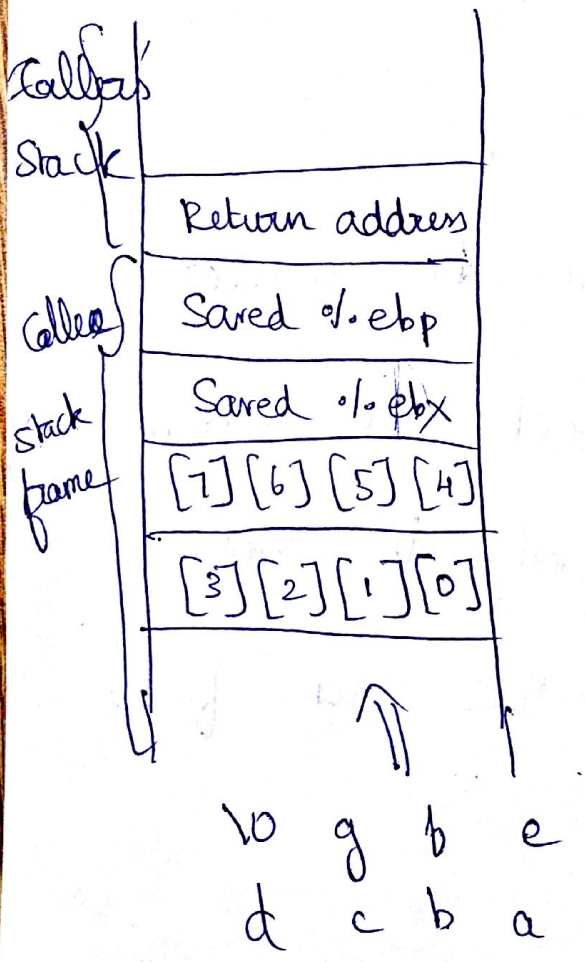
Today

① Buffer Overflow. ✓

② Assembly wrap-up.

→ No bounds-checking for array references.

```
int a[5];
printf("%d", a[10]); //
```




```
1: char buf[8];
2: gets(buf);
```

If I enter something that has 7 characters,

a b c d e f g \0

What will be corrupted

0 – 7	None.
8 – 11	Saved value of %ebx
12 – 15	Saved value of ebp
16 – 19	Return address
20 +	— —

Serious issue !

① Program will break.

② Security issue !

We know that → when the callee returns by using the ret instruction, it basically returns control to the address that's stored on the stack frame.

What if I am an attacker?

I just the overwrite the return address.

→ Load my own code (Exploit code)

→ Modify the return address so that it returns to the exploit code.

→ Now the exploit code is executing ~~code~~ ~~is~~ executing under the guise of your non-malicious code.

→ read materials (private)

→ get priviledged access.

→ start another program that basically gives access to some remote computer.

1988

Morris Worm (Robert Morris).

(Stack smashing)

How could we avoid this?

① Do not use vulnerable library functions.

gets ✗

fgets ✓ → Discards the excess chars entered.

② We could make the OS better!

The stack structure looks exactly the same on different runs the program.

(2a) Stack Randomization!

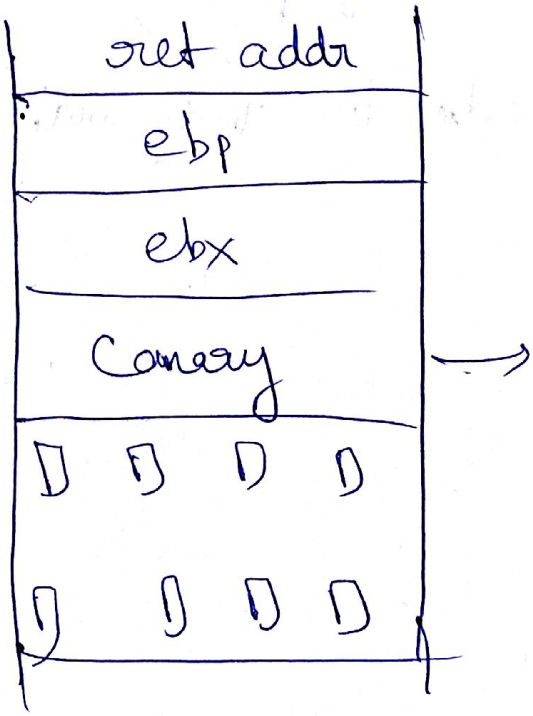
→ making the starting address of the stack vary between different

runs of the program.

