C and Compilation

CS 2130: Computer Systems and Organization 1

March 20, 2023

Announcements

- Homework 5 due tonight at 11pm
- Homework 6 due Monday at 11pm
  - Full details later today
  - You'll start it in lab tomorrow!

Review from last week

Patents and Copyright

Copyright

- “Everyone is a copyright owner. Once you create an original work and fix it, like taking a photograph, writing a poem or blog, or recording a new song, you are the author and the owner.”

from https://www.copyright.gov/what-is-copyright/

Patent

- “Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.”

from 35 U.S.C. 101

Patents

In software and hardware, patents become messy

- Code is a description of a process we want the computer to do
- Do not have to implement the process to patent it

Question: Should we patent something like our ISA?

Patents

In software and hardware, patents become messy

- Code is a description of a process we want the computer to do
- Do not have to implement the process to patent it

Question: Should we patent something like our ISA?

What is the current state of the art?

Common Approaches to Software

How can we get value from what we create?

- Copyright - distribute closed source software
- License Agreements (in contract law)
- Always innovate

Calling Conventions: Parameters

Calling conventions - recommendations for making function calls

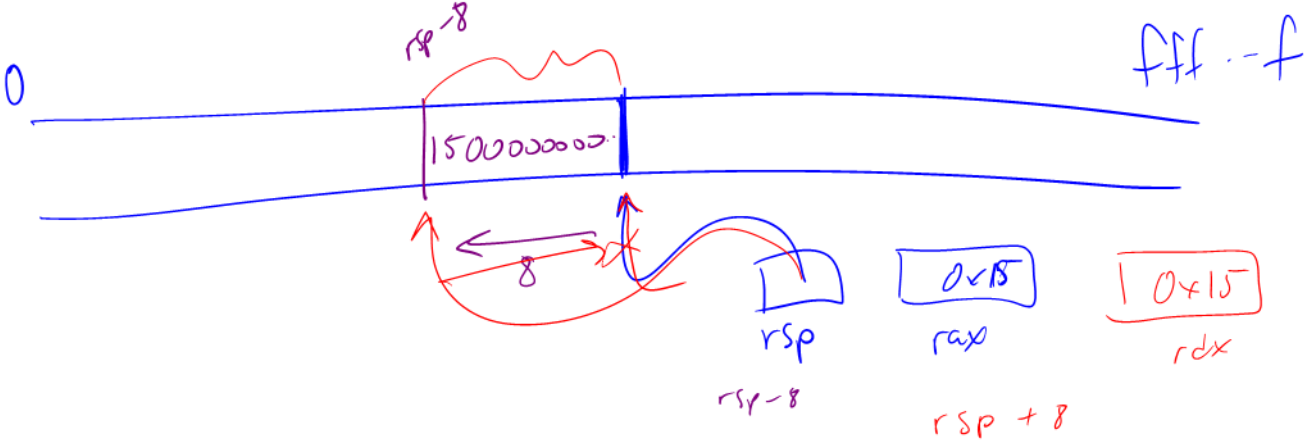
- Where to put arguments/parameters for the function call?
  - First 6 arguments (in order): `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`
  - If more arguments, push onto stack (last to first)
- Where to put return value? in `rax` before calling `retq`
- What happens to values in the registers?
  - **Callee-save** - 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

Most Common x86-64 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

The Stack

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



stack.s and lea.s

Compilation Pipeline

Turning our code into something that runs

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

Why did we discuss assembly?

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

Simple C Example

```
int main() {
    int y = 5;
    return 0;
}
```

Compilation Pipeline

Earlier, we saw:

- C files (`.c`) compiled to assembly (`.s`)
- Assembly (`.s`) assembled into object files (`.o`)
- Object files (`.o`) linked into a program / executable

Compiling C to Assembly

Multiple stages to compile C to assembly

- Preprocess - produces C
  - C is actually implemented as 2 languages:
    C preprocessor language, C language
  - Removes comments, handles preprocessor directives (`#`)
  - `#include`, `#define`, `#if`, `#else`, …
- Lex - breaks input into individual tokens
- Parse - assembles tokens into intended meaning (parse tree)
- Type check - ensures types match, adds casting as needed
- Code generation - creates assembly from parse tree

Compiling C to Assembly

Compiling C to Assembly

Errors

Compile-time errors

- Errors we can catch during compilation (this process)
- **Before** running our program

Runtime errors

- Errors that occur when running our programs

Simple C Example

```
int main() {
    return 0;
}
```

The `main` function

- Start running the `main()` function
- `main` must return an integer - **exit code**
  - `0` = everything went okay
  - Anything else = something went wrong
- There *should* be arguments to main

Example