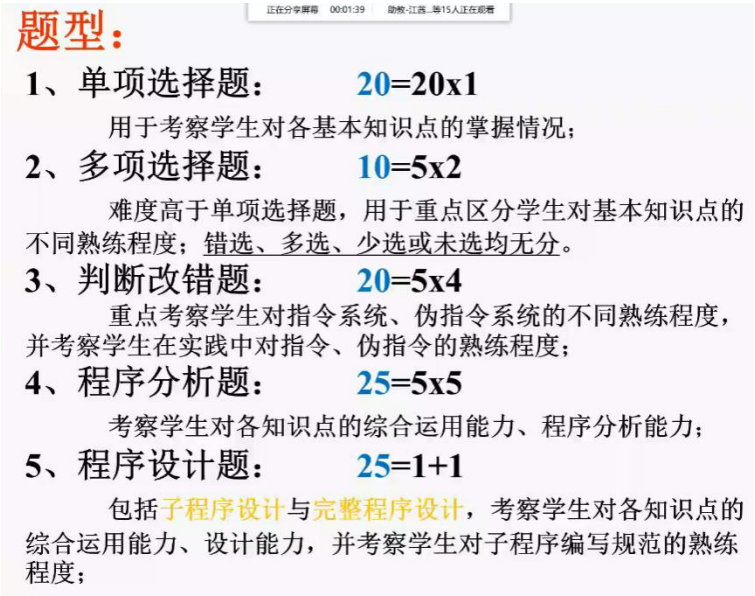
补码的加减运算结果是否具有正确的符号位与补码运算是否溢出这两个命题是等价的

1）“正数补码+负数补码”：这种情况是正数与负数抵消的过程，不会出现溢出。

2）“正数补码+正数补码”：若运算结果的符号为正，则没有溢出，反之，有溢出。

3）“负数补码+负数补码”：若运算结果的符号为负，则没有溢出，反之，有溢出。



子程序设计 完整程序设计（分段。系统调用，第一个返回 dos，字符字符串的输入输出）

一．逻辑地址转化为物理地址（段基值左移四位转换为 20 位的段基址，段基址仅仅是一个段的起始位置，再加偏移量。）

段：一个段是指位于内部存储器中的一块连续存储空间，它由物理地址连续的多个字节单元构成。

逻辑地址：由两个 16 位的地址分量构成，一个段基值，另一个偏移量，都是无符号数编码

段基址=段基值*16,（左移四位），物理地址=段基址+偏移量

二．CPU 寄存器组

数据寄存器组（AX，BX，CX，DX）

段寄存器组（4 个 16 位段寄存器，CS，DS，SS，ES，用于保存 4 个段的段基值，为生成内存单元的物理地址提供必要条件）

地址指针寄存器

地址指针寄存器	与段寄存器的搭配关系
BX	默认与 DS 搭配，但在程序设计时可以修改，可与任意段寄存器搭配
SP	只能与 SS 搭配
BP	默认与 SS 搭配，但在程序设计时可以修改，可与任意段寄存器搭配
SI	默认与 DS 搭配，但在程序设计时可以修改，可与任意段寄存器搭配
DI	一般指令中默认与 DS 搭配，但在程序设计时可以修改，可与任意段寄存器搭配；在串操作指令中只能与 ES 搭配

图 4.2.2 地址指针寄存器与段寄存器的搭配关系

数据寄存器与地址指针寄存器并称为通用寄存器。

控制寄存器*****

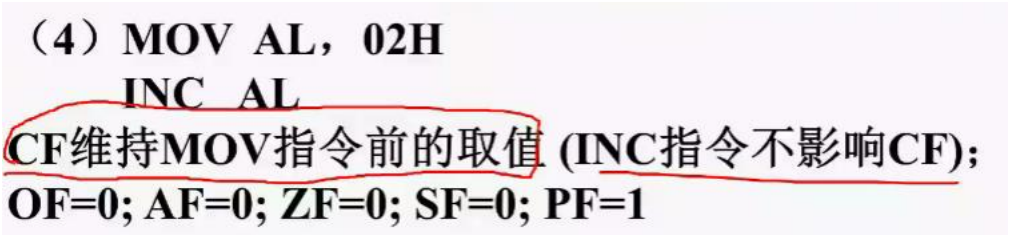
控制寄存器是指能够直接或者间接控制程序执行流程的寄存器

指令指针寄存器 IP，保存 CUP 下一条将要从内存中读取指令在当前代码段（CS）中的首字节偏移量（IP 固定与 CS 搭配使用），当执行某条机器指令时，IP 中的偏移量已经指向下一条指令。

