

# Secure Software Development

Introduction and Low Level

**Daniel Gruss, Vedad Hadzic, Andreas Kogler, Martin Schwarzl, Marcel Nageler**

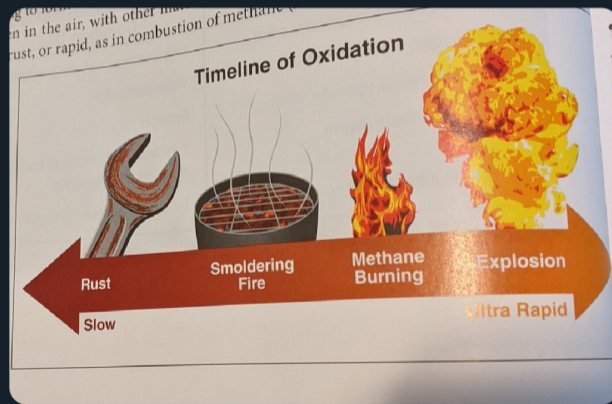
October 13, 2022



edunhome  
@QEDunham

Fun fact rust is basically just an extremely slow explosion

[Tweet übersetzen](#)





Karl  
@supersat

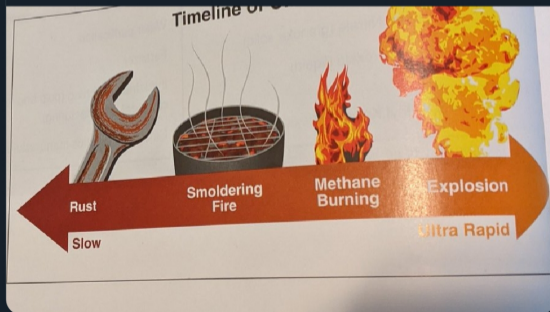
if you need a faster explosion, use  
C++

[Tweet übersetzen](#)



edunhome @QEDunham · 10 Std.

Fun fact rust is basically just an extremely slow explosion







1. Privileges and Context Switches
2. Memory Layout and ELF
3. Calling Conventions
4. C++ Classes

## **Privileges and Context Switches**

---

Memory configured through page tables

- Read / write
- Non-executable
  
- Separate configurations for different mappings and processes
- Separate configuration for kernel

Act as if:

- Thread was running already
- We are returning from an interrupt

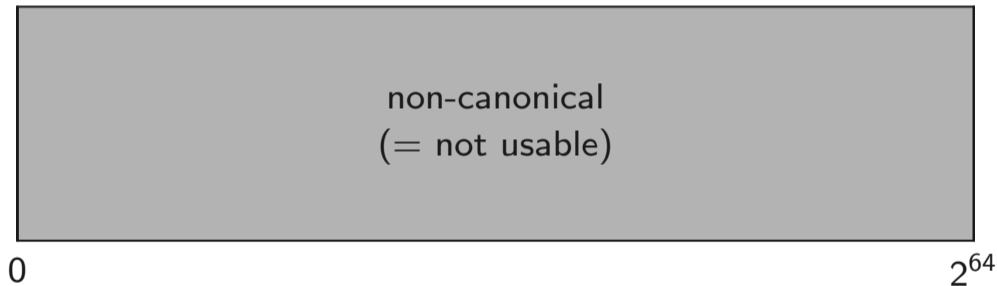


1. “Restore” CPU register values
  - Push stored register values to stack (modifies registers)
    - `iret` (interrupt return) expects `ss`, `esp`, `eflags`, `cs`, `eip` on the stack
    - `iret` pops values from stack into the registers
2. Instruction pointer has a new value, execution continues there

