X86_64, Patents

CS 2130: Computer Systems and Organization 1

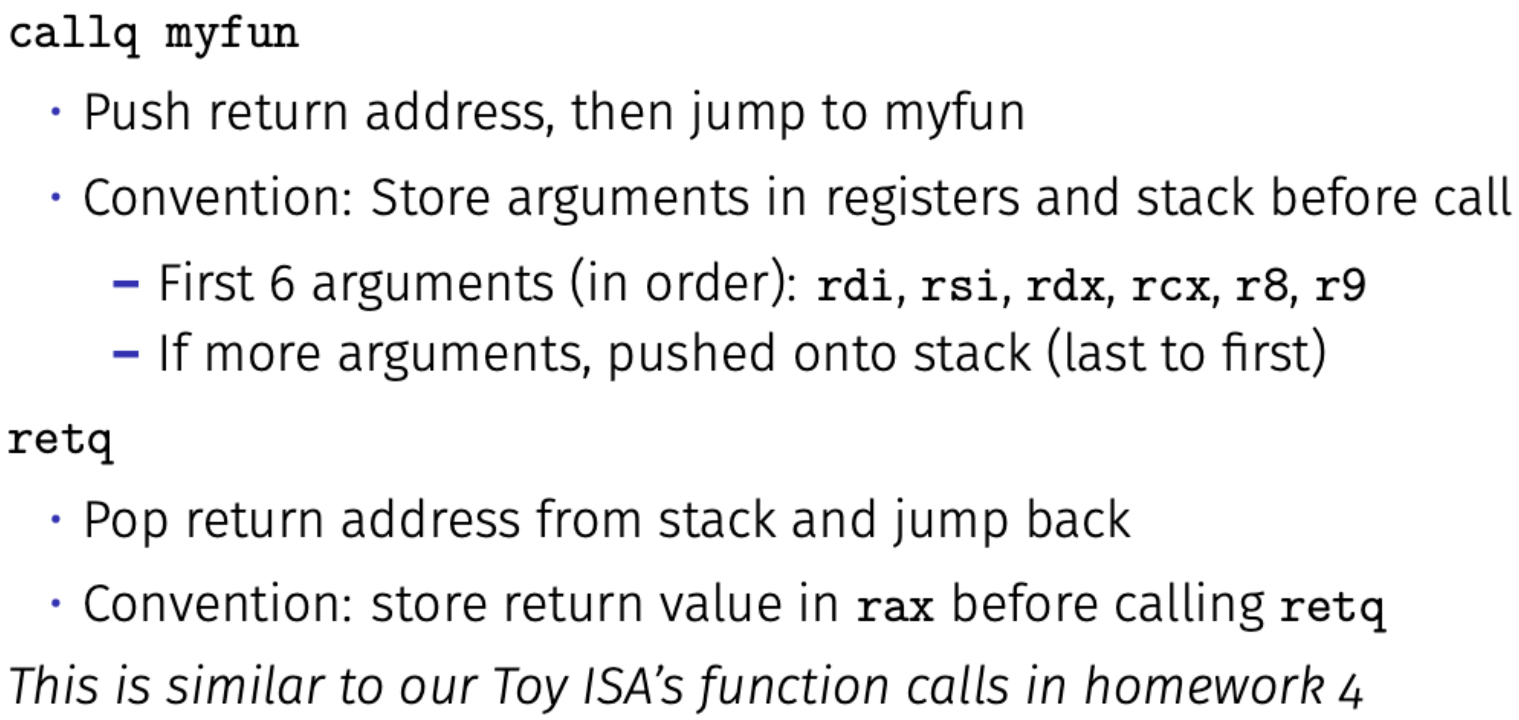
Xinyao Yi Ph.D.
Assistant Professor

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 for callee-save:

```
pushq  %rbx
moveq  $10, %rbx
popq   %rbx
retq
```

example for caller-save:

```
pushq  %rdi
callq  f
popq   %rdi
```

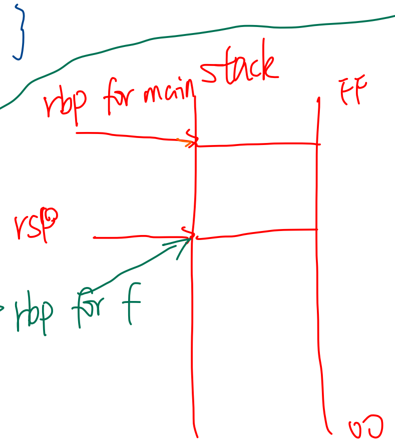
Example: Functions

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

```
int f(int x, int y) {
      return 4;
  }

int main ( ) {
    int z = f(2,5);
}
```

