CS33 Discussion 4

Brandon Wu
10.31.2014

HW2: Shifting and Dividing

- y = x / (2^k)
- y = x >> k works sometimes
  - When x is positive
  - When x is negative but requires no rounding
- (x<0 ? x+(1<<k-1) : x) >> k;

Useful Functions for Lab2

void* malloc(unsigned num_bytes);

- Allocates size num_bytes chunk in memory
- Use free(void* ptr) to deallocate

memcpy(void* dest, void* src, unsinged num_bytes);

- Copies num_bytes Bytes from src to dest

Register Saving Convention

- Example Procedure P() calls Q()
- Caller saved registers:
  - %eax, %edx, %ecx
  - P must save these to stack before calling Q
- Callee saved registers
  - %ebx, %esi, %edi
  - Q must save these to stack before changing
  - Q must restore these before exiting

Swap Example

```
void swap(int* x, int* y) {
        int temp = *x;
        *x = *y;
        *y = temp;
}
```

Swap Disassembled

- The stack setup...

```
0000006c <swap>:
  6c:   55                      push   %ebp
  6d:   89 e5                   mov    %esp,%ebp
  6f:   83 ec 10                sub    $0x10,%esp
```

Pictorial Swap: 1

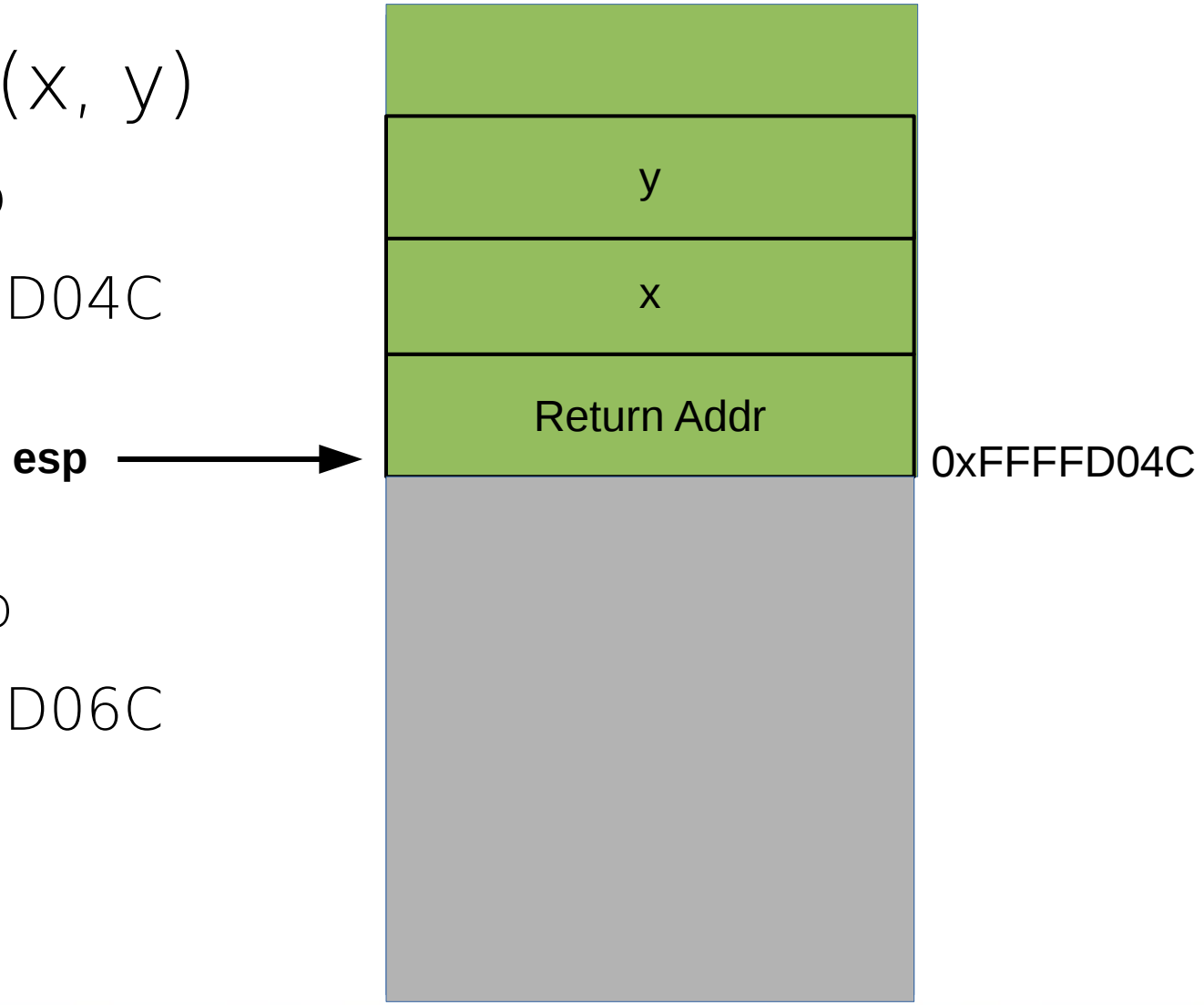
z = swap(x, y)