(2b) Add a stack protector!

↓

Canary value.



→ random number.

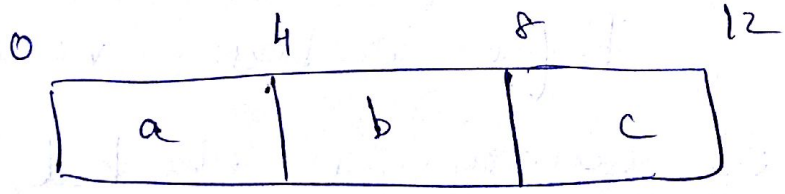
Before return, value of canary is checked.

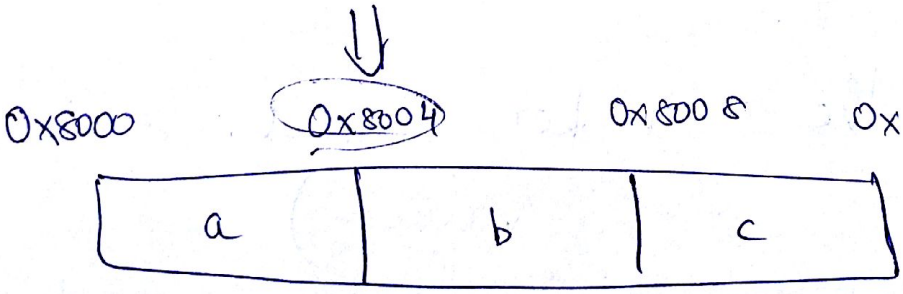
(Remember: the attacker cannot read values).

Review !

Struct Addressing

```
Struct s1
{
   int a;
char ~~int~~ b;
   int c;
};
```



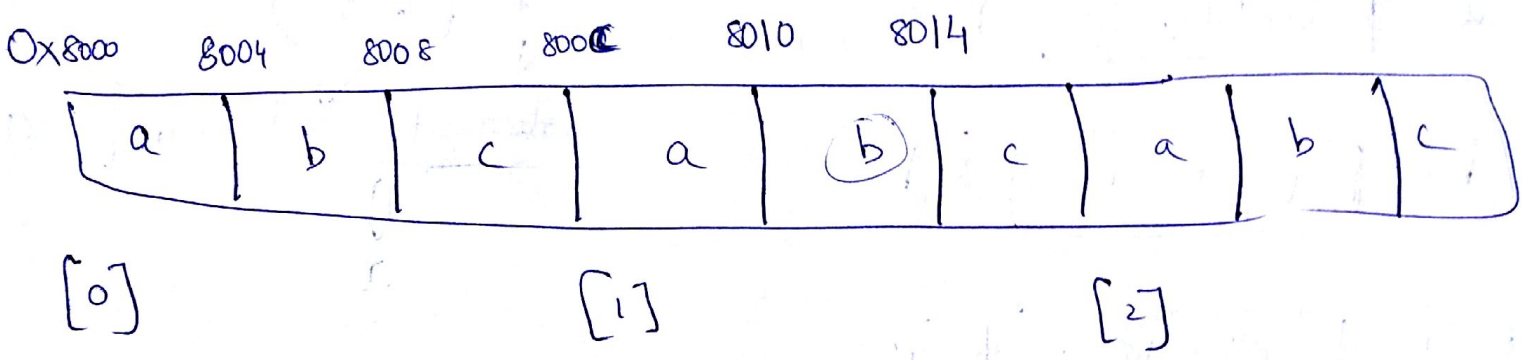



```
struct s1 x;
printf("%p", &x.b)   => 0x8004
printf("%p", &x.a)   => 0x8000
```

Things to Review

1. Conversion (Hex to decimal)
   (Decimal to binary)
2. 2.4 (Floating point) — Study only what we discussed in class
3. No needs to study Gdb.
4.

