running the command `pdf @main` Where pdf means print disassembly function. Doing so will give us the following view:

As seen here, there actually is a function at main. Let's examine the assembly code at main by running the command `pdf @main` Where pdf means print disassembly function. Doing so will give us the following view:

3. Register me this, register me that...

The core of assembly language involves using registers to do the following:

- Transfer data between memory and register, and vice versa
- Perform arithmetic operations on registers and data
- Transfer control to other parts of the program Since the architecture is x86-64, the registers are 64 bit and Intel has a list of 16 registers:

Initial Data Type	Suffix	Size (bytes)
Byte	b	1
Word	w	2
Double Word	l	4
Quad	q	8
Single Precision	s	4
Double Precision	l	8

When dealing with memory manipulation using registers, there are other cases to be considered:

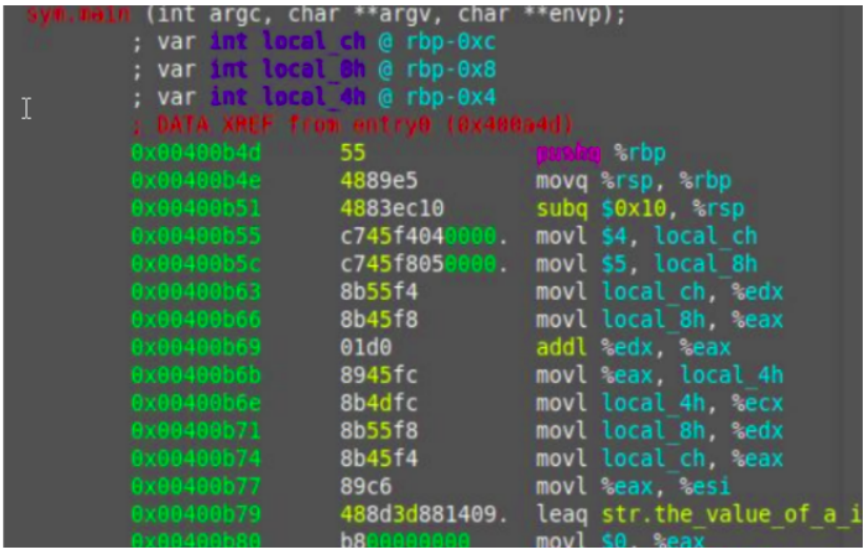
- (Rb, Ri) = MemoryLocation[Rb + Ri]
- D(Rb, Ri) = MemoryLocation[Rb + Ri + D]
- (Rb, Ri, S) = MemoryLocation(Rb + S * Ri]
- D(Rb, Ri, S) = MemoryLocation[Rb + S * Ri + D]

4. Read the instructions!

Some other important instructions are:

- *leaq source, destinatio*n: this instruction sets destination to the address denoted by the expression in source
- *addq source, destination*: destination = destination + source
- *subq source, destination*: destination = destination - source
- *imulq source, destination*: destination = destination * source
- *salq source, destination*: destination = destination << source where << is the left bit shifting operator
- *sarq source, destination*: destination = destination >> source where >> is the right bit shifting operator
- *xorq source, destination*: destination = destination XOR source
- andq source, destination: destination = destination & source
- *orq source, destination*: destination = destination | source

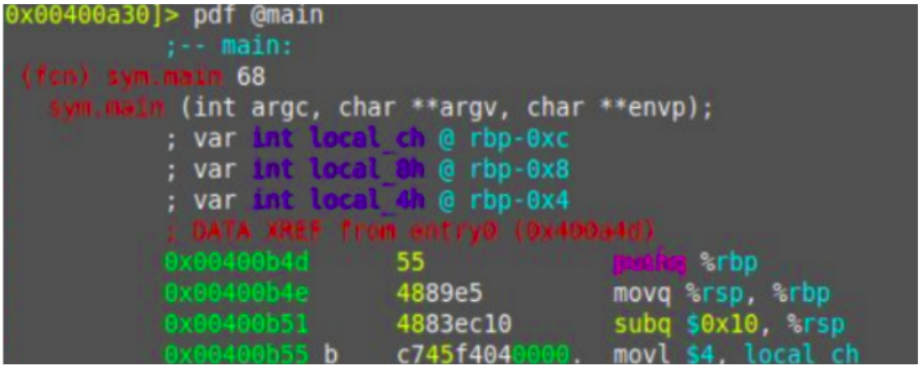
Now let's actually walk through the assembly code to see what the instructions mean when combined.



The line starting with **sym.main** indicates we're looking at the **main function**. The next 3 lines are used to represent the variables stored in the function. The second column indicates that they are integers(*int*), the 3rd column specifies the name that `r2` uses to reference them and the 4th column shows the actual memory location.

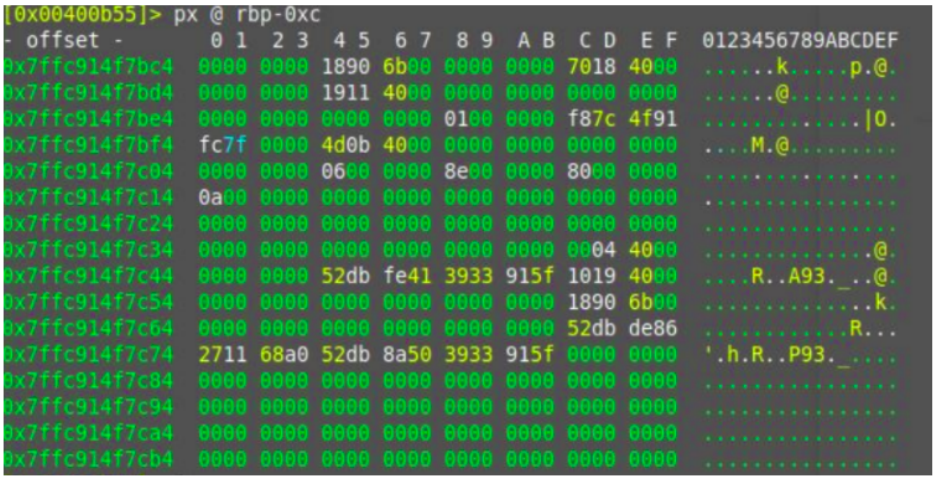
The first 3 instructions are used to allocate space on that stack (ensures that there's enough room for variables to be allocated and more). We'll start looking at the program from the 4th instruction (`movl $4`). We want to analyse the program while it runs and the best way to do this is by using **breakpoints**.

A **breakpoint** specifies where the program should stop executing. This is useful as it allows us to look at the state of the program at that particular point. So let's set a breakpoint using the command `db` in this case, it would be `db 0x00400b55` To ensure the breakpoint is set, we run the `pdf @main` command again and see a little **b** next to the instruction we want to stop at.