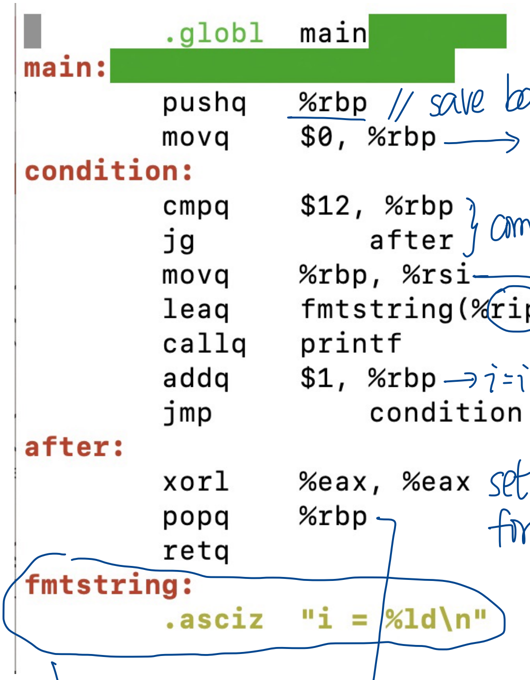
```
…
z = f(2,5)
```



```
.global f
f:
  pushq %rbp          //store old base pointer
  movq %rsp, %rbp     //create a new stack for f
  movl $4, %eax       // put return value 4 to %eax
  popq %rbp           // pop old base pointer (for main)
  retq                // return.
.global main

main:
  pushq %rbp
  movq %rsp, %rbp
```

```
  movl $2, %edi       //put parameter 2 to %edi
  movl $5, %esi       // put 5 to %esi
  callq f
  molv %eax, -4(%rbp) //store return value to local
                                  variable z.
```



```
        .globl  main [redacted]
main: [redacted]
        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"
```

// save base pointer.

use %rbp for i (not normal, but valid).

i=0

compare i with 12, if i>12, then jump out the loop, else, do the while loop.

while(i<=12)

put i to the register %rsi, which is used for the second parameter of printf

put fmtstring(%rip) to the register %rdi, which is used for the first parameter of printf.

i=i+1;

set eax=0 for return value

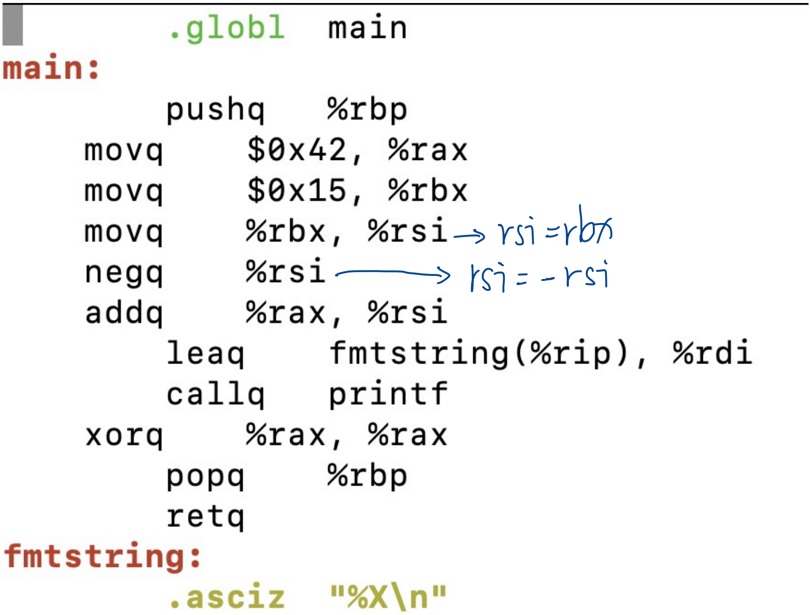
pop old base address

C style format printing

2 things to know:

①. %rip has the address of current instruction.

②. fmtstring(%rip) will calulate the address of fmtstring label using offset.



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