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

指令说明

add 指令表示整数加法。
add 指令分为 8 位 addb、16 位 addw、32 位 addl、64 位 addq。
add 指令可以操作立即数、寄存器、内存。

语法格式 `add ee,ff` 表示 `ff = ff + ee` 。

简单的加法

编写代码： add.s

```
.data

num_int64:
    .quad 0x0  # 64 位

str_int8:
    .string "  int8   value = %#X \n"

str_int16:
    .string "  int16  value = %#X \n"

str_int32:
    .string "  int32  value = %#X \n"

str_int64:
    .string "  int64  value = %lld \n"

.text
.global main

main :
    pushq %rbp
    movq %rsp, %rbp
    subq $64 , %rsp

    # 64 位。操作寄存器
    movq $2000, %rcx
    addq $3, %rcx       # 应用于寄存器
    movq $str_int64, %rdi
    movq %rcx, %rsi
    callq printf

    # 64 位。操作栈
```

```
    movq $2000, -8(%rbp)
    addq $5, -8(%rbp)   # 应用于栈
    movq $str_int64, %rdi
    movq -8(%rbp) , %rsi
    callq printf

    # 64 位。操作数据段
    movq $2000, num_int64(%rip)
    addq $7, num_int64(%rip)    # 应用于数据段
    leaq str_int64(%rip), %rdi
    movq num_int64(%rip) , %rsi
    callq printf

    # 8 位。操作寄存器
    movb $0x22, %al     # 写 1 个字节
    addb $0x11, %al     # 加法，8 位
    movzbl %al, %esi    # 把 1 个字节扩展为 4 个字节
    leaq str_int8(%rip), %rdi
    callq printf

    # 16 位。操作寄存器
    movw $0x3333, %ax   # 写 2 个字节
    addw $0x2222, %ax   # 加法，16 位
    movzwl %ax, %esi    # 把 2 个字节扩展为 4 个字节
    leaq str_int16(%rip), %rdi
    callq printf

    # 32 位。操作寄存器
    movl $0x55555555, %esi # 写 4 个字节
    addl $0x2222, %esi     # 加法，32 位
    leaq str_int32(%rip), %rdi
    callq printf

    addq $64 , %rsp
    popq %rbp
    retq
```

编译代码：

```
gcc add.s -o add
```

运行代码：

```
[root@local int]# ./add
  int64  value = 2003
  int64  value = 2005
  int64  value = 2007
  int8   value = 0X33
  int16  value = 0X5555
  int32  value = 0X55557777
```

