

C Introduction

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

Announcements

- Homework 6 due Tonight at 11:59pm

C

C is a thin wrapper around assembly

- This is by design!
- Invented to write an operating system
 - Can write inline assembly in C
- Many other languages decided to look like C

C

C Helpful Resources

- Wikipedia
- Our Reference and Summary

`sizeof()` - returns size in bytes

- `sizeof(int)` returns 4

Data Types in C

Integer data types

Data type	Size (bits)	Size (bytes)
char	8	1
short	16	2
int	32	4
long	64	8
long long	64	8

Each has 2 versions: *signed* and *unsigned*

by default, short, int, long, long long are signed.

Example of creating a variable: unsigned long long x = 5.

Data Types in C *(check "Readings → C Reference" for exponent bits and fraction bits)*

Floating point

- float
- double

Data Types in C

We stored things like arrays of memory addresses.
We called functions by giving it the memory address.
.....

Data Types in C

Pointers - how C uses addresses!

Data Types in C

Pointers - how C uses addresses!

- Hold the address of a position in memory
- Need to know the kind of information stored at that location

The size of a pointer? How many bytes? — 64 bits (8 bytes) (Registers are 64 bits because we had 64-bit memory addresses)

If I want to have a pointer to an int: `int *y;`
↳ the star references the fact that this is not an integer, it's a pointer.

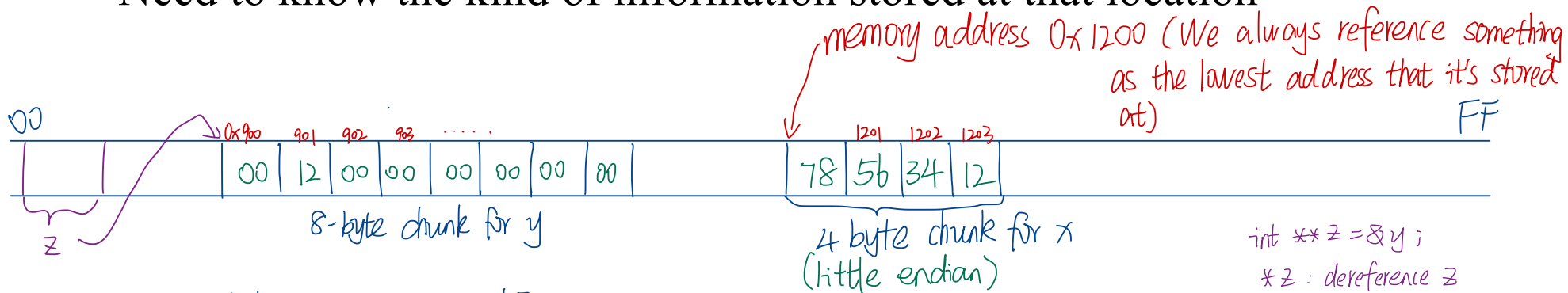
`int x=5;`  (4 bytes)
`int *y;`  (8 bytes)
`y=x;` → doesn't work

`y=&x;` It works!
`*y=25;` will change `x` to 25.

Data Types in C

Pointers - how C uses addresses!

- Hold the address of a position in memory
- Need to know the kind of information stored at that location



`int x = 0x12345678`

`int *y;`

`y = &x;` (I want y point to x, which means store the address of x in y).

`int **z = &y;`
`*z` : dereference z
`**z` : dereference *z
`**z = 0x00000000;`
 (set x to 0x00000000)
`*z = 25;` (Warning: you're trying to assign an integer to a pointer)
 (But later, when you use it as an address, may seg fault).

`y = 0xABCDEF01;` (I'm changing the pointer) → I may don't want to do this. Sometimes seg fault?

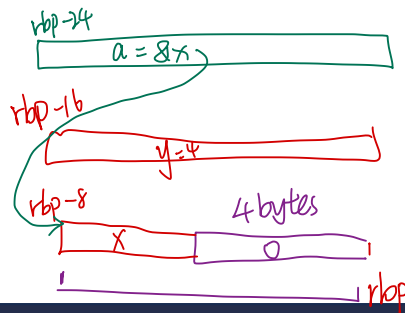
`*y = 25;` (When I use an asterisk, it will say is follow the pointer on y, follow the address to the place in memory that y points to and change that value)

Example

```
int main() {  
    int x = 3;  
    long y = 4;  
    int *a = &x;  
    long *b = &y;  
    long z = *a;  
    int w = *b;  
    return 0;  
}
```

Example

```
int main() {
    int x = 3;
    long y = 4;
    int *a = &x;
    long *b = &y;
    long z = *a;
    int w = *b;
    return 0;
}
```



0000000000000000 <main>:

```
0: 55
1: 48 89 e5
4: 31 c0
6: c7 45 fc 00 00 00 00
d: c7 45 f8 03 00 00 00
14: 48 c7 45 f0 04 00 00
1b: 00
1c: 48 8d 4d f8
20: 48 89 4d e8
24: 48 8d 4d f0
28: 48 89 4d e0
2c: 48 8b 4d e8
30: 48 63 09
33: 48 89 4d d8
37: 48 8b 4d e0
3b: 48 8b 09
3e: 89 4d d4
41: 5d
42: c3
```

Signed extend
1 to 4
long to quad

```
push    %rbp
mov     %rsp,%rbp
xor     %eax,%eax
movl    $0x0,-0x4(%rbp)
movl    $0x3,-0x8(%rbp)
movq    $0x4,-0x10(%rbp)

lea     -0x8(%rbp),%rcx
mov     %rcx,-0x18(%rbp)
lea     -0x10(%rbp),%rcx
mov     %rcx,-0x20(%rbp)
mov     -0x18(%rbp),%rcx
movslq  (%rcx),%rcx
mov     %rcx,-0x28(%rbp)
mov     -0x20(%rbp),%rcx
mov     (%rcx),%rcx
mov     %ecx,-0x2c(%rbp)
pop     %rbp
retq
```

(compiler knows we are working on a 64-bit machine. I'll probably try to line everything to 8 bytes if I can, to make things faster for you)

loading the address of x into rcx
y is treated as the same way.
→ &x

b going from long to int.
move the value of y into rcx, but I will just read out the value of ecx.

Swap Example:

```
void swap(int *a, int *b) {
```

```
    int tmp = *a;
```

```
    *a = *b;
```

```
    *b = tmp;
```

```
}
```

