Lecture 12: ROP & Review

January 27, 2020 Chris Stone

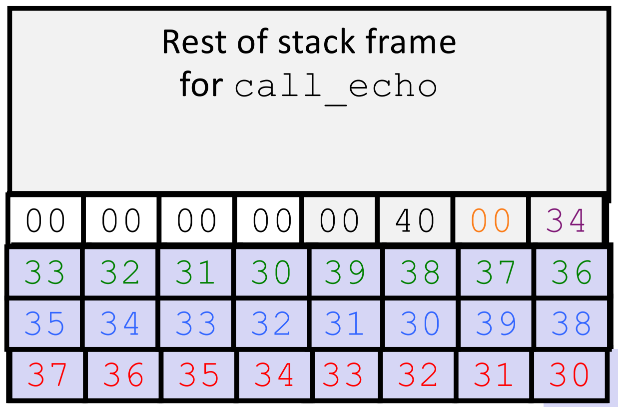
Lab 3 (Bomb) Due 1:15pm Tomorrow

Lab 4 (Attack) Starts Tomorrow — New Partner!

Take-Home Midterm available by 5pm Tomorrow Afternoon (75-minute exam due 5pm next Friday)

Security: The Story So Far

Observation




```
unix> ./bufdemo-nsp
Type a string:012345678901234567890123 4
Segmentation Fault
```

The program crashed because the code "returned" (jumped) to address 0x400034, which didn't contain valid machine code.

And by typing in a carefully-chosen 32-character string, we can make echo() "return" (jump) to any address we want!

Code Injection Attacks

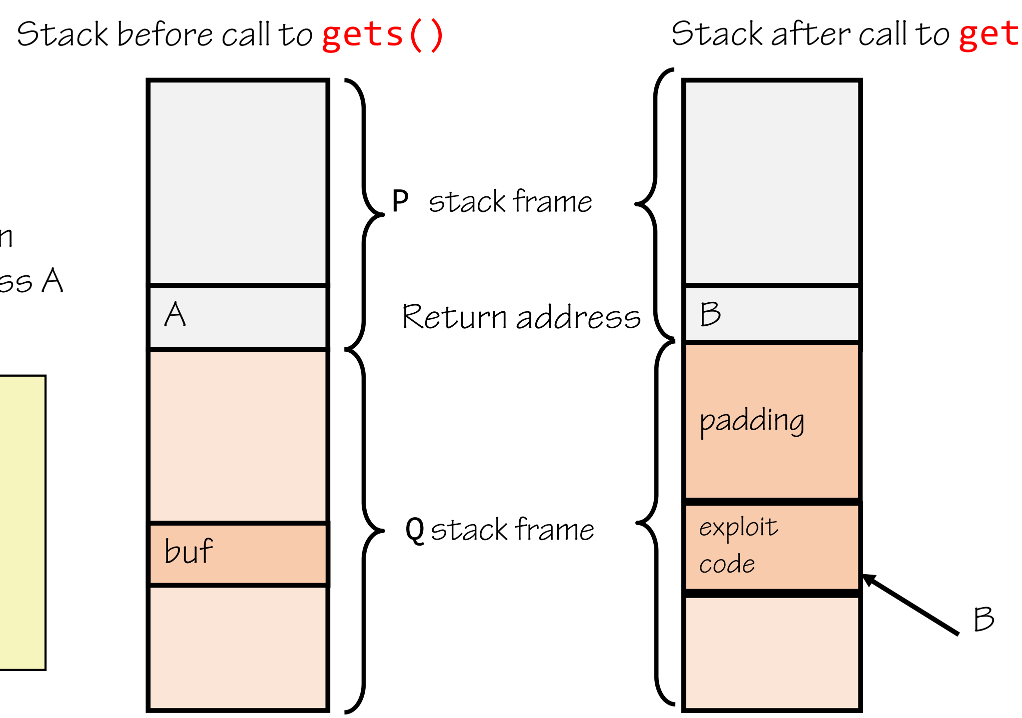
Input string includes bytes encoding machine code