(gdb) p $esp

$1 = 0xFFFFD04C

(gdb) p $ebp

$1 = 0xFFFFD06C



Pictorial Swap: 2

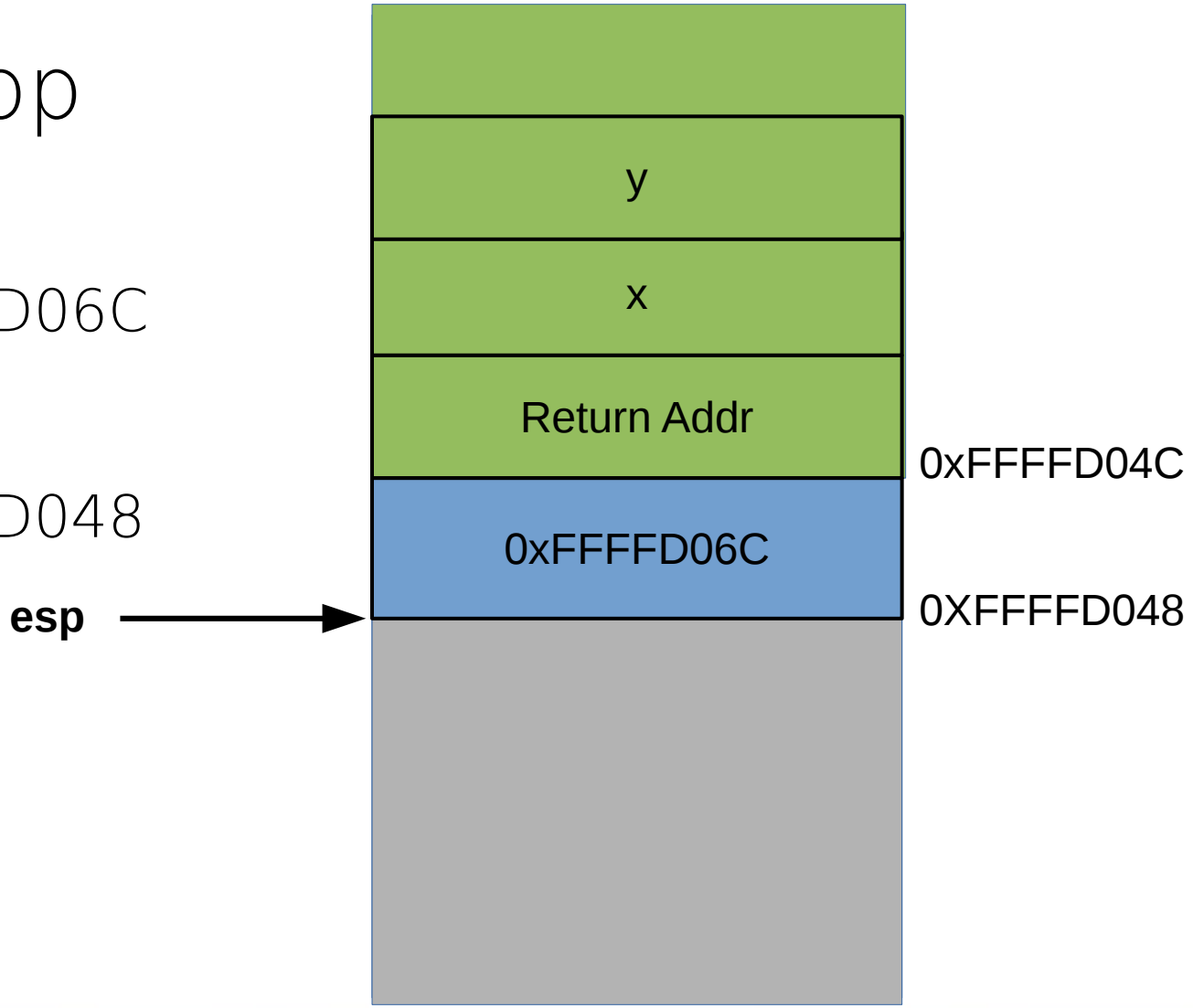
push %ebp

(gdb) p $ebp

$1 = 0xFFFFD06C

(gdb) p $esp

$1 = 0xFFFFD048



Pictorial Swap: 3

mov %esp,%ebp

(gdb) p $ebp

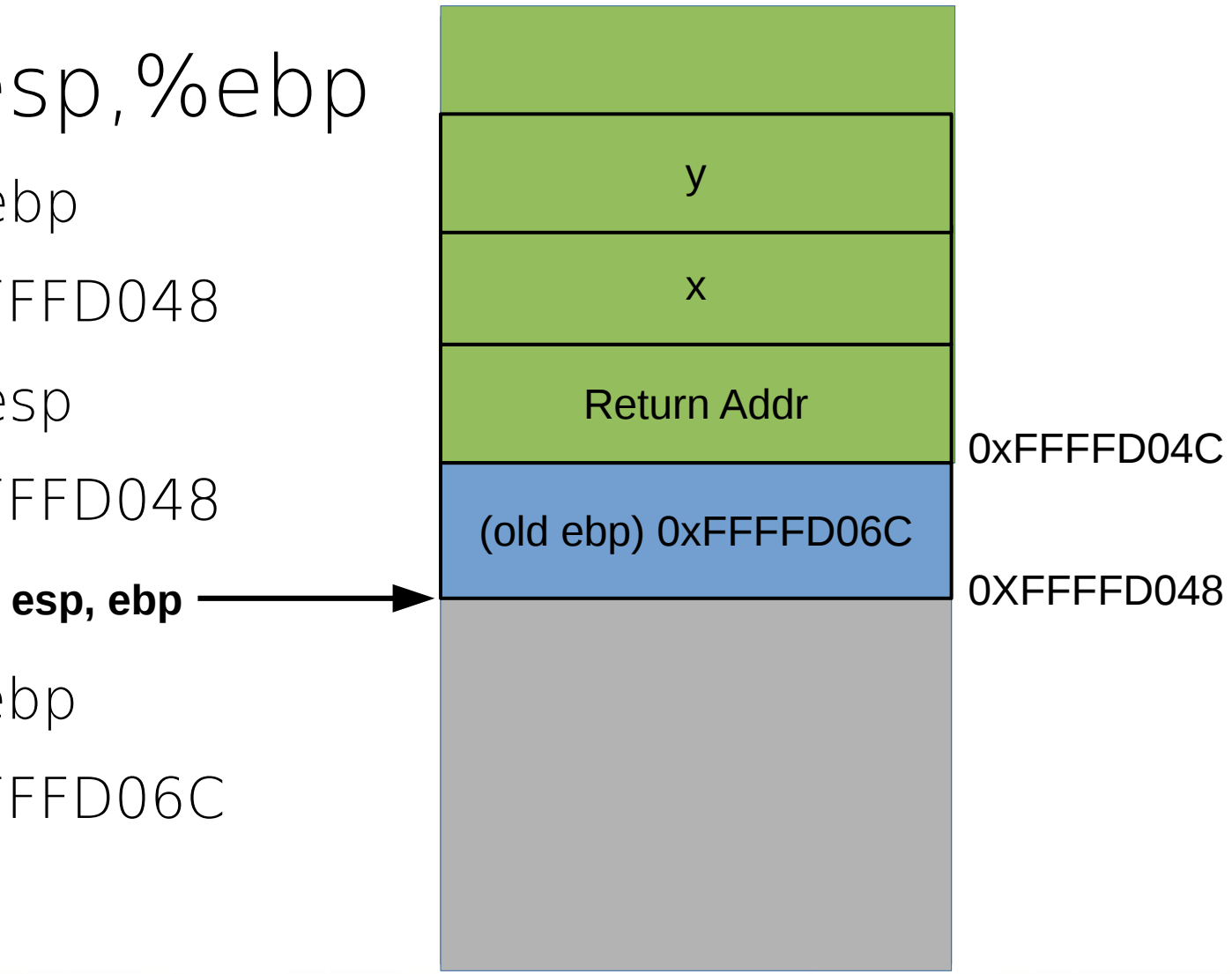
$1 = 0xFFFFD048

(gdb) p $esp

$1 = 0xFFFFD048

(gdb) x $ebp

$1 = 0xFFFFD06C



Pictorial Swap: 4

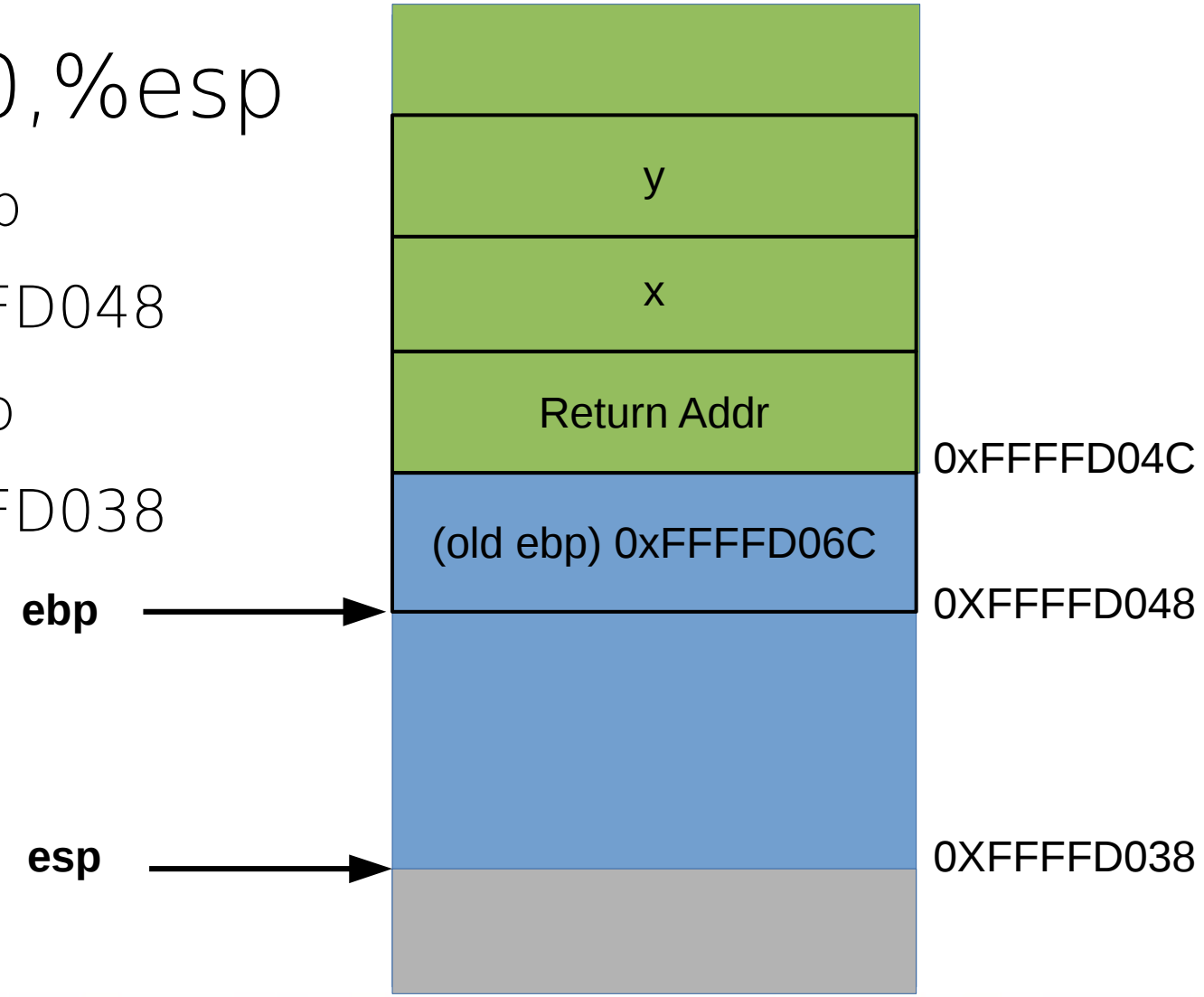
sub $0x10,%esp

(gdb) p $ebp

$1 = 0xFFFFD048

(gdb) p $esp

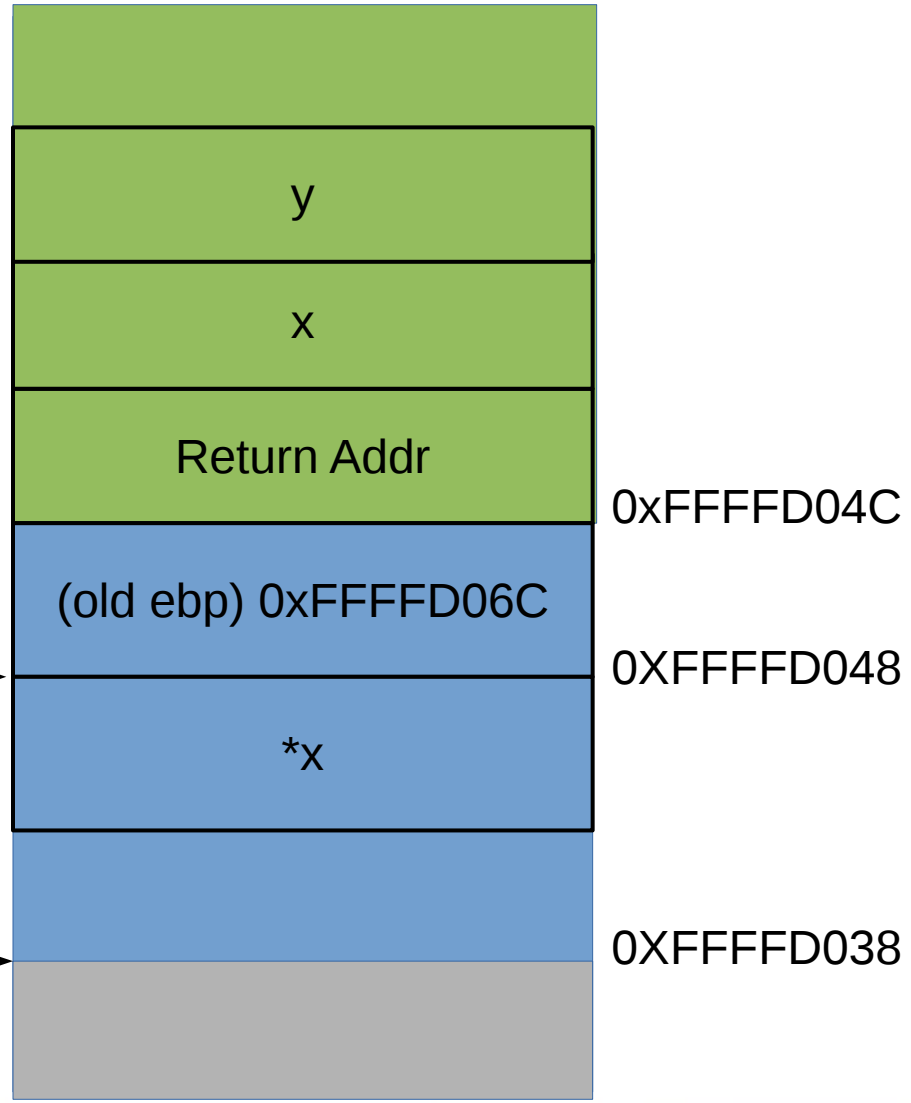
$1 = 0xFFFFD038



Pictorial Swap: 5

```
mov 0x8(%ebp),%eax
mov (%eax),%eax
mov %eax,-0x4(%ebp)
```

```
(gdb) p $eax
$1=0x00000004 (*x)
```



The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)
84:   8b 45 0c        mov   0xc(%ebp),%eax
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

The Swap code

```
72:   8b 45 08      mov   0x8(%ebp),%eax
75:   8b 00         mov   (%eax),%eax
77:   89 45 fc      mov   %eax,-0x4(%ebp)
7a:   8b 45 0c      mov   0xc(%ebp),%eax
7d:   8b 10         mov   (%eax),%edx
7f:   8b 45 08      mov   0x8(%ebp),%eax
82:   89 10         mov   %edx,(%eax)
84:   8b 45 0c      mov   0xc(%ebp),%eax
87:   8b 55 fc      mov   -0x4(%ebp),%edx
8a:   89 10         mov   %edx,(%eax)
```

NOTE: x and y are POINTERS

%eax = x

The Swap code

```
72:   8b 45 08        mov    0x8(%ebp),%eax
75:   8b 00           mov    (%eax),%eax
77:   89 45 fc        mov    %eax,-0x4(%ebp)
7a:   8b 45 0c        mov    0xc(%ebp),%eax
7d:   8b 10           mov    (%eax),%edx
7f:   8b 45 08        mov    0x8(%ebp),%eax
