CSO-Recitation 09

CSCI-UA 0201-007

R09: Assessment 07 & More Assembly

Today’s Topics

- Assessment 07
- More Assembly
  - Procedure calls & data segment
- Breakout exercises

Procedure calls

Calling functions

How do you call functions?

- How do you actually start executing the code of a function?
  - Well, we know about jmp, does that help us? Why not?
- Do you need to do something before calling a function?
  - What?

How do you call functions?

```
mystrlen:
movl $0, %eax
jmp .condition
.loop:
addl $1, %eax
.condition:
movb (%rdi,%rax), %bl
cmp $0, %bl
jne .loop
```

```
main:
jmp mystrlen
```

How do you call functions?

```
mystrlen:
movl $0
jmp .
.loo
ad      eax
.co
mov     %rax), %bl
cmp $
jne .loop
```

```
main:
    mystrlen
```

How do you call functions?

```
mystrlen:
movl $0, %eax
jmp .condition
.loop:
addl $1, %eax
.condition:
movb (%rdi,%rax), %bl
cmp $0, %bl
jne .loop
// How do we get back?
```

```
main:
//Where are the arguments?
jmp mystrlen
```

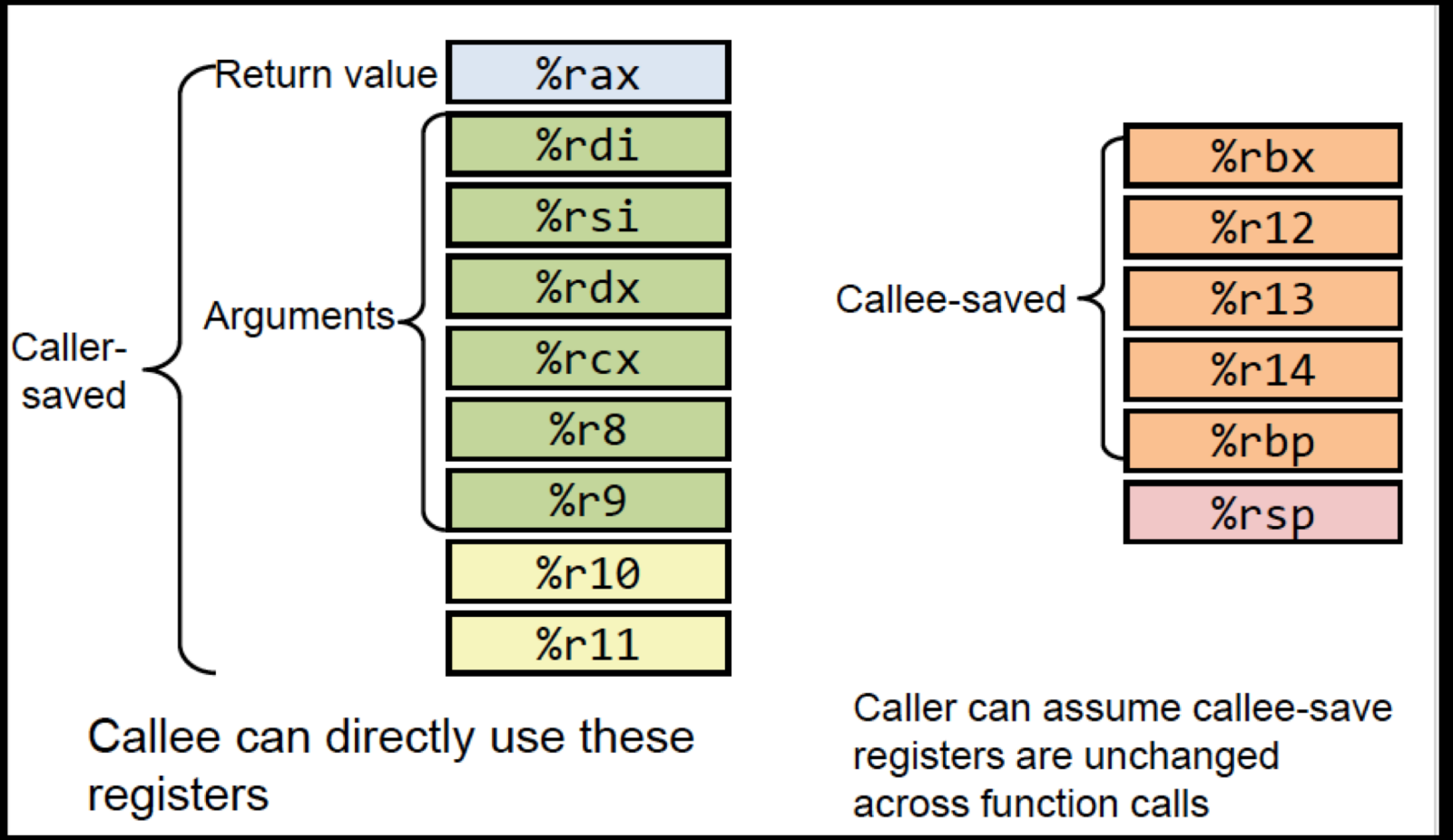
Remember where we came from

- A function that calls another (a caller) knows what it is calling
- A function that is called (a callee) does not know who its caller is
  - But it needs to know where to restore execution when it is done
  - It is the responsibility of the caller to tell the callee where to restore execution
  - We want to restore execution on the instruction after we called the function
  - We store this return address on the stack
    - callq handles this for us