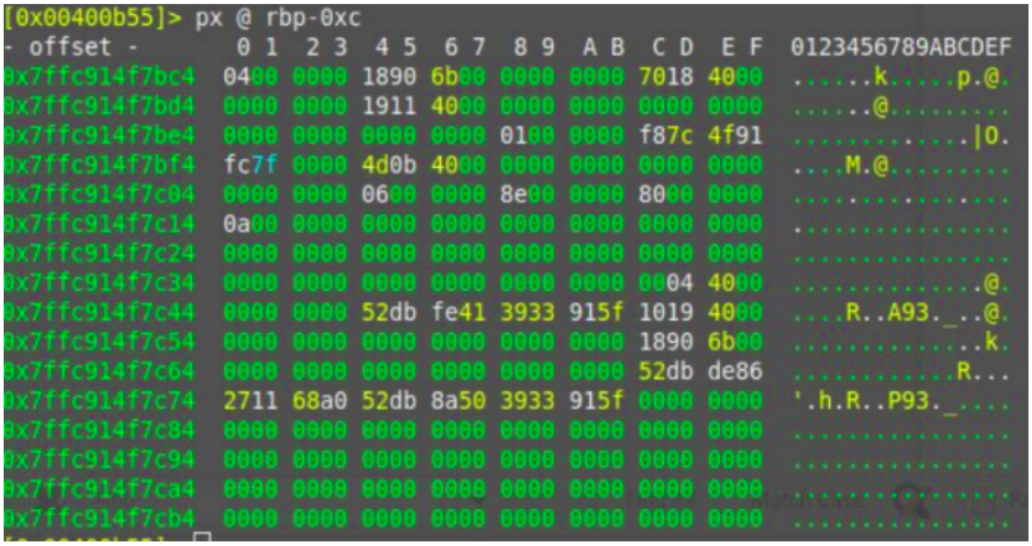


Now that we've set a breakpoint, let's run the program using `dc`

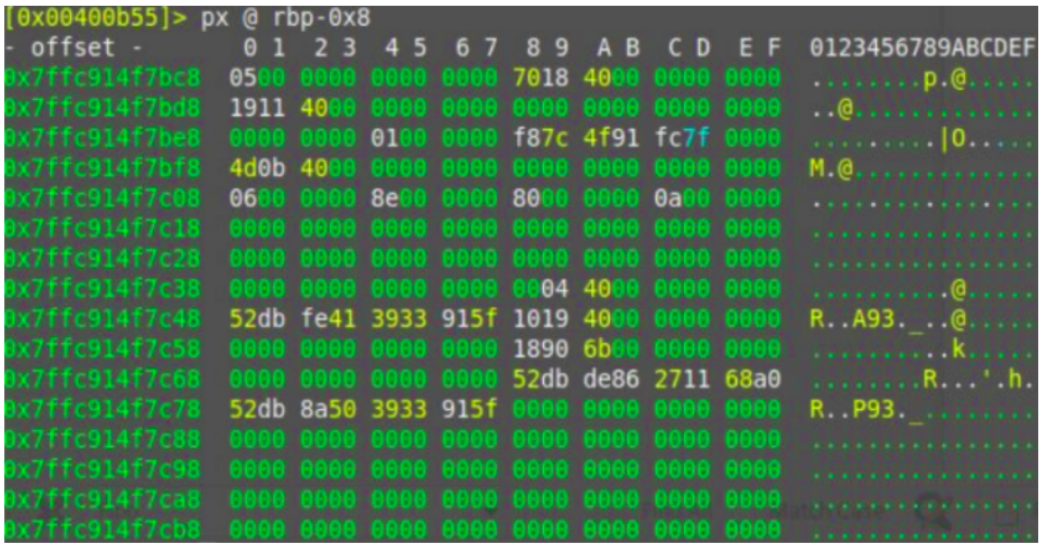
Running `dc` will execute the program until we hit the breakpoint. Once we hit the breakpoint and print out the main function, the rip which is the current instruction shows where execution has stopped. From the notes above, we know that the **mov** instruction is used to transfer values. This statement is transferring the value 4 into the `local_ch` variable. To view the contents of the `local_ch` variable, we use the following instruction `px @memory-address` In this case, the corresponding memory address for `local_ch` will be `rbp-0xc` (from the first few lines of **@pdf main**) This instruction prints the values of memory in hex:



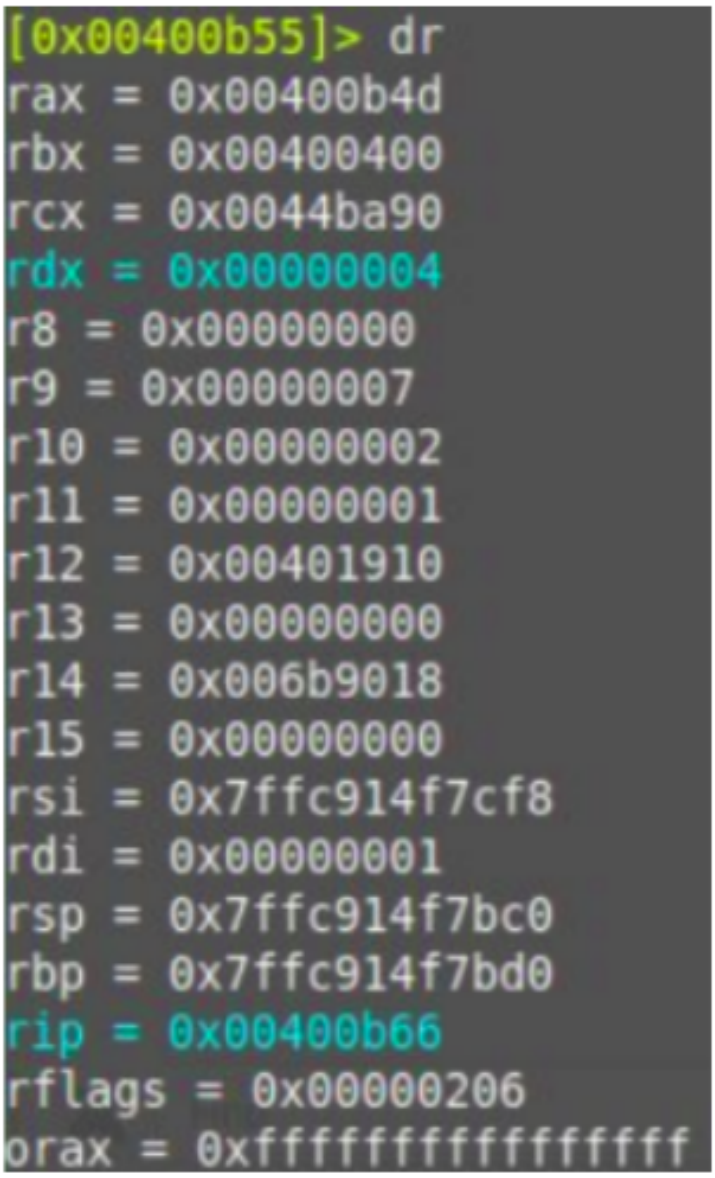
This shows that the variable currently doesn't have anything stored in it (it's just **0000**). Let's execute this instruction and go to the next one using the following command (which only goes to the next instruction) `ds` If we view the memory location after running this command, we get the following:



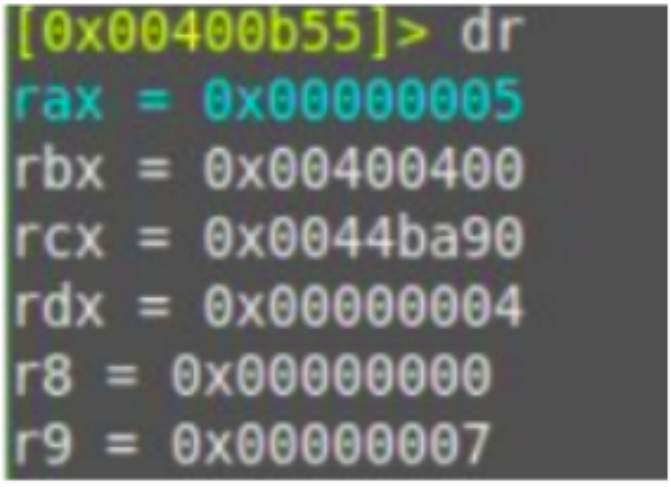
We can see that the first 2 bytes have the value **4**! If we do the same process for the next instruction, we'll see that the variable `local_8h` has the value **5**.



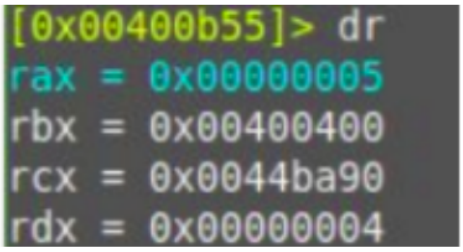
If we go to the instruction `movl local_8h, %eax`, we know from the notes that this moves the value from `local_8h` to the `%eax` register. To see the value of the `%eax` register, we can use the command:



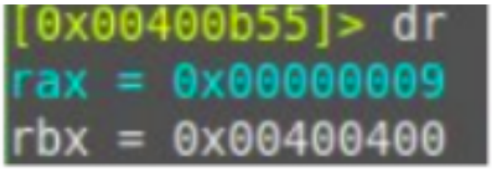
If we execute the instruction and run the `dr` command again, we get:



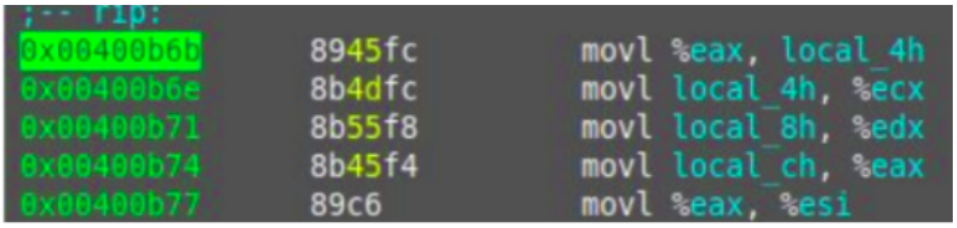
This technically skips the previous instruction `movl local_ch, %edx` but the same process can be applied to it. Showing the value of *rax* (the 64 bit version) to be 5. We can do the same for similar instructions and view the values of the registers changing. When we come to the `addl %edx, %eax`, we know that this will add the values in **edx** and **eax** and store them in **eax**. Running `dr` shows us the **rax** contains **5** and **rdx** contains **4**, so we'd expect **rax** to contain **9** after the instruction is executed:



Executing `ds` to move to the next instruction then executing `dr` to view register variable shows us we are correct:



The next few instructions involve moving the values in registers to the variables and vice versa:



After that, a string (which is the output is loaded into a register and the `printf` function is called in the *3rd* line. The second line clears the value of **eax** as **eax** is sometimes used to store results from functions. The *4th* line clears the value of **eax**. The *5th* and *6th* lines are used to exit the main function.

5. To finalise our workflow...

The general formula for working through something like this is:

1. set appropriate breakpoints
2. use ds to move through instructions and check the values of register and memory
3. if you make a mistake, you can always reload the program using the `ood` command

You may find this radare2 cheatsheet useful in your adventures...