82:   89 10           mov    %edx,(%eax)
84:   8b 45 0c        mov    0xc(%ebp),%eax
87:   8b 55 fc        mov    -0x4(%ebp),%edx
8a:   89 10           mov    %edx,(%eax)
```

%eax = *x

The Swap code

```
72:   8b 45 08        mov    0x8(%ebp),%eax
75:   8b 00           mov    (%eax),%eax
77:   89 45 fc        mov    %eax,-0x4(%ebp)
7a:   8b 45 0c        mov    0xc(%ebp),%eax
7d:   8b 10           mov    (%eax),%edx
7f:   8b 45 08        mov    0x8(%ebp),%eax
82:   89 10           mov    %edx,(%eax)
84:   8b 45 0c        mov    0xc(%ebp),%eax
87:   8b 55 fc        mov    -0x4(%ebp),%edx
8a:   89 10           mov    %edx,(%eax)
```

temp = *x

The Swap code

```
72:   8b 45 08        mov    0x8(%ebp),%eax
75:   8b 00           mov    (%eax),%eax
77:   89 45 fc        mov    %eax,-0x4(%ebp)
7a:   8b 45 0c        mov    0xc(%ebp),%eax
7d:   8b 10           mov    (%eax),%edx
7f:   8b 45 08        mov    0x8(%ebp),%eax
82:   89 10           mov    %edx,(%eax)
84:   8b 45 0c        mov    0xc(%ebp),%eax
87:   8b 55 fc        mov    -0x4(%ebp),%edx
8a:   89 10           mov    %edx,(%eax)
```

temp = *x
%eax = y

The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)
84:   8b 45 0c        mov   0xc(%ebp),%eax
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

temp = *x
%eax = y

%edx = *y

The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)
84:   8b 45 0c        mov   0xc(%ebp),%eax
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

temp = *x

%edx = *y

%eax = x

The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)     temp = *x
