CSE 351
Section 6

2/9/12

Agenda

- Midterm Review!
  - Words/Data Types
  - Byte Ordering
  - Bit-Level and Logic Operations
  - Integers
  - Floating Points
  - C and Assembly
  - Calling Conventions
  - Stack

Word Sizes

- Machines have word sizes
  - A 64-bit machine will have 64-bit pointers and addresses
- Specify locations of words in memory by their first byte
  - E.g. “int x” lives at address 0x1000, but actually takes up bytes 0x1000, 0x1001, 0x1002, 0x1003
- In C, get address of an object with “&” and dereference with “*”
  - E.g. int* x = &y; *y=5;

C Data Types

C Data Type	Number of Bytes in x86-64
bool	1
char	1
short int	2
int	4
float	4
long int	8
double	8
long long	8
pointer *	8

Byte Ordering

- Want to store 0xaabbccdd at memory address 0x100
- Little Endian
  - **Least** significant byte first
  - 0x100 has byte dd
- Big Endian
  - **Most** significant byte first
  - 0x100 has byte aa

Bit-Level Operations

- & - Bit level AND
  - 1011 & 0101 = 0001
- | - Bit level OR
  - 1011 | 0101 = 1111
- ^ - Bit level XOR
  - 1011 ^ 0101 = 1110
- ~ - Bit level NOT
  - ~1011 = 0100
- << - Left Shift
  - 1011 << 2 = 1100
- Right Shift
  - Logical
    - 1011 >> 2 = 0010
  - Arithmetic
    - 1011 >> 2 = 1110

Logical Operations

- 0 = False, anything non-zero = True
- && - Logical AND
  - 1011 && 0101 = 1
  - 1011 && 0000 = 0
- || - Logical OR
  - 1011 || 0101 = 1
  - 1011 || 0000 = 1
- ! – Logical NOT
  - !(1011) = 0
  - !(0000) = 1

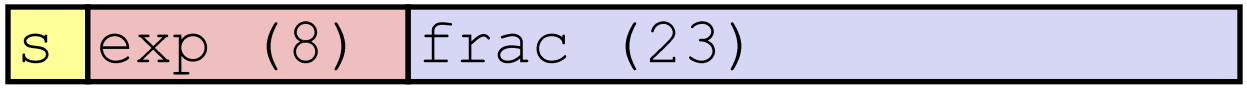
Integers

- Unsigned
  - Always positive
  - From 0 to 2^n-1
- Signed
  - Negative represented as 2's complement
  - From -2^{n-1} to 2^{n-1}-1
  - Negative calculated by taking positive number, flipping all bits and adding 1
- Any mix of unsigned and signed values in a single expression, all signed values are implicitly cast to unsigned
- Casting a smaller data type to a bigger data type, sign gets extended to upper bits

Floating Point

- $(-1)^S * M * 2^E$
- S = sign
- M = mantissa (encoded in `frac` field)
  - In range of [1,2)
- E = exponent (encoded in `exp` field)
- `frac` = M-1
  - Fractional part of mantissa
  - Each bit is negative power of 2
  - Extended with zeros to fit field
  - 23 bits for float, 52 for double
- `exp` = E + Bias
  - Bias = $2^{k-1}-1$ where k is the size of the `exp` field
  - 8 bits for float, 11 for double
- **Never test floating point values for equality! Mathematically equivalent expressions may yield different results.**

Floating Point Example



- How is float 12345.0 represented?
- Value

  $12345.0_{10} = 11000000111001_2$

  $= 1.1000000111001_2 * 2^{13}$

- Mantissa

  M = $1.\underline{1000000111001}_2$

  `frac` = $\underline{1000000111001}0000000000_2$ (Need to extend to fill all 23 bits)

- Exponent

  E = 13

  Bias = $2^7 - 1 = 127$

  `exp` = $140_{10} = 10001100_2$

x86-64 (aka x64) ISA

- Three main classes of instructions
  - Moving data
    - mov, push, pop
  - Computing data
    - add, sub, imul
  - Branching
    - cmp, jmp, je
- 16 registers
  - rax, rcx, rbx, rdx, rsp, rbp, rsi, rdi, r8 through r14
  - All 64-bits wide, but can access smaller fields
    - E.g. eax = lower 32 bits of rax. ax=lower 16 bits. al = lower 8 bits.

