Assembly x86-64

Sicurezza (CT0539) 2024-25
Università Ca' Foscari Venezia

Riccardo Focardi
www.unive.it/data/persone/5590470
secgroup.dais.unive.it

Università
Ca'Foscari
Venezia

Program exploitation

Making a program do something **unexpected** and not planned

The *right* ***bugs*** can be used to **subvert** code execution

⇒ It is essential to understand how programs are **compiled** and **executed**

Executables are written in **machine code**
⇒ depends on computer architecture

Assembly language makes machine code more readable
⇒ depends on computer architecture

We focus on x86-64 assembly (also known as **x64**, **AMD64** and **Intel 64**)
⇒ one of the most popular

Registers

General purpose: **`rax`**, **`rbx`**, **`rcx`**, **`rdx`**, **`r8`**, …, **`r15`** : used to temporarily store values and addresses used in the computation

Stack: **`rsp`**, **`rbp`**, corresponding to **stack pointer** and **base pointer**, i.e., the two addresses delimiting the current stack

Instruction pointer: **`rip`**; points to the instruction to be executed

Indexes: **`rsi`**, **`rdi`**, source index and destination index: used for array, string copy and parameters

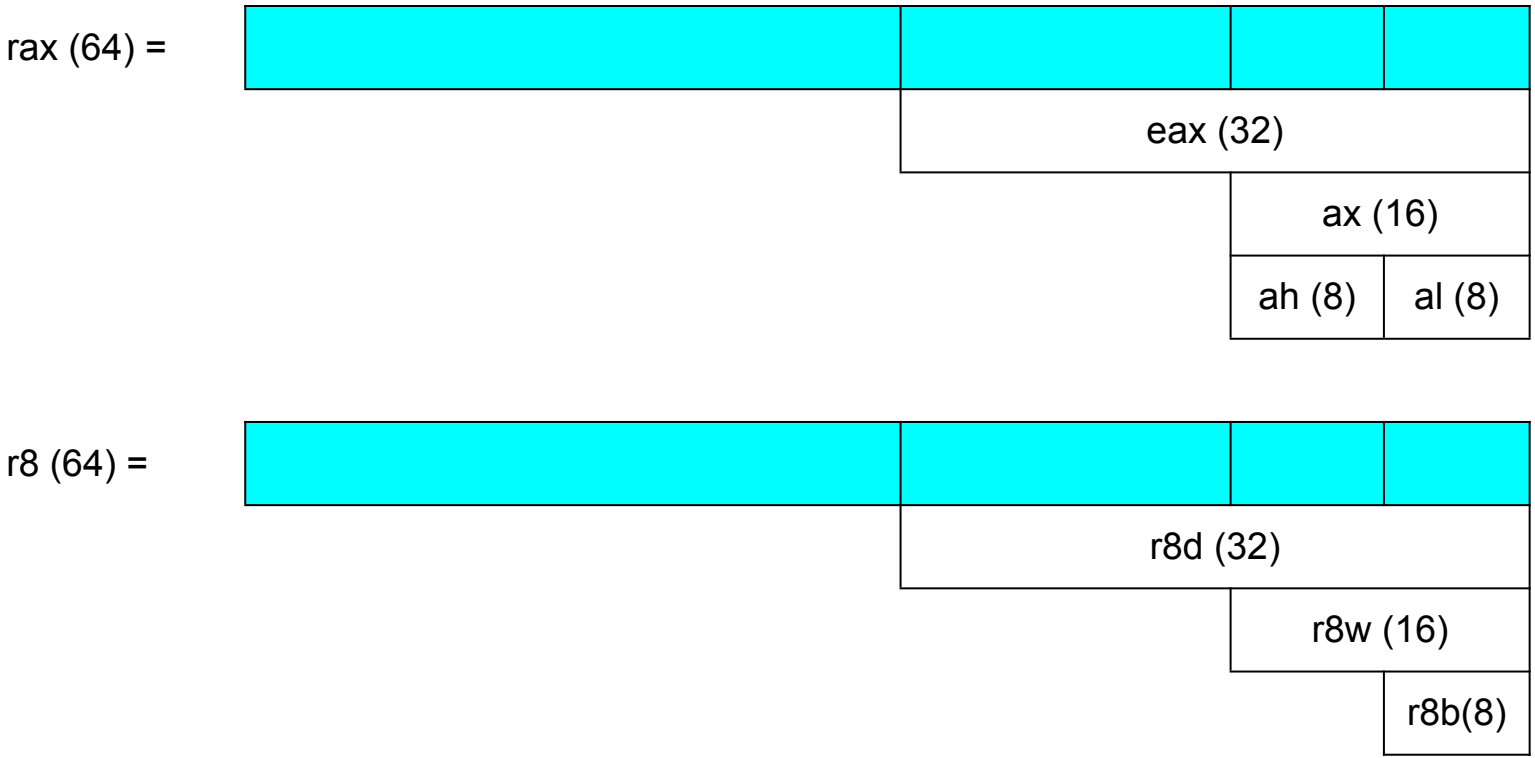
Single Instruction Multiple Data (SIMD): **`xmm0`**, …, **`xmm15`**, 128 bits (up to 512 in AVX-512, **`zmm0`**-**`zmm31`**)

Flag register: **`rFlag`**, for status, es:
ZF - *zero* flag, when result is zero
CF - *carry* flag, result too large/small
SF - *sign* flag, when result is negative

Portions of registers

64 bits	lowest 32 bits	lowest 16 bits	lowest 8 bits	2nd lowest 8 bits
rax	eax	ax	al	ah
rbx	ebx	bx	bl	bh
...	...	...	...	
r8	r8d	r8w	r8b	-
r9	r9d	r9w	r9b	-
...	...	...	...	...

Registers portions



Assembly syntax

AT&T Syntax:

command <source>, <destination>

Example:

```
mov $5, rax
```

Intel Syntax:

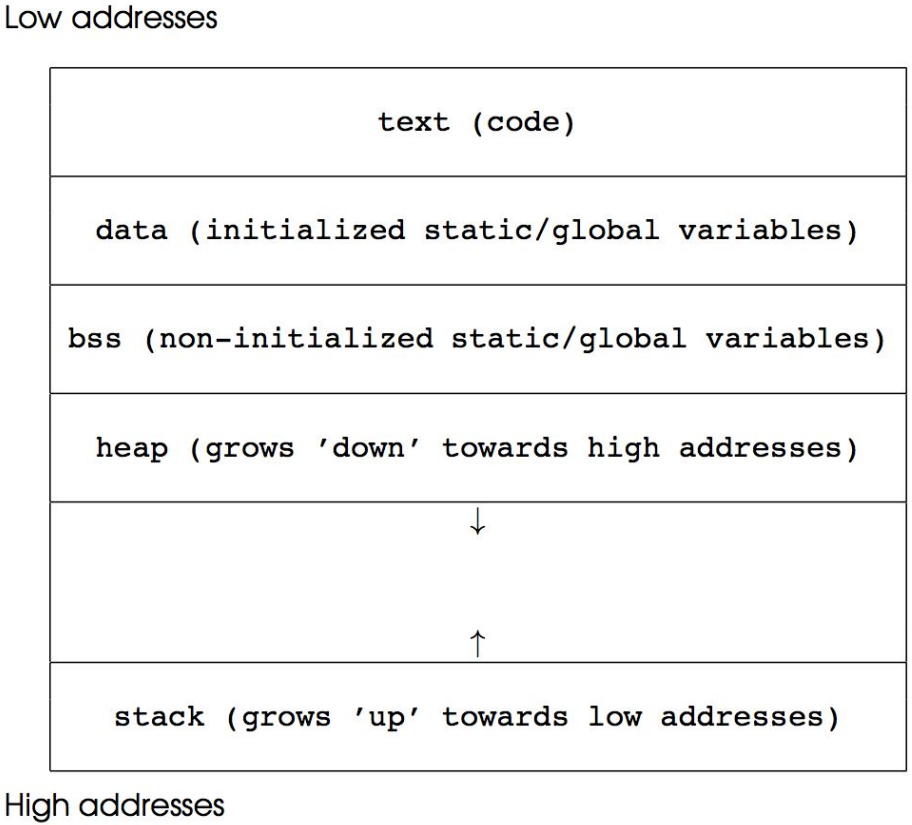
command <destination>, <source>

Example:

```
mov rax, 5
```

⇒ We will use the Intel syntax
- More documented
- More explicit (e.g. size)

Memory layout



Stack and calling convention

Region of memory where **local variables** are stored

Supports **push** and **pop** operations

Grows towards **lower** memory addresses
⇒ pushed values have lower address

When a function is called a ***stack frame*** is set up

rbp contains the address of the base of the current stack frame

rsp contains the address of the top element of the current stack frame

System V AMD64 ABI: Every function call stores first 6 arguments in **`rdi`**, **`rsi`**, **`rdx`**, **`rcx`**, **`r8`**, **`r9`**, pushes extra arguments on the stack, return value up to 128 bits in **`rax`** and **`rdx`**

Main commands

Commands

`mov <dst>, <src>`: **moves** the `<src>` value to `<dst>`

`add <dst>, <src>`: **adds** the value in `<src>` to `<dst>`

`sub <dst>, <src>`: **subtracts** the value in `<src>` from `<dst>`

`and <dst>, <src>`: performs a **logical and** between `<src>` and `<dst>`, placing the result in `<dst>`

`push <target>`: **pushes** the value in `<target>` to the stack

`pop <target>`: **pops** a value from the stack into `<target>`

`cmp <dst>, <src>`: **compares** `<src>` with `<dst>`. This is done by subtracting `<src>` from `<dst>` and updating **flags** that can be checked by subsequent conditional operations

Commands (2) - control flow

`call <address>`: **calls** the function at `<address>`. Before jumping to the function, the address of the next instruction is **pushed** to the stack in order to be able to return

`ret`: pops the return address and **returns** control to it

`leave`: **restores** the stack frame (rsp←rbp and old rbp is popped)

`jle <target>`: **jumps** to the address in `<target>` if the previously compared `<src>` was **less than or equal** to <dst>. The test is done on the flags set by cmp

`jge <target>`: **jumps** to the address in `<target>` if the previously compared <src> was **greater than or equal** to <dst>. The test is done on the flags set by cmp

Commands (3)

`jmp <target>`: **jumps** to the address in `<target>`. Copies target address into the **`rip`** register

`lea <dst>, <src>` stands for “load effective address”: **loads the address** of <src> into <dst>;

`int <value>`: generates software **interrupt** `<value>`. This is commonly used to invoke system calls

`nop`: no-operation, does **nothing**

NOTE: There are various *addressing modes*. For example:

Register indirect:

```
mov DWORD PTR [rbp - 4], eax
```

Immediate: `mov eax, 3`

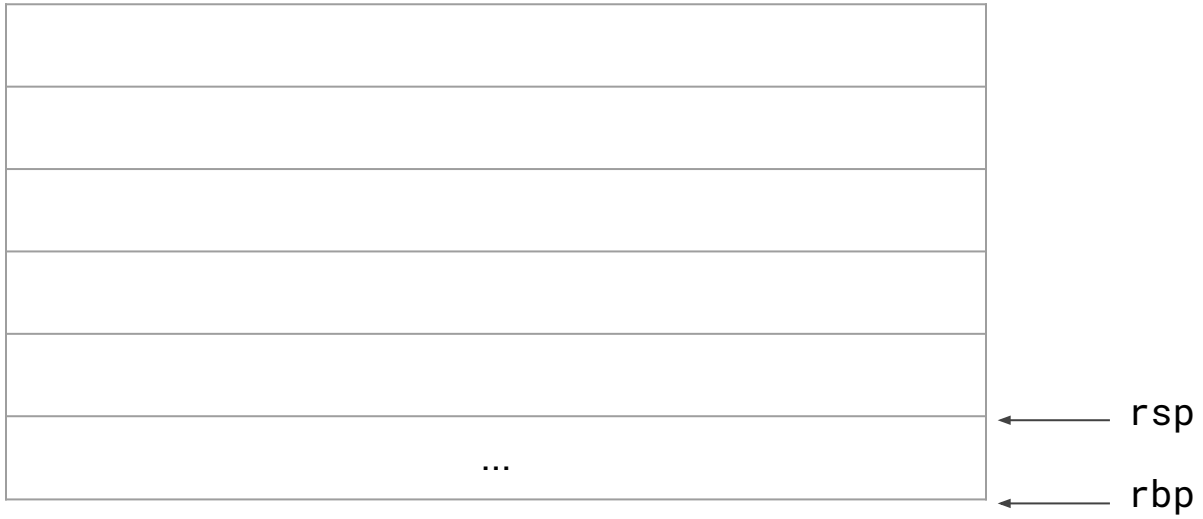
Note: DWORD is like a type: indicates a 32-bit “double word” value

Function calls

Example: function call

```
int main() {
    func(10);
    ...
}
```

```
mov    edi,0xa # moves 10 to rdi
call   <func>  # calls func
...
```

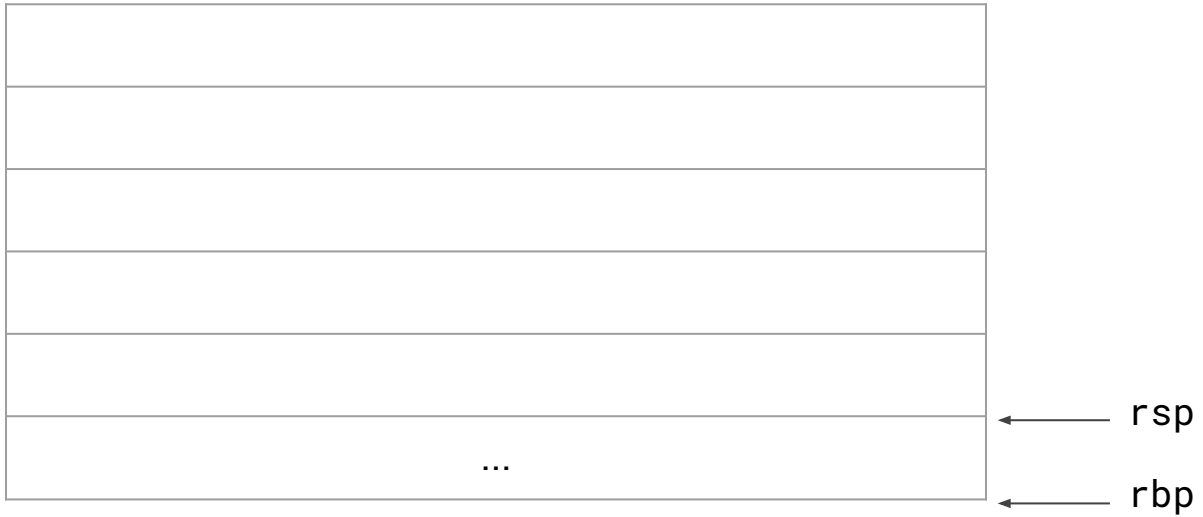


Example: function call

```
int main() {
    func(10);
    ...
}
```

param. in `rdi`
(32 bits zeroed)