Set up registers

- The first six arguments are stored in this order:
  - %rdi, %rsi, %rdx, %rcx, %r8, %r9
  - So when calling a function, you must set those registers to the correct value for that argument
- The return value is stored in %rax
- Functions may feel free to use the argument registers and the return value register, as well as %r10, and %r11
- If the caller was using the argument registers for something, it must save them first, as the callee may use those registers for any purpose
  - It can save them to the stack
  - This is also true of the registers %r10, %r11, and %rax
- The callee must save certain registers if it plans on using them
  - They are %rbx, %r12, %13, %r14, %rbp, and %rsp

Set up registers



The stack

- The register %rsp points to the top of the stack
- The stack grows downwards
- We use it to store return addresses as well as registers whose values we don’t want to lose
- We use it to store the 7th, 8th, 9th etc. function arguments
- We also use it to store local variables
- You can use pushq and popq to add and remove things from the stack

The Stack

- pushq src
- Takes one operand
- DECREASES %rsp by 8
- THEN stores the operand at the memory location given by the new %rsp

The Stack

- popq dst
- Takes one operand
- Takes the value in memory located at %rsp and stores it in the operand
- THEN INCREASES %rsp by 8

The Stack

- callq label
- Takes one operand
- DECREASES %rsp by 8
- THEN stores the return address at the memory location given by the new %rsp
- THEN jumps to the operand

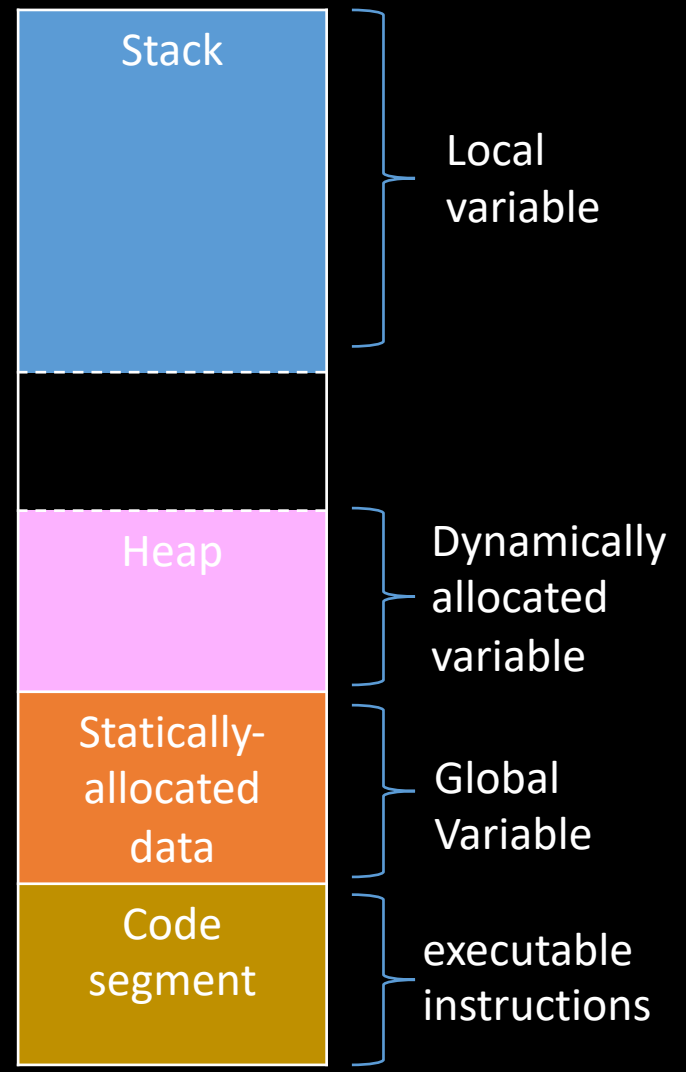
Push return address on stack

The Stack

- retq
- Takes no operands
- Jumps to the location given by the value in memory located at %rsp
- THEN INCREASES %rsp by 8

Data segment

- Local variables
  - Stack
    - C's primitive data type and pointer – registers whenever possible
    - Array, struct
- Global variables
  - global variable / static global variable
- Dynamic allocated variables
  - e.g. malloc
  - Heap



Example of Array/Struct accessing

- Array Accessing Example
  - int getnum(int *arr, long i) { return arr[i];}
  - Suppose %rdi contains arr; %rsi contains i; %eax is to contain arr[i]
  - movl (%rdi, %rsi, 4), %eax

  - char* getpointer(char **arr, long i) { return arr[i];}
  - Suppose %rdi contains arr; %rsi contains I; %rax is to contain arr[i]
  - movq (%rdi, %rsi, 8), %rax

Example of Array/Struct accessing

```
typedef struct node {
        long id;
        char *name;
        struct node *next;
}node;

void init_node(node*n, long id, char *name){
        n->id=id;
        n->name=name;
        n->next=NULL;
}
```

```
movq %rsi, (%rdi)
movq %rdx, 8(%rdi)
movq $0, 16(%rdi)
```

Exercise