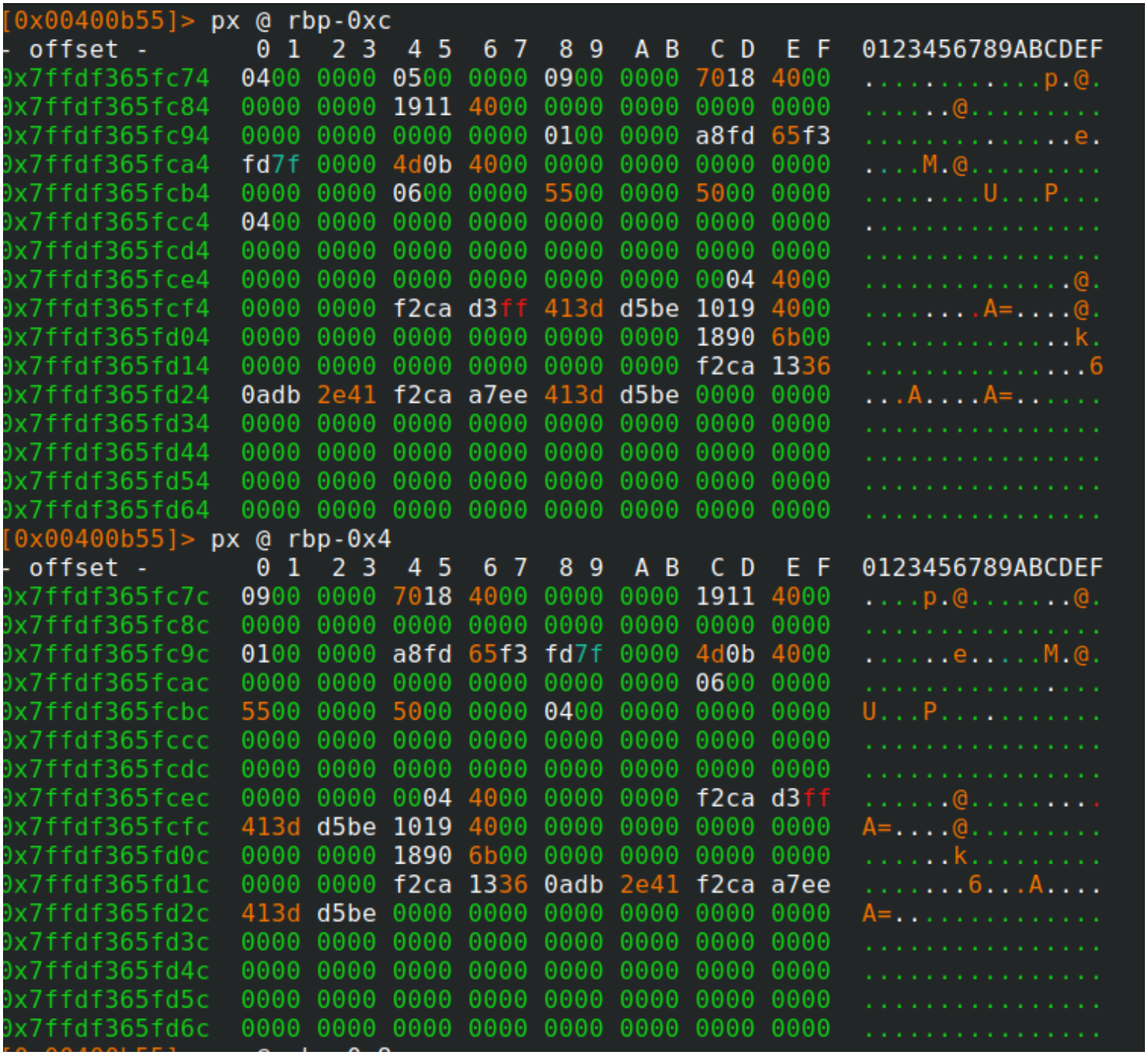
Steps:

```
elfmceager@tbfc-day-17:~$ r2 -d ./file1
Process with PID 1617 started...
= attach 1617 1617
bin.baddr 0x00400000
Using 0x400000
Warning: Cannot initialize dynamic strings
asm.bits 64
[0x00400a30]> aa
[ WARNING : block size exceeding max block size at 0x006ba220
[+] Try changing it with e anal.bb.maxsize
 WARNING : block size exceeding max block size at 0x006bc860
[+] Try changing it with e anal.bb.maxsize
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00400a30]> afl | grep main
0x00400b4d    1 68           sym.main
0x00400e10   10 1007 -> 219  sym.__libc_start_main
0x00403870   39 661  -> 629  sym._nl_find_domain
0x00403b10  308 5366 -> 5301 sym._nl_load_domain
0x00415fe0    1 43           sym._IO_switch_to_main_get_area
0x0044cf00    1 8            sym._dl_get_dl_main_map
0x00470520    1 49           sym._IO_switch_to_main_wget_area
0x0048fae0    7 73   -> 69   sym._nl_finddomain_subfreeres
0x0048fb30   16 247  -> 237  sym._nl_unload_domain
[0x00400a30]>
```

```
[0x00400a30]> pdf @main
            ;-- main:
/ (fcn) sym.main 68
|   sym.main ();
|           ; var int local_ch @ rbp-0xc
|           ; var int local_8h @ rbp-0x8
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x00400a4d (entry0)
|           0x00400b4d      55             push rbp
|           0x00400b4e      4889e5         mov rbp, rsp
|           0x00400b51      4883ec10       sub rsp, 0x10
|           0x00400b55      c745f4040000.  mov dword [local_ch], 4
|           0x00400b5c      c745f8050000.  mov dword [local_8h], 5
|           0x00400b63      8b55f4         mov edx, dword [local_ch]
|           0x00400b66      8b45f8         mov eax, dword [local_8h]
|           0x00400b69      01d0           add eax, edx
|           0x00400b6b      8945fc         mov dword [local_4h], eax
|           0x00400b6e      8b4dfc         mov ecx, dword [local_4h]
|           0x00400b71      8b55f8         mov edx, dword [local_8h]
|           0x00400b74      8b45f4         mov eax, dword [local_ch]
|           0x00400b77      89c6           mov esi, eax
|           0x00400b79      488d3d881409.  lea rdi, qword str.the_value_of_a_is__d__the_value_of_b_is
, the value of b is %d and the value of c is %d"
|           0x00400b80      b800000000     mov eax, 0
|           0x00400b85      e8f6ea0000     call sym.__printf
|           0x00400b8a      b800000000     mov eax, 0
|           0x00400b8f      c9             leave
\           0x00400b90      c3             ret
[0x00400a30]> db 0x00400b55
[0x00400a30]>
```