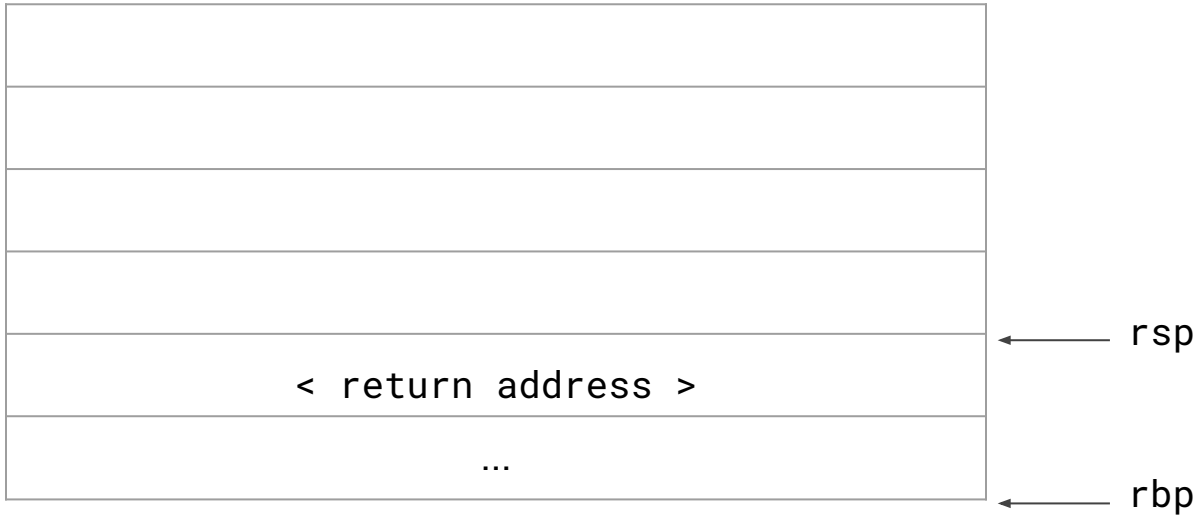
```
mov    edi,0xa # moves 10 to rdi
call   <func>  # calls func
...
```



Example: function call

```
int main() {
    func(10);
    ...
}
```

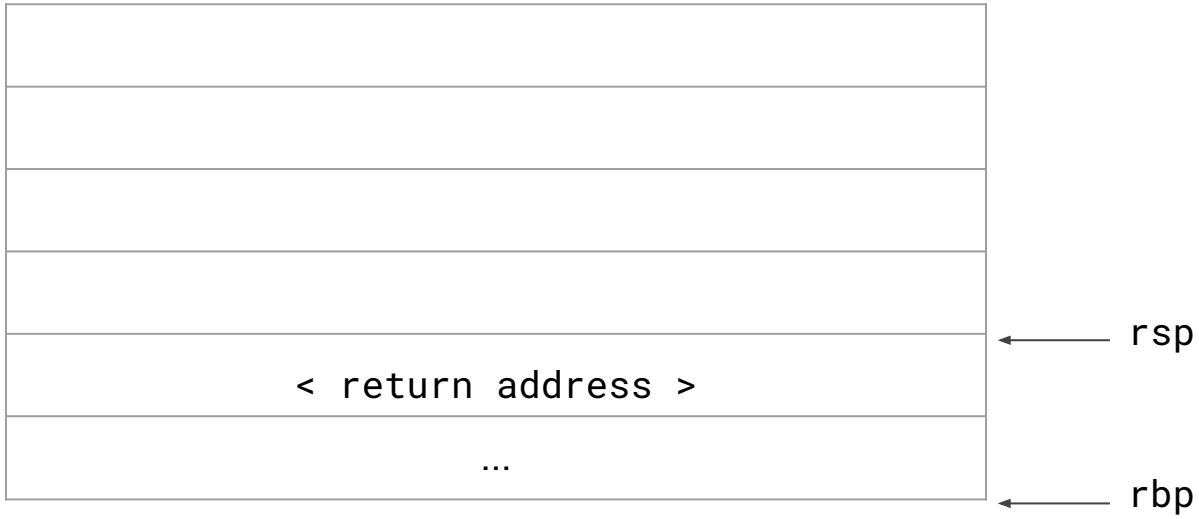
```
mov    edi,0xa # moves 10 to rdi
call   <func>  # calls func
...
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

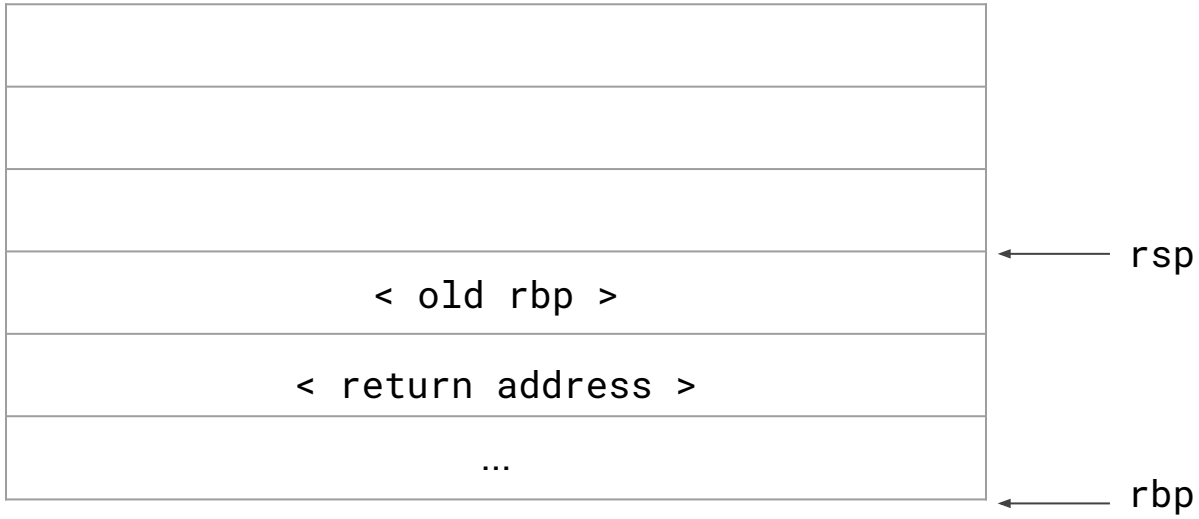
```
push    rbp
mov     rbp,rsp
sub     rsp,0x18
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