Overwrite return address A with address of that code!

```
void P(){
  Q();
  ...
}
```

Return address A

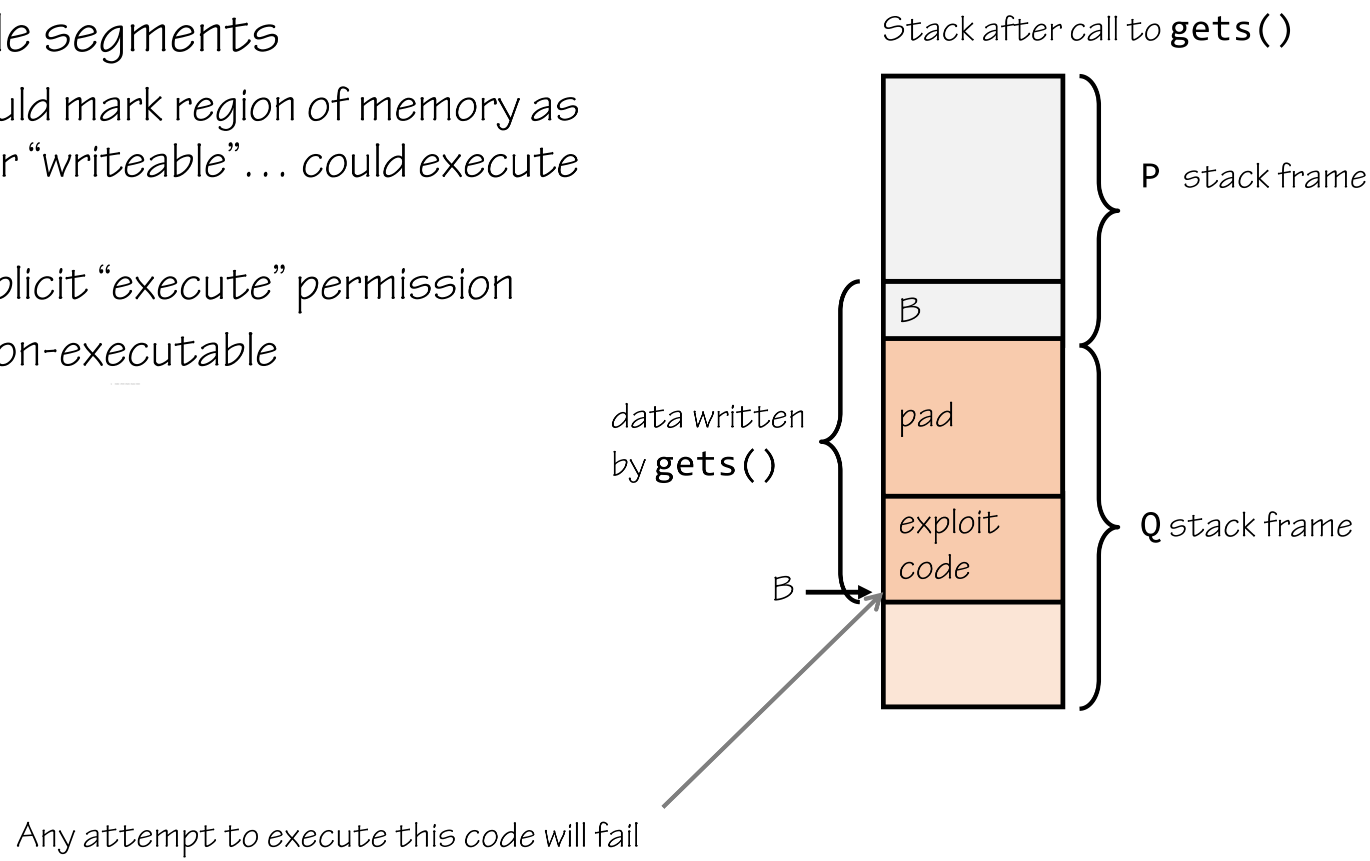
```
int Q() {
  char buf[64];
  gets(buf);
  ...
  return ...;
}
```



What happens when Q returns?