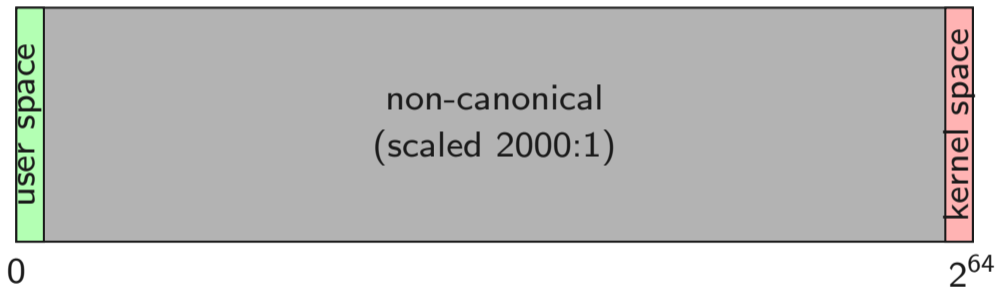
- Caused only by an Interrupt → Privilege level change
- CPU pushes to stack: `ss`, `esp`, `eflags`, `cs`, `eip`
- Store register values
  - Push all CPU register values on the stack
  - Only `eip` and `esp` are modified (which are already saved)
  - Copy all CPU register values from the stack into a struct
- Old thread executes scheduling code (“the Scheduler”)
- Context switch to new thread as before

# Memory Layout and ELF

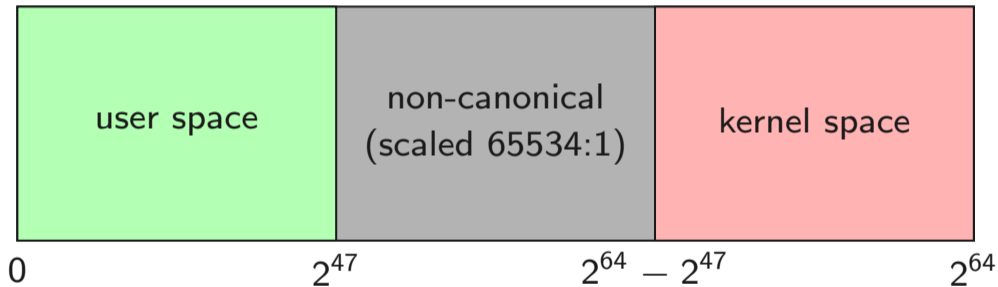
---



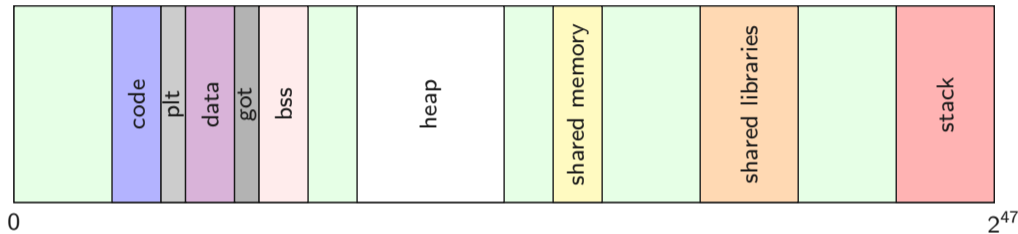
# Memory Layout (scaled)



# Memory Layout (scaled)



# Memory Layout User Space



Let's take a closer look at these 8 regions...



- Executable
- Usually readable
- Usually not writable



```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                34 00 20 00 01 00
40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00
60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

## ELF HEADER

IDENTIFY AS AN ELF TYPE  
SPECIFY THE ARCHITECTURE

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1ELFCLASS32, 1ELFDATA2LSB
EI_VERSION	1EV_CURRENT
e_type	2ET_EXEC
e_machine	3EM_386
e_version	1EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001
p_type	1PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5PF_R PF_X

## PROGRAM HEADER TABLE

EXECUTION INFORMATION

## CODE

X86 ASSEMBLY      EQUIVALENT C CODE

```
mov ebx, 42
mov eax, 1
int 80h
```

→ SC\_EXIT → return 42;



**Write an ELF binary which outputs your immatriculation number and exits**

- Size of the binary must be  $\leq 100$  bytes
- Binary must be a 32-bit or 64-bit ELF binary which runs in the reference VM
- Binary does not have to be 100% valid as specified in the standard
- Use whatever tool(s) you like, source code is not required

## **PLT and GOT**

---

Dynamic loader (part of the OS)

- Loads dynamically linked libraries / shared libraries
- Maps corresponding memory on-demand
- Makes addresses valid on-demand (GOT)

→ But what address is compiled into the binary then?