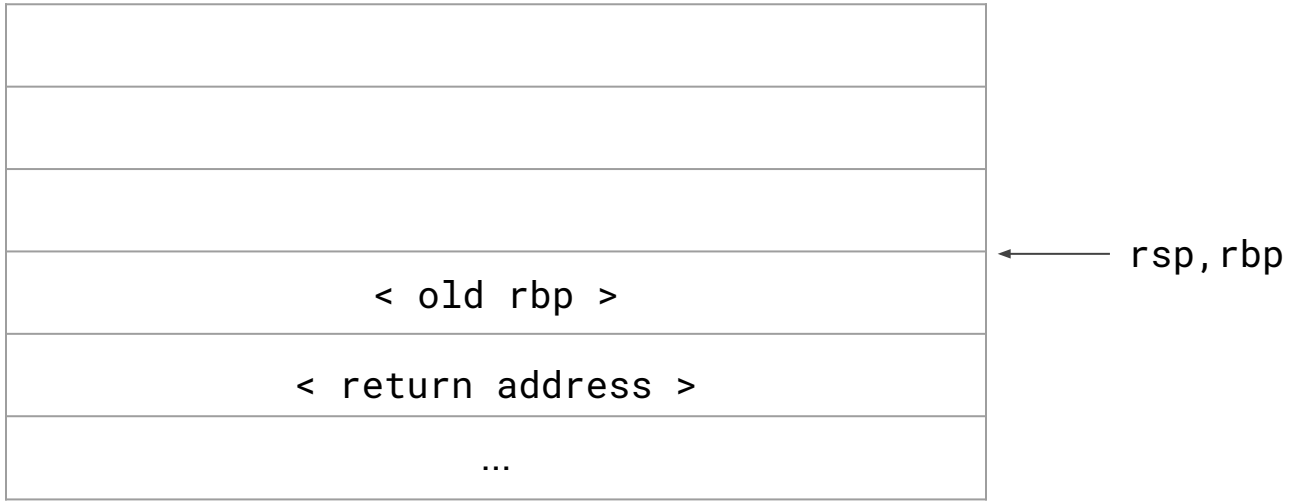
```
push    rbp
mov     rbp,rsp
sub     rsp,0x18
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

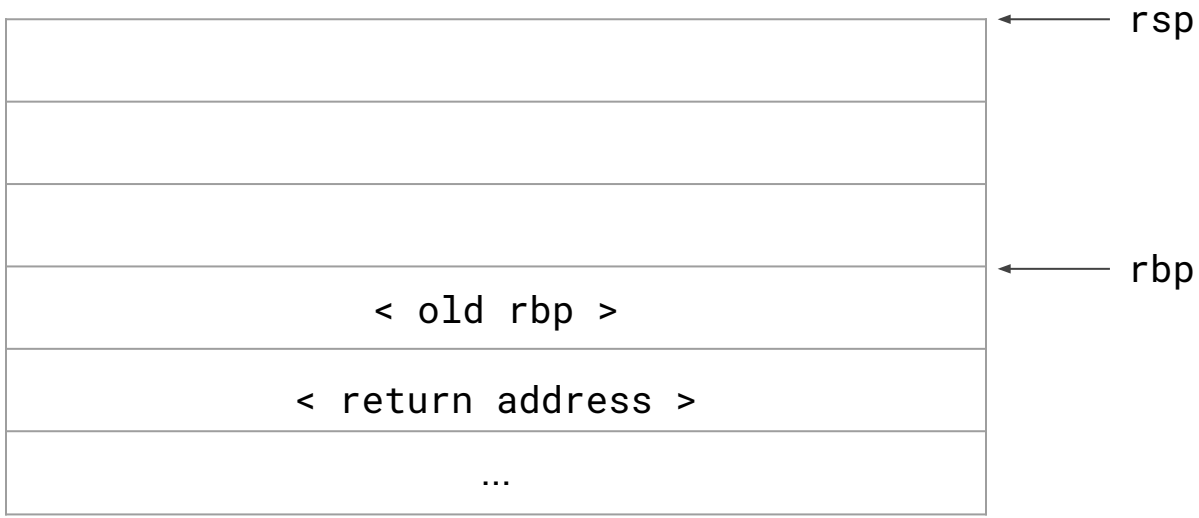
```
push    rbp
mov     rbp,rsp
sub     rsp,0x18
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

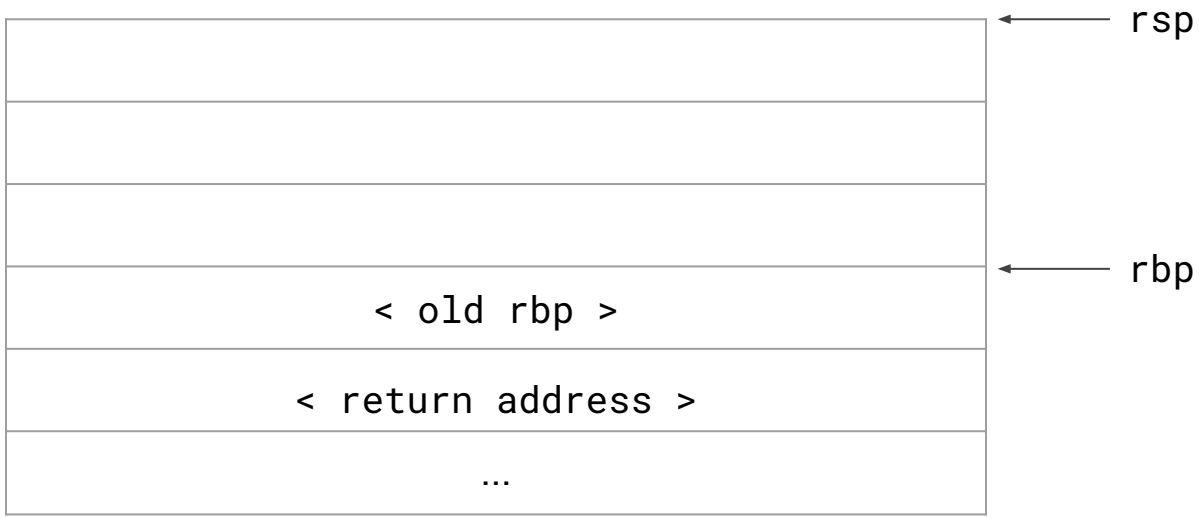
```
push    rbp
mov     rbp,rsp
sub     rsp,0x18
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

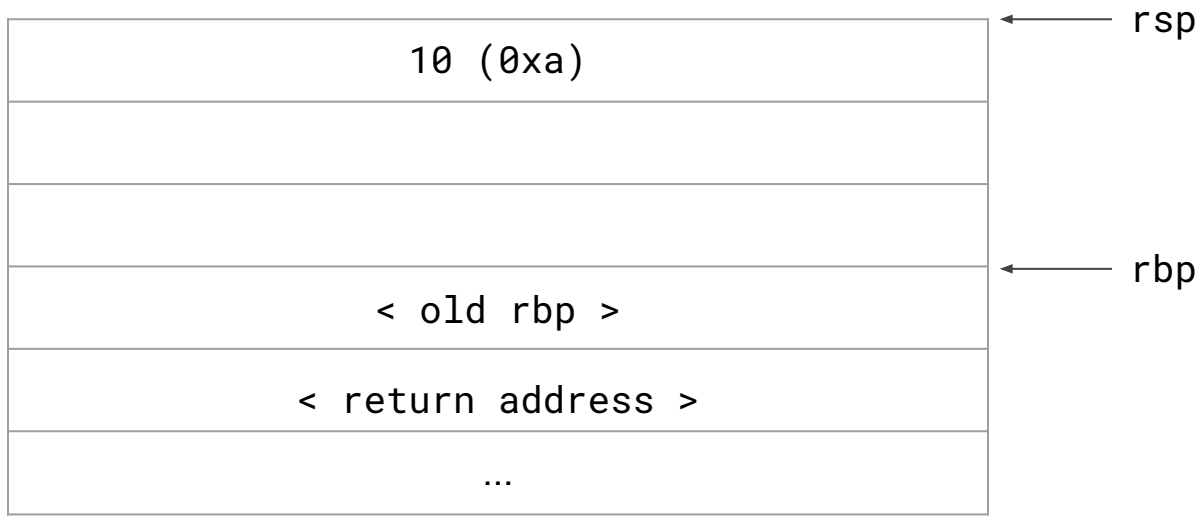
```
mov     QWORD PTR [rbp-0x18],rdi
mov     QWORD PTR [rbp-0x10],0x0
mov     rax,QWORD PTR [rbp-0x18]
mov     QWORD PTR [rbp-0x8],rax
```



Example: function call

