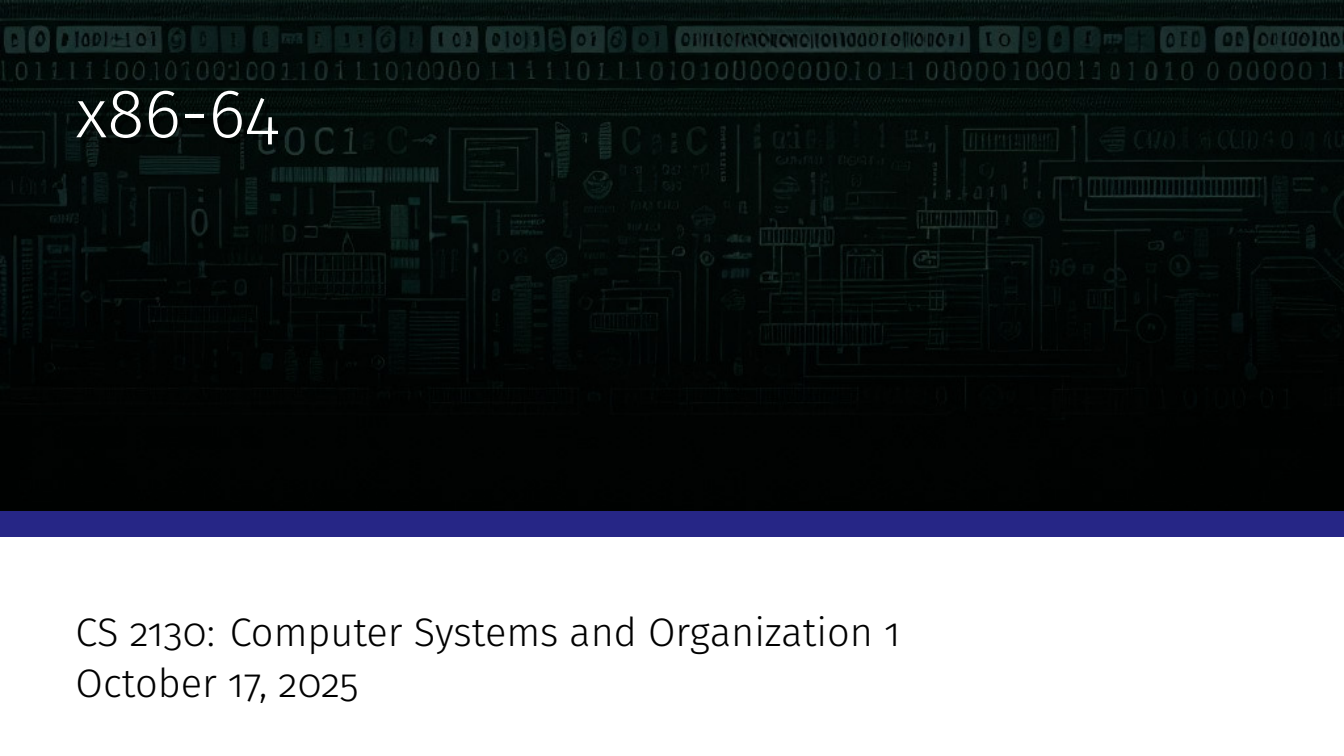




x86-64

CS 2130: Computer Systems and Organization 1
October 17, 2025

Announcements

- Homework 5 **due Monday at 11:59pm** on Gradescope
- Quiz 5 this weekend

Debugger

Debugger - step through code!

- Helpful for Homework 5
- You will be using this for lab 7
- Experience seeing results of these instructions step-by-step
- **Please read the x86-64 summary reading!**

Debugger

Example with lldb

Example: Loops

```
while (i < 42)
    i += 1
```

Functions

```
f(x,y):  
    ...  
    ...  
    return 4
```

```
...  
z = f(2,5)
```

Function Calls: Calling Conventions

`callq myfun`

- Push return address, then jump to `myfun`
- Convention: Store arguments in registers and stack before call
 - First 6 arguments (in order): `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`
 - If more arguments, pushed onto stack (last to first)

`retq`

- Pop return address from stack and jump back
- Convention: store return value in `rax` before calling `retq`

This is similar to our Toy ISA's function calls in homework 4

Calling Conventions: Registers

Calling conventions - recommendations for making function calls

- Where to put arguments/parameters for the function call?
- Where to put return value? in `rax` before calling `retq`
- What happens to values in the registers?
 - **Callee-save** - The function should ensure the values in these registers are unchanged when the function returns
 - * `rbx, rsp, rbp, r12, r13, r14, r15`
 - **Caller-save** - Before making a function call, save the value, since the function may change it

The Stack

```
pushq %rax  
popq %rdx
```

example.s

Most Common Instructions

- `mov` - =
- `lea` - load effective address
- `call` - push PC and jump to address
- `add` - +=
- `cmp` - set flags as if performing subtract
- `jmp` - unconditional jump
- `test` - set flags as if performing &
- `je` - jump iff flags indicate == 0
- `pop` - pop value from stack
- `push` - push value onto stack
- `ret` - pop PC from the stack

Compilation Pipeline

Turning our code into something that runs

- **Pipeline** - a sequence of steps in which each builds off the last