2. System-Level Protections can help

- Non-executable code segments
  - In previous x86, could mark region of memory as either "read-only" or "writeable"… could execute anything readable
  - X86-64 added explicit "execute" permission
  - Stack marked as non-executable



Are We Still in Danger?

If the stack is marked "don't execute"

- we can't write machine code into the buffer and jump to it.
- but we can still overwrite the return address
- we can force a "return" (jump!) anywhere in the code that is running.

Is that really so bad?

Yes!

Question 1

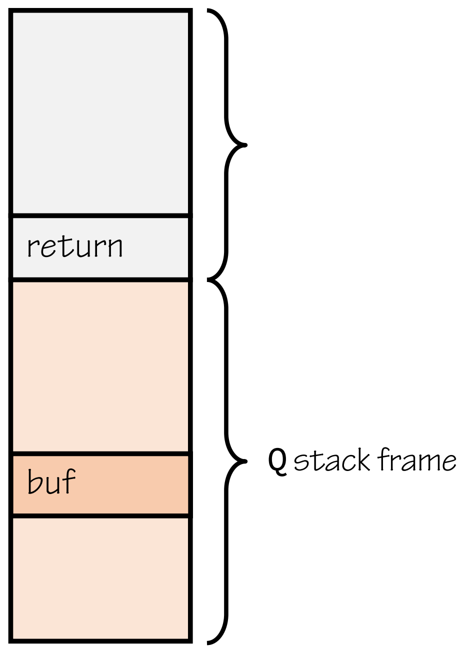
There are *lots* of instructions in a typical program.

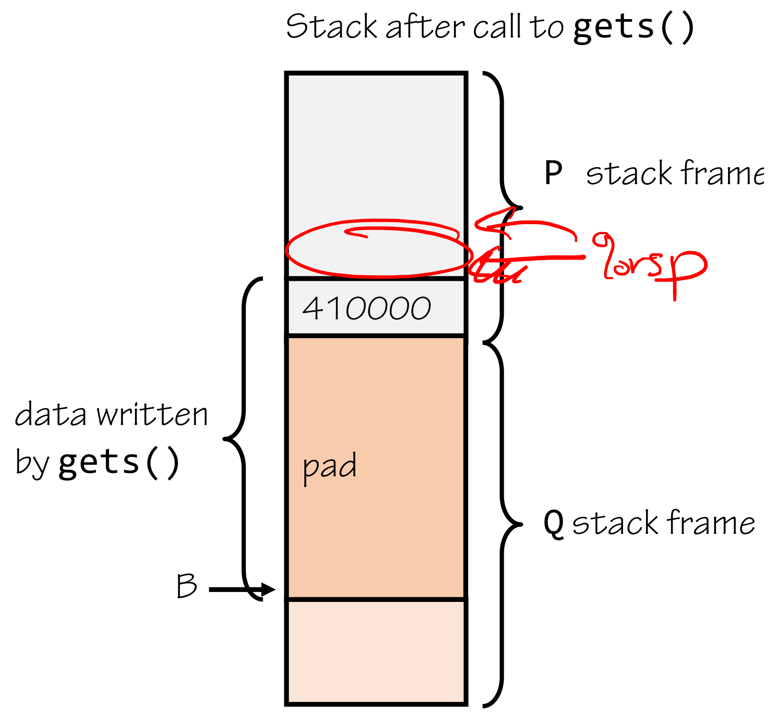
Suppose that at address 0x410000 there are two consecutive instructions

```
inc %ebp
ret
```

Suppose we overwrite the return address with 0x410000.