```
14c0: ff 35 72 79 20 00  pushq  0x207972(%rip)
14c6: ff 25 74 79 20 00  jmpq   *0x207974(%rip)
14cc: 0f 1f 40 00        nopl   0x0(%rax)
```

- Trampoline to real code
- Indirect jumps
- (typically) 16 bytes of code

- Addresses of exported/external variables and functions
- Absolute address (64 bit pointers → 8 bytes)
- Special part of the GOT:  
.got.plt: Addresses of exported/external functions used by the PLT

# But what if I want to call a GOT function?



- .plt.got stubs
- Fixed offset stub for GOT entries
- Typically 8 bytes per stub



```
#include <stdio.h>
int main()
{
    puts("hello world\n");
    return 0;
}
```



# GOT/PLT Example (ASM)

```
00000000000000570 <_start>:
570:   xor    %ebp,%ebp
572:   mov    %rdx,%r9
575:   pop   %rsi
576:   mov   %rsp,%rdx
579:   and   $0xfffffffffffffffff0,%rsp
57d:   push  %rax
57e:   push  %rsp
57f:   lea   0x1aa(%rip),%r8 # __libc_csu_fini
586:   lea   0x133(%rip),%rcx # __libc_csu_init
58d:   lea   0x10c(%rip),%rdi # main
>594:   callq *0x200a3e(%rip) # __libc_start_main@GLIBC_2.2.5
59a:   hlt
59b:   nopl  0x0(%rax,%rax,1)
```

$$0x200a3e + 0x59a = 0x200fd8$$

Relative addressing: GOT offset  $\leftrightarrow$  instruction offset is constant

# GOT/PLT Example (ASM)

```
000000000200fb0 <_GLOBAL_OFFSET_TABLE_>:  
 200fb0: f0 0d 20 00 00 00 00 00  
 200fb8: 00 00 00 00 00 00 00 00  
 200fc0: 00 00 00 00 00 00 00 00  
 200fc8: 00 00 00 00 00 00 00 00  
 200fd0: 00 00 00 00 00 00 00 00  
> 200fd8: 00 00 00 00 00 00 00 00  
 200fe0: 00 00 00 00 00 00 00 00  
 200fe8: 00 00 00 00 00 00 00 00  
 200ff0: 00 00 00 00 00 00 00 00  
 200ff8: 00 00 00 00 00 00 00 00
```

GOT is filled at runtime.

# GOT/PLT Example (ASM)

```
000000000020300 <__libc_start_main@@GLIBC_2.2.5>:
# ...
203de:    48 8b 74 24 08      mov     0x8(%rsp),%rsi
203e3:    8b 7c 24 14        mov     0x14(%rsp),%edi
203e7:    48 8b 10          mov     (%rax),%rdx
203ea:    48 8b 44 24 18    mov     0x18(%rsp),%rax
>203ef:    ff d0            callq  *%rax                # main
203f1:    89 c7            mov     %eax,%edi
203f3:    e8 b8 9e 01 00    callq  3a2b0 <exit@@GLIBC_2.2.5>
# ...
```

# GOT/PLT Example (ASM)

00000000000000570 <\_start>:

6a0: **sub** \$0x8,%rsp

>6a4: **lea** 0x99(%rip),%rdi

6ab: **callq** 560 <.plt.got>

6b0: **mov** \$0x0,%eax

6b5: **add** \$0x8,%rsp

6b9: **retq**

6ba: **nopw** 0x0(%rax,%rax,1)

$$0x6ab + 0x99 = 0x744$$



```
>744: 68 65 6c 6c 6f 20 77 6f 72 6c 64 0a 00 "hello world\n"
```

# GOT/PLT Example (ASM)

```
00000000000000570 <_start>:  
6a0:   sub    $0x8,%rsp  
6a4:   lea   0x99(%rip),%rdi  
>6ab:   callq 560 <.plt.got>  
6b0:   mov   $0x0,%eax  
6b5:   add   $0x8,%rsp  
6b9:   retq  
6ba:   nopw  0x0(%rax,%rax,1)
```

0000000000000560 <.plt.got>:

