Assembly

Where we stand and where we go

Cauchy Pu

Outline

Overview

Computers execute machine code, sequences of bytes encoding the low-level operations that manipulate data, manage memory, read and write data on storage devices, and communicate over networks.

Bits and Where

Information = Bits + Context

So we study what bits and in which we view them

Assembly!

Why assembly

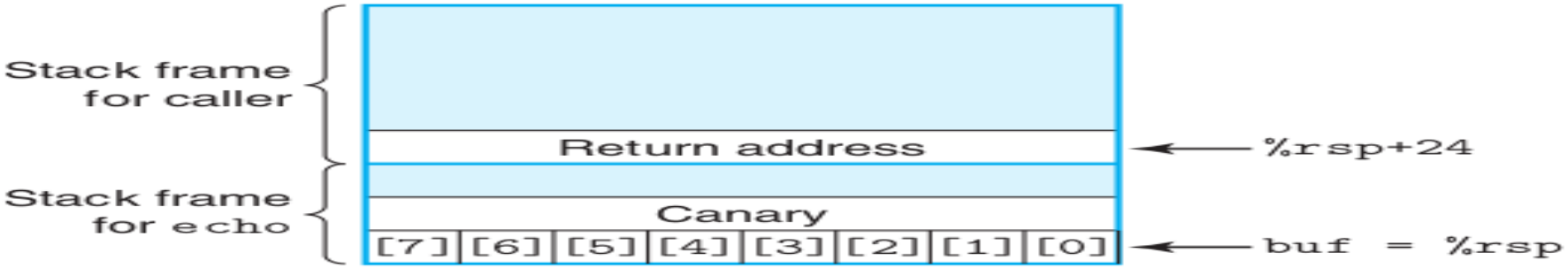
But, compilers do most of the work in generating assembly code, so why sould we spend our time?

1. Optimization. Sometimes we have to try the assembly code corresponding to various forms of upper language.
2. Some bugs more obvious in assembly. For exmaple, concurrent programs.
3. Security. Overwrite information is a common attack method.
4. Some code must be assembly, context switch.
5. When you encounter core dump, what the corresponding high level code?
6. And you told me, we are low-level engineer, right? So, here we go, assembly!
7. .....

Think like a computer

Why assembly

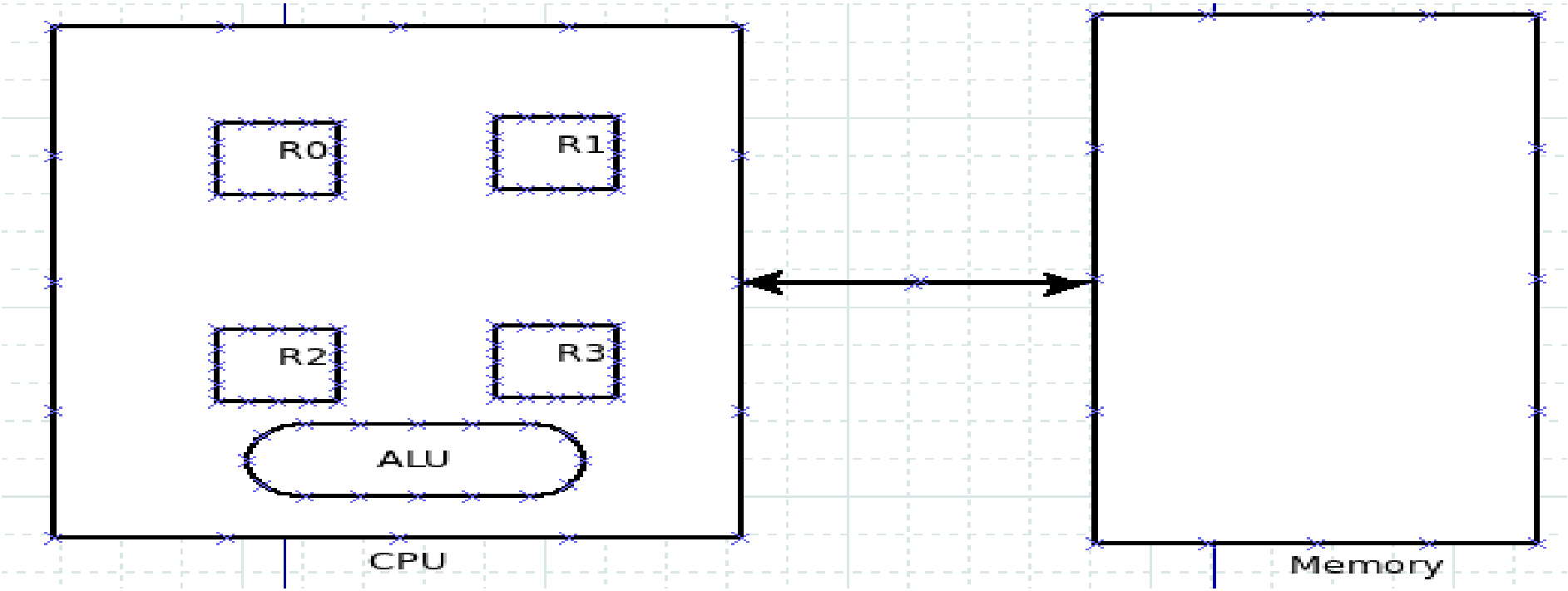
Example




```
1     echo:
2       subq    $24, %rsp               Allocate 24 bytes on stack
3       movq    %fs:40, %rax            Retrieve canary
4       movq    %rax, 8(%rsp)           Store on stack
5       xorl    %eax, %eax              Zero out register
6       movq    %rsp, %rdi              Compute buf as %rsp
7       call    gets                    Call gets
8       movq    %rsp, %rdi              Compute buf as %rsp
9       call    puts                    Call puts
10      movq    8(%rsp), %rax           Retrieve canary
11      xorq    %fs:40, %rax            Compare to stored value
12      je      .L9                     If =, goto ok
13      call    __stack_chk_fail        Stack corrupted!
14    .L9:                            ok:
15      addq    $24, %rsp               Deallocate stack space
16      ret
```

A simple world

In the world of assembly...