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx         %edx = *y
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)         *x = *y
84:   8b 45 0c        mov   0xc(%ebp),%eax
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)     temp = *x
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx         %edx = *y
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)         *x = *y
84:   8b 45 0c        mov   0xc(%ebp),%eax      %eax = y
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)
84:   8b 45 0c        mov   0xc(%ebp),%eax
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

temp = *x

%edx = *y

*x = *y

%eax = y

%edx = temp

The Swap code

```
72:   8b 45 08        mov   0x8(%ebp),%eax
75:   8b 00           mov   (%eax),%eax
77:   89 45 fc        mov   %eax,-0x4(%ebp)
7a:   8b 45 0c        mov   0xc(%ebp),%eax
7d:   8b 10           mov   (%eax),%edx
7f:   8b 45 08        mov   0x8(%ebp),%eax
82:   89 10           mov   %edx,(%eax)
84:   8b 45 0c        mov   0xc(%ebp),%eax
87:   8b 55 fc        mov   -0x4(%ebp),%edx
8a:   89 10           mov   %edx,(%eax)
```

temp = *x

%edx = *y

*x = *y

%eax = y

%edx = temp

*y = temp

Cleanup

- Exit Stack frame...

```
8c: c9          leave
8d: c3          ret
```

The call Instruction

call <label>

- Label = Base address of function to call
- Push return address onto stack
- Jump (change PC) to start of function to call

The ret Instruction

- Pop return address off the stack
- Jump to that location
  - next instruction **after** function call