```
>560:  jmpq  *0x200a6a(%rip)  # puts@GLIBC_2.2.5
```

```
566:  xchg  %ax,%ax
```

$$0x200a6a + 0x566 = 0x200fd0$$

```
0000000000200fb0 <_GLOBAL_OFFSET_TABLE_>:  
 200fb0: f0 0d 20 00 00 00 00 00  
 200fb8: 00 00 00 00 00 00 00 00  
 200fc0: 00 00 00 00 00 00 00 00  
 200fc8: 00 00 00 00 00 00 00 00  
> 200fd0: 00 00 00 00 00 00 00 00  
 200fd8: 00 00 00 00 00 00 00 00  
 200fe0: 00 00 00 00 00 00 00 00  
 200fe8: 00 00 00 00 00 00 00 00  
 200ff0: 00 00 00 00 00 00 00 00  
 200ff8: 00 00 00 00 00 00 00 00
```

And finally arrived in the shared library through absolute address in `0x200fd0`.



```
int test()  
{  
    return 4 + 4;  
}
```

000000000000006a0 <test>:

6a0:	55	<b>push</b>	<b>%rbp</b>	# prologue
6a1:	48 89 e5	<b>mov</b>	<b>%rsp,%rbp</b>	# prologue
6a4:	b8 08 00 00 00	<b>mov</b>	<b>\$0x8,%eax</b>	# body
6a9:	5d	<b>pop</b>	<b>%rbp</b>	# epilogue
6aa:	c3	<b>retq</b>		# epilogue

# Calling Conventions

---



- Many different calling conventions
  - passing arguments
    - register contention vs. slow stack memory accesses
  - responsibility
    - is it easier if the caller/callee save something?
- focus on cdecl (x86\_32) and System V AMD64 ABI (x86\_64)

```
size_t add(size_t a, size_t b) {  
    return a + b;  
}
```

```
int main() {  
    size_t a = 7;  
    size_t b = 14;  
    size_t c = 0;  
  
    c = add(a, b);  
  
    return c;  
}
```

**cdecl (x86\_32)**

# Assembler Program (gcc -S -m32)

main:

```
> pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl $7, -12(%ebp)
movl $14, -8(%ebp)
movl $0, -4(%ebp)
pushl -8(%ebp)
pushl -12(%ebp)
call add
addl $8, %esp
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

add:

```
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %edx
movl 12(%ebp), %eax
addl %edx, %eax
popl %ebp
ret
```

0xffff0	<ebp>	← %esp
0xffec		
0xffe8		
0xffe4		
0xffe0		
0xffdc		
0xffd8		
0xffd4		
0xffd0		

%ebp	<ebp>
%esp	0xffff0
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

main:

```
    pushl   %ebp
>   movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl  -4(%ebp), %eax
    leave
    ret
```

add:

```
    pushl  %ebp
    movl  %esp, %ebp
    movl  8(%ebp), %edx
    movl  12(%ebp), %eax
    addl  %edx, %eax
    popl  %ebp
    ret
```

0xffff0	<ebp>	← %esp, %ebp
0xffec		
0xffe8		
0xffe4		
0xffe0		
0xffdc		
0xffd8		
0xffd4		
0xffd0		

%ebp	<ebp>	0xffff0
%esp		0xffff0
%eax		??
%edx		??



# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    > subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl   -8(%ebp)
    pushl   -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %esp, %ebp
0xffec		
0xffe8		
0xffe4		
0xffe0		← %esp
0xffdc		
0xffd8		
0xffd4		
0xffd0		

%ebp		0xffff0
%esp	0xffff0	0xffe0
%eax		??
%edx		??

