x86-64 Assembly

CS 2130: Computer Systems and Organization 1
March 13, 2026

Announcements

- Homework 5 **due Monday at 11:59pm** on Gradescope
- Prof Hott office hour updates!
  - Rice 401 (no longer in 210)
  - Tuesdays 9-10am (earlier!), Wednesdays 2-3:30pm

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

Debugger

Debugger - step through code!

- Helpful for Homework 5, 6, and when we get to C
- Experience seeing results of these instructions step-by-step
- **Please read the x86-64 summary reading!**

Example: Loops

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

Debugger

`example.s` – Example with lldb

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
```

The Stack

`stack.s` – Example with lldb