struct s1 arr [3] ;



printf ("%p", &arr [1].b) ; =>0x8010

---

(2) Array addressing

Assume var a starting at memory location

0x80490000

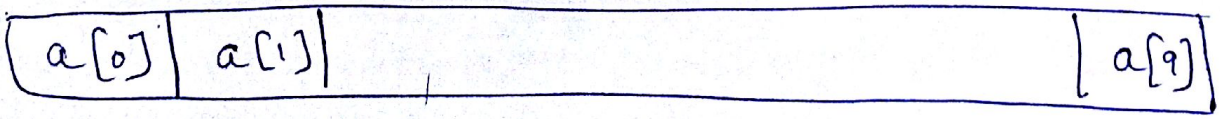
1: int a [10] ;

2: int *p = &a [7] ;

What is the value of p ?

0x8049000 → 0x8049004

0x804901C



Element at index at 7 = Start Address + 4 x 7

↓ ↓ ↓

A L i

(3) · Assembly to C.

% eax holds & a (address of a)

% ebx holds & b (address of b).

1: movl (%eax), %esi => *(& a) => a

2: movl (%esi), %edi => *(a)

~~3: movl (~~

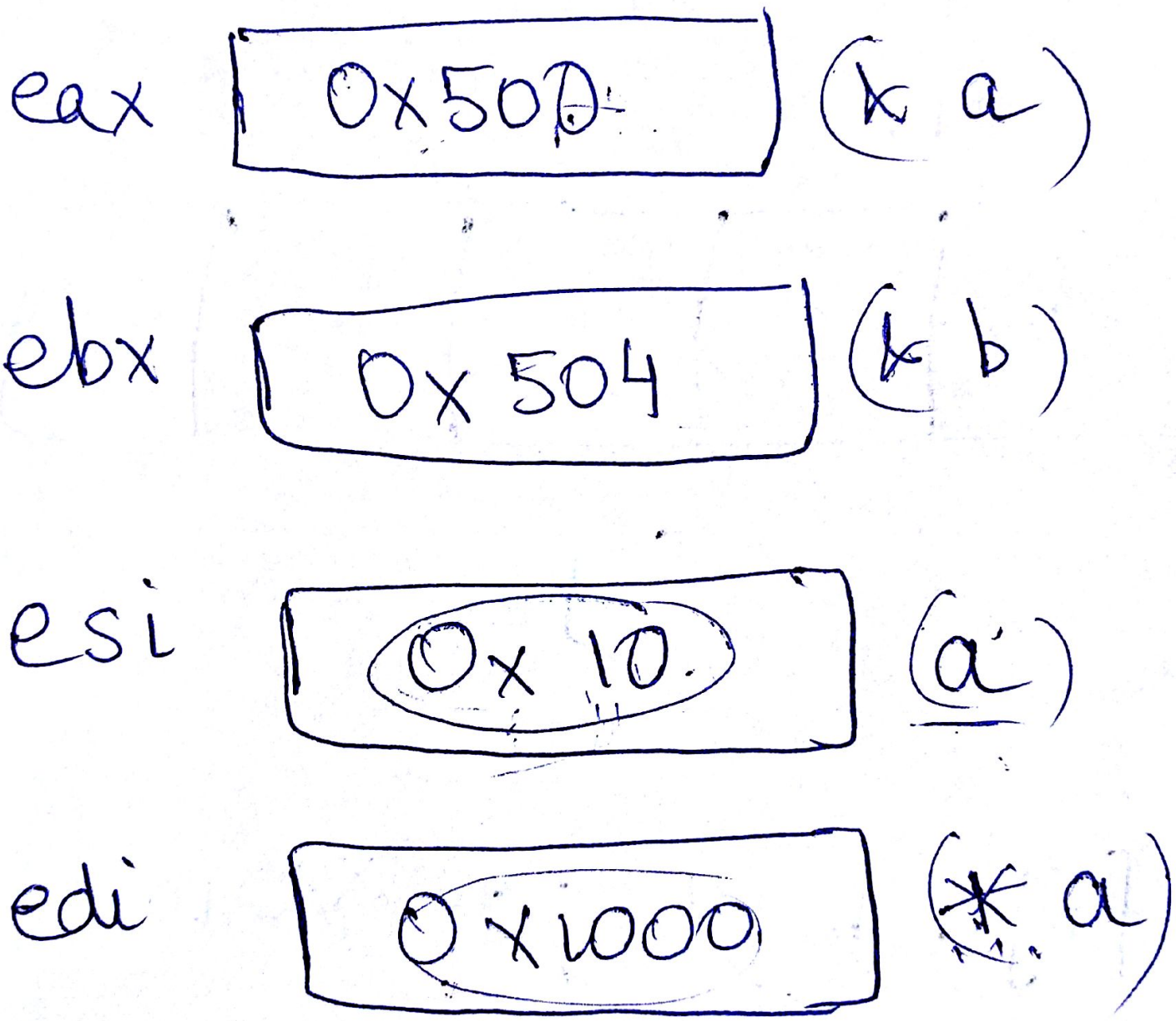
3: movl %edi, (%ebx) => (* a) => *(& b)

