X86_64, Patents

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

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

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

Example: Functions

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



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

```
        .globl  main
main:
        pushq   %rbp
        movq    $0, %rbp
condition:
        cmpq    $12, %rbp
        jg          after
        movq    %rbp, %rsi
        leaq    fmtstring(%rip), %rdi
        callq   printf
        addq    $1, %rbp
        jmp         condition
after:
        xorl    %eax, %eax
        popq    %rbp
        retq
fmtstring:
        .asciz  "i = %ld\n" _
```

```
        .globl  main
main:
        pushq   %rbp
    movq    $0x42, %rax
    movq    $0x15, %rbx
    movq    %rbx, %rsi
    negq    %rsi
    addq    %rax, %rsi
        leaq    fmtstring(%rip), %rdi
        callq   printf
    xorq    %rax, %rax
        popq    %rbp
        retq
fmtstring:
        .asciz  "%X\n"
```

The Stack

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

```
 .globl  main
main:
    pushq   %rbp

    # Set some example values in registers
    movq    $0x42, %rax
    movq    $0x15, %rbx
    movl    $4, %esi

    # Push 64-bit rax, then 16-bit si
    pushq   %rax
    pushw   %si

    # Pop -- oops!
    popq    %rdi
    popw    %si

    # Return 0 (all is well)
    xorq    %rax, %rax
    popq    %rbp
    retq
```

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!**

Patents and Copyright

Patents and Copyright

Remember our Toy ISA. Can we patent our ISA? Should we?

icode	b	meaning
0		`rA = rB`
1		`rA &= rB`
2		`rA += rB`
3	0	`rA = ~rA`
	1	`rA = !rA`
	2	`rA = -rA`
	3	`rA = pc`
4		`rA` = read from memory at address `rB`
5		write `rA` to memory at address `rB`
6	0	`rA` = read from memory at `pc + 1`
	1	`rA` &= read from memory at `pc + 1`
	2	`rA` += read from memory at `pc + 1`
	3	`rA` = read from memory at the address stored at `pc + 1`
		For icode 6, increase `pc` by 2 at end of instruction
7		Compare `rA` as 8-bit 2's-complement to 0
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal

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/

Patents and 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