

Recitation 8: Exam Stack Review

15-213: Introduction to Computer Systems
Oct 16, 2017

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 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?**
 - 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:

<code>%rax</code>
<code>%rbx</code>
<code>%rcx</code>
<code>%rdx</code>
<code>%rsi</code>
<code>%rdi</code>
<code>%r8</code>
<code>%r9</code>
<code>%rbp</code>

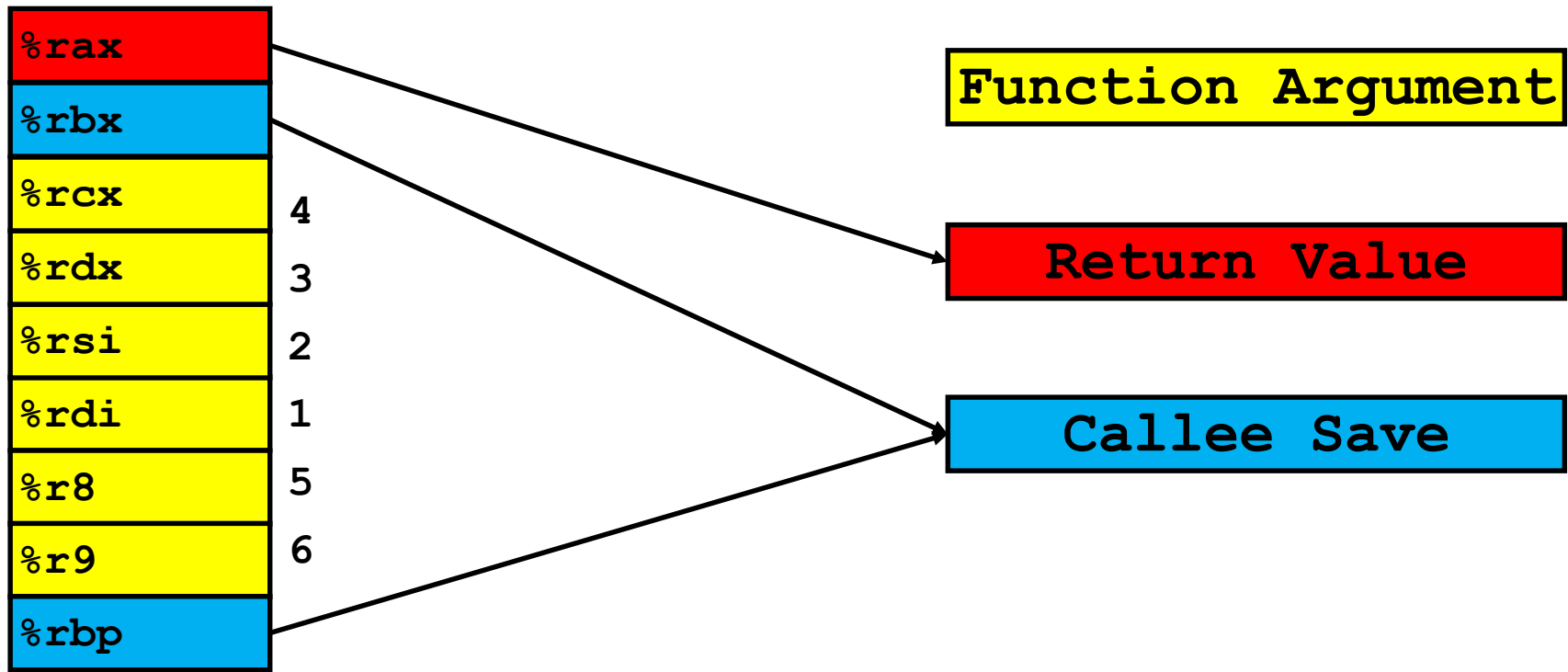
Function Argument

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

How many arguments does “rsr” take?

How many registers are changed before the function call?

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

```

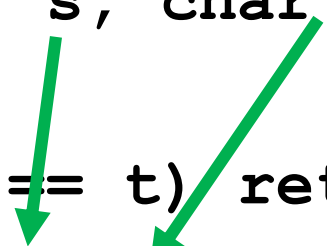
0x0400596 <+0>:      cmp      %sil, (%rdi,%rdx,1)
0x040059a <+4>:      je       0x4005ae <rsr+24>
0x040059c <+6>:      sub     $0x8,%rsp
0x04005a0 <+10>:     sub     $0x1,%rdx
0x04005a4 <+14>:     callq  0x400596 <rsr>
0x04005a9 <+19>:     add     $0x8,%rsp
0x04005ad <+23>:     retq
0x04005ae <+24>: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

```

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)
    ...
```