

# Assembly Language

SiREN

## 1 Jump and Call

### 1.1 Flag 标志位

在汇编语言和低级计算机架构中，标志位是处理器状态寄存器（标志寄存器）中的单个位，用来指示上一条指令执行的某些结果。Flag Z 和 Flag C 是两个特别重要的标志位：

1. Flag Z (零标志位) (Zero Flag), 当算术或逻辑操作的结果为零时，ZF 会被设置为 1；如果结果非零，ZF 会被清除为 0。
  - JZ d (Jump if Zero) : 如果 ZF=1，表示上一个操作的结果为零，则跳转到标签或地址 d。
  - JNZ d (Jump if Not Zero) : 如果 ZF=0，表示上一个操作的结果非零，则跳转到标签或地址 d。
2. Flag C (进位标志位) (Carry Flag), 在加法运算中，如果最高位产生了进位，则 CF 设置为 1；在减法运算中，如果发生了借位，则同样设置 CF 为 1。如果这些情况没有发生，CF 会被清除为 0。
  - JC d (Jump if Carry) : 如果 CF=1，表示有进位发生，则跳转到标签或地址 d。
  - JNC d (Jump if Not Carry) : 如果 CF=0，表示没有进位发生，则跳转到标签或地址 d。

### 1.2 Jump Command 跳转指令

Command	Action	Flag Condition
JMP d	IP = d	-
JZ d	If Z=1 $\rightarrow$ IP = d	Flag Z
JNZ d	If Z=0 $\rightarrow$ IP = d	Flag Z
JC d	If C=1 $\rightarrow$ IP = d	Flag C
JNC d	If C=0 $\rightarrow$ IP = d	Flag C
CALL d	[++SP] := IP, IP=d	-
RET	IP = [SP- -]	-
LOOP d	CX- -, If NZ $\rightarrow$ IP = d	Flag Z

- JMP d

动作: 无条件跳转到地址 d，标志位: 不适用。

- JZ d

动作: 如果零标志位 (Z) 为 1，则跳转到地址 d，标志位: Z。

- JNZ d

动作: 如果零标志位 (Z) 为 0，则跳转到地址 d，标志位: Z。

- JC d

动作: 如果进位标志位 (C) 为 1, 则跳转到地址 d, 标志位: C。

- JNC d

动作: 如果进位标志位 (C) 为 0, 则跳转到地址 d, 标志位: C。

d 是指令的参数, 但具有不同的含义。它可以是一个地址或者一个标签。

跳转行号是一个十六进制数 (例如, 100)。

标签是以 ? 和一个数字开头, 如果有的话 (例如, ?01)。

汇编命令的一般语法是 <optional label>:<command> <arguments>, 其中标签后面跟着一个冒号。

## 1.3 Test

### 1. Zero Comparison (Example)

Run a program that compares BX and CX and sets AX=1 when BX = CX

```
1 CMP BX, CX      ; 比较寄存器BX和CX的值
2 JNZ ?1          ; 如果最后一次比较的结果不为零 (即BX和CX不相等), 则跳转到标签?1
3 MOV AX, 1       ; 将数值1移动到寄存器AX中
4 ?1:            ; 标签?1的位置
5 END             ; 汇编语言程序结束
```

### 2. Comparing two numbers

Write a program that puts 1 in register AX if  $BX \geq CX$  and 0 otherwise.

```
1 CMP BX, CX      ; 比较寄存器BX和CX的值
2 JC ?01          ; 如果上一次的比较结果产生了进位 (即BX小于CX), 则跳转到标签?01
3 MOV AX, 1       ; 将数值1移动到寄存器AX中
4 JMP ?02         ; 无条件跳转到标签?02
5 ?01:           ; 标签?01的位置
6 MOV AX, 0       ; 将数值0移动到寄存器AX中
7 ?02:           ;
8 END             ; 汇编语言程序结束
```

### 3. Absolute value

Write a program to put the absolute value of the difference between the numbers in BX and CX into register AX.

```
1 CMP BX, CX      ; 比较寄存器BX和CX的值
2 JNC ?01         ; 如果比较结果没有产生进位 (即BX大于等于CX), 则跳转到标签?01
3 SUB CX, BX      ; 从CX中减去BX的值, 结果存回CX
4 MOV AX, CX      ; 将寄存器CX的值移动到AX中
5 JMP ?02         ; 无条件跳转到标签?02
6 ?01:           ; 标签?01的位置
7 SUB BX, CX      ; 从BX中减去CX的值, 结果存回BX
8 MOV AX, BX      ; 将寄存器BX的值移动到AX中
9 ?02:           ;
10 END            ; 汇编语言程序结束
```