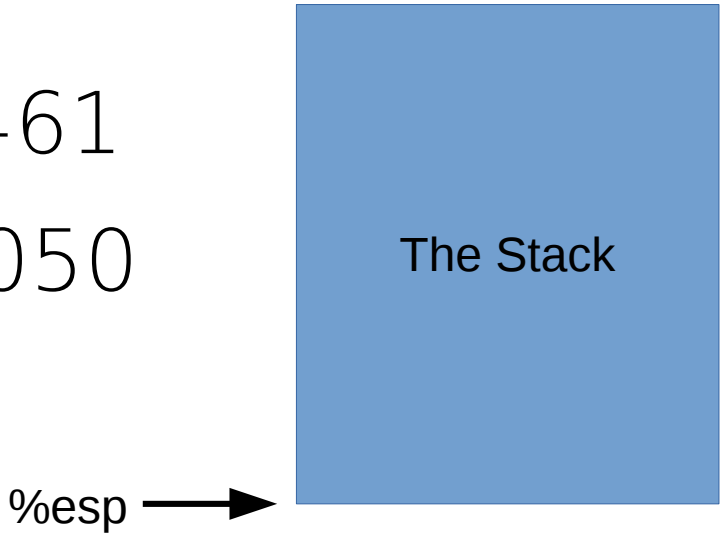
“call” and “ret” in Swap Example

0x08048461 <main+68>:

e8 23 00 00 00 call 0x8048489 <swap>

%eip = 0x08048461

%esp = 0xFFFFD050



“call” and “ret” in Swap Example

0x080484aa <swap+33>: c3 ret

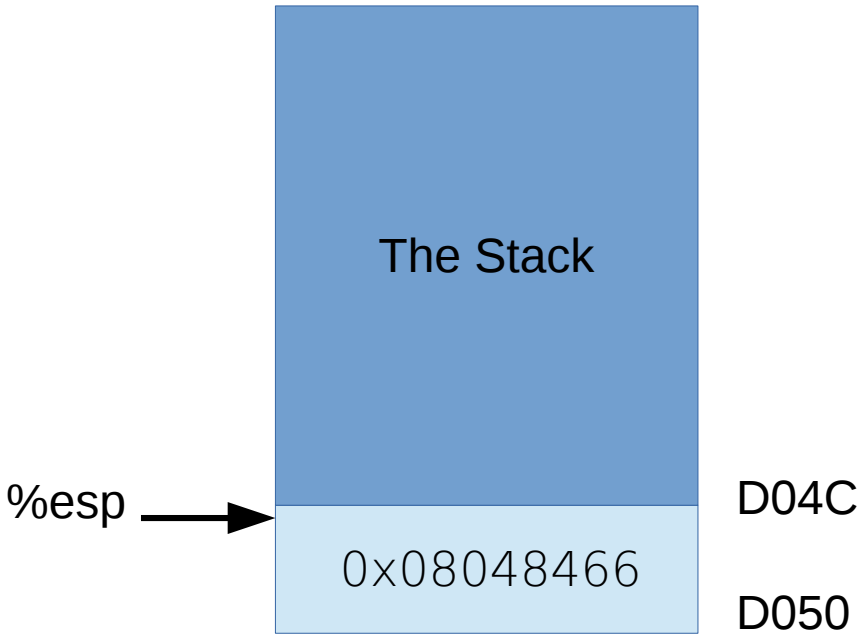
- Kind of like: popl %eip (pop & jump)
- Before:

  %eip = 0x08048489

  %esp = 0xFFFFD04C
- After

  %eip = 0x8048466

  %esp = 0xFFFFD050



Load Effective address

```
leal $0x8(%ebp, %eax, 4), $edx
```

- Like mov Mem → Reg but does not access memory

JUMPS and BRANCHES

Conditional Expressions

- Single bit condition flags
- CF: unsigned overflow
- ZF: result is zero
- SF: result is negative
- OF: signed overflow

Example: Condition Flags

- What condition flags are set:

  %eax = 0x12, %ecx = 0x0c
- subl %eax, %eax

Compare and Test

cmp b, a

- **Computes** a - b
- **Forms:** cmpb, cmpw, cmpl

test b, a **computes** a & b

- How do we use this?
- What condition codes are set?

Compare

cmpl %eax, %edx

if edx == eax

ZF = 1

if edx < eax

SF = 1

The SET Instructions

set{e, ne, s, ns, g, ge, ...} R

- Set single byte of **R** to 1 or 0 depending on current value of one or more condition codes

The SET Instructions

sete	Set equal	ZF
setne	Set not equal	~ZF
setg	Set greater	~(SF^OF)&~ZF
setl	Set less than	SF^OF
seta	Set above	~CF
setb	Set below	CF

Jump Instructions

- How to model if, while, for...
- Need to be able to modify Flow of Control
  - i.e. “Jump” around in our code
- Change the instruction pointer %eip based on condition codes

Direct v.s Indirect Jump

jmp <Label>

- Direct Jump
- Harded coded jump location
  - e.g. jmp 8a goes to instr at address 8a

jmp *<Operand>

- Indirect Jump
- Jump targets can be Register value or Mem

jmp *(%eax)

Conditional Jumps

- Like SET operations but sets %eip to value of operand based on Condition Flags

```
cmpl %eax, %edx
jge a <myFunction+0xa>
```

Conditional Jumps

je	Jump equal to	?
jne	Jump not equal	?
jg	Jump greater than	~(SF^OF)&~ZF
jge	Jump greater than/equal to	?
jl	Jump less than	SF^OF
ja	Jump above	~CF & ~ZF

```
2a:8b 45 08          mov    0x8(%ebp),%eax
2d:3b 45 0c          cmp    0xc(%ebp),%eax
30:7e 0f             jle    41 <unknown+0x1d>
32:8b 45 0c          mov    0xc(%ebp),%eax
35:8b 55 08          mov    0x8(%ebp),%edx
38:29 c2             sub    %eax,%edx
3a:89 d0             mov    %edx,%eax
3c:89 45 fc          mov    %eax,-0x4(%ebp)
3f: eb 0d            jmp    4e <unknown+0x2a>
41:8b 45 08          mov    0x8(%ebp),%eax
44:8b 55 0c          mov    0xc(%ebp),%edx
47:29 c2             sub    %eax,%edx
49:89 d0             mov    %edx,%eax
4b:89 45 fc          mov    %eax,-0x4(%ebp)
4e:8b 45 fc          mov    -0x4(%ebp),%eax
```

What does this function do?

