Recitation 7: Exam Stack Review

15-213: Introduction to Computer Systems
March 5, 2018

Instructor:

Your TA(s)

Midterm Exam This Week

- **3 hours + 1 hour for regrade requests**
- **1 double-sided page of notes**
  - No preworked problems from prior exams
- **7 questions**

- **Report to the room**
  - TA will verify your notes and ID
  - TAs will give you your exam server password
  - Login via Andrew, then navigate to exam server and use special exam password

Stack Review

- **In the following questions, treat them like the exam**
  - Can you answer them from memory?
  - Write down your answer
  - Talk to your neighbor, do you agree?

- **Discuss:**

  What is the stack used for?

Stack Manipulation

- **We execute:**

```
mov $0x15213, %rax
pushq %rax
```

- **Which of the following instructions will place the value 0x15213 into %rcx?**

```
1) mov (%rsp), %rcx
2) mov 0x8(%rsp), %rcx
3) mov %rsp, %rcx
4) popq %rcx
```

Stack Manipulation

- **We execute:**

```
mov $0x15213, %rax
pushq %rax
```

- **Which of the following instructions will place the value 0x15213 into %rcx?**

```
1) mov (%rsp), %rcx
2) mov 0x8(%rsp), %rcx
3) mov %rsp, %rcx
4) popq %rcx
```

Stack is memory

- **We execute:**

```
mov $0x15213, %rax
pushq %rax
popq %rax
```

- **If we now execute:** `mov -0x8(%rsp), %rcx` **what value is in %rcx?**

  1) 0x0 / NULL

  2) Seg fault

  3) Unknown

  4) 0x15213

Stack is memory

- **We execute:**

```
mov $0x15213, %rax
pushq %rax
popq %rax
```

- **If we now execute:** `mov -0x8(%rsp), %rcx` **what value is in %rcx?**

  1) 0x0 / NULL

  2) Seg fault

  3) Unknown

  [✓ **4) 0x15213**]

x86-64 Calling Convention

- What does the calling convention govern (select all that apply)?

  1) How large each type is.

  2) How to pass arguments to a function.

  3) The alignment of fields in a struct.

  4) When registers can be used by a function.

  5) Whether a function can call itself.

x86-64 Calling Convention

- What does the calling convention govern (select all that apply)?

    1) How large each type is.

    [✓ 2) How to pass arguments to a function.]

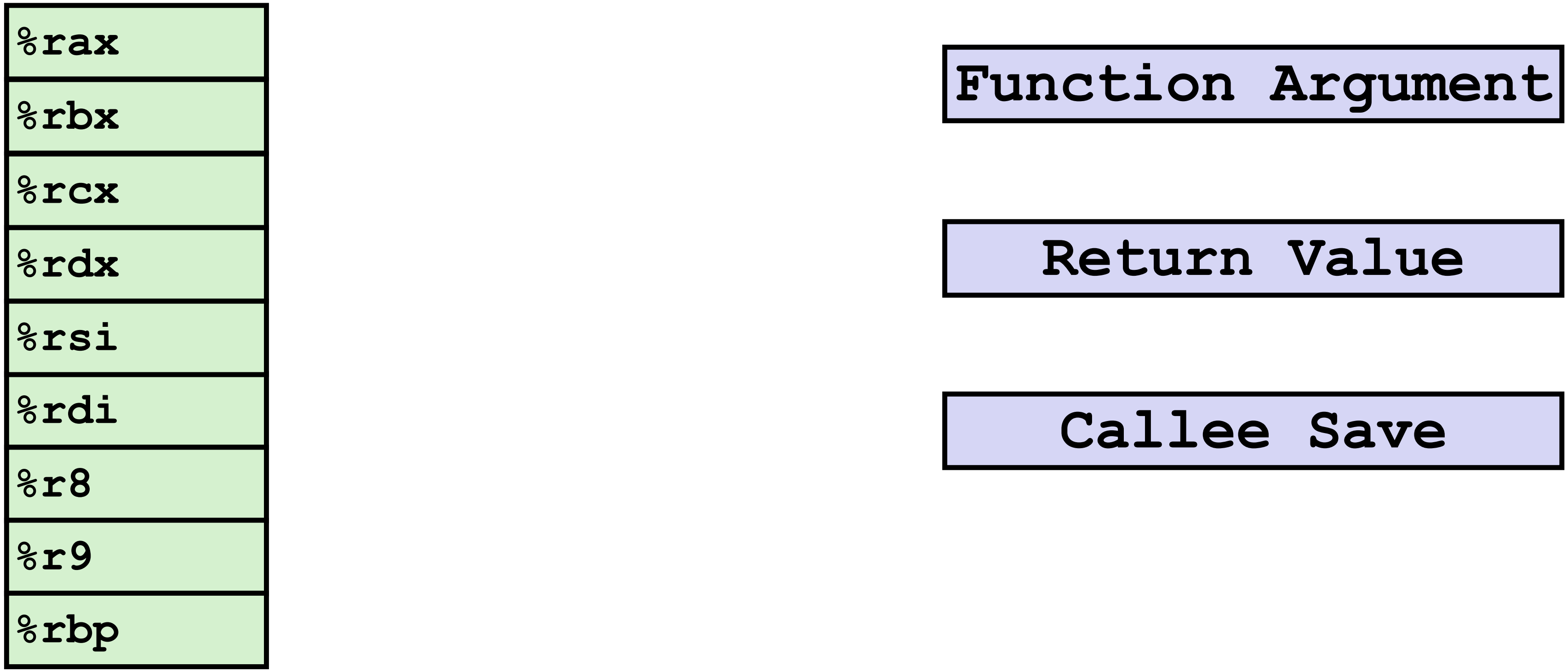
    3) The alignment of fields in a struct.