# Assembler Program (gcc -S -m32)

main:

```
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
>    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

add:

```
    pushl  %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec		
0xffe8		
0xffe4	7	
0xffe0		← %esp
0xffdc		
0xffd8		
0xffd4		
0xffd0		

%ebp	0xffff0
%esp	0xffe0
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

main:

```
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    > movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl   -8(%ebp)
    pushl   -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

add:

```
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec		
0xffe8	14	
0xffe4	7	
0xffe0		← %esp
0xffdc		
0xffd8		
0xffd4		
0xffd0		

%ebp	0xffff0
%esp	0xffe0
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    > movl   $0, -4(%ebp)
    pushl   -8(%ebp)
    pushl   -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		← %esp
0xffdc		
0xffd8		
0xffd4		
0xffd0		

%ebp	0xffff0
%esp	0xffe0
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

main:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl $7, -12(%ebp)
movl $14, -8(%ebp)
movl $0, -4(%ebp)
> pushl -8(%ebp)
pushl -12(%ebp)
call add
addl $8, %esp
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

add:

```
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %edx
movl 12(%ebp), %eax
addl %edx, %eax
popl %ebp
ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		← %esp
0xffdc	14	← %esp
0xffd8		
0xffd4		
0xffd0		

%ebp	0xffff0
%esp	0xffe00xffdc
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    > pushl -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl  %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		
0xffdc	14	← %esp
0xffd8	7	← %esp
0xffd4		
0xffd0		

%ebp	0xffff0
%esp	0xffdc0xffd8
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    > call  add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl  %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		
0xffdc	14	
0xffd8	7	← %esp
0xffd4	<main+11>	← %esp
0xffd0		

%ebp	0xffff0
%esp	0xffd80xffd4
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
> pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	←
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		
0xffdc	14	
0xffd8	7	
0xffd4	<main+11>	←
0xffd0	0xffff0 ← %esp	

%ebp	0xffff0
%esp	0xffd40xffd0
%eax	??
%edx	??



# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl  %ebp
    > movl  %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		
0xffdc	14	
0xffd8	7	
0xffd4	<main+11>	
0xffd0	0xffff0	← %esp,%ebp

%ebp	0xffff00ffd0
%esp	0xffd0
%eax	??
%edx	??

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    > movl  8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>
0xffec	0
0xffe8	14
0xffe4	7
0xffe0	
0xffdc	14
0xffd8	7
0xffd4	<main+11>
0xffd0	0xffff0 ← %esp,%ebp

%ebp	0xffd0
%esp	0xffd0
%eax	??
%edx	???

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl  %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    > movl  12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>
0xffec	0
0xffe8	14
0xffe4	7
0xffe0	
0xffdc	14
0xffd8	7
0xffd4	<main+11>
0xffd0	0xffff0 ← %esp,%ebp

%ebp	0xffd0
%esp	0xffd0
%eax	??14
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    > addl  %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>
0xffec	0
0xffe8	14
0xffe4	7
0xffe0	
0xffdc	14
0xffd8	7
0xffd4	<main+11>
0xffd0	0xffff0 ← %esp,%ebp

%ebp	0xffd0
%esp	0xffd0
%eax	1421
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    > popl  %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		
0xffdc	14	
0xffd8	7	
0xffd4	<main+11>	← %esp
0xffd0	0xffff0	← %esp,%ebp

%ebp	0xffd00xffff0
%esp	0xffd00xffd4
%eax	21
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
> ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		
0xffdc	14	
0xffd8	7	← %esp
0xffd4	<main+11>	← %esp
0xffd0	0xffff0	

%ebp	0xffff0
%esp	0xffd40xffd8
%eax	21
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    > addl  $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl  %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp
0xffec	0	
0xffe8	14	
0xffe4	7	
0xffe0		← %esp
0xffdc	14	
0xffd8	7	← %esp
0xffd4	<main+11>	
0xffd0	0xffff0	

%ebp	0xffff0
%esp	0xffd80xffe0
%eax	21
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    > movl  -4(%ebp), %eax
    movl  -4(%ebp), %eax
    leave
    ret

add:
    pushl  %ebp
    movl  %esp, %ebp
    movl  8(%ebp), %edx
    movl  12(%ebp), %eax
    addl  %edx, %eax
    popl  %ebp
    ret
```

0xffff0	<ebp> ← %ebp
0xffec	021
0xffe8	14
0xffe4	7
0xffe0	← %esp
0xffdc	14
0xffd8	7
0xffd4	<main+11>
0xffd0	0xffff0