```
long func(long x) {
   long a = 0;
   long b = x;
   ...
   return b;
}
```

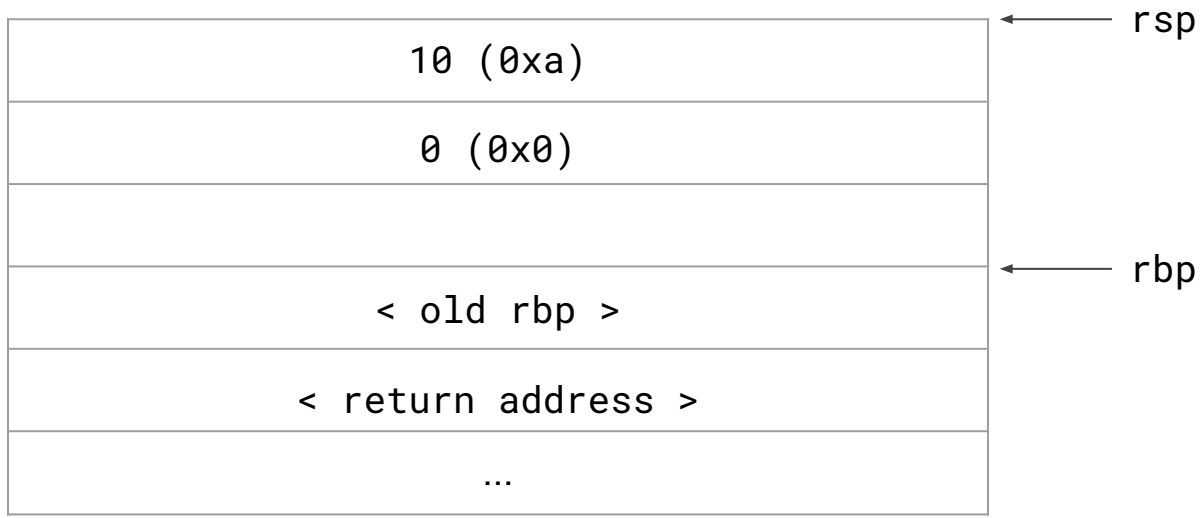
```
mov     QWORD PTR [rbp-0x18],rdi
mov     QWORD PTR [rbp-0x10],0x0
mov     rax,QWORD PTR [rbp-0x18]
mov     QWORD PTR [rbp-0x8],rax
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

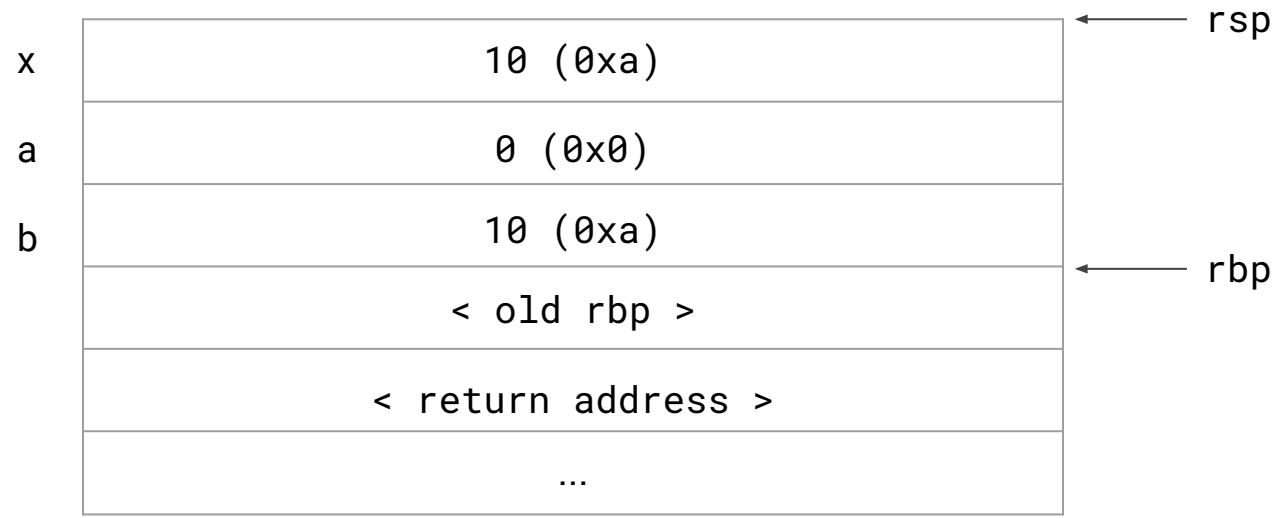
```
mov     QWORD PTR [rbp-0x18],rdi
mov     QWORD PTR [rbp-0x10],0x0
mov     rax,QWORD PTR [rbp-0x18]
mov     QWORD PTR [rbp-0x8],rax
```



