C Introduction

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 7 due tonight on Gradescope
- Midterm 2 is Friday
  - Similar format to midterm 1
  - Schedule with SDAC early if needed
  - Review session on Wednesday in class

C Reference Guide

Switch

```
void describe(int age) {
    switch (age) {
        case 1:
            puts("You're one.");
            break;
        case 2:
            puts("You're two.");
            break;
        case 4:
            puts("You're four.");
        case 5:
            puts("You're four or five.");
            break;
        default:
            puts("You're not 1, 2, 4 or 5!");
    }
}
```

```
    .text
    .globl  describe
describe:
    pushq   %rax
    movl    %edi, %eax
    addl    $-1, %eax
    cmpl    $4, %eax
    ja  Label5
    leaq    Text0, %rdi
    leaq    JumpTable, %rcx
    movslq  (%rcx,%rax,4), %rax
    addq    %rcx, %rax
    jmpq    *%rax
Label2:
    leaq    Text1, %rdi
    jmp Label6
Label5:
    leaq    Text4, %rdi
    jmp Label6
Label3:
    leaq    Text2, %rdi
    xorl    %eax, %eax
    callq   puts
```

```
Label6:
    xorl    %eax, %eax
    callq   puts
    popq    %rax
    retq
```

```
    .section    .rodata
JumpTable:
    .long   Label6-JumpTable
    .long   Label2-JumpTable
    .long   Label5-JumpTable
    .long   Label3-JumpTable
    .long   Label4-JumpTable

Text0:
    .asciz  "You're one."
Text1:
    .asciz  "You're two."
Text2:
    .asciz  "You're four."
Text3:
    .asciz  "You're four or five."
Text4:
    .asciz  "You're not 1, 2, 4 or 5!"
```

It looks like a bunch of labels. And it's going to work exactly like we've seen labels. Where I jumped to a label and then I start executing code.

Not like a function: I go there, run things, then return.

Not like if statement: I do a bunch of conditional jumps.

One thing to note. we have to switch on an integer. That integer is going to be the index into our array of labels. (Indexing into an array of labels and picking which one we want to jump to).

```
    .globl  describe
describe:
    pushq   %rax
    movl    %edi, %eax
    addl    $-1, %eax
    cmpl    $4, %eax
    ja  Label5
    leaq    Text0, %rdi
    leaq    JumpTable, %rcx
    movslq  (%rcx,%rax,4), %rax
    addq    %rcx, %rax
    jmpq    *%rax
Label2:
    leaq    Text1, %rdi
    jmp Label6
Label5:
    leaq    Text4, %rdi
    jmp Label6
Label3:
    leaq    Text2, %rdi
    xorl    %eax, %eax
    callq   puts
Label4:
    leaq    Text3, %rdi
```

first parameter (age).

The index of an array starts from 0, so we want to map 1~5 to 0~4

if index > 4, then jump to label 5

It's preparing the argument for a later call like puts (Text0).

Load the address of the jump table into %rcx.

Loads a 32-bit offset from the jump table entry indexed by %rax and sign-extends it to 64 bits. Each entry is 4 bytes long (hence the *4).

Adds the base address of the jump table to that offset.

indirect jump to the address in %rcx.

load Text1 and jump to label6.

load Text4 and jump to label6

```
Label3:
    leaq    Text2, %rdi
    xorl    %eax, %eax
    callq   puts
Label4:
    leaq    Text3, %rdi
Label6:
    xorl    %eax, %eax
    callq   puts
    popq    %rax
    retq
```

empty out eax and outputs.

in assembly a long type is 32 bites.

```
    .section    .rodata
JumpTable:
    .long   Label6-JumpTable
    .long   Label2-JumpTable
    .long   Label5-JumpTable
    .long   Label3-JumpTable
    .long   Label4-JumpTable

Text0:
    .asciz  "You're one."
Text1:
    .asciz  "You're two."
Text2:
    .asciz  "You're four."
Text3:
    .asciz  "You're four or five."
Text4:
    .asciz  "You're not 1, 2, 4 or 5!"
```

jump table. Why they choose this order ?
I don't know but it doesn't matter.

read only section

Calling Functions

The C code

```
long a = f(23, "yes", 34uL);
```

follow the calling conventions if I don't know anything else about function f.

compiles to

```
movl $23, %edi
leaq label_of_yes_string, %rsi
movq $34, %rdx
callq f
# %rax is "long a" here
```

put the address of "yes" to rsi.

without respect to how `f` was defined. It is the calling convention, not the type declaration of `f`, that controls this.

Calling Functions

But, if the C code has access to the type declaration of `f`, then it might perform some implicit casting first; for example, if we declared

```
long f(double a, const char *b, double c);
```

```
long a = f(23, "yes", 34uL);
```

We need to make sure that we have the declaration of the function before we call it. So the type checker knows how the function supposed to look

then the call would be interpreted by C as having implicit casts in it:

```
long a = f((double)23, "yes", (double)34uL);
```

Calling Functions

When I call a function with floating pointer numbers, there's a whole set of registers that just yield floating point numbers.

and the arguments would be passed in floating-point registers, like so:

```
movl $23, %eax
cvtsi2sd %eax, %xmm0          # first floating-point argument
leaq label_of_yes_string, %rdi # first integer/pointer argument

movl $34, %eax
cvtsi2sd %eax, %xmm1          # second floating-point argument

callq f
# %rax is "long a" here
```

Not edi anymore. It's a general-purpose integer, not an argument.

the first floating point number register. (follow the calling conventions for floating point numbers).

cvtsi2sd convert signed integer to signed double.

For different types of the parameters, use different kinds of registers in order.

Functions Declaration

We want to know what f is so that we can call it appropriately.

```
int f(int x);
```

→ just a function header with a semicolon, no body.

just different names

- Declaration of the function — We leave it's implementation somewhere else.
- Function header
- Function signature
- Function prototype

We want this in every file that invokes `f()`

We want to know what f looks like in every single one of our C files.

Functions Definition We only want the function definition to appear in one place

```
int f(int x) {
    return 2130 * x;
}
```

- Definition of the function

We only want this in **one** `.c` file

- Do not want 2 definitions
- Which one should the linker choose?