%ebp	0xffff0
%esp	0xffe0
%eax	21
%edx	7



# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
> movl   -4(%ebp), %eax
    leave
    ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp> ← %ebp
0xffec	21
0xffe8	14
0xffe4	7
0xffe0	← %esp
0xffdc	14
0xffd8	7
0xffd4	<main+11>
0xffd0	0xffff0

%ebp	0xffff0
%esp	0xffe0
%eax	21
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    > leave
    ret

add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp,%esp
0xffec	21	
0xffe8	14	
0xffe4	7	
0xffe0		← %esp
0xffdc	14	
0xffd8	7	
0xffd4	<main+11>	
0xffd0	0xffff0	

%ebp	0xffff0
%esp	0xffe00xffff0
%eax	21
%edx	7

# Assembler Program (gcc -S -m32)

```
main:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   $7, -12(%ebp)
    movl   $14, -8(%ebp)
    movl   $0, -4(%ebp)
    pushl  -8(%ebp)
    pushl  -12(%ebp)
    call   add
    addl   $8, %esp
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave
> ret
```

```
add:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %edx
    movl   12(%ebp), %eax
    addl   %edx, %eax
    popl   %ebp
    ret
```

0xffff0	<ebp>	← %ebp,%esp
0xffec	21	
0xffe8	14	
0xffe4	7	
0xffe0		← %esp
0xffdc	14	
0xffd8	7	
0xffd4	<main+11>	
0xffd0	0xffff0	

%ebp	0xffff0
%esp	0xffff0
%eax	21
%edx	7

**System V AMD64 ABI (x86\_64)**

# Assembler Program (gcc -S -m64)

main:

```
> pushq %rbp
movq %rsp, %rbp
subq $32, %rsp
movq $7, -24(%rbp)
movq $14, -16(%rbp)
movq $0, -8(%rbp)
movq -16(%rbp), %rdx
movq -24(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call add
movq %rax, -8(%rbp)
movq -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
addq %rdx, %rax
popq %rbp
ret
```

0xffff0	<rbp>	← %rsp
0xffe8		
0xffe0		
0xffd8		
0xffd0		
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	<rbp>
%rsp	0xffff0
%rax	??
%rdx	??
%rdi	??
%rsi	??

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
```

```
> movq %rsp, %rbp
```

```
subq $32, %rsp
```

```
movq $7, -24(%rbp)
```

```
movq $14, -16(%rbp)
```

```
movq $0, -8(%rbp)
```

```
movq -16(%rbp), %rdx
```

```
movq -24(%rbp), %rax
```

```
movq %rdx, %rsi
```

```
movq %rax, %rdi
```

```
call add
```

```
movq %rax, -8(%rbp)
```

```
movq -8(%rbp), %rax
```

```
leave
```

```
ret
```

add:

```
pushq %rbp
```

```
movq %rsp, %rbp
```

```
movq %rdi, -8(%rbp)
```

```
movq %rsi, -16(%rbp)
```

```
movq -8(%rbp), %rdx
```

```
movq -16(%rbp), %rax
```

```
addq %rdx, %rax
```

```
popq %rbp
```

```
ret
```

0xffff0	<rbp>	← %rsp, %rbp
0xffe8		
0xffe0		
0xffd8		
0xffd0		
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	<rbp>	0xffff0
%rsp		0xffff0
%rax		??
%rdx		??
%rdi		??
%rsi		??

# Assembler Program (gcc -S -m64)

main:

```
    pushq %rbp
    movq  %rsp, %rbp
>   subq  $32, %rsp
    movq  $7, -24(%rbp)
    movq  $14, -16(%rbp)
    movq  $0, -8(%rbp)
    movq  -16(%rbp), %rdx
    movq  -24(%rbp), %rax
    movq  %rdx, %rsi
    movq  %rax, %rdi
    call  add
    movq  %rax, -8(%rbp)
    movq  -8(%rbp), %rax
    leave
    ret
```

add:

```
    pushq %rbp
    movq  %rsp, %rbp
    movq  %rdi, -8(%rbp)
    movq  %rsi, -16(%rbp)
    movq  -8(%rbp), %rdx
    movq  -16(%rbp), %rax
    addq  %rdx, %rax
    popq  %rbp
    ret
```

0xffff0	<rbp>	← %rsp, %rbp
0xffe8		
0xffe0		
0xffd8		
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp		0xffff0
%rsp	0xffff0	0xffd0
%rax		??
%rdx		??
%rdi		??
%rsi		??

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
> movq $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8		
0xffe0		
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??



# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
> movq $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8		
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq %rsp, %rbp
subq $32, %rsp
movq $7, -24(%rbp)
movq $14, -16(%rbp)
> movq $0, -8(%rbp)
movq -16(%rbp), %rdx
movq -24(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call add
movq %rax, -8(%rbp)
movq -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
addq %rdx, %rax
popq %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	??
%rdx	??
%rdi	??
%rsi	??

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
> movq -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	??
%rdx	??14
%rdi	??
%rsi	??

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
> movq -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	???
%rdx	14
%rdi	??
%rsi	??

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
> movq %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	??
%rsi	??14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
> movq %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8		
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd0
%rax	7
%rdx	14
%rdi	???
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
> call add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	← %rsp
0xffc0		
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffd00xffc8
%rax	7
%rdx	14
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
> pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0	0xffff0	← %rsp
0xffb8		
0xffb0		

%rbp	0xffff0
%rsp	0xffc80xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14



# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
> movq %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xffff0	← %rsp,%rbp
0xffb8		
0xffb0		

%rbp	0xffff00xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
> movq %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
> movq %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	14
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
> movq -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	7
%rdx	147
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
> movq -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	714
%rdx	7
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
> addq %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>
0xffe8	0
0xffe0	14
0xffd8	7
0xffd0	
0xffc8	<main+13>
0xffc0	0xffff0 ← %rsp,%rbp
0xffb8	7
0xffb0	14

%rbp	0xffc0
%rsp	0xffc0
%rax	1421
%rdx	7
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
> popq %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	← %rsp
0xffc0	0xffff0	← %rsp,%rbp
0xffb8	7	
0xffb0	14	

%rbp	0xffc00xffff0
%rsp	0xffc00xffc8
%rax	21
%rdx	7
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq %rsp, %rbp
subq $32, %rsp
movq $7, -24(%rbp)
movq $14, -16(%rbp)
movq $0, -8(%rbp)
movq -16(%rbp), %rdx
movq -24(%rbp), %rax
movq %rdx, %rsi
movq %rax, %rdi
call add
movq %rax, -8(%rbp)
movq -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq -8(%rbp), %rdx
movq -16(%rbp), %rax
addq %rdx, %rax
popq %rbp
> ret
```

0xffff0	<rbp>	← %rbp
0xffe8	0	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	← %rsp
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%rsp	0xffc80xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14



# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
> movq %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	021	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
> movq -8(%rbp), %rax
leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp
0xffe8	21	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%rsp	0xffd0
%rax	21
%rdx	7
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
> leave
ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp,%rsp
0xffe8	21	
0xffe0	14	
0xffd8	7	
0xffd0		← %rsp
0xffc8	<main+13>	
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%rsp	0xffd00xffff0
%rax	21
%rdx	7
%rdi	7
%rsi	14

# Assembler Program (gcc -S -m64)

main:

```
pushq %rbp
movq  %rsp, %rbp
subq  $32, %rsp
movq  $7, -24(%rbp)
movq  $14, -16(%rbp)
movq  $0, -8(%rbp)
movq  -16(%rbp), %rdx
movq  -24(%rbp), %rax
movq  %rdx, %rsi
movq  %rax, %rdi
call  add
movq  %rax, -8(%rbp)
movq  -8(%rbp), %rax
leave
```

```
> ret
```

add:

```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  -8(%rbp), %rdx
movq  -16(%rbp), %rax
addq  %rdx, %rax
popq  %rbp
ret
```

