作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师

入参的规则

入参的规则：
入参的数量，可以有 0 个、1 个、多个。入参少，使用寄存器传参，一个寄存器对应一个入参。
入参的传递，优先使用寄存器。如果寄存器不够用，再使用函数栈。
整数入参，使用整数寄存器 `rdi rsi rdx rcx r8 r9` 。
浮点数入参，使用浮点数寄存器 `xmm0 xmm1 xmm2 xmm3 xmm4 xmm5 xmm6` 。
入参是指针，表示引用传递。
入参是非指针，表示值传递。
入参支持自定义的寄存器、函数栈。

用 C 和汇编分析入参数量、整数入参、浮点数入参

编写代码： param.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

// 函数没有入参
void func_no_param()
{
    printf("函数没有入参 \n");
}

// 函数有 1 个入参
void func_1_param(int age)
```

```
{
    printf("函数有 1 个入参 %d \n", age);
}

// 函数有 5 个入参
void func_5_param(int age, int age2, int age3, int age4, int age5)
{
    printf("函数有 5 个入参 %d %d %d %d %d \n", age, age2, age3, age4, age5);
}

// 函数有 9 个入参
void func_9_param(int age, int age2, int age3, int age4, int age5,
                  int age6, int age7, int age8, int age9)
{
    printf("函数有 9 个入参 %d %d %d %d %d %d %d %d %d \n", age, age2, age3,
           age4, age5, age6, age7, age8, age9);
}

// 函数有 double/int 入参
void func_double_int_param(double height, double height2, int age, int age2)
{
    printf("函数有 double/int 入参 %f %f %d %d \n", height, height2, age, age2);
}

int main()
{
    func_no_param();
    func_1_param(100);
    func_5_param(100, 200, 300, 400, 500);
    func_9_param(100, 200, 300, 400, 500, 600, 700, 800, 900);
    func_double_int_param(11.11, 22.22, 100, 200);
    return 0;
}
```

编译代码：

```
gcc param.c -o param
gcc -S param.c -o param.s
```

运行代码：

```
[root@localhost func]# ./param
函数没有入参
函数有 1 个入参 100
函数有 5 个入参 100 200 300 400 500
函数有 9 个入参 100 200 300 400 500 600 700 800 900
函数有 double/int 入参 11.110000 22.220000 100 200
```

查看汇编文件 param.s，分别查看每个函数。这里截取与入参有关的部分。

入参分类	函数定义	函数调用	分析
函数没有入参	func_no_param: movl $.LC0, %edi call puts	call func_no_param	没有入参。 没有使用寄存器
函数有 1 个入参	func_1_param: movl %edi, -4(%rbp)	movl $100, %edi call func_1_param	使用 1 个寄存器 edi。

函数有 5 个入参	func_5_param:
	movl %edi, -4(%rbp)
	movl %esi, -8(%rbp)
	movl %edx, -12(%rbp)
	movl %ecx, -16(%rbp)
	movl %r8d, -20(%rbp)
		movl $500, %r8d
		movl $400, %ecx
		movl $300, %edx
		movl $200, %esi
		movl $100, %edi
		call func_5_param
			使用 5 个寄存器 edi、esi、edx、ecx、r8d。

函数有 9 个入参	func_9_param:
	movl %edi, -4(%rbp)
	movl %esi, -8(%rbp)
	movl %edx, -12(%rbp)
	movl %ecx, -16(%rbp)
	movl %r8d, -20(%rbp)
		subq $32, %rsp
		movl $900, 16(%rsp)
		movl $800, 8(%rsp)
		movl $700, (%rsp)
		movl $600, %r9d
		movl $500, %r8d
			使用 6 个寄存器 edi、esi、edx、ecx、r8d、r9d。
			使用函数栈传递了 3 个入参 700、800、900。

	movl %r9d, -24(%rbp)
	movl 32(%rbp), %esi
	movl 24(%rbp), %esi
	movl 16(%rbp), %esi
		movl $400, %ecx
		movl $300, %edx
		movl $200, %esi
		movl $100, %edi
		call func_9_param	

函数有
double/int 入参
	func_double_int_param:
	movsd %xmm0, -8(%rbp)
	movsd %xmm1, -16(%rbp)
	movl %edi, -20(%rbp)
	movl %esi, -24(%rbp)
		movabsq $4626947591664639672, %rdx
		movabsq $4622443992037269176, %rax
		movl $200, %esi
		movl $100, %edi
		movq %rdx, -8(%rbp)
		movsd -8(%rbp), %xmm1
		movq %rax, -8(%rbp)
		movsd -8(%rbp), %xmm0
		call func_double_int_param
			使用 2 个整数寄存器 edi、esi。
			使用 2 个浮点数寄存器 xmm0、xmm1。
			
			4626947591664639672、4622443992037269176，这 2 个奇怪的数字，是浮点数的 8 个字节被转成整数。

问题：汇编代码，函数调用时传递入参，入参的顺序为什么是反向的？
C 语言代码， `func_5_param(100, 200, 300, 400, 500)` ，入参顺序为 `100 200 300 400 500` 。
汇编代码，入参顺序为 `500 400 300 200 100` 。
汇编代码，要求把入参写到对应的寄存器、栈内存，对写入参的汇编代码的顺序没有要求，顺序可以改变。

用汇编实现入参使用自定义的寄存器

上面的寄存器顺序，是默认的约定。用高级语言(C/C++/Java 等)写程序，编译器自动把入参对应寄存器。
如果手写一个汇编函数，可以自定义使用哪些寄存器。

编写代码： param_reg.s

