Machine-level representation of programs: procedures, recursion, arrays, structs

Yipeng Huang

Rutgers University

March 31, 2022
Table of contents

Announcements

Procedures and function calls
   Memory stack frames

Procedures and function calls: Transferring control
   Procedure call and return: call and ret
   Example in GDB

Procedures and function calls: Transferring data
   Data transferred via registers
   Data transferred via memory

Architecture support for recursive programming
Announcements

PA4 bomb lab
▶ PA4 bomb lab out and live. Due Tuesday, April 5.
▶ Class discussion on what each phase demonstrates.

Short quiz this week
Short quiz on assembly basics and control spanning Tuesday 3/29 to Friday 4/1.

Class session plan
▶ Today, Thursday, 3/31: Arrays and data structures in assembly. (Book chapter 3.8)
▶ Starting next week, Chapter 6, The Memory Hierarchy
Table of contents

Announcements

Procedures and function calls
   Memory stack frames

Procedures and function calls: Transferring control
   Procedure call and return: call and ret
   Example in GDB

Procedures and function calls: Transferring data
   Data transferred via registers
   Data transferred via memory

Architecture support for recursive programming
Procedures and function calls

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)

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

Structure of stack for currently executing function Q()

- P() calls Q(). P() is the caller function. Q() is the callee function.
Stack instructions: \texttt{push} \texttt{src} \texttt{and} \texttt{pop} \texttt{dest}

Initially

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>0x123</td>
</tr>
<tr>
<td>%rdx</td>
<td>0</td>
</tr>
<tr>
<td>%rsp</td>
<td>0x108</td>
</tr>
</tbody>
</table>

\texttt{pushq} \%rax

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>0x123</td>
</tr>
<tr>
<td>%rdx</td>
<td>0</td>
</tr>
<tr>
<td>%rsp</td>
<td>0x100</td>
</tr>
</tbody>
</table>

\texttt{popq} \%rdx

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>0x123</td>
</tr>
<tr>
<td>%rdx</td>
<td>0x123</td>
</tr>
<tr>
<td>%rsp</td>
<td>0x108</td>
</tr>
</tbody>
</table>

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.
Table of contents

Announcements

Procedures and function calls
   Memory stack frames

Procedures and function calls: Transferring control
   Procedure call and return: call and ret
   Example in GDB

Procedures and function calls: Transferring data
   Data transferred via registers
   Data transferred via memory

Architecture support for recursive programming
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
- Code
- Data
- Stack

Relevant state in CPU:
- %rip register / instruction pointer / program counter
- %rsp register / stack pointer

Relevant state in Memory:
- Stack
Procedure call and return: `call` and `ret`

(a) Executing `call`

(b) After `call`

(c) After `ret`

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

```
#include <stdio.h>

int return_neg_one() {
    return -1;
}

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

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` info stack
- `gdb` print /a $rip
- `gdb` print /a $rsp
- `gdb` x /a $rsp

Step past return instruction, and inspect again:

- `gdb` stepi
Announcements

Procedures and function calls
  Memory stack frames

Procedures and function calls: Transferring control
  Procedure call and return: call and ret
  Example in GDB

Procedures and function calls: Transferring data
  Data transferred via registers
  Data transferred via memory

Architecture support for recursive programming
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 Lab.

Passing parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Register / stack</th>
<th>Subset registers</th>
<th>Mnemonic¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>%rdi</td>
<td>%edi, %di</td>
<td>Diane’s</td>
</tr>
<tr>
<td>2nd</td>
<td>%rsi</td>
<td>%esi, %si</td>
<td>silk</td>
</tr>
<tr>
<td>3rd</td>
<td>%rdx</td>
<td>%edx, %dx, %dl</td>
<td>dress</td>
</tr>
<tr>
<td>4th</td>
<td>%rcx</td>
<td>%ecx, %cx, %cl</td>
<td>cost</td>
</tr>
<tr>
<td>5th</td>
<td>%r8</td>
<td>%r8d</td>
<td>$8</td>
</tr>
<tr>
<td>6th</td>
<td>%r9</td>
<td>%r9d</td>
<td>9</td>
</tr>
<tr>
<td>7th and beyond</td>
<td>Stack</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PA4 Defusing a Binary Bomb: `sscanf()` ;

```c
int sscanf ( 
    const char *str, // 1st arg, %rdi
    const char *format, // 2nd arg, %rsi
    ...
)
```
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

Example from textbook slides on assembly procedures
Slides 33 through 38.
Data transferred via memory

Structure of stack for currently executing function Q()

- P() calls Q(). P() is the caller function. Q() is the callee function.

Example from textbook slides on assembly procedures

Slides 40 through 44.
Table of contents

Announcements

Procedures and function calls
  Memory stack frames

Procedures and function calls: Transferring control
  Procedure call and return: call and ret
  Example in GDB

Procedures and function calls: Transferring data
  Data transferred via registers
  Data transferred via memory

Architecture support for recursive programming
Discussion points

▶ Use info stack, info args in GDB to see recursion depth
▶ Difference between compiling with and without -g for debugging information.
▶ Memory costs of recursion.
▶ Compilers can recognize tail recursive calls to reduce memory use. Enabled with -foptimize-sibling-calls, -O2, -O3, and -Os.