

# Assembly Language

SiREN

## 1 Intro. Registers

### 1. 0 (Example)

```
1 MOV AX,0 ; 将0移动至寄存器AX, 即赋值  
2 END
```

### 2. 2000 Write a program that puts 2000 in the AX register.

```
1 MOV AX, 2000  
2 END
```

### 3. Data Write a program that puts data from BX in the AX register.

```
1 MOV AX, BX  
2 END
```

### 4. Sum (Example) Run the program that puts the sum of data from BX and CX into the AX register.

```
1 MOV AX, BX  
2 ADD AX, CX  
3 END
```

### 5. Subtract Write a program that places the result of subtracting data CX from BX into register AX.

```
1 SUB BX, CX  
2 MOV AX, BX
```

### 6. Calculate Write a program that places the result of operation into register AX:

Let's AX=0

Let's BX=FF00

Let's CX=0F

Let's DX=F1

Let's CX=CX AND DX

Let's CX=CX OR 80

Let's AX=BX OR C0

Let's AX=AX+CX

Let's AX=-AX (NEG)

Remark: Do not save registers before operations (registers values matter)!

```

1 MOV AX, 0
2 MOV BX, FF00
3 MOV CX, F
4 MOV DX, F1
5
6 AND CX, DX
7 OR CX, 80
8
9 MOV AX, BX
10 OR AX, C0
11 ADD AX, CX
12 NEG AX
13
14 END

```

7. Swap Write a program that swaps two numbers in registers BX and CX.

```

1 MOV AX, BX
2 MOV BX, CX
3 MOV CX, AX
4
5 END

```

8. Swap Return Write a program that swaps two numbers in registers BX and CX (Not use Push/Pop or other registers)!

```

1 ADD BX, CX
2 SUB CX, BX
3 NEG CX
4 SUB BX, CX
5
6 END

```

9. NOT Bitwise Unary NOT ( ) performs complementation or negation operation; inverts all the bits of the number, i.e. 0→1 and 1→0.

Write a program that places the result of operation NOT into register AX. (Not use Push/Pop or other registers)!

```

1 NEG AX
2 DEC AX
3
4 END

```

10. Remember! Write a program that stores two numbers in registers BX and CX between operations.

```

1 PUSH BX
2 PUSH CX
3
4 MOV DX, 0
5 MOV CX, 0
6 MOV BX, 0

```

```

7 MOV AX,0
8
9 POP CX
10 POP BX
11
12 END

```

11. XOR Exclusive or or exclusive disjunction is a logical operation that is true if and only if its arguments differ (one is 1, the other is 0). Exclusive disjunction is often used for bitwise operations. Examples:

1 XOR 1 = 0

1 XOR 0 = 1

0 XOR 1 = 1

0 XOR 0 = 0

Write a program that places the result of operation AX XOR BX into register AX. (Not use Push/Pop or all other registers, you can use only one extra register - CX)!

```

1 ; A XOR B = (A OR B) AND NOT(A AND B)
2 MOV CX, AX
3 OR AX, BX
4 AND BX, CX
5 NEG BX
6 DEC BX
7 AND AX, BX
8
9 END

```

12. Left Shift (Example) Run the program that multiply by 2 a number in the AX register.

```

1 CLC ; Clear Carry Flag, 清除进位标志 (Carry Flag)
2 RCL AX
3 END

```

13. Right Shift Write a program that divides by 2 the number in register CX and puts the result in register AX.

Do Not forget CLC before any RCR!

```

1 CLC
2 RCR CX
3 MOV AX, CX
4
5 END

```

14. MultiShift Write a program that puts the value of AX multiplied by 4 into register BX and the value of AX divided by 4 into register CX.

Use the shift operations RCL and RCR.

Do Not forget CLC before any RCL and RCR!

```

1 MOV BX, AX
2 MOV CX, AX
3
4 CLC
5 RCL BX
6 CLC
7 RCL BX
8
9 CLC
10 RCR CX
11 CLC
12 RCR CX
13
14 END

```

15. Sum Write a program that puts into register AX the sum of two 8-bit numbers stored in 16-bit register BX.

Example:  $BX = 032E_{16}$ , result  $AX = 03_{16} + 2E_{16} = 31_{16}$ .

If  $BX=0$ , then  $AX=0$ .

Use the shift operations RCL or RCR.