[0x00400a30]> pdf @main
;-- main:
/ (fcn) sym.main 68
sym.main ();
; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; var int local_4h @ rbp-0x4
; DATA XREF from 0x00400a4d (entry0)
0x00400b4d 55 push rbp
0x00400b4e 4889e5 mov rbp, rsp
0x00400b51 4883ec10 sub rsp, 0x10
0x00400b55 b c745f4040000. mov dword [local_ch], 4
0x00400b5c c745f8050000. mov dword [local_8h], 5
0x00400b63 8b55f4 mov edx, dword [local_ch]
0x00400b66 8b45f8 mov eax, dword [local_8h]
0x00400b69 01d0 add eax, edx
0x00400b6b 8945fc mov dword [local_4h], eax
0x00400b6e 8b4dfc mov ecx, dword [local_4h]
0x00400b71 8b55f8 mov edx, dword [local_8h]
0x00400b74 8b45f4 mov eax, dword [local_ch]
0x00400b77 89c6 mov esi, eax
0x00400b79 488d3d881409. lea rdi, qword str.the_value_of_a_is__d__the_value_of_b_is__d_a
, the value of b is %d and the value of c is %d"
0x00400b80 b800000000 mov eax, 0
0x00400b85 e8f6ea0000 call sym.__printf
0x00400b8a b800000000 mov eax, 0
0x00400b8f c9 leave
0x00400b90 c3 ret
[0x00400a30]> dc
hit breakpoint at: 400b55
[0x00400b55]>

```
[0x00400b55]> px  @ rbp-0x8
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7ffdf365fc78  1890 6b00 0000 0000 7018 4000 0000 0000  ..k.....p.@.....
0x7ffdf365fc88  1911 4000 0000 0000 0000 0000 0000 0000  ..@.............
0x7ffdf365fc98  0000 0000 0100 0000 a8fd 65f3 fd7f 0000  ..........e.....
0x7ffdf365fca8  4d0b 4000 0000 0000 0000 0000 0000 0000  M.@.............
0x7ffdf365fcb8  0600 0000 5500 0000 5000 0000 0400 0000  ....U...P.......
0x7ffdf365fcc8  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fcd8  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fce8  0000 0000 0000 0000 0004 4000 0000 0000  ..........@.....
0x7ffdf365fcf8  f2ca d3ff 413d d5be 1019 4000 0000 0000  ....A=....@.....
0x7ffdf365fd08  0000 0000 0000 0000 1890 6b00 0000 0000  ..........k.....
0x7ffdf365fd18  0000 0000 0000 0000 f2ca 1336 0adb 2e41  ...........6...A
0x7ffdf365fd28  f2ca a7ee 413d d5be 0000 0000 0000 0000  ....A=..........
0x7ffdf365fd38  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd48  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd58  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd68  0000 0000 0000 0000 0000 0000 0000 0000  ................
[0x00400b55]> ds
[0x00400b55]> ds
[0x00400b55]> ds
[0x00400b55]> px @ rbp-0x4
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7ffdf365fc7c  0000 0000 7018 4000 0000 0000 1911 4000  ....p.@.......@.
0x7ffdf365fc8c  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fc9c  0100 0000 a8fd 65f3 fd7f 0000 4d0b 4000  ......e.....M.@.
0x7ffdf365fcac  0000 0000 0000 0000 0000 0000 0600 0000  ................
0x7ffdf365fcbc  5500 0000 5000 0000 0400 0000 0000 0000  U...P...........
0x7ffdf365fccc  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fcdc  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fcec  0000 0000 0004 4000 0000 0000 f2ca d3ff  ......@.........
0x7ffdf365fcfc  413d d5be 1019 4000 0000 0000 0000 0000  A=....@.........
0x7ffdf365fd0c  0000 0000 1890 6b00 0000 0000 0000 0000  ......k.........
0x7ffdf365fd1c  0000 0000 f2ca 1336 0adb 2e41 f2ca a7ee  .......6...A....
0x7ffdf365fd2c  413d d5be 0000 0000 0000 0000 0000 0000  A=..............
0x7ffdf365fd3c  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd4c  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd5c  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd6c  0000 0000 0000 0000 0000 0000 0000 0000  ................
```


[0x00400b55]> px @ rbp-0xc
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x7ffdf365fc74 0400 0000 0500 0000 0900 0000 7018 4000 ....p.@.
0x7ffdf365fc84 0000 0000 1911 4000 0000 0000 0000 0000 ......@.........
0x7ffdf365fc94 0000 0000 0000 0000 0100 0000 a8fd 65f3 ....e.
0x7ffdf365fca4 fd7f 0000 4d0b 4000 0000 0000 0000 0000 ....M.@.........
0x7ffdf365fcb4 0000 0000 0600 0000 5500 0000 5000 0000 ........U...P...
0x7ffdf365fcc4 0400 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fcd4 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fce4 0000 0000 0000 0000 0000 0000 0004 4000 ....@.
0x7ffdf365fcf4 0000 0000 f2ca d3ff 413d d5be 1019 4000 ....A=....@.
0x7ffdf365fd04 0000 0000 0000 0000 0000 0000 1890 6b00 ....k.
0x7ffdf365fd14 0000 0000 0000 0000 0000 0000 f2ca 1336 6
0x7ffdf365fd24 0adb 2e41 f2ca a7ee 413d d5be 0000 0000 ...A....A=......
0x7ffdf365fd34 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fd44 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fd54 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fd64 0000 0000 0000 0000 0000 0000 0000 0000 ....
[0x00400b55]> px @ rbp-0x4
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x7ffdf365fc7c 0900 0000 7018 4000 0000 0000 1911 4000 ....p.@.......@.
0x7ffdf365fc8c 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fc9c 0100 0000 a8fd 65f3 fd7f 0000 4d0b 4000 ......e.....M.@.
0x7ffdf365fcac 0000 0000 0000 0000 0000 0000 0600 0000 ....
0x7ffdf365fcbc 5500 0000 5000 0000 0400 0000 0000 0000 U...P...........
0x7ffdf365fccc 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fcdc 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fcec 0000 0000 0004 4000 0000 0000 f2ca d3ff ......@.........
0x7ffdf365fcfc 413d d5be 1019 4000 0000 0000 0000 0000 A=....@.........
0x7ffdf365fd0c 0000 0000 1890 6b00 0000 0000 0000 0000 ......k.........
0x7ffdf365fd1c 0000 0000 f2ca 1336 0adb 2e41 f2ca a7ee ....6...A....
0x7ffdf365fd2c 413d d5be 0000 0000 0000 0000 0000 0000 A=..............
0x7ffdf365fd3c 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fd4c 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fd5c 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7ffdf365fd6c 0000 0000 0000 0000 0000 0000 0000 0000 ....

```
[0x00400b55]> px @ rbp-0x8
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7ffdf365fc78  0500 0000 0900 0000 7018 4000 0000 0000  ........p.@.....
0x7ffdf365fc88  1911 4000 0000 0000 0000 0000 0000 0000  ..@.............
0x7ffdf365fc98  0000 0000 0100 0000 a8fd 65f3 fd7f 0000  ..........e.....
0x7ffdf365fca8  4d0b 4000 0000 0000 0000 0000 0000 0000  M.@.............
0x7ffdf365fcb8  0600 0000 5500 0000 5000 0000 0400 0000  ....U...P.......
0x7ffdf365fcc8  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fcd8  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fce8  0000 0000 0000 0000 0004 4000 0000 0000  ..........@.....
0x7ffdf365fcf8  f2ca d3ff 413d d5be 1019 4000 0000 0000  ....A=....@.....
0x7ffdf365fd08  0000 0000 0000 0000 1890 6b00 0000 0000  ..........k.....
0x7ffdf365fd18  0000 0000 0000 0000 f2ca 1336 0adb 2e41  ...........6...A
0x7ffdf365fd28  f2ca a7ee 413d d5be 0000 0000 0000 0000  ....A=..........
0x7ffdf365fd38  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd48  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd58  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffdf365fd68  0000 0000 0000 0000 0000 0000 0000 0000  ................
[0x00400b55]> dr
rax = 0x00000004
rbx = 0x00400400
rcx = 0x00000009
rdx = 0x00000005
r8 = 0x00000000
r9 = 0x00000003
r10 = 0x00000002
r11 = 0x00000000
r12 = 0x00401910
r13 = 0x00000000
r14 = 0x006b9018
r15 = 0x00000000
rsi = 0x7ffdf365fda8
rdi = 0x00000001
rsp = 0x7ffdf365fc70
rbp = 0x7ffdf365fc80
rip = 0x00400b77
rflags = 0x00000206
orax = 0xffffffffffffffff
[0x00400b55]>
```

CHALLENGE:

6. Challenge

Use your new-found knowledge of Radare2 to **analyse the "challenge1" file** in the Instance `10.10.204.229` that is attached to this task to answer the questions below.