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```

cmp y, x

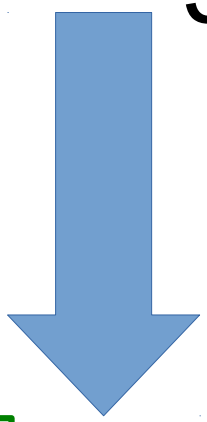
Assume
arg1 = x
arg2 = y

```
2a:8b 45 08        mov   0x8(%ebp),%eax
2d:3b 45 0c        cmp   0xc(%ebp),%eax
30:7e 0f           jle   41 <unknown+0x1d>
32:8b 45 0c        mov   0xc(%ebp),%eax
35:8b 55 08        mov   0x8(%ebp),%edx
38:29 c2           sub   %eax,%edx
3a:89 d0           mov   %edx,%eax
3c:89 45 fc        mov   %eax,-0x4(%ebp)
3f: eb 0d          jmp   4e <unknown+0x2a>
41:8b 45 08        mov   0x8(%ebp),%eax
44:8b 55 0c        mov   0xc(%ebp),%edx
47:29 c2           sub   %eax,%edx
49:89 d0           mov   %edx,%eax
4b:89 45 fc        mov   %eax,-0x4(%ebp)
4e:8b 45 fc        mov   -0x4(%ebp),%eax
```

if (x <= y)

Assume
arg1 = x
arg2 = y

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```



if (x <= y)

JUMP

Assume
arg1 = x
arg2 = y

```
2a:8b 45 08        mov   0x8(%ebp),%eax
2d:3b 45 0c        cmp   0xc(%ebp),%eax
30:7e 0f           jle   41 <unknown+0x1d>
32:8b 45 0c        mov   0xc(%ebp),%eax
35:8b 55 08        mov   0x8(%ebp),%edx
38:29 c2           sub   %eax,%edx
3a:89 d0           mov   %edx,%eax
3c:89 45 fc        mov   %eax,-0x4(%ebp)
3f: eb 0d          jmp   4e <unknown+0x2a>
41:8b 45 08        mov   0x8(%ebp),%eax
44:8b 55 0c        mov   0xc(%ebp),%edx
47:29 c2           sub   %eax,%edx
49:89 d0           mov   %edx,%eax
4b:89 45 fc        mov   %eax,-0x4(%ebp)
4e:8b 45 fc        mov   -0x4(%ebp),%eax
```

if (x <= y)

eax = x

Assume
arg1 = x
arg2 = y

```
2a:8b 45 08        mov   0x8(%ebp),%eax
2d:3b 45 0c        cmp   0xc(%ebp),%eax
30:7e 0f           jle   41 <unknown+0x1d>
32:8b 45 0c        mov   0xc(%ebp),%eax
35:8b 55 08        mov   0x8(%ebp),%edx
38:29 c2           sub   %eax,%edx
3a:89 d0           mov   %edx,%eax
3c:89 45 fc        mov   %eax,-0x4(%ebp)
3f: eb 0d          jmp   4e <unknown+0x2a>
41:8b 45 08        mov   0x8(%ebp),%eax
44:8b 55 0c        mov   0xc(%ebp),%edx
47:29 c2           sub   %eax,%edx
49:89 d0           mov   %edx,%eax
4b:89 45 fc        mov   %eax,-0x4(%ebp)
4e:8b 45 fc        mov   -0x4(%ebp),%eax
```

if (x <= y)

eax = x
edx = y

Assume
arg1 = x
arg2 = y

```
2a:8b 45 08        mov   0x8(%ebp),%eax
2d:3b 45 0c        cmp   0xc(%ebp),%eax
30:7e 0f           jle   41 <unknown+0x1d>
32:8b 45 0c        mov   0xc(%ebp),%eax
35:8b 55 08        mov   0x8(%ebp),%edx
38:29 c2           sub   %eax,%edx
3a:89 d0           mov   %edx,%eax
3c:89 45 fc        mov   %eax,-0x4(%ebp)
3f: eb 0d          jmp   4e <unknown+0x2a>
41:8b 45 08        mov   0x8(%ebp),%eax
44:8b 55 0c        mov   0xc(%ebp),%edx
47:29 c2           sub   %eax,%edx
49:89 d0           mov   %edx,%eax
4b:89 45 fc        mov   %eax,-0x4(%ebp)
4e:8b 45 fc        mov   -0x4(%ebp),%eax
```

if (x <= y)

eax = x
edx = y
edx = y - x

Assume
arg1 = x
arg2 = y

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```

if (x <= y)

eax = x
edx = y
edx = y – x
z = y – x

return z

```
2a:8b 45 08        mov   0x8(%ebp),%eax
2d:3b 45 0c        cmp   0xc(%ebp),%eax
30:7e 0f           jle   41 <unknown+0x1d>
32:8b 45 0c        mov   0xc(%ebp),%eax
35:8b 55 08        mov   0x8(%ebp),%edx
38:29 c2           sub   %eax,%edx
3a:89 d0           mov   %edx,%eax
3c:89 45 fc        mov   %eax,-0x4(%ebp)
3f: eb 0d          jmp   4e <unknown+0x2a>
41:8b 45 08        mov   0x8(%ebp),%eax
44:8b 55 0c        mov   0xc(%ebp),%edx
47:29 c2           sub   %eax,%edx
49:89 d0           mov   %edx,%eax
4b:89 45 fc        mov   %eax,-0x4(%ebp)
4e:8b 45 fc        mov   -0x4(%ebp),%eax
```

!(x <= y)

eax = y

Assume

arg1 = x

arg2 = y

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```

!(x <= y)

eax = y

edx = x

Assume

arg1 = x

arg2 = y

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```

!(x <= y)

eax = y

edx = x

edx = x - y

Assume

arg1 = x

arg2 = y

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```

!(x <= y)

eax = y

edx = x

edx = x - y

z = x - y

Assume

arg1 = x

arg2 = y

```
2a:8b 45 08      mov   0x8(%ebp),%eax
2d:3b 45 0c      cmp   0xc(%ebp),%eax
30:7e 0f         jle   41 <unknown+0x1d>
32:8b 45 0c      mov   0xc(%ebp),%eax
35:8b 55 08      mov   0x8(%ebp),%edx
38:29 c2         sub   %eax,%edx
3a:89 d0         mov   %edx,%eax
3c:89 45 fc      mov   %eax,-0x4(%ebp)
3f: eb 0d        jmp   4e <unknown+0x2a>
41:8b 45 08      mov   0x8(%ebp),%eax
44:8b 55 0c      mov   0xc(%ebp),%edx
47:29 c2         sub   %eax,%edx
49:89 d0         mov   %edx,%eax
4b:89 45 fc      mov   %eax,-0x4(%ebp)
4e:8b 45 fc      mov   -0x4(%ebp),%eax
```

!(x <= y)