Example: function call

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

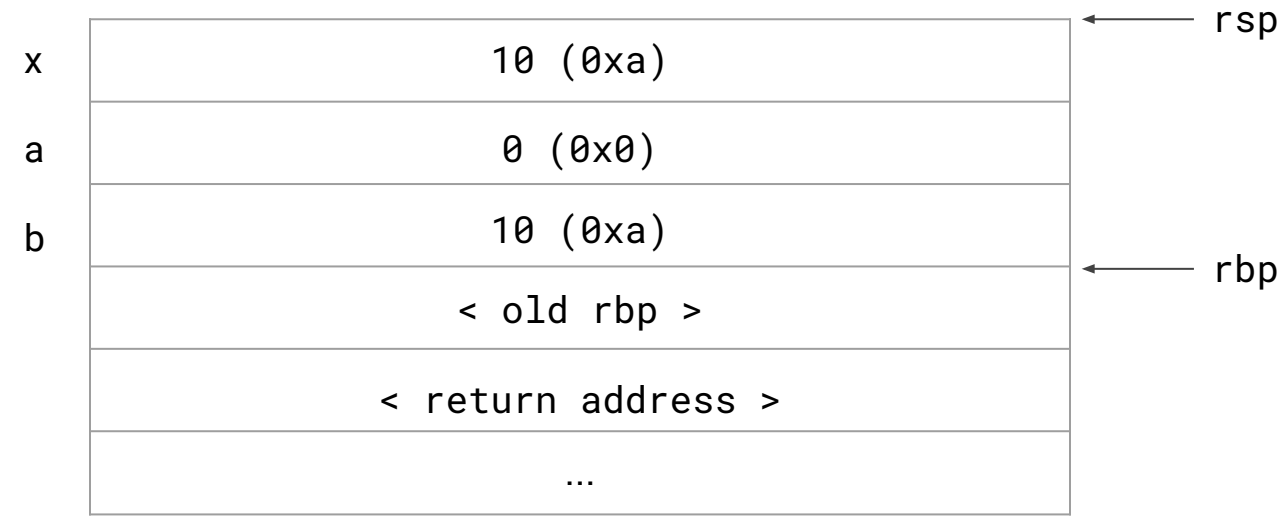
```
mov     QWORD PTR [rbp-0x18],rdi
mov     QWORD PTR [rbp-0x10],0x0
mov     rax,QWORD PTR [rbp-0x18]
mov     QWORD PTR [rbp-0x8],rax
```



Example: function return

```
long func(long x) {
   long a = 0;
   long b = x;
   ...
   return b;
}
```

```
mov     rax,DWORD PTR [rbp-0x8]
leave
ret
```

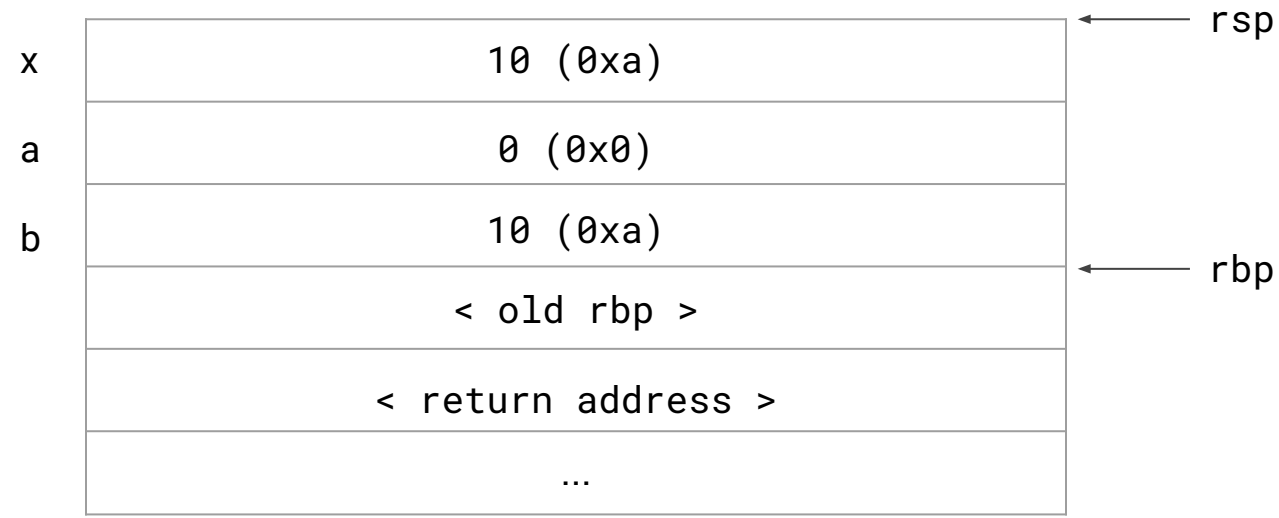


Example: function return

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

return value (b) ⟹

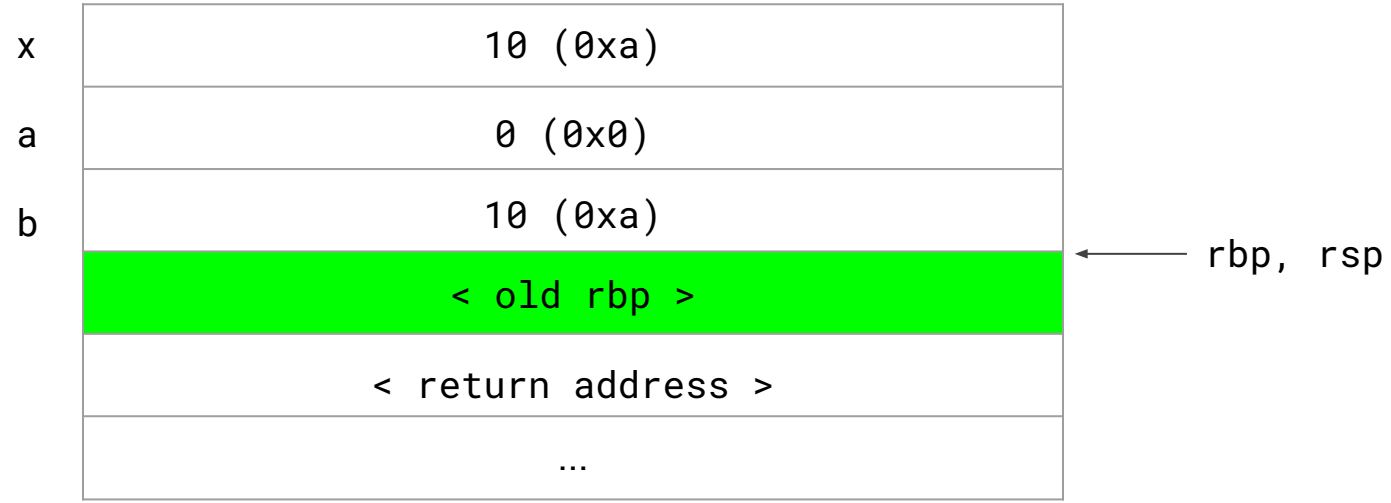
```
mov     rax,DWORD PTR [rbp-0x8]
leave
ret
```



Example: function return

```
long func(long x) {
    long a = 0;
    long b = x;
    ...
    return b;
}
```