x86-64 Integer Registers: Usage Conventions

%rax	Return value
%rbx	Callee saved
%rcx	Argument #4
%rdx	Argument #3
%rsi	Argument #2
%rdi	Argument #1
%rsp	Stack pointer
%rbp	Callee saved

%r8	Argument #5
%r9	Argument #6
%r10	Caller saved
%r11	Caller Saved
%r12	Callee saved
%r13	Callee saved
%r14	Callee saved
%r15	Callee saved

Calling Conventions

- First 6 integer arguments to a function passed through registers

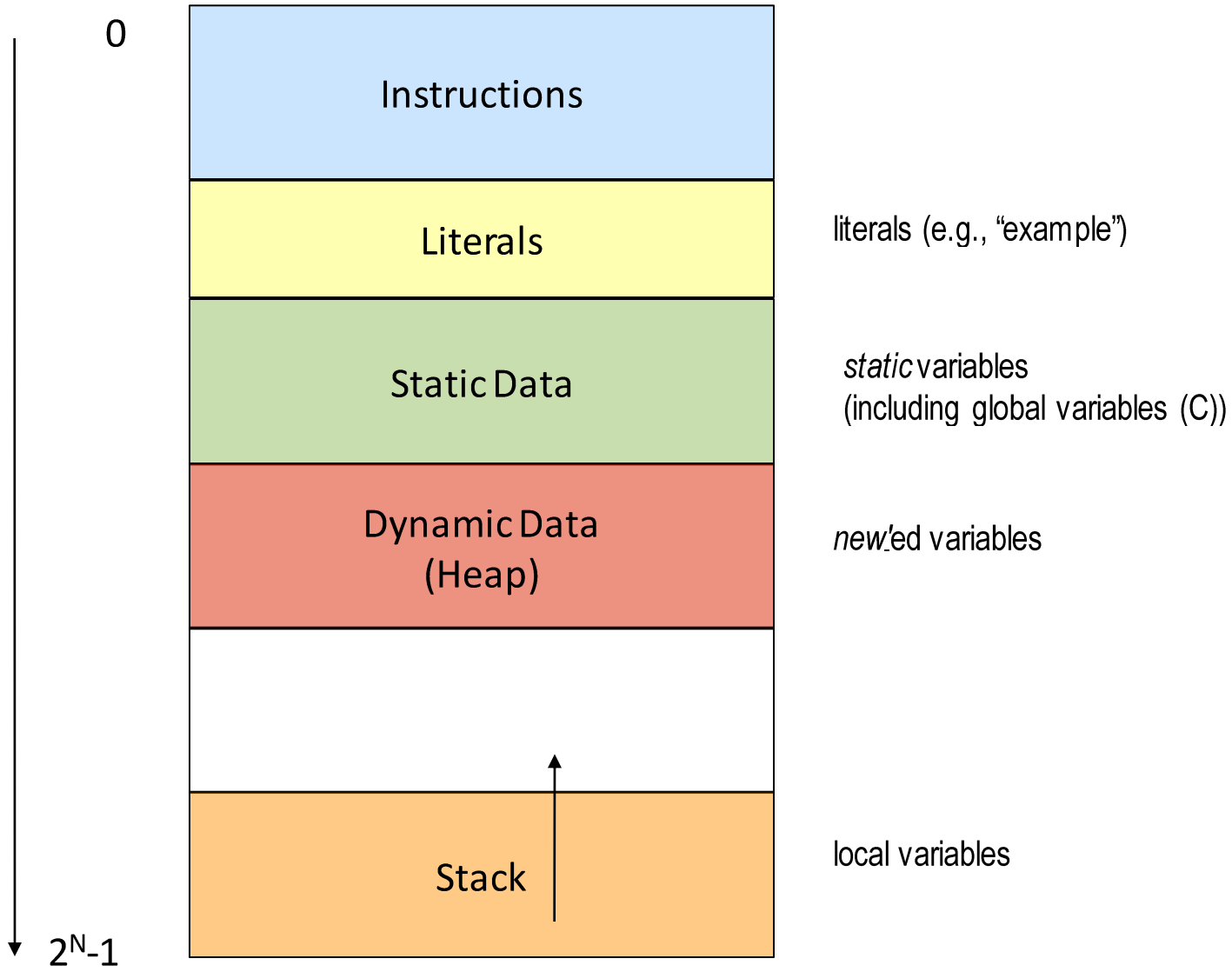
```
uint64_t foobar(uint64_t a1 uint64_t a2, uint64_t
        a3, uint64_t a4, uint64_t a5,
        uint64_t a6){
        return 10;
}
```