    [✓ 4) When registers can be used by a function.]

    5) Whether a function can call itself.

Register Usage

- **The calling convention gives meaning to every register, describe the following 9 registers:**



`%rax`

`%rbx`

`%rcx`

`%rdx`

`%rsi`

`%rdi`

`%r8`

`%r9`

`%rbp`

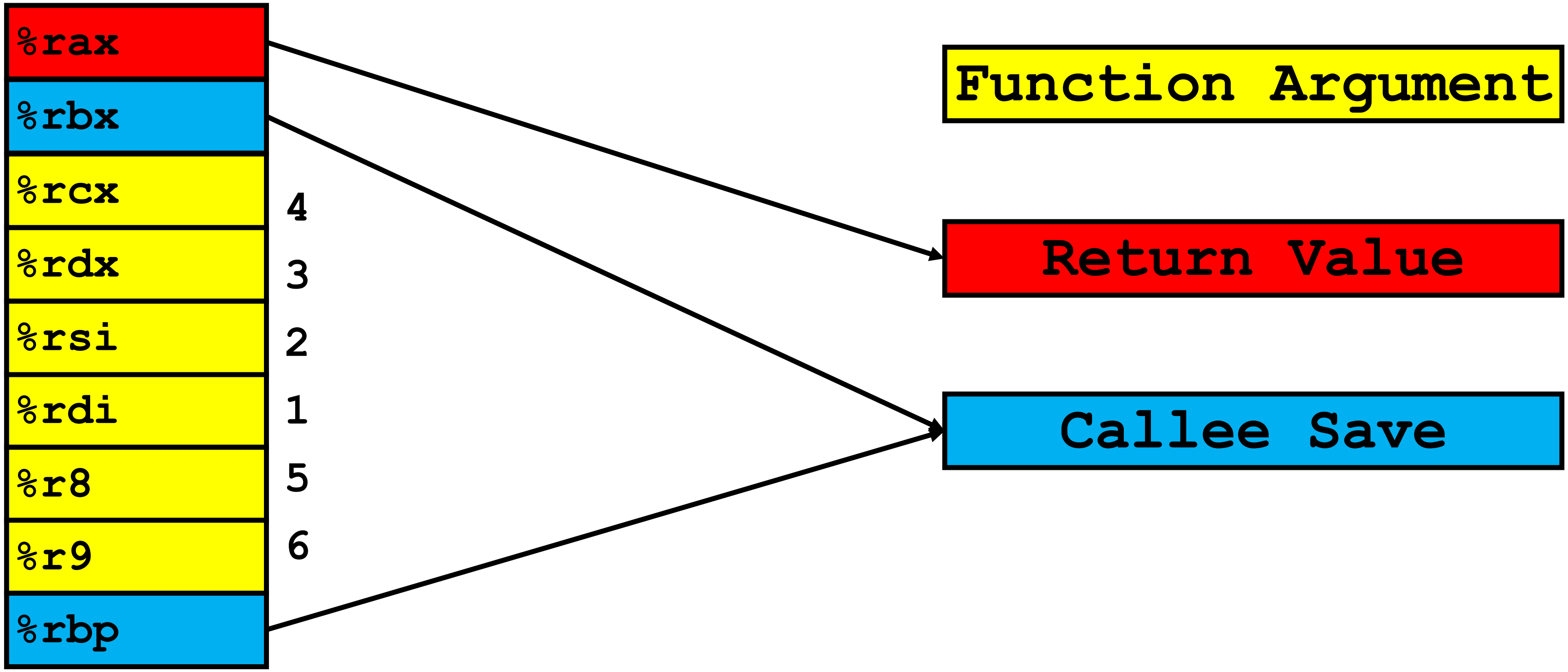
`Function Argument`

`Return Value`

`Callee Save`

Register Usage

- The calling convention gives meaning to every register, describe the following 9 registers:



Register Usage

- Which line is the first violation of the calling convention?

```
mov $0x15213, %rax
push %rax
mov 0x10(%rsp), %rcx
mov %rbx, %rax
pop %rdx
push %rax
pop %rbx
mov %rcx, %rbx
```

Register Usage

- **Which line is the first violation of the calling convention?**

```
mov $0x15213, %rax
push %rax
mov 0x10(%rsp), %rcx
mov %rbx, %rax
pop %rdx
push %rax
pop %rbx