Simple, right?

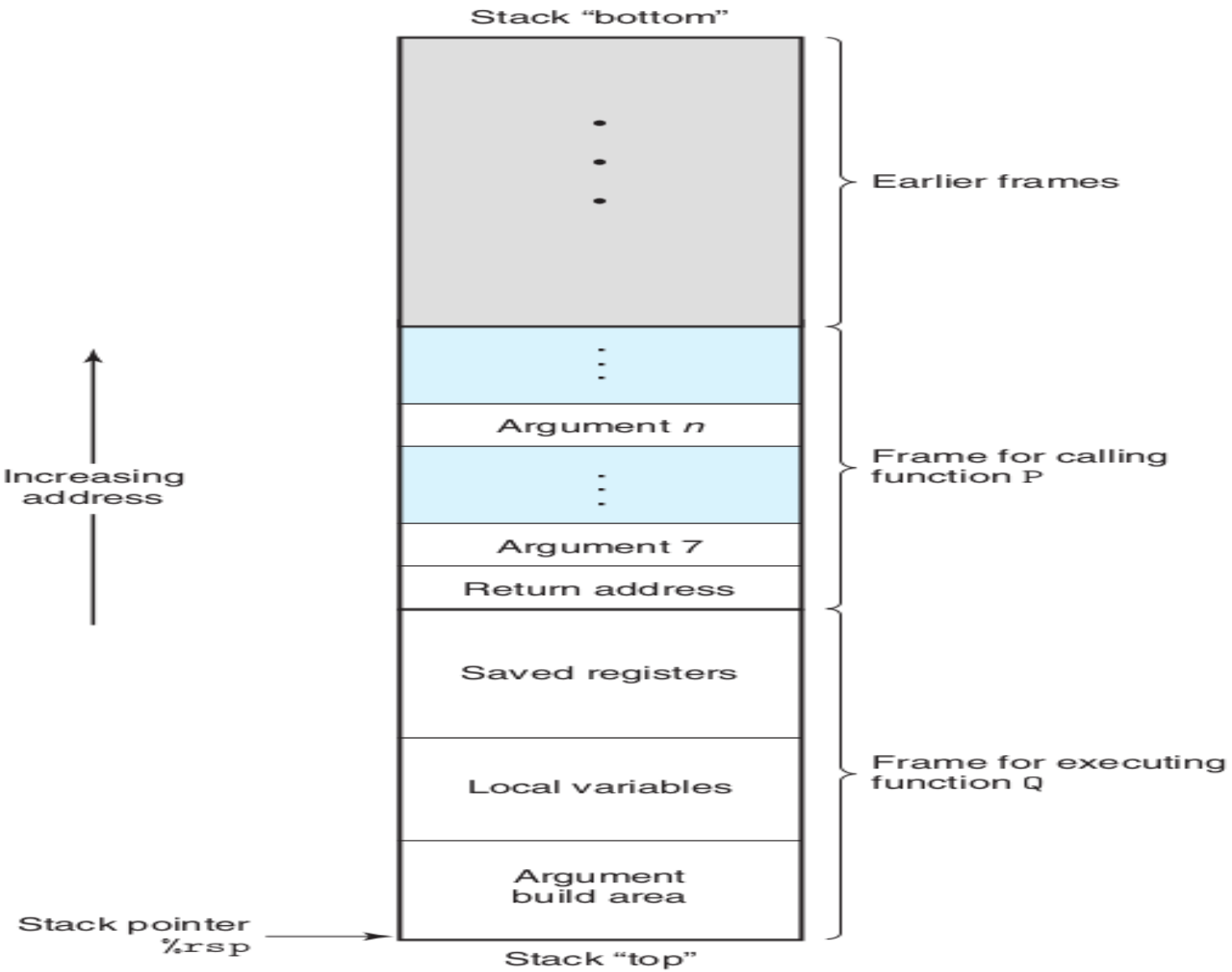
A simple world

The class of instructions:

1. assessing
   - mov
   - ldl
   - stl
2. arithmetic and logical operations
   - add
   - sub
3. control & procedures
   - jump
   - call
   - ret

A simple world

How stack operations...



A simple world

I want struct, array and many functions...

A simple world

Now some real feed...

Example

```
.section .data
data_items: .long 3,67,34,222,45,75,54,34,44,33,22,11,66,0
.section .text
```

A simple world

Some caveat

Inline Assembly

When you travel happily in the kernel source...

Damn!

```
__asm__ ("swp %0, %0, [%1]" : : "r"(val), "r"(addr));
```

This is inline assembly

How many architectures

Agreements

Reverse Engineering

Some friends you need...

Questions

ANY
QUESTIONS?

References I


Randal E. Bryant, David R. O'Hallaron
Computer Systems, A prorgammer's perspective.
Pearson, 2018


Jonathan Bartlett
Programming from the Ground Up.


John R. Levine
Linkers & Loaders.


Sandeep.S
GCC-Inline-Assembly-HOWTO, 2003.
https://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html

Thanks

Thank you ;)!

pqy7172@gmail.com