- x=foobar(1,2,3,4,5,6)
  - rdi=1, rsi=2, rdx=3, rcx=4, r8=5, r9=6
  - Upon returning, rax=10
- Also if type is less than 64 bits
  - char foobar(char x, int64_t y)
  - edi=2, rsi=-3
- Remainder of arguments go on the stack
- xmm registers (floating point) used for passing FP arguments

Calling Conventions

- When a function is called
  - Caller saved registers must be saved by the function doing the calling
    - Assume function being called will destroy the data in those registers
  - Callee saved registers must be saved by the function being called
    - Restore contents of registers before returning
- rsp is special and used to point to the bottom of the stack

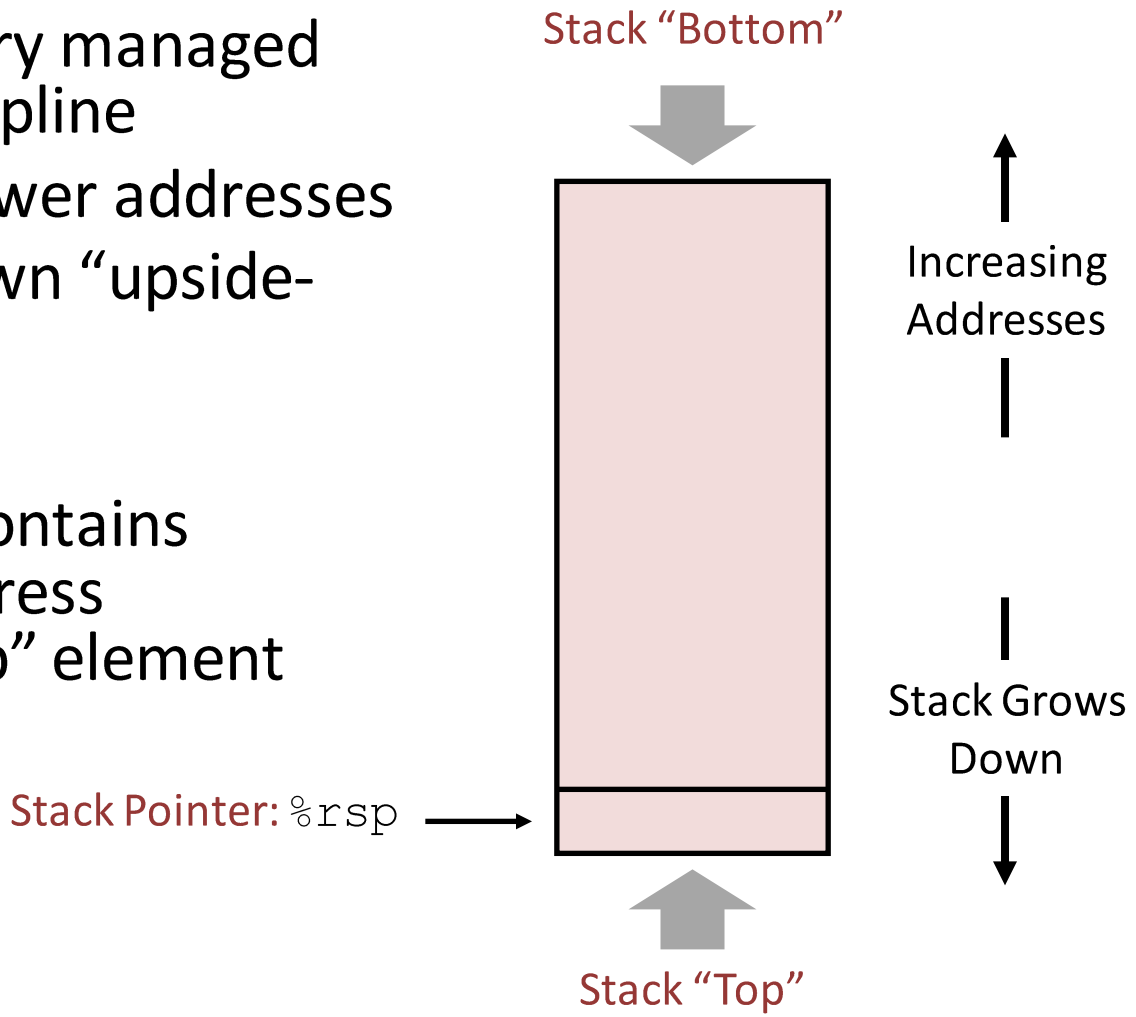
Memory Layout



Stack

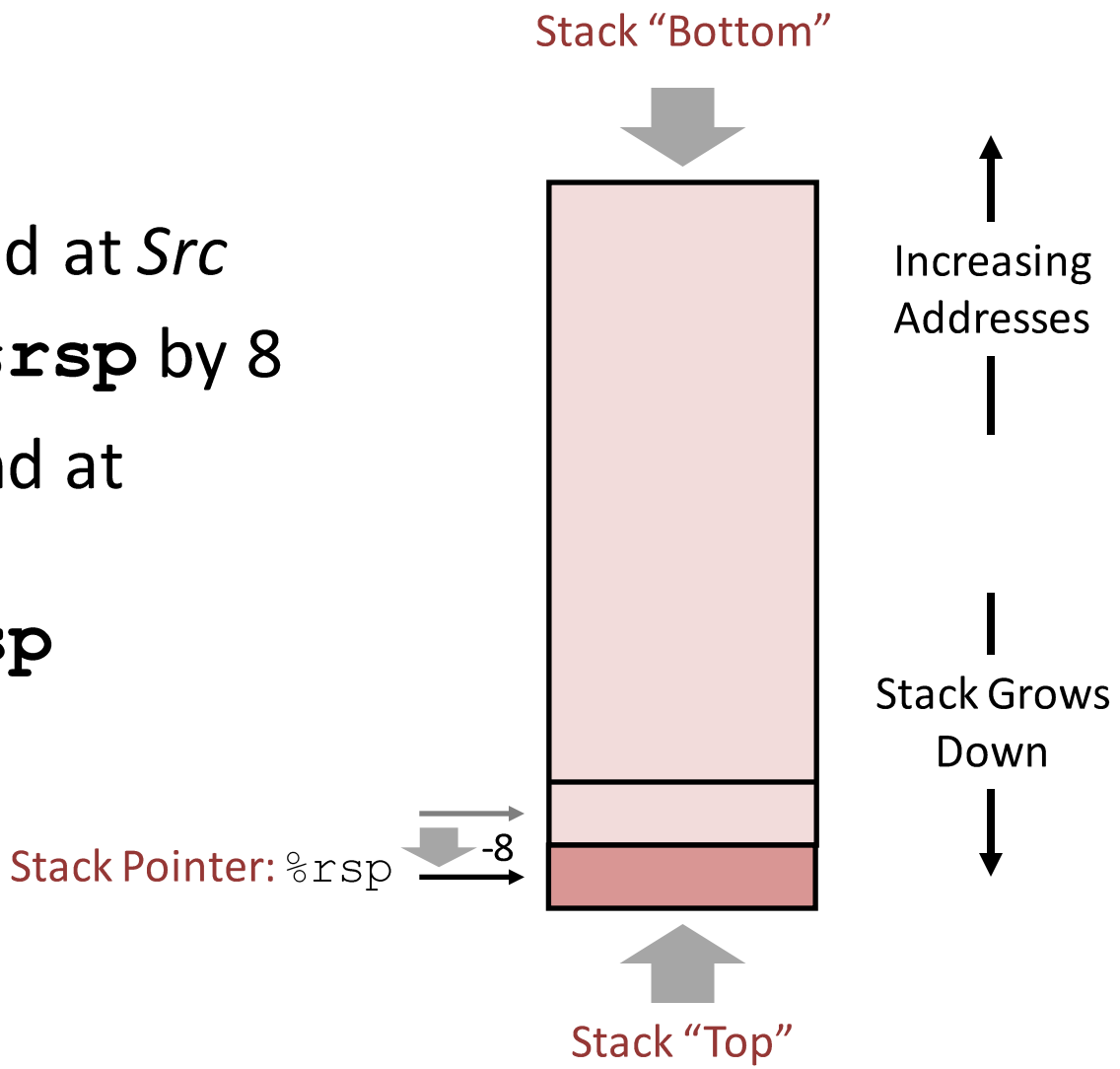
- Region of memory managed with a stack discipline
- Grows toward lower addresses
- Customarily shown "upside-down"

