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

整数计算器的功能

整数计算器，输入 2 个整数和 1 个运算符，执行整数运算，输出结果。
功能包括输入、输出、加法指令、减法指令、乘法指令、除法指令、函数指令、寻址指令、比较指令、跳转指令等。

整数计算器的实现

编写代码： int_computer.s

```
.data

int64_aa :  # 64 位整数
    .quad 0x0

int64_bb :  # 64 位整数
    .quad 0x0

int64_xx :  # 运算的结果
    .quad 0x0

input_op :  # 运算符
    .byte 0x0,0x0,0x0,0x0,0x0

str_tips :  # 提示
    .string ">> Please input like  1 + 2 \n"

str_read :  # 输入
    .string "%lld  %s  %lld"

str_out : # 输出
    .string "output  :  %lld  %s  %lld  =  %lld \n\n"

str_error :  # 错误
    .string " wrong params : %lld  %s  %lld \n\n"

# 运算符
op_add : .string "+"    # 加法
op_sub : .string "-"    # 减法
op_mul : .string "*"    # 乘法
op_div : .string "/"    # 除法取商
op_yu  : .string "%"    # 除法取余

.text
.global main
```

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

    # 提示
    movq $str_tips, %rdi
    callq printf

    # 输入
    movq $str_read , %rdi
    movq $int64_aa, %rsi    # 变量 aa
    movq $input_op, %rdx    # 运算符
    movq $int64_bb, %rcx    # 变量 bb
    callq scanf

    # 加法
    movq $input_op, %rdi
    movq $op_add, %rsi
    callq strcmp        # 比较运算符
    cmpq $0, %rax
    je mark_add

    # 减法
    movq $input_op, %rdi
    movq $op_sub, %rsi
    callq strcmp        # 比较运算符
    cmpq $0, %rax
    je mark_sub

    # 乘法
    movq $input_op, %rdi
    movq $op_mul, %rsi
    callq strcmp        # 比较运算符
    cmpq $0, %rax
    je mark_mul

    # 除法取商
    movq $input_op, %rdi
    movq $op_div, %rsi
    callq strcmp        # 比较运算符
    cmpq $0, %rax
    je mark_div

    # 除法取余
    movq $input_op, %rdi
    movq $op_yu, %rsi
    callq strcmp        # 比较运算符
    cmpq $0, %rax
    je mark_yu
```

```
    # 参数错误
    jmp mark_error

mark_add :
    movq int64_aa(%rip), %r8    # 变量 aa
    movq int64_bb(%rip), %r9    # 变量 bb
    addq %r9, %r8               # r8 = r8 + r9
    movq %r8, int64_xx(%rip)    # 保存结果
    jmp mark_out

mark_sub :
    movq int64_aa(%rip), %r8    # 变量 aa
    movq int64_bb(%rip), %r9    # 变量 bb
    subq %r9, %r8               # r8 = r8 - r9
    movq %r8, int64_xx(%rip)    # 保存结果
    jmp mark_out

mark_mul :
    movq int64_aa(%rip), %rax   # 变量 aa
    movq int64_bb(%rip), %r9    # 变量 bb
    imulq %r9                   # rax = rax * r9
    movq %rax, int64_xx(%rip)   # 保存结果
    jmp mark_out

mark_div :
    movq int64_aa(%rip), %rax   # 变量 aa
    movq int64_bb(%rip), %r9    # 变量 bb
    cqto                        # 符号扩展
    idivq %r9                   # rax = rax / r9
    movq %rax, int64_xx(%rip)   # 保存结果
    jmp mark_out

mark_yu :
    movq int64_aa(%rip), %rax   # 变量 aa
    movq int64_bb(%rip), %r9    # 变量 bb
    cqto                        # 符号扩展
    idivq %r9                   # rdx = rax % r9
    movq %rdx, int64_xx(%rip)   # 保存结果
    jmp mark_out

mark_error :  # 错误
    movq $str_error, %rdi
    movq int64_aa(%rip), %rsi
    movq $input_op, %rdx
    movq int64_bb(%rip), %rcx
    callq printf
    jmp mark_last
```