#### 4. Minimum

Write a program that puts the minimum number of numbers from BX and CX into AX.

```
1  CMP BX, CX      ; 比较寄存器BX和CX的值
2  JC  ?01         ; 如果BX小于CX（即比较结果有进位），则跳转到标签?01
3  MOV AX, CX      ; 将CX的值赋给AX（如果没有跳转发生，即BX大于等于CX时执行）
4  JMP ?02         ; 无条件跳转到标签?02
5  ?01:           ; 标签?01的位置
6  MOV AX, BX      ; 将BX的值赋给AX（如果发生了跳转，即BX小于CX时执行）
7  ?02:           ;
8  END            ; 汇编语言程序结束
```

#### 5. Divisible by 3

Write a program that puts 1 in register AX if BX is evenly divisible by 3, and 0 otherwise.

```
1  MOV AX, BX      ; 将BX的值赋给AX
2  MOV CX, 3       ; 将数值3赋给CX
3  JZ  ?03         ; 如果之前的操作结果使零标志位(ZF)被设置（即AX为0），则跳转到标签?03
4
5  ?01:           ; 标签?01的位置，循环开始的地方
6  SUB AX, 0       ; AX减0，这条指令实际上没有改变AX的值，但会影响标志位
7  JZ  ?03         ; 如果AX为0（即SUB操作结果为0），则跳转到标签?03
8  CMP AX, CX      ; 比较AX和CX的值
9  JC  ?02         ; 如果AX小于CX（即比较结果有进位），则跳转到标签?02
10 JZ  ?03         ; 如果AX等于CX（即比较结果为0），则跳转到标签?03
11 SUB AX, CX      ; 从AX中减去CX的值，结果存回AX
12 JMP ?01         ; 无条件跳转回标签?01，继续循环
13
14 ?02:           ; 标签?02的位置
15 MOV AX, 0       ; 将0赋给AX
16 JMP ?04         ; 无条件跳转到标签?04
17
18 ?03:           ; 标签?03的位置
19 MOV AX, 1       ; 将1赋给AX
20
21 ?04:           ; 标签?04的位置，循环结束后的操作或处理
22
23 END            ; 汇编语言程序结束
```

#### 6. Divisible by 2

Write a program that puts 1 in register AX if BX is evenly divisible by 2, and 0 otherwise. (Do not use all other registers, except AX and BX or any Jump command)!

```
1  MOV AX, 1       ; 首先，假设结果为1并放入AX
2  AND BX, 1       ; 将BX和1进行AND操作，结果在BX中。如果BX是偶数，BX变为0；如果是奇数，BX变为1。
3  SUB AX, BX      ; 从AX中减去BX的结果。如果BX为0（即偶数），AX保持为1；如果BX为1（即奇数），AX变为0。
```

## 7. Bit number

Bits in a 16-bit register are numbered from 1 (least significant bit) to 16. Write a program that finds the least significant non-zero bit number of register BX and puts the answer into register AX. If BX=0, then AX=0. Use the RCR shift operation.

Example:  $BX=0006_{16}(0000000000000110_2)$ ,  $AX=2$ .

```

1 MOV CX, 0           ; 将 CX 寄存器清零，准备计数
2 MOV AX, BX          ; 将 BX 寄存器的值复制到 AX 寄存器
3
4 OR AX, AX           ; 使用或操作检查 AX 的值是否为 0
5 JC ?01              ; 如果结果为零（即 BX 为 0），跳转到标签 ?01
6 MOV CX, 1           ; 初始化 CX 为 1，因为至少有一个非零位
7
8 ?02:
9 RCR BX, 1           ; 右循环移位 BX，考虑进位
10 JC ?03              ; 如果最后一个移出的位是 1（即发现非零位），跳转到 ?03
11 INC CX              ; 如果没有跳转，则增加 CX 的值，继续查找非零位
12 JMP ?02             ; 无条件跳回 ?02，继续循环
13
14 ?03:
15 MOV AX, CX          ; 将找到的非零位的位置（计数）移至 AX
16 JMP ?04             ; 跳转到结束标签 ?04，实际上这一步可能是多余的
17
18 ?01:
19 MOV AX, 0           ; 如果 BX 为 0，设置 AX 也为 0
20
21 ?04:
22 END                 ; 程序结束