What happens when function Q returns?





Question 2

There are *lots* of instructions in a typical program.

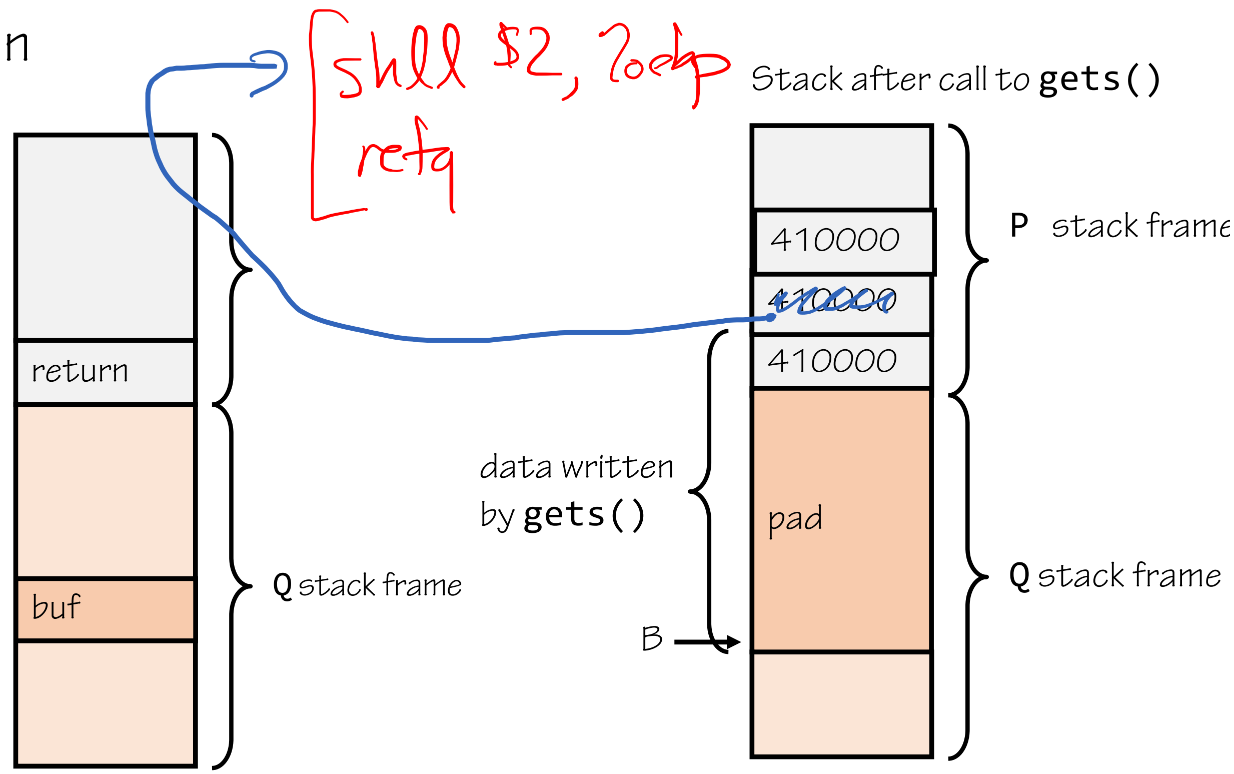
Suppose that at address 0x410000 there are two consecutive instructions

0x45 `incl %ebp`

0xC3 `retq`

Suppose we overwrite the return address with three copies of 0x410000

What happens when function Q returns?



Return-Oriented Programming (ROP)

Idea:

- Find *existing* machine code instructions followed by retq
  (These are called *gadgets*)
- Put a sequence of gadgets addresses on the stack.
  (where the sequence of gadgets does our evil work)

The computer returns (jumps) from each gadget to the next!

- It reads addresses from the stack, but executes code in the text segment.

