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

指令说明

jmp 指令表示跳转。
通用跳转：jmp 无条件跳转，je 相等就跳转，jne 不相等就跳转。
有符号整数跳转：jg 大于就跳转，jge 大于等于就跳转，jl 小于就跳转，jle 小于等于就跳转等。
无符号整数跳转：ja 大于就跳转，jae 大于等于就跳转，jb 小于就跳转，jbe 小于等于就跳转等。
浮点数跳转：ja 大于就跳转，jae 大于等于就跳转，jb 小于就跳转，jbe 小于等于就跳转等。

g 表示 greater。
l 表示 less。
e 表示 equal。
a 表示 above。
b 表示 below。
n 表示 not。
jg 可以理解为 `if greater then jmp` 。
jge 可以理解为 `if greater or equal then jmp` 。
jne 可以理解为 `if not equal then jmp` 。

用跳转指令实现乱序输出

编写代码：reorder_jmp.s

```
.data

int32_aa :   # 记录入参
    .long 0x0

str_tip :    # 输入 1 个数字。 0 表示 false，其他表示 true
    .string "Please input num : 0 = false , other = true  \n"

str_input :  # 输入 1 个数字。
    .string "%d"

str_step :   # 打印顺序
    .string " step = %d \n"

.text
.global main

main :
    pushq %rbp
    movq %rsp, %rbp

    # 提示
    movq $str_tip, %rdi
```

```
    callq printf

    # 输入
    movq $str_input, %rdi
    movq $int32_aa, %rsi
    callq scanf

# ------------------------------------
    cmpl $0x0, int32_aa(%rip) # 比较
    jne mark_step_3     # 不相等就跳转 3

mark_step_1 :           # step 1
    movl $1, %esi
    movq $str_step, %rdi
    callq printf
    cmpl $0x0, int32_aa(%rip)  # 比较
    jne mark_step_4     # 不相等就跳转 4

mark_step_2 :           # step 2
    movl $2, %esi
    movq $str_step, %rdi
    callq printf
    cmpl $0x0, int32_aa(%rip)  # 比较
    jne mark_step_out   # 不相等就跳转 out

mark_step_3 :           # step 3
    movl $3, %esi
    movq $str_step, %rdi
    callq printf
    cmpl $0x0, int32_aa(%rip)  # 比较
    jne mark_step_1     # 不相等就跳转 1

mark_step_4 :           # step 4
    movl $4, %esi
    movq $str_step, %rdi
    callq printf
    cmpl $0x0, int32_aa(%rip)  # 比较
    jne mark_step_2     # 不相等就跳转 2

mark_step_out :         # step out

    popq %rbp
    retq
```

编译代码：

```
gcc reorder_jmp.s -o reorder_jmp
```

运行代码：

```
[root@local jmp]# ./reorder_jmp
Please input num : 0 = false , other = true
0
 step = 1
 step = 2
 step = 3
 step = 4

[root@local jmp]# ./reorder_jmp
Please input num : 0 = false , other = true
3
 step = 3
 step = 1
 step = 4
 step = 2
```

分析结果：
规则为，输入 0，按照默认顺序输出；输入其他数，使用跳转乱序输出。
使用函数 scanf，输入 1 个数，写到变量 int32_aa。
比较变量 int32_aa 和 0 的大小 `cmpl $0x0, int32_aa(%rip)` 。
如果 int32_aa 与 0 不相等，执行跳转，比如 `jne mark_step_2` 。
如果 int32_aa 与 0 相等，不执行跳转，顺序执行指令。

输入 0，输出顺序为 1、2、3、4。
输入 3，输出顺序为 3、1、4、2。

用跳转指令实现循环累加

编写代码： loop_jmp.s

```
.data

int64_aa :
    .quad 0x0

str_tip :  # 从 0 累加到输入的数
    .string "Sum from 0 to input number \n"

str_tip2 :
    .string "Please input num that is bigger than 0 \n"

str_input :  # 输入 1 个数
    .string "%lld"

str_sum :
    .string "Sum = %lld \n\n"
```

```
.text
.global main

main :
    pushq %rbp
    movq %rsp , %rbp

    # 提示
    movq $str_tip, %rdi
    callq printf
    movq $str_tip2, %rdi
    callq printf

    # 输入
    movq $str_input, %rdi
    movq $int64_aa, %rsi
    callq scanf

    # 把 1 到 aa 的数，累加

    movq $0, %r8    # 下标，依次递增
    movq $0, %r9    # 累加的结果

# ------------------------------------

tmp_loop :              # 循环
    cmpq int64_aa(%rip), %r8  # 判断边界
    jg  tmp_loop_out    # 大于就跳转，退出循环

    addq %r8, %r9       # 累加 1 个数
    incq %r8            # 下标+1
    jmp tmp_loop        # 无条件跳转，继续循环

# ------------------------------------

tmp_loop_out :          # 退出循环

    movq $str_sum, %rdi
    movq %r9, %rsi      # 打印结果
    callq printf

    popq %rbp
    retq
```