```
mark_out : # 输出
    callq func_print_out  # 调用函数

mark_last :

    popq %rbp
    retq

# ---------   输出函数
func_print_out :
    pushq %rbp
    movq %rsp, %rbp

    movq $str_out, %rdi
    movq int64_aa(%rip), %rsi   # 变量 aa
    movq $input_op, %rdx        # 运算符
    movq int64_bb(%rip), %rcx   # 变量 bb
    movq int64_xx(%rip), %r8    # 结果
    callq printf

    popq %rbp
    retq
```

编译代码：

```
gcc int_computer.s -o int_computer
```

运行代码：

```
[root@local zong]# ./int_computer
>> Please input like  1 + 2
22 + 33
output  :  22  +  33  =  55

[root@local zong]# ./int_computer
>> Please input like  1 + 2
22 - 33
output  :  22  -  33  =  -11

[root@local zong]# ./int_computer
>> Please input like  1 + 2
22 * 7
output  :  22  *  7  =  154

[root@local zong]# ./int_computer
>> Please input like  1 + 2
55 / 6
output  :  55  /  6  =  9

[root@local zong]# ./int_computer
```

```
>> Please input like  1 + 2
55 % 6
output  :  55  %  6  =  1
```

分析结果：
加法。输入 `22 + 33` ，输出 `22  +  33  =  55` 。
减法。输入 `22 - 33` ，输出 `22  -  33  =  -11` 。
乘法。输入 `22 * 7` ，输出 `22  *  7  =  154` 。
除法取商。输入 `55 / 6` ，输出 `55  /  6  =  9` 。
除法取余。输入 `55 % 6` ，输出 `55  %  6  =  1` 。

汇编代码	结果和分析

`int64_aa :  # 64 位整数`
`    .quad 0x0`
 	
`int64_bb :  # 64 位整数`
`    .quad 0x0`
 	
`int64_xx :  # 运算的结果`
`    .quad 0x0`
 	
`input_op :  # 运算符`
`    .byte 0x0,0x0,0x0,0x0,0x0`
	用变量承接 2 个输入整数、1 个运算符、1 个输出整数。
	运算符，用字节数组变量。

`op_add : .string "+"    # 加法`
`op_sub : .string "-"    # 减法`
`op_mul : .string "*"    # 乘法`
`op_div : .string "/"    # 除法取商`
`op_yu  : .string "%"    # 除法取余`
	运算符包括加法、减法、乘法、除法取商、除法取余。
	运算符格式为字符串。

`    # 输入`
`    movq $str_read , %rdi`
`    movq $int64_aa, %rsi    # 变量 aa`
`    movq $input_op, %rdx    # 运算符`
`    movq $int64_bb, %rcx    # 变量 bb`
`    callq scanf`
	使用 scanf 函数，输入变量。

`    # 加法`
`    movq $input_op, %rdi`
`    movq $op_add, %rsi`
`    callq strcmp        # 比较运算符`
`    cmpq $0, %rax`
`    je mark_add`
	使用 strcmp 函数，比较运算符和加法字符串。
	如果返回值等于 0，说明字符串匹配，跳转到加法语句块。
	其他运算符，也是类似逻辑。

`mark_add :`
`    movq int64_aa(%rip), %r8    # 变量 aa`
`    movq int64_bb(%rip), %r9    # 变量 bb`
`    addq %r9, %r8               # r8 = r8 + r9`
`    movq %r8, int64_xx(%rip)    # 保存结果`
`    jmp mark_out`
	加法语句块。
	把 2 个整数相加，结果写到变量 int64_xx。
	跳转到输出语句块。
	其他运算符，也是类似逻辑。