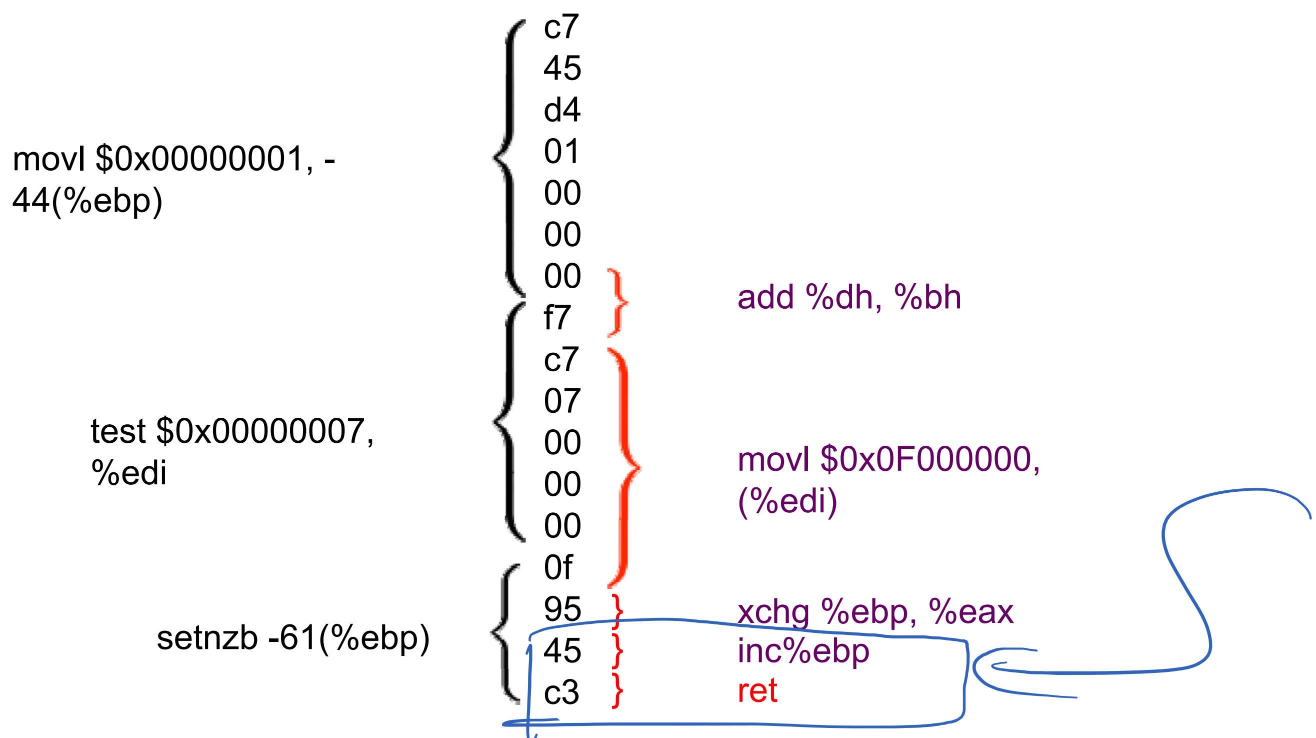
But most of our retq instructions immediately follow addq $..., %rsp.

- Can attacker find enough gadgets to do evil?

Yes!

We don't need **retq**; we need **0xc3** !

Unintended instructions — ecb_crypt()

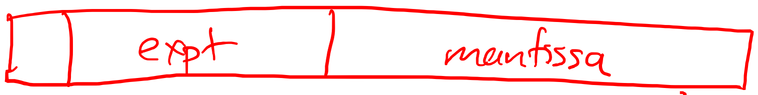


https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf

Have Fun with Lab 4!