mov %rcx, %rbx
```

Until this point, the callee has preserved the callee-save value.

Sometimes arguments are implicit

What is the minimum number of arguments that “rsr” takes?

How many of those registers are changed in the function before the function call?

(Note, %sil is the low 8 bits of %rsi)

```
0x0400596 <+0>:     cmp    %sil,(%rdi,%rdx,1)
0x040059a <+4>:     je     0x4005ae <rsr+24>
0x040059c <+6>:     sub    $0x8,%rsp
0x04005a0 <+10>:    sub    $0x1,%rdx
0x04005a4 <+14>:    callq  0x400596 <rsr>
0x04005a9 <+19>:    add    $0x8,%rsp
0x04005ad <+23>:    retq
0x04005ae <+24>:    mov    %edx,%eax
0x04005b0 <+26>:    retq
```

Sometimes arguments are implicit

What is the minimum number of arguments that “rsr” takes? 3

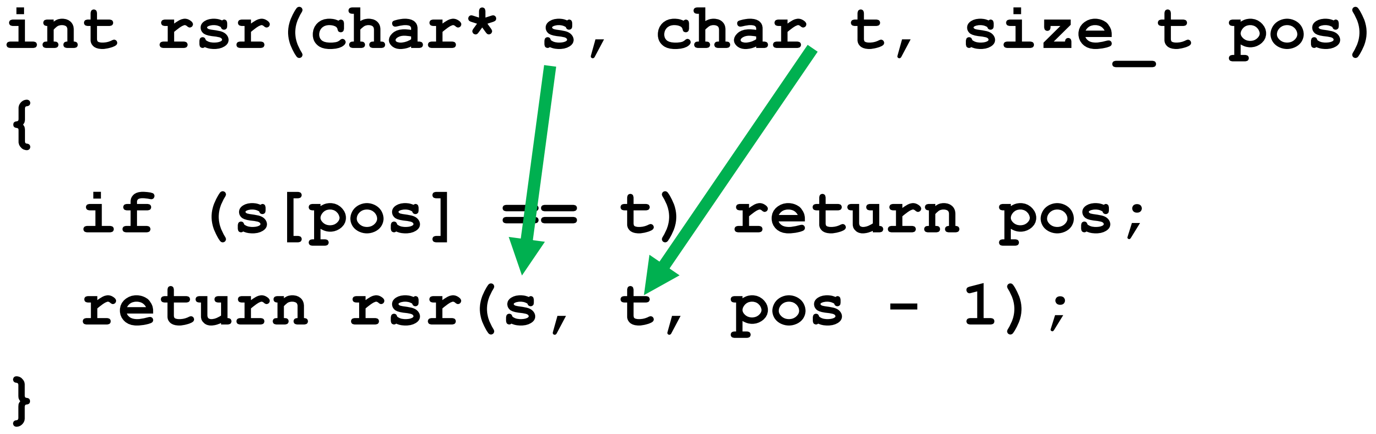
How many of those registers are changed in the function before the function call? 1