```

- **反码 (Ones' Complement)** 反码是一种表示负数的方式。对于一个二进制数，其反码是将所有的 0 变为 1，所有的 1 变为 0。例如，数 5 在 8 位二进制中表示为 00000101，其反码是 11111010。对于正数，反码与原码相同。对于负数，反码是通过取其正数的二进制表示，然后对每一位进行反转 (0 变 1, 1 变 0) 得到的。
- **补码**也是一种表示负数的方法，是当前计算机系统中最常用的方法。一个数的补码是其反码加 1。例如，数 -5 的补码在 8 位二进制中是 11111011。这是因为 5 的二进制表示为 00000101，反码是 11111010，加 1 后得到补码 11111011。在现代计算机系统中，负数通常使用补码 (Twos' Complement) 表示
- 补码应用到减法中时，减法通常是通过加上一个数的负数 (即补码) 来实现的。例如，计算  $5 - 3$  可以转换为  $5 + (-3)$ 。在二进制中，5 表示为 00000101，3 表示为 00000011，-3 的补码是 11111101 (因为 3 的二进制表示为 00000011，反码是 11111100，加 1 后得到补码 11111101)

$$0 \quad 00000101 \quad (1)$$

$$+ \quad 11111101 \quad (2)$$

$$- \quad - \quad - \quad - \quad - \quad - \quad (3)$$

$$0 \quad 00000010 \quad (4)$$

由于我们通常使用固定位数（如 8 位），最左边的进位被丢弃，结果就是 00000010，即二进制的 2，这是正确的结果。

## 8. Number of different numbers

Write a program that puts in register AX the number of different numbers in BX, CX, DX.

```
1 MOV AX, 3          ; 将 AX 寄存器的值设置为 3, 假设 BX, CX, DX 中有 3 个不同的数
2 CMP BX, CX         ; 比较 BX 和 CX 寄存器的值
3 JNZ ?01            ; 如果 BX 和 CX 不相等, 则跳转到标签 ?01
4 DEC AX             ; 如果 BX 和 CX 相等, 则将 AX 寄存器的值减 1
5
6 ?01:
7 CMP BX, DX         ; 比较 BX 和 DX 寄存器的值
8 JNZ ?02            ; 如果 BX 和 DX 不相等, 则跳转到标签 ?02
9 DEC AX             ; 如果 BX 和 DX 相等, 则将 AX 寄存器的值减 1
10
11 ?02:
12 CMP CX, DX        ; 比较 CX 和 DX 寄存器的值
13 JNZ ?03            ; 如果 CX 和 DX 不相等, 则跳转到标签 ?03
14 DEC AX             ; 如果 CX 和 DX 相等, 则将 AX 寄存器的值减 1
15
16 ?03:
17 CMP AX, 1          ; 比较 AX 寄存器的值与 1
18 JNC ?04            ; 如果 AX 大于或等于 1, 则跳转到标签 ?04
19 MOV AX, 1          ; 如果 AX 小于 1, 则将 AX 寄存器的值设置为 1
20
21 ?04:
22 END               ; 程序结束
```

## 9. Number of maximum

Write a program that counts the maximums in the sequence BX, CX, DX and places the result in the AX register. For example, in the sequence of numbers 1,2,3 - the maximum is unique and equals 3 (BX=1, CX=2, DX=3 → AX=1) In the sequence of numbers 3,1,3 - the maximum is 3 and there are 2 such numbers in the sequence (BX=3, CX=1, DX=3 → AX=2).

```
1 MOV AX, 1          ; 初始化 AX 为 1, 假设 BX 是最大的
2
3 ; 比较 BX 和 CX
4 ?00:
5 CMP BX, CX
6 JC ?01             ; 如果 BX < CX, 则需要更新
7 JZ ?02             ; 如果 BX = CX, 则 AX 需要增加, 并跳过到 DX 的比较
8 ; 如果 BX > CX, 继续比较 BX 和 DX
9
10 ?03:
11 CMP BX, DX
12 JC ?04             ; 如果 BX < DX, 则需要更新
13 JZ ?05             ; 如果 BX = DX, 则 AX 需要增加
14 ; 如果 BX > DX 或 BX = DX, 完成比较
15 JMP ?06
16
17 ?01:
```

```

18 MOV BX, CX          ; 将 BX 更新为 CX
19 MOV AX, 1           ; 重置 AX 为 1，因为我们找到了一个新的最大值
20 JMP ?03             ; 跳转去比较更新后的 BX（现在是 CX 的值）和 DX
21
22 ?02:
23 INC AX              ; 由于 BX = CX，增加 AX 的值
24 JMP ?03             ; 现在跳到 BX 与 DX 的比较
25
26 ?04:
27 MOV BX, DX          ; 将 BX 更新为 DX
28 MOV AX, 1           ; 重置 AX 为 1
29 JMP ?06             ; 完成，直接跳到结束
30
31 ?05:
32 INC AX              ; 由于 BX = DX 或 CX = DX，增加 AX 的值
33 JMP ?06             ; 完成，跳到结束
34
35 ?06:
36 ; 此时 AX 包含最大值的出现次数

```