Do Not forget CLC before any RCL and RCR!

```

1 MOV AX, BX      ; 将BX寄存器的值复制到AX寄存器
2 AND AX, 00FFh   ; 将AX寄存器的高8位清零，仅保留低8位
3 MOV CX, BX      ; 将BX寄存器的值复制到CX寄存器
4
5 ; 下面的一系列操作是将CX寄存器中的内容右移8位
6 ; 这是通过8次单位的右循环移位（RCR）完成的，每次移动1位
7 ; 在每次RCR操作前，使用CLC清除进位标志以确保正确的右移行为
8
9 CLC             ; 清除进位标志
10 RCR CX, 1      ; 通过进位标志将CX寄存器右旋转1位
11 CLC           ; 清除进位标志
12 RCR CX, 1      ; 再次将CX寄存器右旋转1位
13 CLC           ; 清除进位标志
14 RCR CX, 1      ; 将CX寄存器右旋转1位
15 CLC           ; 清除进位标志
16 RCR CX, 1      ; 将CX寄存器右旋转1位
17 CLC           ; 清除进位标志
18 RCR CX, 1      ; 将CX寄存器右旋转1位
19 CLC           ; 清除进位标志
20 RCR CX, 1      ; 将CX寄存器右旋转1位
21 CLC           ; 清除进位标志
22 RCR CX, 1      ; 将CX寄存器右旋转1位
23 CLC           ; 清除进位标志
24 RCR CX, 1      ; 将CX寄存器右旋转1位
25
26 ADD AX, CX      ; 将AX和CX寄存器的值相加，结果存储在AX寄存器中
27

```

## 2 Jump and Call

### 2.1 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。

### 2.2 Jump Command 跳转指令

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

## 2.3 Computer Command 计算指令

Command	Action	Flag Z	Flag C
DEC d      减 1	d = d - 1	+	
INC d      加 1	d = d + 1	+	
ADD d,s    按位加法	d = d + s	+	+
SUB d,s    按位减法	d = d - s	+	+

## 2.4 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 ≥ 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)

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

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

$$\text{---} \quad \text{---} \quad \text{---} \quad \text{---} \quad \text{---} \quad \text{---} \quad (3)$$

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

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

```

## 10. Sides of a triangle

Can non-negative integers in BX, CX, DX be sides of a triangle? The result will be AX=1 if YES and AX=0 if NOT.

```

1 ; BX和CX相加，检查它们的和是否大于DX
2 ADD BX, CX          ; 将BX和CX相加，结果存储在BX
3 MOV AX, BX          ; 将相加后的结果BX移动到AX，为后续比较做准备
4 SUB BX, CX          ; 从BX中减去CX，恢复BX的原始值
5 SUB AX, DX          ; 从AX中减去DX，为了检查BX + CX是否大于DX
6 JZ ?00              ; 如果结果是0（AX为0），跳转到?00，表示不满足三角形条件
7 JC ?00              ; 如果有借位（AX < DX），跳转到?00，表示不满足三角形条件
8 MOV AX, 1           ; 如果没有跳转，设置AX为1，表示目前检查满足三角形条件
9
10 ; BX和DX相加，检查它们的和是否大于CX
11 ADD BX, DX          ; 将BX和DX相加，结果存储在BX
12 MOV AX, BX          ; 将相加后的结果BX移动到AX，为后续比较做准备
13 SUB BX, DX          ; 从BX中减去DX，恢复BX的原始值
14 SUB AX, CX          ; 从AX中减去CX，为了检查BX + DX是否大于CX
15 JZ ?00              ; 如果结果是0（AX为0），跳转到?00，表示不满足三角形条件
16 JC ?00              ; 如果有借位（AX < CX），跳转到?00，表示不满足三角形条件
17 MOV AX, 1           ; 如果没有跳转，设置AX为1，表示目前检查满足三角形条件
18
19 ; CX和DX相加，检查它们的和是否大于BX
20 ADD CX, DX          ; 将CX和DX相加，结果存储在CX
21 MOV AX, CX          ; 将相加后的结果CX移动到AX，为后续比较做准备
22 SUB CX, DX          ; 从CX中减去DX，恢复CX的原始值
23 SUB AX, BX          ; 从AX中减去BX，为了检查CX + DX是否大于BX
24 JZ ?00              ; 如果结果是0（AX为0），跳转到?00，表示不满足三角形条件
25 JC ?00              ; 如果有借位（AX < BX），跳转到?00，表示不满足三角形条件
26 MOV AX, 1           ; 如果没有跳转，设置AX为1，表示目前检查满足三角形条件

```

```

27 JMP ?03          ; 无条件跳转到?03，结束检查流程
28
29 ?00:
30 MOV AX, 0         ; 设置AX为0，表示不能构成三角形
31
32 ?03:
33 END              ; 检查结束

```

## 11. Right triangle

Can non-negative integers in BX, CX, DX be sides of a right triangle? The result will be AX=1 if yes, and AX=0 if not.

Note. A triangle with sides 0,0,0 is not a right triangle.