- Register `%rsp` contains lowest stack address = address of "top" element



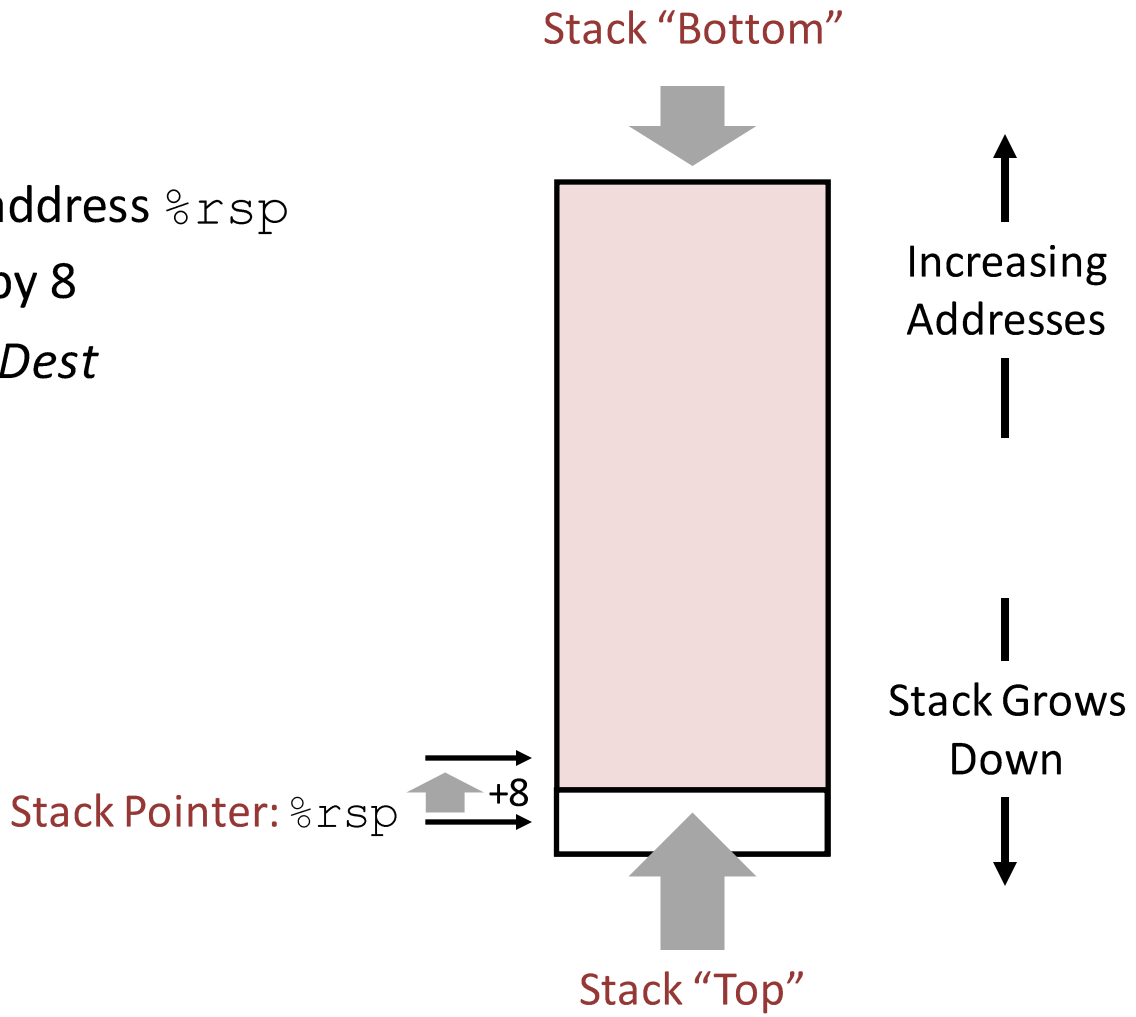
Push

- `pushq` *Src*
  - Fetch operand at *Src*
  - Decrement **`%rsp`** by 8
  - Write operand at address given by **`%rsp`**



Pop

- **popq** *Dest*
  - Read operand at address `%rsp`
  - Increment `%rsp` by 8
  - Write operand to *Dest*



Procedure Control Flow

- Use stack to support procedure call and return
- `call` *`label`*
  - Push return address on stack
    - Address of instruction after the call
  - Jump to *`label`*
- `ret`
  - Pop address from stack
  - Jump to address

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	<= rsp, rbp
0xFF8	
0xFF0	
0xFE8	
0xFE0	
0xFD8	
0xFD0	
0xFC8	
0xFC0	
0xFB8	
0xFB0	
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	

rax	rdi	rsi	rbp	rsp
0	0	0	0x1000	0x1000

For illustration purposes, assume that before call to main, rsp = rbp = 0x1000. This example does sum(3). Note: each slide shows result **after** instruction executes.

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
     => push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		<= rbp
0xFF8	0x1000	<= rsp
0xFF0		
0xFE8		
0xFE0		
0xFD8		
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
0	0	0	0x1000	0xFF8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
     => mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	
0xFE8	
0xFE0	
0xFD8	
0xFD0	
0xFC8	
0xFC0	
0xFB8	
0xFB0	
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp, rbp

rax	rdi	rsi	rbp	rsp
0	0	0	0xFF8	0xFF8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
     => mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	
0xFE8	
0xFE0	
0xFD8	
0xFD0	
0xFC8	
0xFC0	
0xFB8	
0xFB0	
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp, rbp

rax	rdi	rsi	rbp	rsp
0	3	0	0xFF8	0xFF8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
     => call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	
0xFE0	
0xFD8	
0xFD0	
0xFC8	
0xFC0	
0xFB8	
0xFB0	
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rbp

<= rsp

rax	rdi	rsi	rbp	rsp
0	3	0	0xFF8	0xFF0

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
     => push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	<= rbp
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rsp
0xFE0		
0xFD8		
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
0	3	0	0xFF8	0xFE8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
     => mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	
0xFD0	
0xFC8	
0xFC0	
0xFB8	
0xFB0	
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
0	3	0	0xFE8	0xFE8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
     => sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8		<= rsp
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
0	3	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
     => mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
0	3	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
    =>  mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	3	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
    =>  cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	3	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
    =>  je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	3	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
     => sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0		
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
    =>  call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	<= rsp
0xFC8		
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFE8	0xFD0

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
     => push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rsp
0xFC0		
0xFB8		
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFE8	0xFC8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
     => mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	
0xFB0	
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
1	2	0	0xFC8	0xFC8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
     => sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8		<= rsp
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
     => mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	<= rsp
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
     => mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	<= rsp
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
     => cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	<= rsp
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
    =>  je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	<= rsp
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	2	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
     => sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	<= rsp
0xFB0		
0xFA8		
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
     => call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8 <= rbp
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>* <= rsp
0xFA8	
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

rax	rdi	rsi	rbp	rsp
1	1	0	0xFC8	0xFB0

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
     => push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	<= rsp
0xFA0		
0xF98		
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFC8	0xFA8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
     => mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xFA8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
     => sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	<= rbp
0xFA0		
0xF98		<= rsp
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xF98