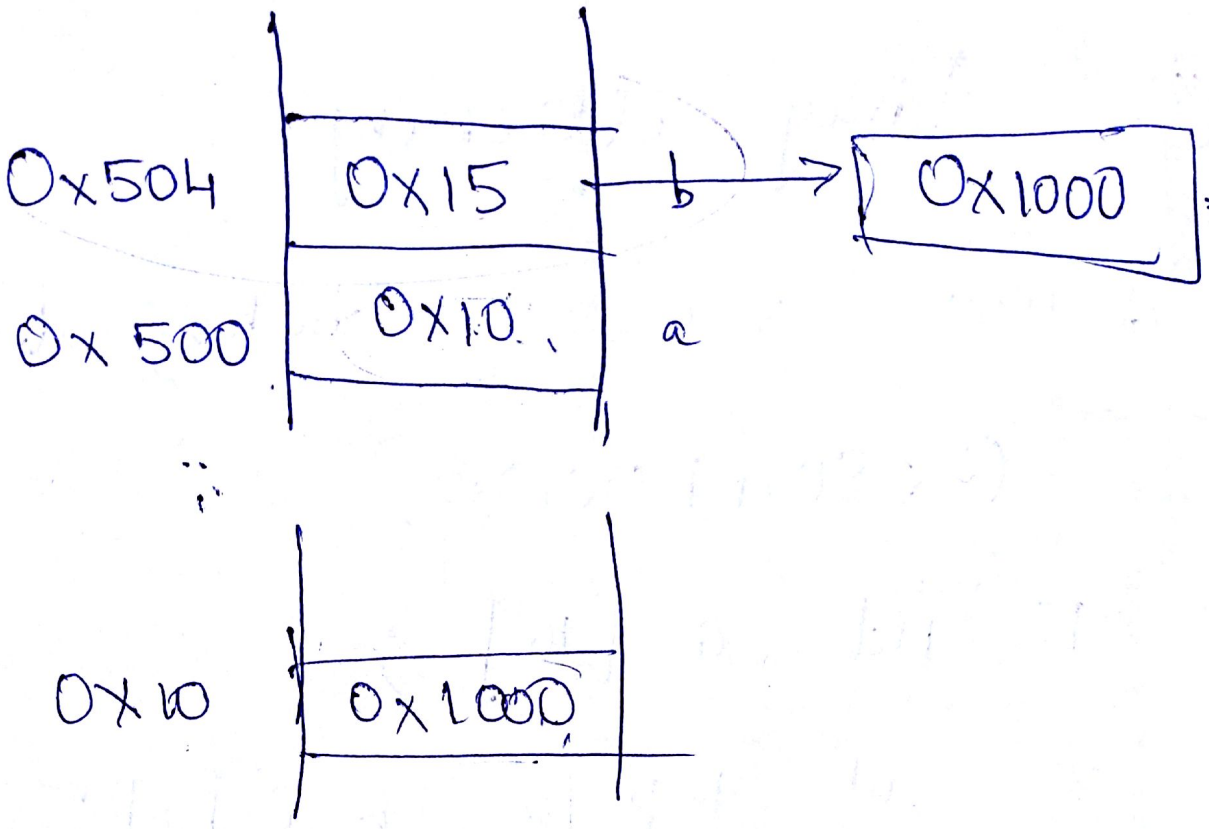
*a => b

What is the equivalent C statement(s) for this set of instructions?

1: move the value at the address stored in %eax to %esi

Check email for additional info about this problem!





b = *a;