```

1 ; 保存寄存器DX和CX的值到堆栈，以便后续恢复
2 PUSH DX
3 PUSH CX
4
5 ; 计算BX的平方，并保存到BX中
6 MOV AX, BX      ; 将BX的值复制到AX
7 MOV CX, BX      ; 将BX的值复制到CX，用于累加
8 DEC AX          ; AX自减1，准备循环
9
10 ?01:
11 ADD BX, CX      ; 累加计算BX的平方
12 DEC AX          ; 循环计数器自减
13 JZ ?02          ; 如果AX为0，跳转到?02
14 JMP ?01         ; 否则继续循环
15
16 ?02:
17 ; 恢复寄存器CX和DX的原始值
18 POP CX
19 POP DX
20
21 ; 计算CX的平方，并保存到CX中
22 PUSH BX        ; 保存BX的值到堆栈
23 PUSH DX        ; 保存DX的值到堆栈
24 MOV AX, CX      ; 将CX的值复制到AX
25 MOV BX, CX      ; 将CX的值复制到BX，用于累加
26 DEC AX          ; AX自减1，准备循环
27
28 ?03:
29 ADD CX, BX      ; 累加计算CX的平方
30 DEC AX          ; 循环计数器自减
31 JZ ?04          ; 如果AX为0，跳转到?04
32 JMP ?03         ; 否则继续循环
33
34 ?04:
35 ; 恢复寄存器DX和BX的原始值
36 POP DX
37 POP BX

```

```

38
39 ; 计算DX的平方，并保存到DX中
40 PUSH BX          ; 保存BX的值到堆栈
41 PUSH CX          ; 保存CX的值到堆栈
42 MOV AX, DX       ; 将DX的值复制到AX
43 MOV BX, DX       ; 将DX的值复制到BX，用于累加
44 DEC AX           ; AX自减1，准备循环
45
46 ?05:
47 ADD DX, BX       ; 累加计算DX的平方
48 DEC AX           ; 循环计数器自减
49 JZ ?06           ; 如果AX为0，跳转到?06
50 JMP ?05          ; 否则继续循环
51
52 ?06:
53 ; 恢复寄存器CX和BX的原始值
54 POP CX
55 POP BX
56
57 ; 初始化AX为0，用于存储最终的结果
58 MOV AX, 0
59
60 ; 检查是否有边长为0，如果有，则直接跳到?07
61 CMP BX, 0
62 CMP CX, 0
63 CMP DX, 0
64 JZ ?07
65
66 ; 检查是否可以构成三角形，如果三个边的平方和相等，则是直角三角形
67 ADD BX, CX
68 CMP BX, DX
69 JZ ?09           ; 如果满足 $BX^2 + CX^2 = DX^2$ ，设置AX为1
70 SUB BX, CX
71
72 ADD BX, DX
73 CMP BX, CX
74 JZ ?09           ; 如果满足 $BX^2 + DX^2 = CX^2$ ，设置AX为1
75 SUB BX, DX
76
77 ADD CX, DX
78 CMP CX, BX
79 JZ ?09           ; 如果满足 $CX^2 + DX^2 = BX^2$ ，设置AX为1
80 SUB CX, DX
81 JMP ?07
82
83 ?09:
84 MOV AX, 1        ; 如果是直角三角形，设置AX为1
85
86 ?07:

```

## 12. Long shift to the left

Run a program that doubles a 32-bit number from the BX AX registers (lower part of AX).

RCL shift operations are used.

```
1 CLC
2 RCL AX
3 RCL BX
4
5 END
```

## 13. Long shift to the right

Write a program that divides a 32-bit number from the BX AX registers (lower part of AX) by 4.

```
1 RCR BX
2 RCR AX
3
4 RCR BX
5 RCR AX
6
7 END
```

## 14. Adding two long

Write a program that adds a 32-bit long number stored in the CX, DX registers with a long number stored in the AX, BX registers.

```
1 ADD BX, DX      ; 将DX加到BX上，结果存回BX
2 JC  ?00         ; 如果上一条指令导致进位（即BX+DX的结果超出了16位能表示的范围），则跳转到
                  ; 标签?00
3 JMP ?01         ; 无条件跳转到标签?01，如果没有因为JC跳转，则此条指令会被执行，绕过INC AX
4 ?00:
5 INC AX          ; 如果执行了JC跳转，意味着有进位，增加AX的值
6 ?01:
7 ADD AX, CX      ; 将CX加到AX上，结果存回AX
8 END
```

## 15. Bus for programmers

Bus tickets for programmers have a hexadecimal number. We call a ticket lucky if the sum of the two left hexadecimal digits is equal to the sum of the two right digits, for example ADE9. Example of an unsuccessful ticket:  $7+8 \neq 9+3$ . How many lucky ticket numbers are found from number BX to number CX, including borders?

---

AX    BX    CX    DX    [1]

BX, CX是不能动的  
BX要++控制遍历主体, CX是循环终点