分析结果：

汇编代码	输出结果

# 64 位。操作寄存器 	
movq $2000, %rcx
addq $3, %rcx # 应用于寄存器
	int64 value = 2003
	
	add 指令操作寄存器 rcx。
	2000 + 3 = 2003

# 64 位。操作栈 	
movq $2000, -8(%rbp)
addq $5, -8(%rbp) # 应用于栈
	int64 value = 2005
	
	add 指令操作操作栈-8(%rbp)。
	2000 + 5 = 2005

# 64 位。操作数据段 	
movq $2000, num_int64(%rip)
addq $7, num_int64(%rip) # 应用于数据段
	int64 value = 2007
	
	add 指令操作数据段 num_int64(%rip)。
	2000 + 7 = 2007

# 8 位。操作寄存器 	
movb $0x22, %al # 写 1 个字节
addb $0x11, %al # 加法，8 位
	int8 value = 0X33
	
	8 位加法使用 addb 指令。
	0x22 + 0x11 = 0X33

# 16 位。操作寄存器 	
movw $0x3333, %ax # 写 2 个字节
addw $0x2222, %ax # 加法，16 位
	int16 value = 0X5555
	
	16 位加法使用 addw 指令。
	0x3333 + 0x2222 = 0X5555

# 32 位。操作寄存器 	
movl $0x55555555, %esi # 写 4 个字节
addl $0x2222, %esi # 加法，32 位
	int32 value = 0X55557777
	
	32 位加法使用 addl 指令。
	0x55555555 + 0x2222 = 0X55557777

复杂的加法

编写代码： add_hard.s

```
.data

num_int64:
    .quad 0x0

str_int64:
    .string "addl  int64  value = %#llX \n"

str_flow_int32:
    .string "flow  int32  value = %#X \n"

str_flow_int64:
    .string "flow  int64  value = %#llX \n"

.text
```

```
.global main

main :
    pushq %rbp
    movq %rsp, %rbp

    # 64 位变量，执行 32 位加法。操作数据段
    movq $0x3333333355555555, %r9
    movq %r9, num_int64(%rip)
    addl $0x11111111, num_int64(%rip)
    movq num_int64(%rip), %rsi
    leaq str_int64(%rip), %rdi
    callq printf

    # 32 位变量，32 位溢出。操作寄存器
    movl $0xFFFFFFFF, %edx
    addl $0x2, %edx
    leaq str_flow_int32(%rip), %rdi
    movl %edx, %esi
    callq printf

    # 64 位变量，32 位溢出。操作寄存器
    movq $0x55555555FFFFFFFF, %rbx
    addl $0x2, %ebx    # 导致高位清零
    leaq str_flow_int64(%rip), %rdi
    movq %rbx, %rsi
    callq printf

    # 64 位变量，16 位溢出。操作寄存器
    movq $0x55555555FFFFFFFF, %rbx
    addw $0x2, %bx
    leaq str_flow_int64(%rip), %rdi
    movq %rbx, %rsi
    callq printf

    # 64 位变量，8 位溢出。操作寄存器
    movq $0x55555555FFFFFFFF, %rbx
    addb $0x2, %bh
    leaq str_flow_int64(%rip), %rdi
    movq %rbx, %rsi
    callq printf

    # 64 位变量，8 位溢出。操作寄存器
    movq $0x55555555FFFFFFFF, %rbx
    addb $0x2, %bl
    leaq str_flow_int64(%rip), %rdi
    movq %rbx, %rsi
    callq printf
```

```
    # 64 位变量，32 位溢出。操作数据段
    movq $0x55555555FFFFFFFF, %r9
    movq %r9, num_int64(%rip)
    addl $0x2, num_int64(%rip)
    leaq str_flow_int64(%rip), %rdi
    movq num_int64(%rip), %rsi
    callq printf

    popq %rbp
    retq
```

编译代码：

```
gcc add_hard.s -o add_hard
```

运行代码：

```
[root@local int]# ./add_hard
addl  int64  value = 0X3333333366666666
flow  int32  value = 0X1
flow  int64  value = 0X1
flow  int64  value = 0X55555555FFFF0001
flow  int64  value = 0X55555555FFFF01FF
flow  int64  value = 0X55555555FFFFFF01
flow  int64  value = 0X5555555500000001
```

分析结果：

汇编代码	输出结果

# 64 位变量，执行 32 位加法。操作数据段 	
movq $0x3333333355555555, %r9
movq %r9, num_int64(%rip)
addl $0x11111111, num_int64(%rip)
	addl int64 value = 0X3333333366666666
	
	把 64 位 0x3333333355555555 写到 num_int64。
	给 num_int64 加上 0x11111111。
	结果为 0X3333333366666666。
	低 32 位受影响，从 55555555 变为 66666666。
	高 32 位不变，都为 33333333。

# 32 位变量，32 位溢出。操作寄存器 	
movl $0xFFFFFFFF, %edx
addl $0x2, %edx
	flow int32 value = 0X1
	
	把 32 位 0xFFFFFFFF 写到 edx。
	edx 加上 0x2，导致溢出，结果为 0X1。

# 64 位变量，32 位溢出。操作寄存器 	
movq $0x55555555FFFFFFFF, %rbx
addl $0x2, %ebx   # 导致高位清零
	flow int64 value = 0X1
	
	把 64 位 0x55555555FFFFFFFF 写到 rbx。
	32 位 ebx 加上 0x2。
	触发高 32 位清零，结果为 0X1。

# 64 位变量，16 位溢出。操作寄存器 	
movq $0x55555555FFFFFFFF, %rbx
addw $0x2, %bx
	flow int64 value = 0X55555555FFFF0001
	
	把 64 位 0x55555555FFFFFFFF 写到 rbx。
	16 位 bx 加上 0x2。
	结果为 0X55555555FFFF0001。
	低 16 位溢出，从 FFFF 变为 0001。
	其他位不变，都为 55555555FFFF。

# 64 位变量，8 位溢出。操作寄存器 	
movq $0x55555555FFFFFFFF, %rbx
addb $0x2, %bh
	flow int64 value = 0X55555555FFFF01FF
	
	把 64 位 0x55555555FFFFFFFF 写到 rbx。
	8 位 bh 加上 0x2。
	结果为 0X55555555FFFF01FF。
	高 8 位溢出，从 FF 变为 01。
	其他位不变，都为 55555555FFFF、FF。

# 64 位变量，8 位溢出。操作寄存器 	
movq $0x55555555FFFFFFFF, %rbx
addb $0x2, %bl
	flow int64 value = 0X55555555FFFFFF01
	
	把 64 位 0x55555555FFFFFFFF 写到 rbx。
	8 位 bl 加上 0x2。
	结果为 0X55555555FFFFFF01。
	低 8 位溢出，从 FF 变为 01。
	其他位不变，都为 55555555FFFFFF。

# 64 位变量，32 位溢出。操作数据段 	
movq $0x55555555FFFFFFFF, %r9
movq %r9, num_int64(%rip)
addl $0x2, num_int64(%rip)
	flow int64 value = 0X5555555500000001
	
	把 64 位 0x55555555FFFFFFFF 写到 num_int64。
	32 位 0x2 加到 num_int64。
	结果为 0X5555555500000001。
	低 32 位溢出，从 FFFFFFFF 变为 00000001。
	其他位不变，都为 55555555。

问题：64 位 0x55555555FFFFFFFF，加上 32 位 0x2，操作寄存器 rbx 和操作内存，为什么结果不相同？
给 64 位通用寄存器执行 32 位写操作指令，触发高 32 位清零。这些通用寄存器包括 rax、rbx、r8 等，这些写操作指令包括 addl、subl、movl 等。
给内存执行 32 位写操作，只影响 32 位内存。