What is the value of **local_ch** when its corresponding movl instruction is called (first if multiple)?

1 — Correct Answer

What is the value of **eax** when the imull instruction is called?

6 — Correct Answer

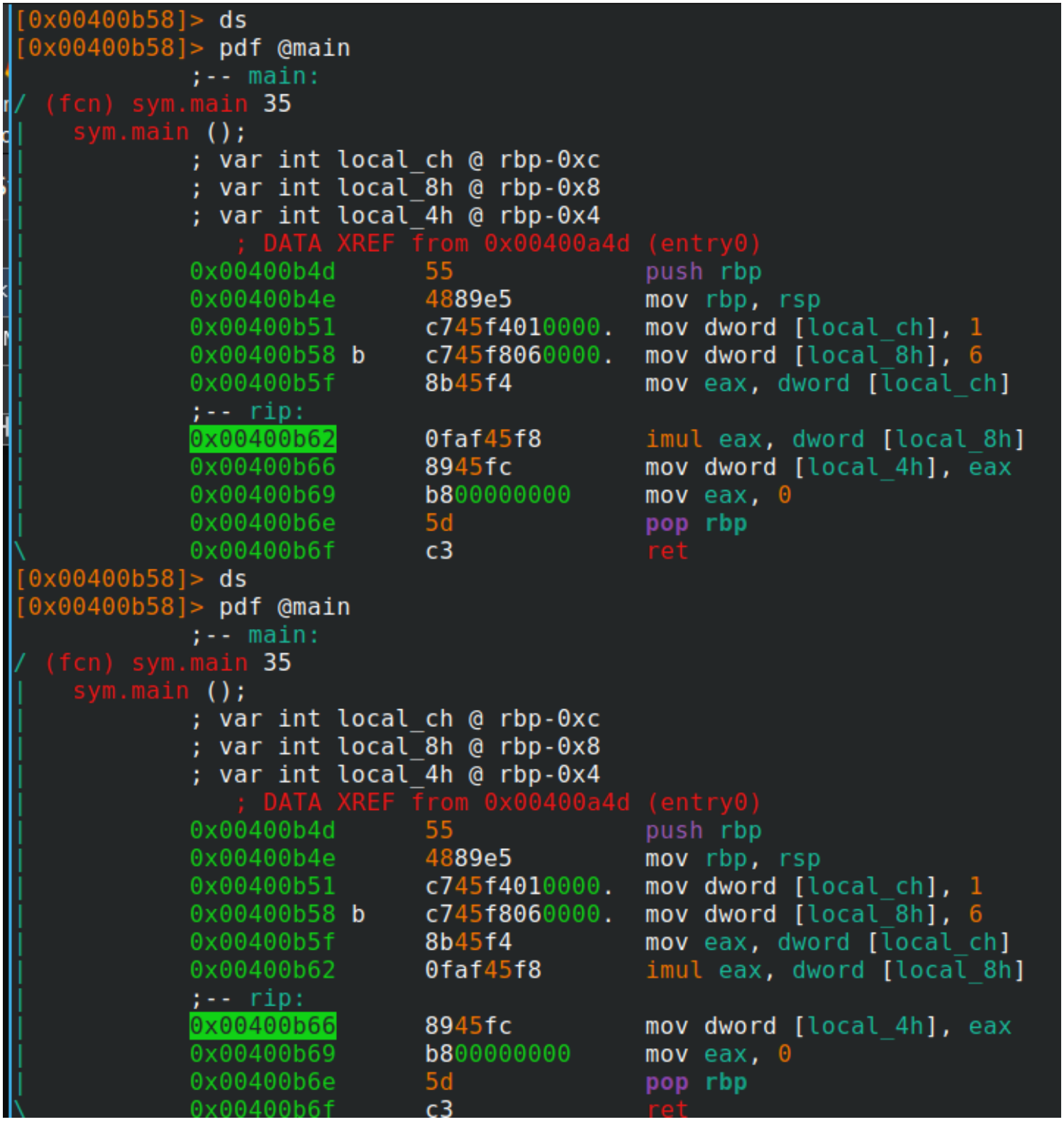
What is the value of **local_4h** before **eax** is set to 0?