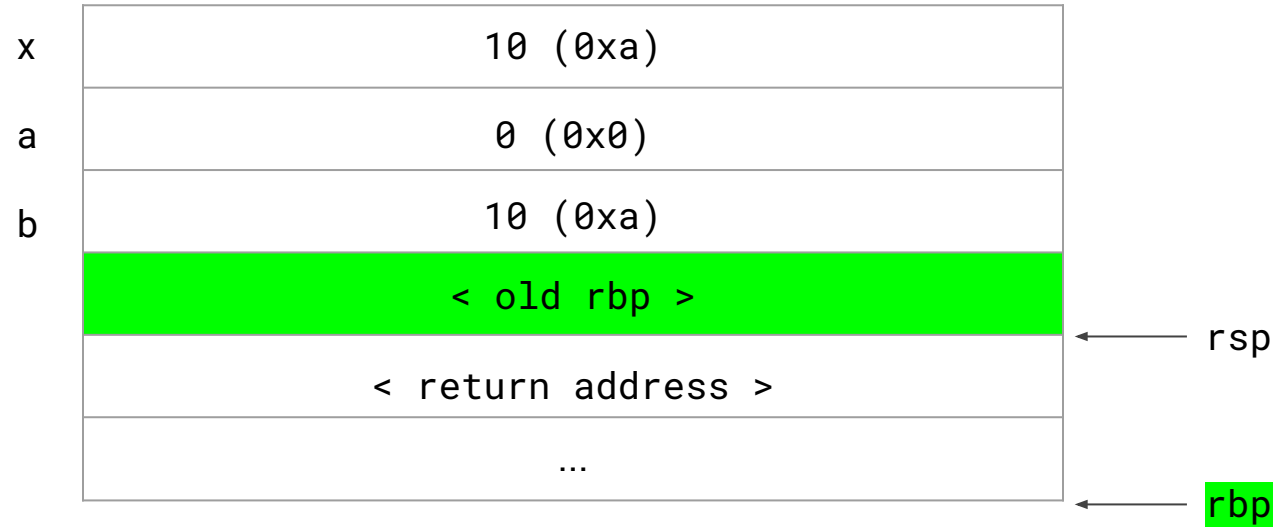
```
mov     rax,DWORD PTR [rbp-0x8]
leave
ret
```



Example: function return

```
long func(long x) {
   long a = 0;
   long b = x;
   ...
   return b;
}
```

```
mov     rax,DWORD PTR [rbp-0x8]
leave
ret
```

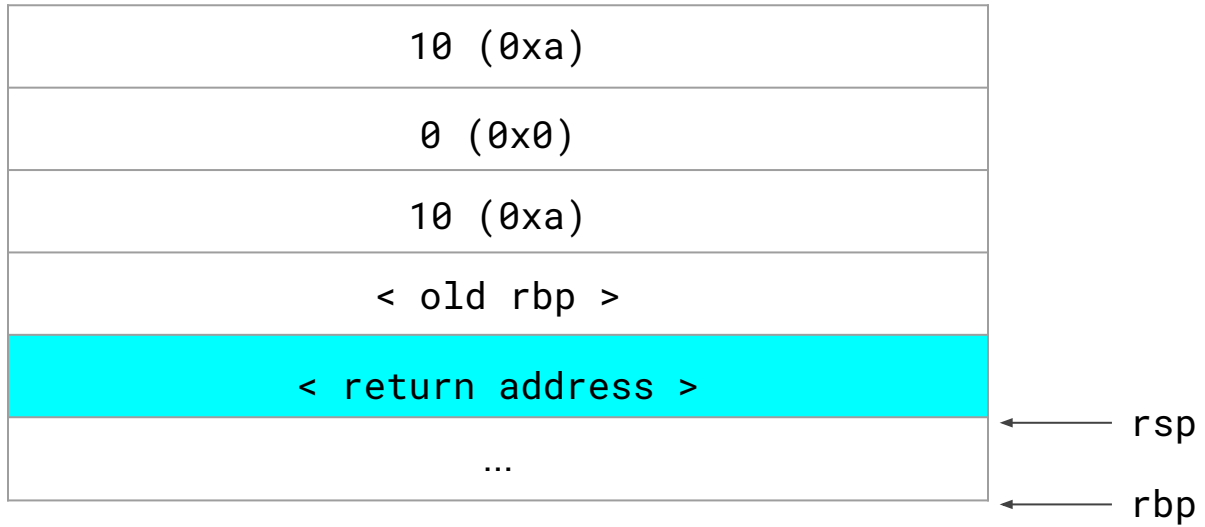


Example: function return

```
long func(long x) {
   long a = 0;
   long b = x;
   ...
   return b;
}
```

```
mov     rax,DWORD PTR [rbp-0x8]
leave
ret
```

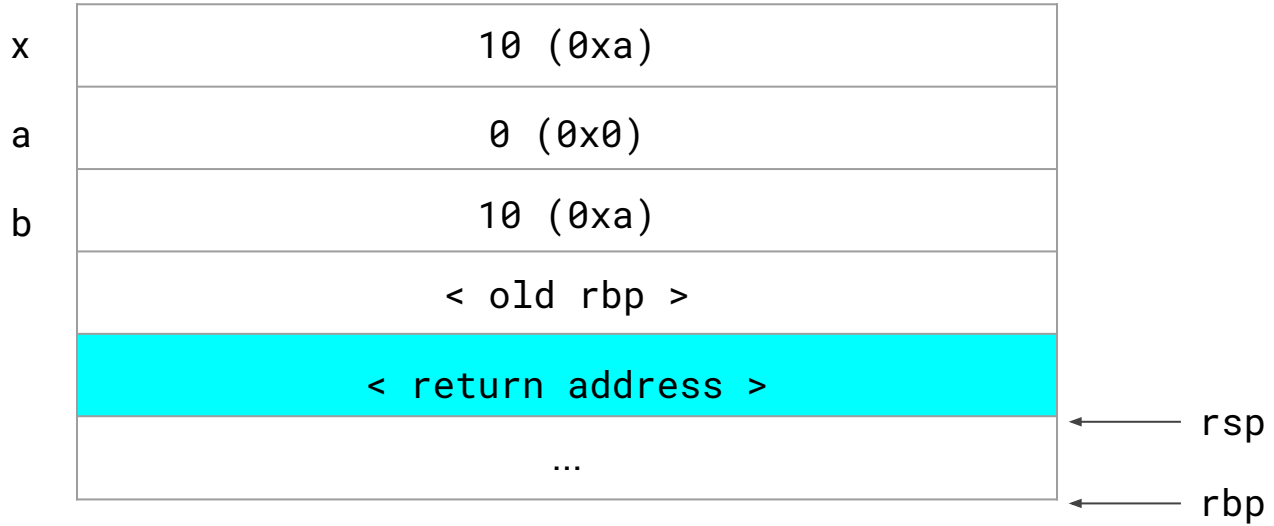
rip is set to ret. address



Example: function return

```
int main() {
    func(10);
    ...
}
```

```
mov    edi,0xa # moves 10 to rdi
call   <func>  # calls func
...
```



Reading machine code

Disassembling with objdump

Simple example program count.c:

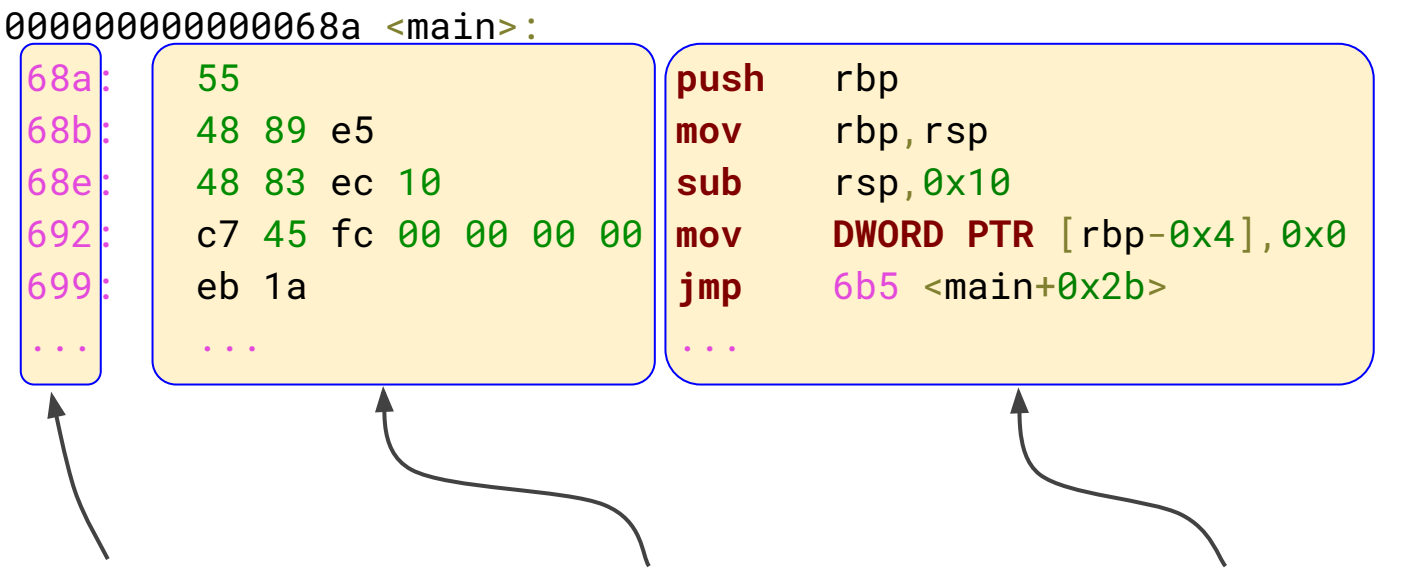
```c
#include <stdio.h>

int main()
{
    int i;
    for (i=0; i<10; i++)
        printf("%d ",i);
    printf("\n");
}
```

```
$ gcc count.c -o count
$ ./count
0 1 2 3 4 5 6 7 8 9
```

With objdump we can produce the assembly code (-d) and display sections (-s) in Intel syntax (-M intel)

```
$ objdump -M intel -ds count > count.s
```

Assembly of count.c



Addresses (can be relocated)

The actual **machine code** (as bytes). Commands have **different lengths**!

Assembly x86-64 Intel syntax

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    DWORD PTR [rbp-0x4],0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
 69e: 89 c6                 mov    esi,eax
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 754 ...
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    DWORD PTR [rbp-0x4],0x1
 6b5: 83 7d fc 09           cmp    DWORD PTR [rbp-0x4],0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    DWORD PTR [rbp-0x4],0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
 69e: 89 c6                 mov    esi,eax
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 754 ...
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    DWORD PTR [rbp-0x4],0x1
 6b5: 83 7d fc 09           cmp    DWORD PTR [rbp-0x4],0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    DWORD PTR [rbp-0x4],0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
 69e: 89 c6                 mov    esi,eax
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 754 ...
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    DWORD PTR [rbp-0x4],0x1
 6b5: 83 7d fc 09           cmp    DWORD PTR [rbp-0x4],0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

NOTE: we replace `DWORD PTR [rbp-0x4]` with `i` to improve readability

Assembly of count.c

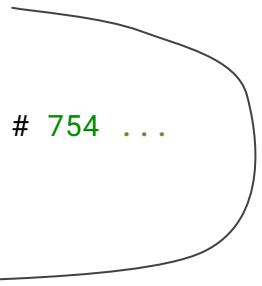
```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]    # 754 ...
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

NOTE: we replace `DWORD PTR [rbp-0x4]` with `i` to improve readability

Assembly of count.c

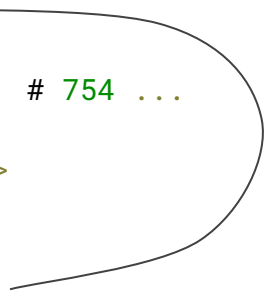
```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]    # 754 ...
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```



```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]    # 754 ...
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```



```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax          # 2nd param ← i
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param addr.
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Search in the data section

```
6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param addr.
6a7: b8 00 00 00 00        mov    eax,0x0
```

```
>>> hex(0x6a7 + 0xad)          ← in python
'0x754'
```

We search in the assembly file an address close to `0x754`:

```
Contents of section .rodata:
 0750 01000200 25642000                    ....%d .
```

address `0x754` contains bytes `0x25 0x64 0x20 0x00` ⇒ the string `"%d "`

```
>>> b'\x25\x64\x20\x00'          ← in python
b'%d \x00'
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax          # 2nd param ← i
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param ← "%d "
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax          # 2nd param ← i
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param ← "%d "
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax          # 2nd param ← i
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param ← "%d "
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax          # 2nd param ← i
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param ← "%d "
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

NOTE: `printf` is implemented with `putchar` , 0xa is \n

Assembly of count.c

```
000000000000068a <main>:
 68a: 55                    push   rbp
 68b: 48 89 e5              mov    rbp,rsp
 68e: 48 83 ec 10           sub    rsp,0x10
 692: c7 45 fc 00 00 00 00  mov    i,0x0
 699: eb 1a                 jmp    6b5 <main+0x2b>
 69b: 8b 45 fc              mov    eax,i
 69e: 89 c6                 mov    esi,eax          # 2nd param ← i
 6a0: 48 8d 3d ad 00 00 00  lea    rdi,[rip+0xad]   # 1st param ← "%d "
 6a7: b8 00 00 00 00        mov    eax,0x0
 6ac: e8 af fe ff ff        call   560 <printf@plt>
 6b1: 83 45 fc 01           add    i,0x1
 6b5: 83 7d fc 09           cmp    i,0x9
 6b9: 7e e0                 jle    69b <main+0x11>
 6bb: bf 0a 00 00 00        mov    edi,0xa
 6c0: e8 8b fe ff ff        call   550 <putchar@plt>
 6c5: b8 00 00 00 00        mov    eax,0x0          # returns 0
 6ca: c9                    leave
 6cb: c3                    ret
```

```c
#include <stdio.h>

int main()
{
      int i;
      for (i=0; i<10; i++)
            printf("%d ",i);
      printf("\n");
}
```

Patching executables

hexdump and back: xxd

Executables are binary files and can be edited using **specific editors** (es. `hexedit`) or standard editors that also support binary files (es. sublime)

xxd allows for producing a **text file** with hex code of bytes, so that they can be edited with any editor

xxd is able to **regenerate the binary** file from the modified hexdump

Example:

```
$ xxd -g 1 count > count.txt
```

Option `-g 1` writes distinct bytes

We edit and modify `count.txt` and we generate a new binary `count2` with option `-r`

```
$ xxd -r count.txt > count2
$ chmod +x count2  # executable
$ ./count2
```

Exercises

Exercise 1: change the branch. Modify the /home/rookie/Assembly/`count` executable file so that the loop ends at 8 instead of 9

Hint: try to change **`jle`** into **`jl`**, this summary of x86 opcodes (sometimes chrome block this site because of http) will help you!

Exercise 2: change the value. Modify the /home/rookie/Assembly/`count` executable file so that it steps of 2 instead of 1 (only prints even values)

Hint: In this case you need to change data

Exercise 3: skip a branch. Modify the /home/rookie/Assembly/`checkPassword` so that it skips the password check

Hint: nop is your friend! (opcode 0x90)