编译代码：

```
gcc loop_jmp.s -o loop_jmp
```

运行代码：

```
[root@local jmp]# ./loop_jmp
Sum from 0 to input number
Please input num that is bigger than 0
3
Sum = 6

[root@local jmp]# ./loop_jmp
Sum from 0 to input number
Please input num that is bigger than 0
100
Sum = 5050

[root@local jmp]# ./loop_jmp
Sum from 0 to input number
Please input num that is bigger than 0
8000
Sum = 32004000
```

分析结果：
规则为，输入 1 个数，从 0 一直累加到该数。
使用函数 scanf，输入 1 个数，写到变量 int64_aa。
设置下标 r8，起始为 0。设置结果 r9，起始为 0。
开启 loop 循环 `tmp_loop :` 。
比较变量 int64_aa 和下标的大小 `cmpq int64_aa(%rip), %r8` 。
如果下标超过边界，就退出循环 `jg  tmp_loop_out` 。
如果下标没有超过边界，就累加下标到结果 `addq %r8, %r9` ，下标加 1 `incq %r8` ，然后继续循环 `jmp tmp_loop` 。

输入 3，输出为 6，累加过程 0+1+2+3=6。
输入 100，输出为 5050，累加过程 0+1+2+3+...+100=5050。
输入 8000，输出为 32004000，累加过程 0+1+2+3+...+8000=32004000。

用跳转指令实现规则匹配

编写代码：multi_jmp.s

```
.data

int64_aa :
    .quad 0x0

int64_bb :
    .quad 0x0

int64_cc :
    .quad 0x0

str_tip :   # 校验 3 个数是否满足规则
    .string "Check rule :    ( aa > 100 && bb > 100 ) || cc > 100  \n"
```

```
str_tip2 :  # 输入 3 个数字
    .string "Please input like : 11 300 22 \n"

str_input : # 输入 3 个数字
    .string "%lld %lld %lld"

str_match_yes :
    .string "Match = true \n\n"

str_match_no :
    .string "Match = false \n\n"

.text
.global main

main :
    pushq %rbp
    movq %rsp, %rbp

    # 提示
    movq $str_tip, %rdi
    callq printf
    movq $str_tip2, %rdi
    callq printf

    # 输入
    movq $str_input, %rdi
    movq $int64_aa, %rsi
    movq $int64_bb, %rdx
    movq $int64_cc, %rcx
    callq scanf

    # 判断规则   ( aa > 100 && bb > 100 ) || cc > 100

    cmpq $100, int64_aa(%rip)   # 判断 aa > 100
    jle tmp_next                # 小于等于，就跳转
    cmpq $100, int64_bb(%rip)   # 判断 bb > 100
    jle tmp_next                # 小于等于，就跳转
    jmp tmp_match_yes           # 无条件跳转。满足规则

tmp_next :
    cmpq $100, int64_cc(%rip)   # 判断 cc > 100
    jg tmp_match_yes            # 大于就跳转。满足规则
    jmp tmp_match_no            # 无条件跳转。不满足规则

tmp_match_yes : # 满足规则
    movq $str_match_yes, %rdi
    callq printf
```

```
    jmp tmp_out

tmp_match_no : # 不满足规则
    movq $str_match_no, %rdi
    callq printf

tmp_out :   # 退出

    popq %rbp
    retq
```

编译代码：

```
gcc multi_jmp.s -o multi_jmp
```

运行代码：

```
[root@local jmp]# ./multi_jmp
Check rule :    ( aa > 100 && bb > 100 ) || cc > 100
Please input like : 11 300 22
200 300 55
Match = true

[root@local jmp]# ./multi_jmp
Check rule :    ( aa > 100 && bb > 100 ) || cc > 100
Please input like : 11 300 22
1 2 555
Match = true

[root@local jmp]# ./multi_jmp
Check rule :    ( aa > 100 && bb > 100 ) || cc > 100
Please input like : 11 300 22
200 3 5
Match = false
```

分析结果：

规则为，输入 3 个数，匹配 `( aa > 100 && bb > 100 ) || cc > 100` 。

使用函数 scanf，输入 3 个数，写到变量 int64_aa、int64_bb、int64_cc。

比较 int64_aa 和 100 的大小 `cmpq $100, int64_aa(%rip)` ，如果 int64_aa 小于等于 100 就跳转 `jle tmp_next` ，否则继续执行。

比较 int64_bb 和 100 的大小 `cmpq $100, int64_bb(%rip)` ，如果 int64_bb 小于等于 100 就跳转 `jle tmp_next` ，否则匹配成功 `jmp tmp_match_yes` 。

比较 int64_cc 和 100 的大小 `cmpq $100, int64_cc(%rip)` ，如果 int64_cc 大于 100 就匹配成功 `jg tmp_match_yes` ，否则匹配失败 `jmp tmp_match_no` 。

输入 200 300 55，输出匹配成功。

输入 1 2 555，输出匹配成功。

输入 200 3 5，输出匹配失败。