6 — Correct Answer

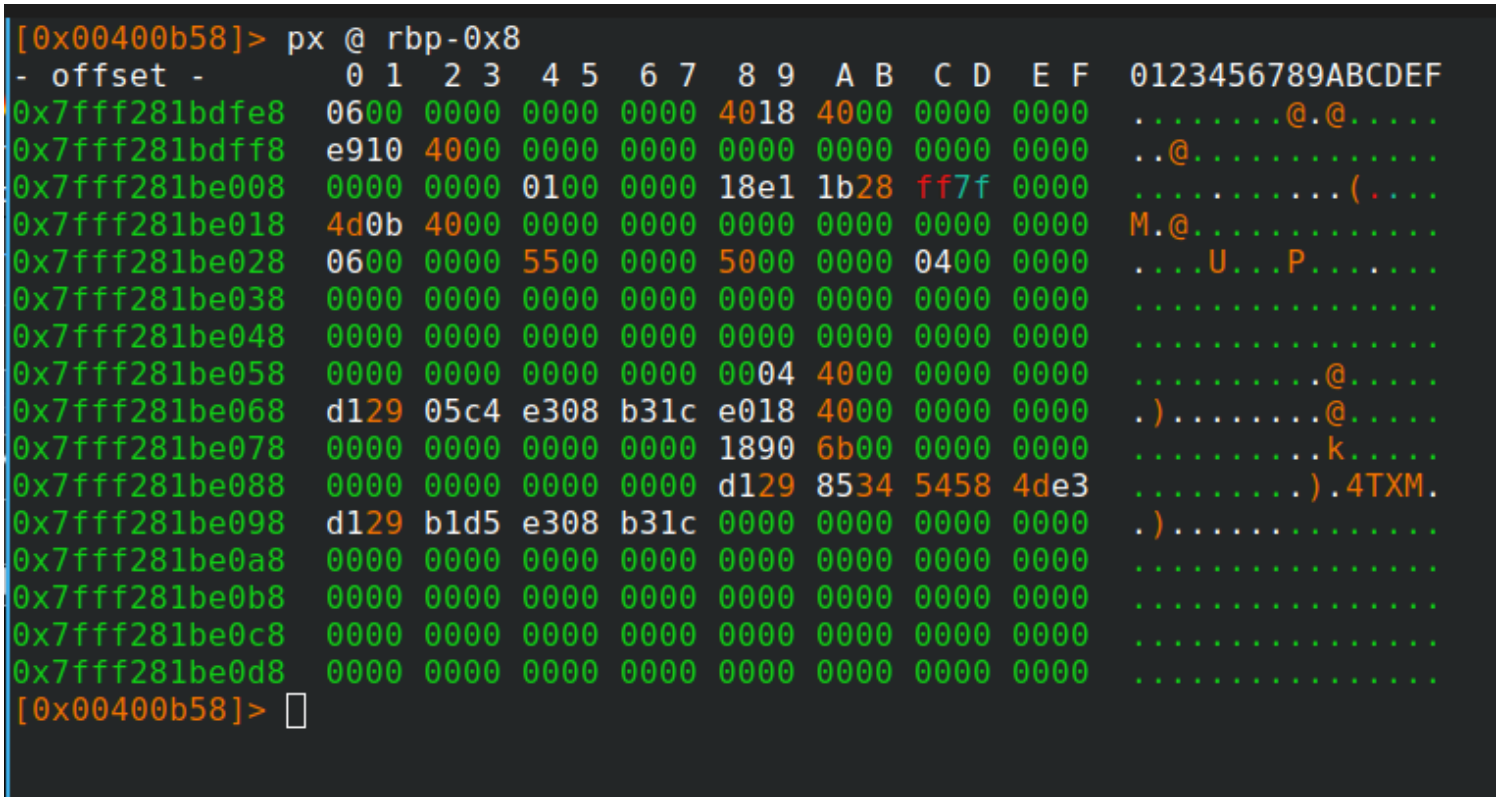
```
elfmceager@tbfc-day-17:~$ r2 -d ./challenge1
Process with PID 1633 started...
= attach 1633 1633
bin.baddr 0x00400000
Using 0x400000
Warning: Cannot initialize dynamic strings
asm.bits 64
[0x00400a30]> aa
[ WARNING : block size exceeding max block size at 0x006ba220
[+] Try changing it with e anal.bb.maxsize
 WARNING : block size exceeding max block size at 0x006bc860
[+] Try changing it with e anal.bb.maxsize
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00400a30]> afl | grep main
0x00400b4d    1 35           sym.main
0x00400de0   10 1007 -> 219  sym.__libc_start_main
0x00403840   39 661  -> 629  sym._nl_find_domain
0x00403ae0  308 5366 -> 5301 sym._nl_load_domain
0x00415ef0    1 43           sym._IO_switch_to_main_get_area
0x0044ce10    1 8            sym._dl_get_dl_main_map
0x00470430    1 49           sym._IO_switch_to_main_wget_area
0x0048f9f0    7 73   -> 69   sym._nl_finddomain_subfreeres
0x0048fa40   16 247  -> 237  sym._nl_unload_domain
[0x00400a30]> pdf @main
            ;-- main:
/ (fcn) sym.main 35
|   sym.main ();
|           ; var int local_ch @ rbp-0xc
|           ; var int local_8h @ rbp-0x8
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x00400a4d (entry0)
|           0x00400b4d      55             push rbp
|           0x00400b4e      4889e5         mov rbp, rsp
|           0x00400b51      c745f4010000.  mov dword [local_ch], 1
|           0x00400b58      c745f8060000.  mov dword [local_8h], 6
|           0x00400b5f      8b45f4         mov eax, dword [local_ch]
|           0x00400b62      0faf45f8       imul eax, dword [local_8h]
|           0x00400b66      8945fc         mov dword [local_4h], eax
|           0x00400b69      b800000000     mov eax, 0
|           0x00400b6e      5d             pop rbp
\           0x00400b6f      c3             ret
```

Ans 1:


[0x00400a30]> db 0x00400b58
[0x00400a30]> pdf @main
;-- main:
/ (fcn) sym.main 35
| sym.main ();
; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; var int local_4h @ rbp-0x4
; DATA XREF from 0x00400a4d (entry0)
0x00400b4d 55 push rbp
0x00400b4e 4889e5 mov rbp, rsp
0x00400b51 c745f4010000. mov dword [local_ch], 1
0x00400b58 b c745f8060000. mov dword [local_8h], 6
0x00400b5f 8b45f4 mov eax, dword [local_ch]
0x00400b62 0faf45f8 imul eax, dword [local_8h]
0x00400b66 8945fc mov dword [local_4h], eax
0x00400b69 b800000000 mov eax, 0
0x00400b6e 5d pop rbp
\ 0x00400b6f c3 ret
[0x00400a30]> dc
hit breakpoint at: 400b58
[0x00400b58]> px @ rbp-0xc
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x7fff281bdfe4 0100 0000 1890 6b00 0000 0000 4018 4000 ......k.....@.@.
0x7fff281bdff4 0000 0000 e910 4000 0000 0000 0000 0000 ......@.........
0x7fff281be004 0000 0000 0000 0000 0100 0000 18e1 1b28 ....(
0x7fff281be014 ff7f 0000 4d0b 4000 0000 0000 0000 0000 ....M.@.........
0x7fff281be024 0000 0000 0600 0000 5500 0000 5000 0000 ........U...P...
0x7fff281be034 0400 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be044 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be054 0000 0000 0000 0000 0000 0000 0004 4000 ....@.
0x7fff281be064 0000 0000 d129 05c4 e308 b31c e018 4000 .....).........@.
0x7fff281be074 0000 0000 0000 0000 0000 0000 1890 6b00 ....k.
0x7fff281be084 0000 0000 0000 0000 0000 0000 d129 8534 ....).4
0x7fff281be094 5458 4de3 d129 b1d5 e308 b31c 0000 0000 TXM..).........
0x7fff281be0a4 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be0b4 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be0c4 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be0d4 0000 0000 0000 0000 0000 0000 0000 0000 ....
[0x00400b58]>