0xffff0	<rbp>	← %rbp,%rsp
0xffe8	21	
0xffe0	14	
0xffd8	7	
0xffd0		
0xffc8	<main+13>	
0xffc0	0xffff0	
0xffb8	7	
0xffb0	14	

%rbp	0xffff0
%rsp	0xffff0
%rax	21
%rdx	7
%rdi	7
%rsi	14

cdecl / System V AMD64 ABI: both allows variable-length argument lists

- cdecl: push all arguments in opposite order on the stack
- AMD64 ABI: fill register first, then push all remaining arguments in opposite order on the stack

How does the function know:

- how many parameters?
- which data types?



```
printf("%c %u %f %s\n", 'a', 123, 3.14, "hello");
```

libc parses format string and pops from stack:

1. a char
2. an unsigned int
3. a float
4. a char\*

What if format string and data on stack do not fit together?

maybe:

- local variables
- function arguments

always:

- return addresses
  - many nested calls → many return addresses

# C++ Classes

---



- Basically structs in memory
- struct contains all member variables
  - including any member objects or parent objects
- By default, pointers for methods (function pointers) are not stored in the object
  - Only used implicitly by the compiler
  - Important exception: virtual methods → vtables (next slide)

- Fundamental feature enabling “polymorphism”
- Table with function pointers
- First member of every object
- Specialized (derived) class simply overwrites previous function pointer

# VTable example

```
class Animal { public:  
> Animal() { p("Animal ctor"); }  
    virtual void makesound() = 0;  
> virtual void move() { p("moving"); }  
> virtual ~Animal() { p("Animal dtor"); }  
};  
class Bird : public Animal { public:  
> Bird() { p("Bird ctor"); }  
> virtual void makesound() { p("cheep"); }  
> virtual ~Bird() { p("Bird dtor"); }  
};  
class Cat : public Animal { public:  
> Cat() { p("Cat ctor"); }  
> virtual void makesound() { p("meow"); }
```

```
int main(void) {  
> Animal* b = new Bird();  
> Animal* c = new Cat();  
> b->makesound();  
> c->makesound();  
> b->move();  
> c->move();  
> delete b;  
> delete c;  
> return 0;  
}
```

vtable for b

[0]	0x0 <Animal::makesound()>0x555555554cbe
-----	---



- Powerful concept for inter-process communication
- Some are similar to exceptions
- Sometimes used for exploits

```
int main()
{
    if (i != -1ULL/4096) {
        printState();
        laststate = currentstate;
        ++i;
        currentstate = VALID;
        *(volatile size_t*)(i*4096);
    }
    return 0;
}
```

```
static void unblock_signal(int signum
    __attribute__((__unused__))) {
    sigset_t sigs;
    sigemptyset(&sigs);
    sigaddset(&sigs, signum);
    sigprocmask(SIG_UNBLOCK, &sigs, NULL);
}
void segfault_handler(int signum) {
    currentstate = INVALID;
    unblock_signal(SIGSEGV);
}
```

```
int main() {
    signal(SIGSEGV, segfault_handler);

    if (i != -1ULL/4096) {
        printState();
        laststate = currentstate;
        ++i;
        currentstate = VALID;
        *(volatile size_t*)(i*4096);
    }
    return 0;
}
```

```
static void unblock_signal(int signum
    __attribute__((__unused__))) {
> sigset_t sigs;
> sigemptyset(&sigs);
> sigaddset(&sigs, signum);
> sigprocmask(SIG_UNBLOCK, &sigs, NULL);
}
void segfault_handler(int signum) {
> currentstate = INVALID;
> unblock_signal(SIGSEGV);
> longjmp(buf, 0);
}
```

```
int main() {
> signal(SIGSEGV, segfault_handler);
> setjmp(buf);
> if (i != -1ULL/4096) {
>     printState();
>     laststate = currentstate;
>     ++i;
>     currentstate = VALID;
>     *(volatile size_t*)(i*4096);
}
return 0;
}
```

**Live Demo**





- CTF platform
- Material page
- Register an account (**student mail address** recommended)
- Hacklets are worth **20%** of your exam
- During the exam **tick** that you solved a certain hacklet