Review Topics

- Bits
- And/Or/Not/Xor
- Arithmetic & logical shifts
- Integers
  - Unsigned ints
  - 2's complement
  - Max/min values
- Negating a signed int
- Signed/unsigned compare
- Zero- vs. sign-extension
- Casting
- Overflow
- Mult/Div vs. Shifting
- IEEE float & double
- Normal, special, and denormal fp numbers
- Memory vs. registers
- Machine code vs. assembly
- x86 assembly
  - arithmetic
  - movq vs. leaq
  - comparisons
  - condition codes
  - conditional jumps
  - conditional moves
- Implementing if, do, while loops using jumps & labels
- Stack frames & %rsp
- Return address
- Arrays, Structs, Unions
  - Padding/alignment
- Buffer overflows
  - Identifying
  - Security implications
  - Prevention techniques



single 32-bit
double 64-bit

ign
= +
= −

$1.\text{mantissa} \times 2^{\text{expt} - \text{“bias”}}$

$\pm\ 1.\_\_\_\_ \times 2^{+127}$
$\pm\ 1.\_\_\_\_ \times 2^{-126}$ through

normals

t : if expt is all 0's then

$\pm\ 0.\underline{\text{mantissa}} \times 2^{-126}$

denormals

if expt is all 1's then
+∞, −∞, or NaN, depending on mantissa bits.

