Assembly: Procedures, function calls, stack discipline, recursion.

Yipeng Huang

Rutgers University

March 25, 2021

Table of contents

Looking ahead

Class plan

1. Today, Thursday, 3/25: Assembly procedures, function calls, stack discipline, recursion.
2. Starting next week: Recitations will have specialized topics for remainder of semester. `https://rutgers.instructure.com/courses/104725/pages/recitation-and-office-hour-information`
3. Starting next week: The memory hierarchy. Reading assignment, CS:APP Chapter 6.

Table of contents

Procedures and function calls

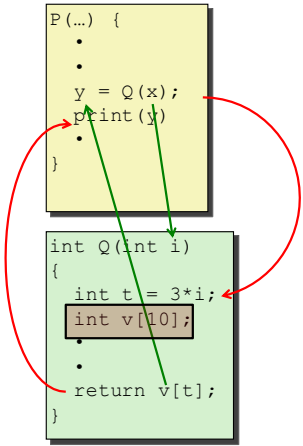


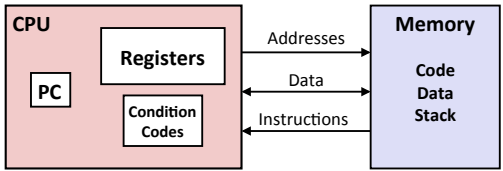
Figure: Steps of a C function call. Image credit CS:APP

To create the abstraction of functions, need to:

- Transfer control to function and back
- Transfer data to function (parameters)
- transfer data from function (return type)

CPU and memory state in support of procedures and functions

Assembly/Machine Code View



Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
  - Called "RIP" (x86-64)
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

Relevant state in CPU:

- %rip register / instruction pointer / program counter
- %rsp register / stack pointer

Relevant state in Memory:

- Stack

Stack instructions: `push` and `pop`

OP SRC DEST

PUSHQ SRC

POPQ DEST

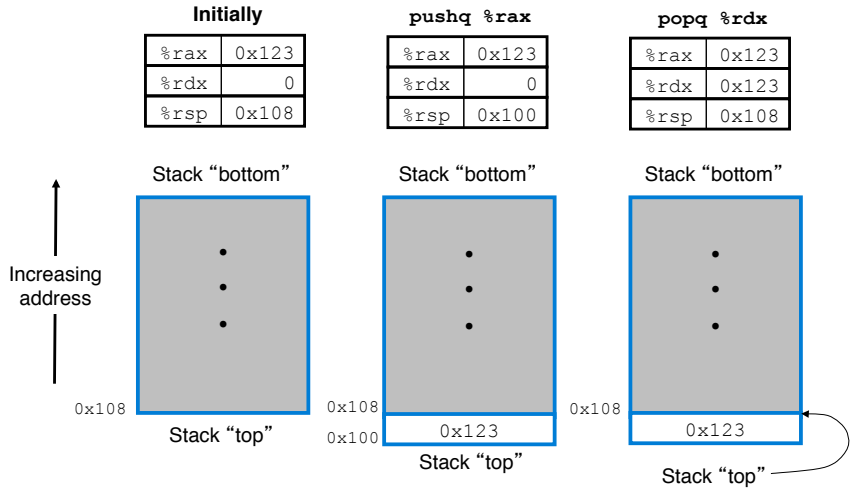
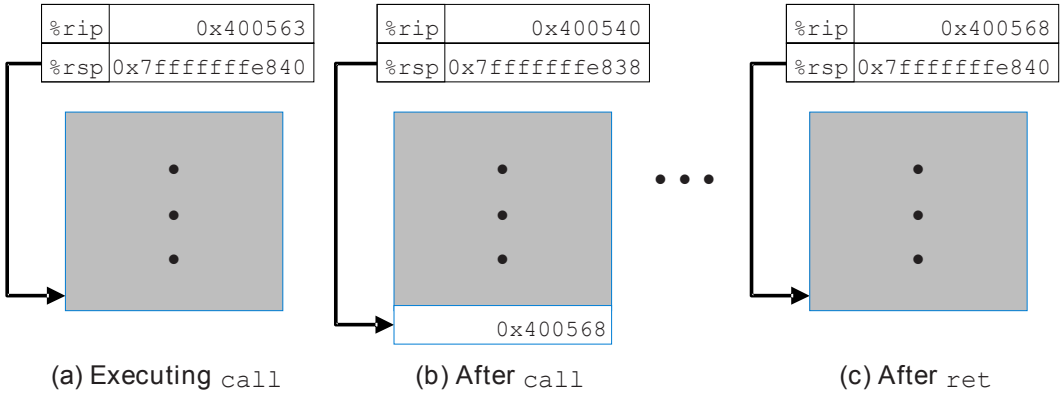


Figure: x86-64 offers dedicated instructions to work with stack in memory. In addition to moving data, the updating of %rsp is implied. Image credit: CS:APP.

Procedure call and return: `call` and `ret`



explode_bomb()

Figure: Effect of `call 0x400540` instruction and subsequent return. `call` and `ret` instructions update the instruction pointer, the stack pointer, and the stack to create the procedure / function call abstraction. Image credit: CS:APP.

Example in GDB

```c
#include <stdio.h>

int return_neg_one() {
  return -1;
}

int main() {
  int num = return_neg_one();
  printf("%d", num);
  return 0;
}
```

```
return_neg_one:
    movl $-1, %eax
    ret
main:
    subq $8, %rsp
    movl $0, %eax
    call return_neg_one
    movl %eax, %edx
    ...
```

Compile, and then run it in GDB:

```
gdb return
```

In GDB, see evolution of %rip, %rsp, and stack:

- `(gdb) layout split`
- `(gdb) break return_neg_one`
- `(gdb) print /a $rip`
- `(gdb) print /a $rsp`
- `(gdb) x /a $rsp`

Step past return instruction, and inspect again:

- `(gdb) stepi`

Table of contents

Procedures and function calls: Transferring data

For purposes of this class, the Bomb Lab, and the CS:APP textbook, we study the x86-64 Linux Application Binary Interface (ABI). Would be different on ARM or in Windows. So, don't memorize this, but it is helpful for PA4 Bomb Lab.

Passing parameters

Parameter	Register / stack	Subset registers	Mnemonic[1]
1st	%rdi	%edi, %di	Diane's
2nd	%rsi	%esi, %si	silk
3rd	%rdx	%edx, %dx, %dl	dress
4th	%rcx	%ecx, %cx, %cl	cost
5th	%r8	%r8d	$8
6th	%r9	%r9d	9
7th and beyond	Stack		

[1] http://csappbook.blogspot.com/2015/08/dianes-silk-dress-costs-89.html

Procedures and function calls: Transferring data

Passing function return data

Function return data is passed via:

- the 64-bit %rax register
- the 32-bit subset %eax register