[0x00400b58]> ds
[0x00400b58]> pdf @main
;-- main:
/ (fcn) sym.main 35
| sym.main ();
; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; var int local_4h @ rbp-0x4
; DATA XREF from 0x00400a4d (entry0)
0x00400b4d 55 push rbp
0x00400b4e 4889e5 mov rbp, rsp
0x00400b51 c745f4010000. mov dword [local_ch], 1
0x00400b58 b c745f8060000. mov dword [local_8h], 6
0x00400b5f 8b45f4 mov eax, dword [local_ch]
;-- rip:
0x00400b62 0faf45f8 imul eax, dword [local_8h]
0x00400b66 8945fc mov dword [local_4h], eax
0x00400b69 b800000000 mov eax, 0
0x00400b6e 5d pop rbp
\ 0x00400b6f c3 ret
[0x00400b58]> ds
[0x00400b58]> pdf @main
;-- main:
/ (fcn) sym.main 35
| sym.main ();
; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; var int local_4h @ rbp-0x4
; DATA XREF from 0x00400a4d (entry0)
0x00400b4d 55 push rbp
0x00400b4e 4889e5 mov rbp, rsp
0x00400b51 c745f4010000. mov dword [local_ch], 1
0x00400b58 b c745f8060000. mov dword [local_8h], 6
0x00400b5f 8b45f4 mov eax, dword [local_ch]
0x00400b62 0faf45f8 imul eax, dword [local_8h]
;-- rip:
0x00400b66 8945fc mov dword [local_4h], eax
0x00400b69 b800000000 mov eax, 0
0x00400b6e 5d pop rbp
\ 0x00400b6f c3 ret

Ans 2:


[0x00400b58]> px @ rbp-0x8
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x7fff281bdfe8 0600 0000 0000 0000 4018 4000 0000 0000 ....@.@.....
0x7fff281bdff8 e910 4000 0000 0000 0000 0000 0000 0000 ..@.............
0x7fff281be008 0000 0000 0100 0000 18e1 1b28 ff7f 0000 ....(....
0x7fff281be018 4d0b 4000 0000 0000 0000 0000 0000 0000 M.@.............
0x7fff281be028 0600 0000 5500 0000 5000 0000 0400 0000 ....U...P.......
0x7fff281be038 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be048 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be058 0000 0000 0000 0000 0004 4000 0000 0000 ....@.....
0x7fff281be068 d129 05c4 e308 b31c e018 4000 0000 0000 .)........@.....
0x7fff281be078 0000 0000 0000 0000 1890 6b00 0000 0000 ....k.....
0x7fff281be088 0000 0000 0000 0000 d129 8534 5458 4de3 ....).4TXM.
0x7fff281be098 d129 b1d5 e308 b31c 0000 0000 0000 0000 .)..............
0x7fff281be0a8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be0b8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be0c8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x7fff281be0d8 0000 0000 0000 0000 0000 0000 0000 0000 ....
[0x00400b58]>

Ans 3:

Note : using radare 2 we can convert any binary file to assembly too, you can see that with this C program binary file :

```
elfmceager@tbfc-day-17:~$ cat hello.c
#include <stdio.h>

int main() {
   int a, b;

   a = 11;
   b = 99;

   // to take values from user input uncomment the below lines -
   // printf("Enter value for A :");
   // scanf("%d", &a);
   // printf("Enter value for B :");
   // scanf("%d", &b);

   if(a > b)
      printf("a is greater than b");
   else
      printf("a is not greater than b");

   return 0;
}
elfmceager@tbfc-day-17:~$ gcc hello.c
elfmceager@tbfc-day-17:~$ r2 -d ./a.out
Process with PID 1697 started...
= attach 1697 1697
bin.baddr 0x5603b0b6c000
Using 0x5603b0b6c000
asm.bits 64
[0x7f156c3bf090]> aa
[ WARNING : block size exceeding max block size at 0x5603b0d6cfe0
[+] Try changing it with e anal.bb.maxsize
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x7f156c3bf090]>
```

```
[0x00400b58]> pdf @main
            ;-- main:
[0x7f156c3bf090]> afl
0x5603b0b6c000    3 72   -> 73   sym.imp.__libc_start_main
0x5603b0b6c4f0    3 23           sym._init
0x5603b0b6c520    1 6            sym.imp.printf
0x5603b0b6c530    1 6            fcn.5603b0b6c530
0x5603b0b6c540    1 43           entry0
0x5603b0b6c570    4 50   -> 40   sym.deregister_tm_clones
0x5603b0b6c5b0    4 66   -> 57   sym.register_tm_clones
0x5603b0b6c600    4 49           sym.__do_global_dtors_aux
0x5603b0b6c640    1 10           entry1.init
0x5603b0b6c64a    4 73           main
0x5603b0b6c6a0    4 101          sym.__libc_csu_init
0x5603b0b6c710    1 2            sym.__libc_csu_fini
0x5603b0b6c714    1 9            sym._fini
0x5603b0d6cfe0    1 1020         reloc.__libc_start_main_224
[0x7f156c3bf090]> pdf @ main
            ;-- main:
/ (fcn) main 73
|   main ();
|           ; var int local_8h @ rbp-0x8
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x5603b0b6c55d (entry0)
|           0x5603b0b6c64a      55             push rbp
|           0x5603b0b6c64b      4889e5         mov rbp, rsp
|           0x5603b0b6c64e      4883ec10       sub rsp, 0x10
|           0x5603b0b6c652      c745f80b0000.  mov dword [local_8h], 0xb ; 11
|           0x5603b0b6c659      c745fc630000.  mov dword [local_4h], 0x63 ; 'c' ; 99
|           0x5603b0b6c660      8b45f8         mov eax, dword [local_8h]
|           0x5603b0b6c663      3b45fc         cmp eax, dword [local_4h]
|       ,=< 0x5603b0b6c666      7e13           jle 0x5603b0b6c67b
|       |   0x5603b0b6c668      488d3db50000.  lea rdi, qword str.a_is_greater_than_b ; 0x5603b0b6c724 ; "a is greater than b"
|       |   0x5603b0b6c66f      b800000000     mov eax, 0
|       |   0x5603b0b6c674      e8a7feffff     call sym.imp.printf     ; int printf(const char *format)
|      ,==< 0x5603b0b6c679      eb11           jmp 0x5603b0b6c68c
|      |`-> 0x5603b0b6c67b      488d3db60000.  lea rdi, qword str.a_is_not_greater_than_b ; 0x5603b0b6c738 ; "a is not greater
|      |    0x5603b0b6c682      b800000000     mov eax, 0
|      |    0x5603b0b6c687      e894feffff     call sym.imp.printf     ; int printf(const char *format)
|      |       ; JMP XREF from 0x5603b0b6c679 (main)
|      `--> 0x5603b0b6c68c      b800000000     mov eax, 0
```