special

if $x \geq 0$ then $-x \leq 0$ ✓

- For 8 bit signed

  $-128 \leq x \leq 127$

if $x \leq 0$ then $-x \geq 0$ ✗

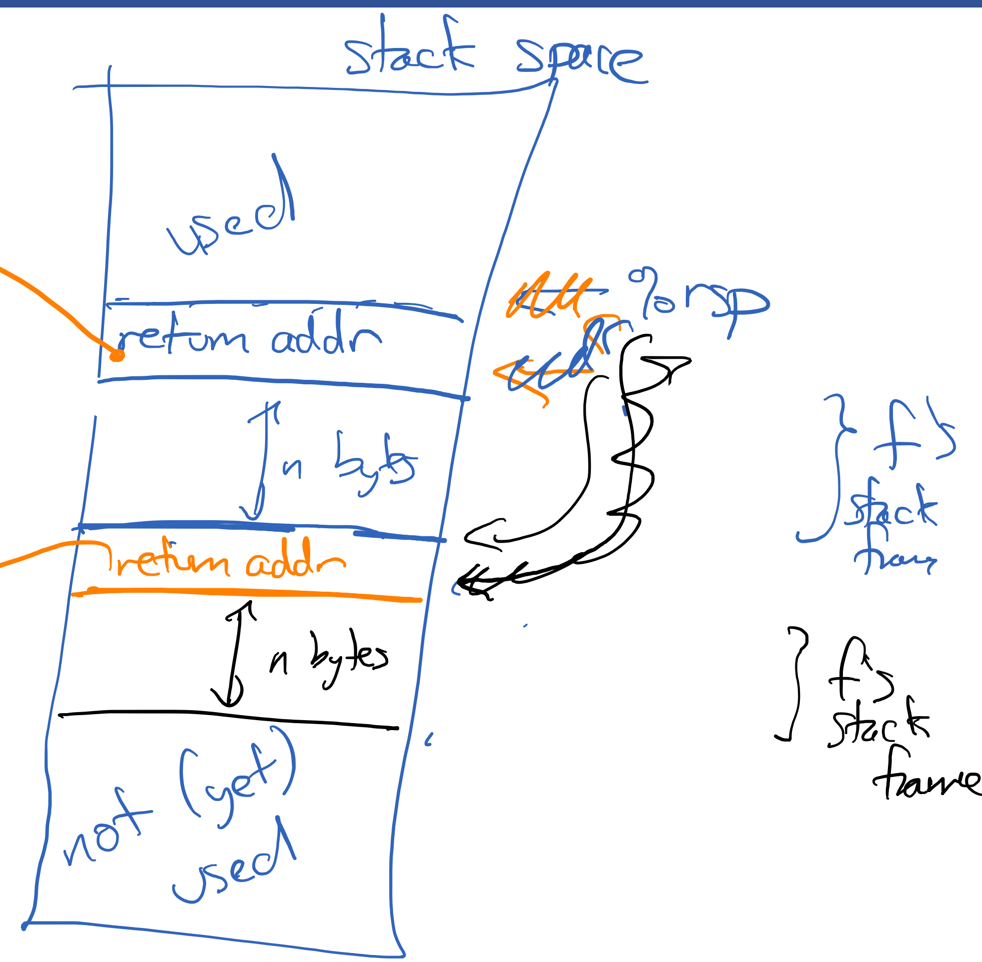
$-(-128) == -128$

n bits

unsigned	signed
$U_{max} = 2^n - 1$	$T_{max} = 2^{n-1} - 1$
⋮	⋮ 0 ⋮
$U_{min} = 0$	$T_{min} = -2^{n-1}$

```
callq f

code for f:
  subq $n, %rsp
  :
    callq f
  addq $n, %rsp
  retq
```



caller-save registers – functions are allowed to use these w/o putting them back the way they

callee-save registers – functions must put the original values back before they return.

Floats

sign	"expt"	"mantissa"

sign
0 = +
1 = −

Single-Precision: 32 bits total

Double: 64 bits

Normals: $\pm 1.\text{"mantissa"} \times 2^{\text{"expt"} - \text{"bias"}}$

Denormals: $\pm 0.\text{"mantissa"} \times 2^{1 - \text{"bias"}}$

if "expt" = 0000000..

Single:

$\pm 1.\_\_\_\_ \times 2^{127}$

↕

$\pm 1.\_\_\_\_ \times 2^{-126}$

$\pm 0.\_\_\_\_ \times 2^{-126}$

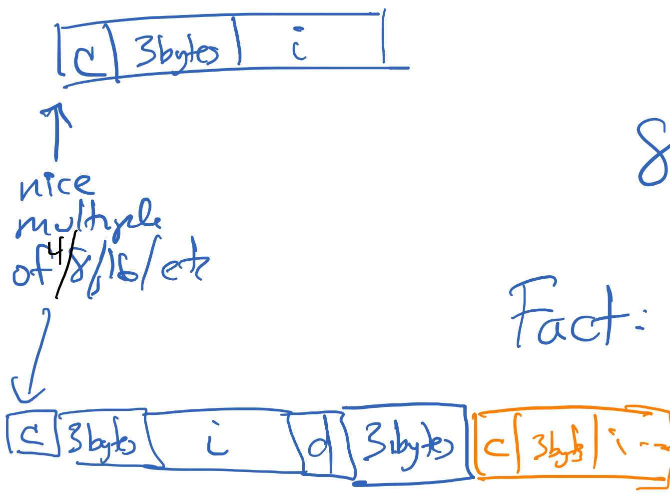
Special (if "expt" = 1111...1)

$+\infty$, $-\infty$, NaN depending on "mantissa" bit pattern.

Structs & alignment

```
struct {
    char c;
    int i;
};
```

```
struct
    char c;
    int i;
    char d;
};
```