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
     => mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8 <= rbp
0xFA0	
0xF98	0x1 <= rsp
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xF98

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
     => mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	<= rbp
0xFA0		
0xF98	0x1	<= rsp
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xF98

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
    =>  cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	<= rbp
0xFA0		
0xF98	0x1	<= rsp
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xF98

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
    =>  je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	<= rbp
0xFA0		
0xF98	0x1	<= rsp
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xF98

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
     => add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rbp,rsp

rax	rdi	rsi	rbp	rsp
1	1	0	0xFA8	0xFA8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
     => pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	<= rsp
0xFA8	0xFC8	
0xFA0		
0xF98	0x1	
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
1	1	0	0xFC8	0xFB0

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
    => ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8 <= rbp
0xFC0	
0xFB8	0x2 <= rsp
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

rax	rdi	rsi	rbp	rsp
1	1	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
     => add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	<= rbp
0xFC0		
0xFB8	0x2	<= rsp
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	
0xFA0		
0xF98	0x1	
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
3	1	0	0xFC8	0xFB8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
     => add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
3	1	0	0xFC8	0xFC8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
     => pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	
0xFD0	*<addr of add instr>*	<= rsp
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	
0xFA0		
0xF98	0x1	
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
3	1	0	0xFE8	0xFD0

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
    => ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	
0xFA0		
0xF98	0x1	
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
3	1	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
    =>  add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000		
0xFF8	0x1000	
0xFF0	*<addr of mov instr>*	
0xFE8	0xFF8	<= rbp
0xFE0		
0xFD8	0x3	<= rsp
0xFD0	*<addr of add instr>*	
0xFC8	0xFE8	
0xFC0		
0xFB8	0x2	
0xFB0	*<addr of add instr>*	
0xFA8	0xFC8	
0xFA0		
0xF98	0x1	
0xF90		
0xF88		
0xF80		
0xF78		
0xF70		
0xF68		

rax	rdi	rsi	rbp	rsp
6	1	0	0xFE8	0xFD8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
     => add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
6	1	0	0xFE8	0xFE8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
    =>  pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rbp

<= rsp

rax	rdi	rsi	rbp	rsp
6	1	0	0xFF8	0xFF0

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
    => ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
6	1	0	0xFF8	0xFF8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
     => mov     $print_string, %edi
        mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
6	& of str	0	0xFF8	0xFF8

Example: Sum

```
.data
print_string: .string "The result is %d\n"

.text
        .globl  main
sum:
        push    %rbp
        mov     %rsp, %rbp
        sub     $16, %rsp
        mov     %rdi, 0(%rsp)
        mov     $1, %eax
        cmp     $1, %rdi
        je      exit_sum
        sub     $1, %rdi
        call    sum
        add     0(%rsp), %rax
exit_sum:
        add     $16, %rsp
        pop     %rbp
        ret
main:
        push    %rbp
        mov     %rsp, %rbp
        mov     $3, %edi
        call    sum
        mov     $print_string, %edi
     => mov     %rax, %rsi
        call    printf
        pop     %rbp
        ret
```

Stack

0x1000	
0xFF8	0x1000
0xFF0	*<addr of mov instr>*
0xFE8	0xFF8
0xFE0	
0xFD8	0x3
0xFD0	*<addr of add instr>*
0xFC8	0xFE8
0xFC0	
0xFB8	0x2
0xFB0	*<addr of add instr>*
0xFA8	0xFC8
0xFA0	
0xF98	0x1
0xF90	
0xF88	
0xF80	
0xF78	
0xF70	
0xF68	

<= rsp,rbp

rax	rdi	rsi	rbp	rsp
6	& of str	6	0xFF8	0xFF8