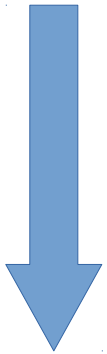
eax = y

edx = x

edx = x - y

z = x - y

JUMP



return z

```
2a:8b 45 08          mov    0x8(%ebp),%eax
2d:3b 45 0c          cmp    0xc(%ebp),%eax
30:7e 0f             jle    41 <unknown+0x1d>
32:8b 45 0c          mov    0xc(%ebp),%eax
35:8b 55 08          mov    0x8(%ebp),%edx
38:29 c2             sub    %eax,%edx
3a:89 d0             mov    %edx,%eax
3c:89 45 fc          mov    %eax,-0x4(%ebp)
3f: eb 0d            jmp    4e <unknown+0x2a>
41:8b 45 08          mov    0x8(%ebp),%eax
44:8b 55 0c          mov    0xc(%ebp),%edx
47:29 c2             sub    %eax,%edx
49:89 d0             mov    %edx,%eax
4b:89 45 fc          mov    %eax,-0x4(%ebp)
4e:8b 45 fc          mov    -0x4(%ebp),%eax
```

If (!(x <= y))

z = x - y

If (x <= y)

z = y - x

return z

Loops

- Need to translate C → Assembly:
  - Do While
  - While
  - For

Do While in C

- C code:

```
do {

   /*Loop Body */

} while (test);
```

Do While in Assembly

- Assembly:

```
.L1:
   /*Loop Body */
   cmpl   <op1>,<op2>
   jg       .L1
```

While in C

- C Code:

```
while (test) {

    /*Loop Body */

}
```

While in C Rewritten

- Modified C code

```
loop:
    if (!test) goto done;
    /*Loop Body */
    goto loop;
done:
```

- Is this equivalent?

While in Assembly

- Assembly code:

```
.L2:
    cmpl   <op1>,<op2>
    jg       .L3
    /*Loop Body */
    jmp     .L2
.L3:                    /*done*/
```

For Loop in C

- C code

```
for (init, test, update) {
   /* Loop body */
}
```

- Hmmm, doesn't look like something we've seen before...

For → While

- For:

```
for (init, test, update) {
   /* Loop body */
}
```

- While:

```
init;
while (test)  {
   /* Loop body */
   update;
}
```

For → While → Goto

- Goto While:

```
init;
loop:
if (!test)  goto done;
   /* Loop body */
   Update;
   goto loop;
done:
```

ARRAYS, STRUCTS and UNIONS

Arrays

- Where do we store a static array?

  ```
  int arr1 [N];
  ```

  - Allocates sizeof(int)*N contiguous bytes
- How do we determine location of x?

  ```
  int x = arr1[i];
  ```

Arrays

- Where do we store a static array?
  - The Stack

```
int arr1 [N];
```

  - Allocates sizeof(int)*N contiguous bytes
- How do we determine location of x?

```
int x = arr1[i];
```

```
x = *(arr1+i*sizeof(int));
```

NOTE: Above statement is **not** correct C code

Multi-Dimensional Arrays

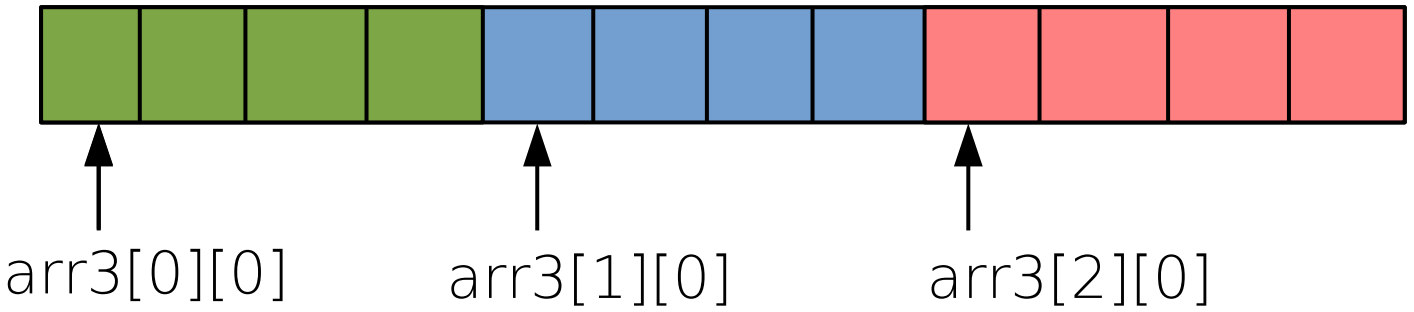
- How about a Matrix?

  ```
  short arr2 [N] [M];
  ```

- How many bytes do we allocate?
- How do I access arr2[i][j]

Multi-Dimensional Array

char arr3 [3] [4];



arr [i] [j] = *(arr + i*M + j)

N = # of rows, M = # of columns

Struct

- Hetrogeneous aggregate data type

```
struct myStruct {
        char c[3];
        int i;
        double d;
}
```

sizeof(myStruct) >= sum of all constituent elements

- Why not equal?

Union

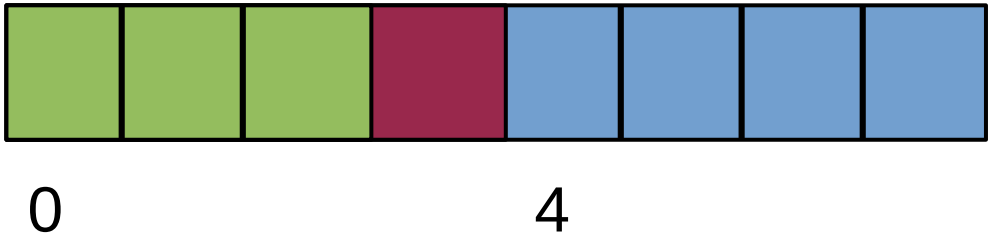
- A single object that can be referenced according to multiple types

```
union myUnion {
        char c[3];
        int i;
        long long ll;
}
```

sizeof(myUnion) = max(sizeof(element_i)) for all elements

Unions in Memory

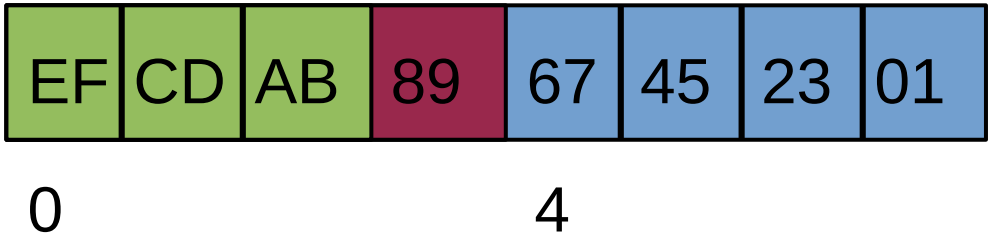
myUnion u;



- u.ll = 0x0123456789ABCDEF;
- u.i = ?
- u.c[1] = ?