```
.global main

.data
```

```
# 字符串，打印多个 int 值
out_str :
    .string "params : %d , %d , %d  \n"

.text

# 自定义函数，入参使用寄存器 rax rbx rcx
func_self_param:
    pushq %rbp
    movq %rsp,%rbp

    movq $out_str, %rdi  # printf 的第一个参数
    movl %eax, %esi  # 接收入参 1
    movl %ebx, %edx  # 接收入参 2
    movl %ecx, %ecx  # 接收入参 3
    callq printf     # 调用函数 printf

    popq %rbp
    retq

# main func
main:
    pushq %rbp
    movq %rsp, %rbp

    movl $111, %eax  # 入参 1，使用 rax
    movl $222, %ebx  # 入参 2，使用 rbx
    movl $333, %ecx  # 入参 3，使用 rcx
    callq func_self_param  # 调用函数 func_self_param

    popq %rbp
    retq
```

编译代码：

```
gcc param_reg.s -o param_reg
```

运行代码：

```
[root@192 func]# ./param_reg
params : 111 , 222 , 333
```

分析结果：
函数 main，调用函数 func_self_param，使用 rax、rbx、rcx 传递了 3 个入参 111、222、333。
函数 func_self_param，打印了 3 个入参。入参没有使用 rdi、rsi、rdx，入参使用 rax、rbx、rcx，功能正常。

问题：上面使用了自定义的寄存器传递入参，不使用寄存器可以传递入参吗？
可以只使用函数栈传递入参。这时，入参不依赖寄存器。

用汇编实现只使用函数栈传递入参

编写代码： param_stack.s

```
.global main

.data

# 字符串，打印多个 int 值
out_str :
    .string "params : %d , %d \n"

.text

# 自定义函数，入参使用函数栈 rbp rsp
```

```
func_self_param :
    pushq %rbp
    movq %rsp, %rbp

    movq $out_str, %rdi  # printf 的第一个参数
    movl 16(%rbp), %esi  # 接收入参 1
    movl 20(%rbp), %edx  # 接收入参 2
    callq printf

    popq %rbp
    retq

main:
    pushq %rbp
    movq %rsp, %rbp

    subq $32, %rsp   # 扩大栈

    movl $777, 0(%rsp)   # 入参 1，使用 rsp+0
    movl $888, 4(%rsp)   # 入参 2，使用 rsp+4
    callq func_self_param  # 调用函数 func_self_param

    addq $32, %rsp   # 缩小栈。必须和上方的扩大栈配合使用。

    popq %rbp
    retq
```

编译代码：

```
gcc param_stack.s -o param_stack
```

运行代码：

```
[root@localhost func]# ./param_stack
params : 777 , 888
```

分析结果：
函数 main，调用函数 func_self_param，使用 rsp 传递了 2 个入参 777、888。
函数 func_self_param，打印 2 个入参。入参没有使用寄存器，入参使用 rbp，功能正常。

问题：写入参 `movl $777, 0(%rsp)` 使用 rsp，读入参 `movl 16(%rbp), %esi` 使用 rbp。为什么一个用 rsp，一个用 rbp？为什么 rsp/rbp 前方的偏移量不一样？
rbp/rsp 是函数栈的寄存器。如果找到相同的内存位置，两者可以交换使用。
偏移量不同，因为在调用函数时，把 rip 入栈使用 8 字节，把 rbp 入栈使用 8 字节，共使用 16 字节。