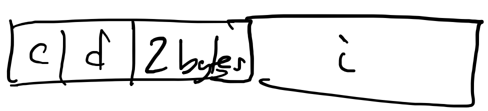


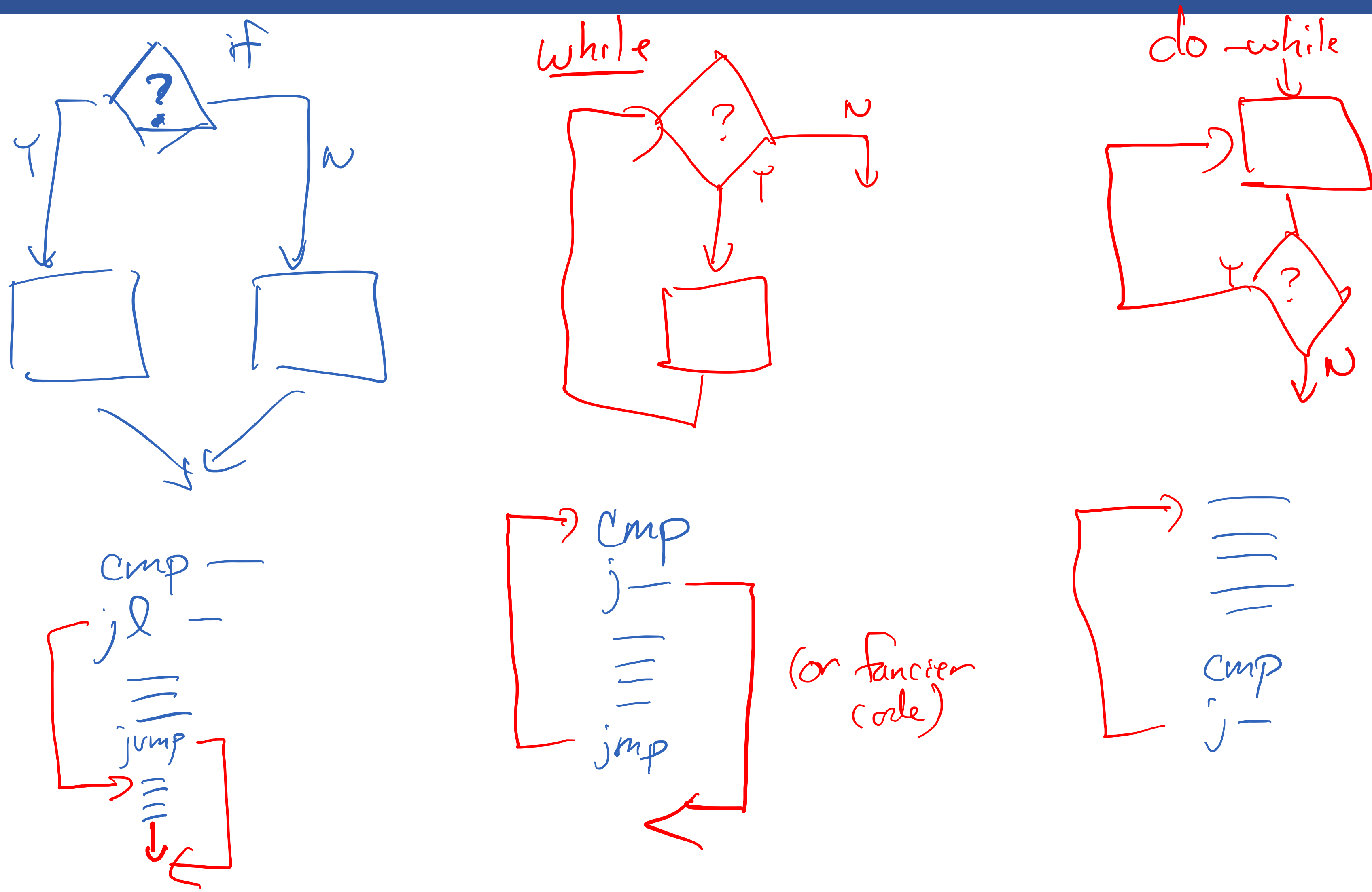
Goal: 4 byte primitive data should live/start at an address that is a multiple of 4

8-byte primitive data ... ... multiple of 8

Fact: structs will be placed so they start at a nice address (multiple of 4, 8, 16, ...)

```
struct {
    char c;
    char d;
    int i
}
```





if
?
Y
N
cmp
jl
jump
while
?
N
Y
cmp
j
jmp
(or fancier code)
do-while
?
Y
N
cmp
j