Unions in Memory

myUnion u;



- u.ll = 0x0123456789ABCDEF;
- u.i = 0x89ABCDEF;
- u.c[1] = 0xCD;

Alignment

- Restrictions on base addr of primitive types to improve efficiency
- e.g. for integer (4B), address must be a multiple of 4

  ```
  int x = 4;
  ```

- &x **can be**: 11100, 10100, etc
- &x **cannot be** 11101, 11110, etc

Alignment

- What does alignment mean for structs?

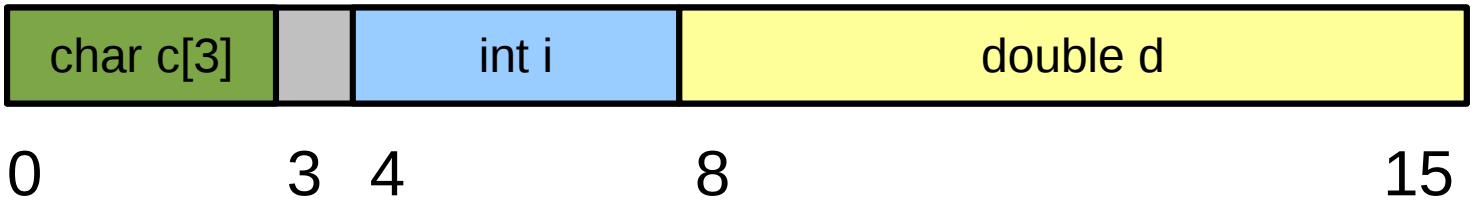
Struct: Alignment

- Each constituent must obey alignment
  - Integer addr must be multiple of 4
  - Double addr must be multiple of 8

```
struct myStruct {
        char c[3];
        int i;
        double d;
}
```

Struct in Memory

myStruct s;

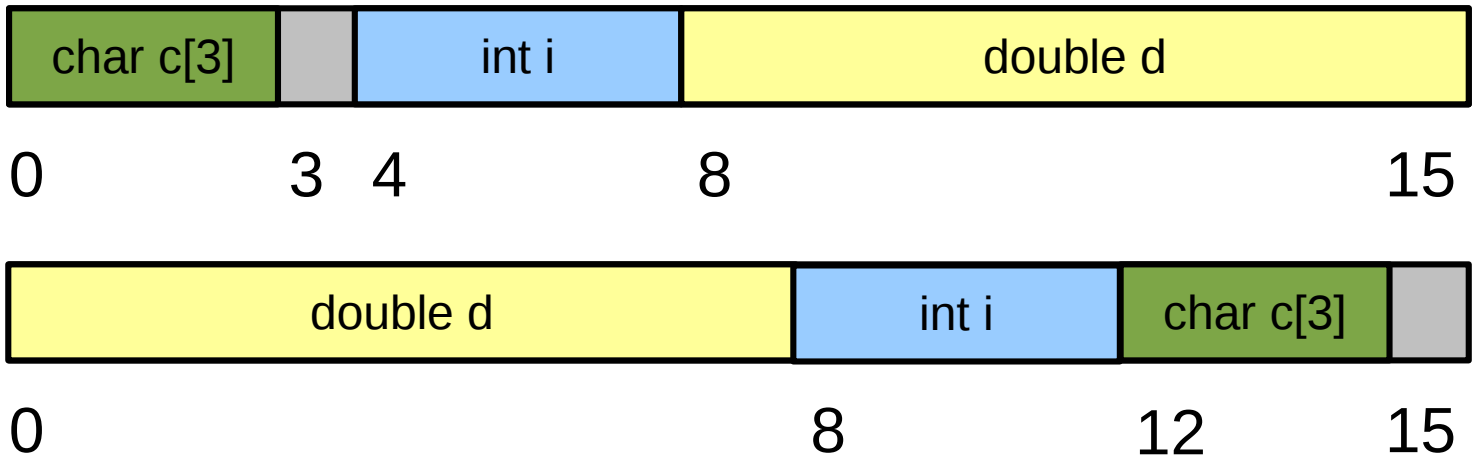


sizeof(s) = 16

- Can I be more efficient?

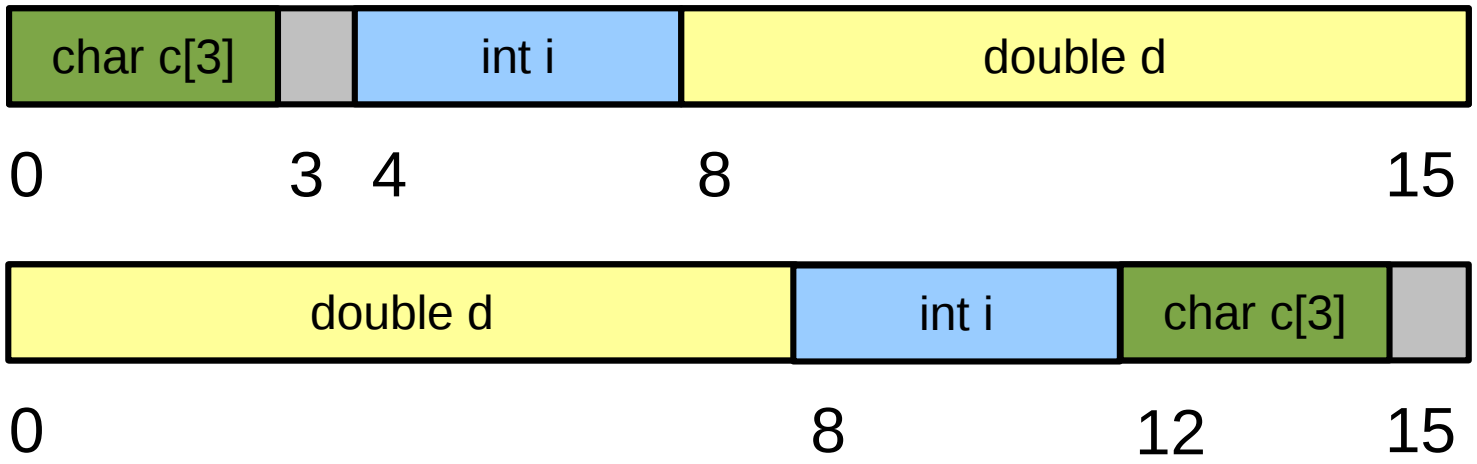
Struct in Memory

- Can I be more efficient?
  - In this case, no. Why?



Struct in Memory

- Can I be more efficient?
  - In this case, no. Why?



- Must consider arrays of structs
- Addr of myStruct must be a multiple max(sizeof(element of myStruct))