(Note, %sil is the low 8 bits of %rsi)

```
0x0400596 <+0>:     cmp    %sil,(%rdi,%rdx,1)
0x040059a <+4>:     je     0x4005ae <rsr+24>
0x040059c <+6>:     sub    $0x8,%rsp
0x04005a0 <+10>:    sub    $0x1,%rdx
0x04005a4 <+14>:    callq  0x400596 <rsr>
0x04005a9 <+19>:    add    $0x8,%rsp
0x04005ad <+23>:    retq
0x04005ae <+24>:    mov    %edx,%eax
0x04005b0 <+26>:    retq
```

Arguments can already be "correct"

- rsr does not modify s and t, so the arguments in those registers are always correct



```
int rsr(char* s, char t, size_t pos)
{
  if (s[pos] == t) return pos;
  return rsr(s, t, pos - 1);
}
```

Recursive calls

- **Describe the stack after doThis(4) returns.**

```
void doThis(int count)
{
    char buf[8];
    strncpy(buf, “Hi 15213”, sizeof(buf));
    if (count > 0) doThis(count – 1);
}
```

```
        push %rbx
        sub $0x10, %rsp
        mov    %edi,%ebx
        movabs $0x3331323531206948,%rax
        mov    %rax,(%rsp)
        . . .
```

Recursive calls

- Describe the stack after doThis(4) returns.

```
void doThis(int count)
{
    char buf[8];
    strncpy(buf, "Hi 15213", sizeof(buf));
    if (count > 0) doThis(count - 1);
}
```

```
push %rbx
sub $0x10, %rsp
mov    %edi,%ebx
movabs $0x3331323531206948,%rax
mov    %rax,(%rsp)
...
```

ascii representation of Hi 15213 in little endian

The stack will be normal – no buffer overflow with the local variables allocated on the stack and the calling function's return address on the stack

Also there will be 4 repeats of the three lines
48692051
35323133
doThis return address
above the current stack pointer (Note the string is stored in array index order in the stack)

Callee, Caller Stack Frames

```
000000000000068a <foo>:
 68a:    48 83 ec 08         sub   $0x8,%rsp
 68e:    e8 cd fe ff ff      callq 560 <rand@plt>
 693:    48 83 c4 08         add   $0x8,%rsp
 697:    c3                  retq

0000000000000698 <main>:
 698:    48 83 ec 08         sub   $0x8,%rsp
 69c:    bf 00 00 00 00      mov   $0x0,%edi
 6a1:    e8 aa fe ff ff      callq 550 <srand@plt>
 6a6:    b8 00 00 00 00      mov   $0x0,%eax
 6ab:    e8 da ff ff ff      callq 68a <foo>
```

At the start of the instruction at 68e, how large is the callee stack frame (the caller stack frame includes the return address to main)?

Callee, Caller Stack Frames

```
000000000000068a <foo>:
 68a:      48 83 ec 08          sub   $0x8,%rsp
 68e:      e8 cd fe ff ff       callq  560 <rand@plt>
 693:      48 83 c4 08          add   $0x8,%rsp
 697:      c3                   retq

0000000000000698 <main>:
 698:      48 83 ec 08          sub   $0x8,%rsp
 69c:      bf 00 00 00 00       mov   $0x0,%edi
 6a1:      e8 aa fe ff ff       callq  550 <srand@plt>
 6a6:      b8 00 00 00 00       mov   $0x0,%eax
 6ab:      e8 da ff ff ff       callq  68a <foo>
```

At the start of the instruction at 68e, how large is the callee stack frame (the caller stack frame includes the return address to main)?

0x8

Callee, Caller Stack Frames

- **Assume the same functions: foo and main (but recompiled with stack randomization)**
- **The output of the command** `gdb x/4gx` **is shown below for the line**

```
callq 560 <rand@plt>
```

```
0x7fffffffe010:
        0x00007fffffffe100
        0x00005555555546b0
0x7fffffffe020:
        0x0000000000000000
        0x00007ffff7a161c1
```

- **What is the return address of foo?**

Callee, Caller Stack Frames

- Assume the same functions: foo and main (but recompiled with stack randomization)
- The output of the command `gdb x/4gx` is shown below for the line

```
callq 560 <rand@plt>

0x7fffffffe010:
    0x00007fffffffe100
    0x00005555555546b0

0x7fffffffe020:
    0x0000000000000000
    0x00007ffff7a161c1
```

- What is the return address of foo?

  0x00005555555546b0