标志寄存器 FR。

含义和影响指令。C，P，A，Z，S，O，六个状态标志位。

进位标志位 CF，最高位产生进位或者借位，CF 置 1，否则清 0.可以用来判断无符号数编码加减是否溢出，实现高低字或者字节间运算的衔接，从而实现长操作数运算，突破机器字长的限制



奇偶标志为 PF，如果运算结果**低八位**中包含偶数个“1”的数据为，PF 置 1，奇数个“1”PF 清 0，用于编码的奇偶校验。

辅助进位标志位 AF，执行加减算术运算时，如果第 3 位（最低位为 0）产生进位或者借位，AF 置 1，否则清 0，BCD 加法有用到

零值标志位 ZF，如果运算结果为 0，ZF 置 1，否则清 0。比较两个编码是否相等，判断计数是否为 0、判断存储单元指定位的取值等

符号标志位 SF，SF 值与运算结果的最高位保持一致，操作数解释为补码时，SF 可以看做运算结果的符号位，未解释为补码，则 SF 无意义，补码运算溢出，符号位会出现错误，SF 是运算结果符号位的直接反应，也会出现错误。SF 与 OF 标志配合使用，判断两个补码间的大小关系。

溢出标志位 OF，按照补码解释的运算出现溢出，OF 置 1，没有溢出，清 0。

三个控制标志

单步跟踪标志位 TF

中断使能标志位 IF，
方向标志位 DF

(20 道单选一般与标志位相关和程序分析题)
算术类指令和位操作指令对标志位的影响
寄存器的隐含使用和特定使用
SP 寄存器　　PUSH，POP，CALL，RET 等

基本指令系统
寻址方式（不会特意考查，编程时会使用）指令获取操作数的方式
寄存器寻址：操作数在寄存器中，则指令中以寄存器名称（地址）的形式给出。
立即数寻址（仅能用于源操作数寻址），操作数包含在机器指令内，以立即数字段的方式给出
存储器寻址方式：操作数位于内存单元中
1．直接寻址方式，内存操作数的偏移量为机器指令中的位移量字段
2.寄存器间接寻址方式（BX，SI，DI 默认与 DS 搭配，BP 默认与 SS 搭配）内存操作数的偏移量由地址指令寄存器 BX，BP，SI，DI 其一给出
3.基址寻址与变址寻址，内存操作数的偏移量由**基址分量**（BX）（BP）或者**变址分量**（DI）（SI）与位移量分量相加得到
4.基址变址寻址，内存操作数的偏移量由**基址分量、变址分量、位移量分量**三种分量**相加**得到。
源操作数是指提供给指令作为原始数据的操作数
目的操作数是指操作完成后的结果数据
偏移量也称作有效地址 EA
有效地址：内存单元中逻辑地址的偏移量，其含义为内存单元（起始字节单元）与段基址间的字节距离
基本指令系统
单操作数指令中，操作数不能使用立即数，只能使用寄存器、内存单元
双操作数指令中仅允许最多一个操作数为内存单元或端口，不能两个操作数同为内存单元或同为端口。
传送类指令　10 条
MOV 指令　若要实现两个内存单元的数据传送，则需要使用两条 MOV 指令，并通过通用寄存器。**MOV 指令也不能直接在两个段寄存器之间传送数据，也不能对段寄存器直接传送立即数，需要通过通用寄存器中转。**
XCHG 指令（交换）不能使用立即数也不能使用段寄存器。
LAHF（取标志位指令）将标志寄存器低八位保存到 AH 中
SAHF（存标志位指令）将（AH）传送到标志寄存器低 8 位保存,**影响标志寄存器**的低 8 位，SF，AF,PF,CF,ZF
PUSH 指令　先将（SP）减 2，使之指向一个空的栈顶，再将源操作数传送至栈顶保存。源操作数必须是 16 位的寄存器或内存单元，不能是 8 位存储单元或者立即数
POP 指令 先将栈顶字单元中的数据传送至目的操作数保存，再将（SP）加 2。
PUSHF　先将（SP）减 2，使之指向一个空的栈顶，然后将 16 位的标志寄存器中的数据传送至栈顶保存

POPF 先将（SP）指示的栈顶字内存单元数据传送孩子 16 位的标志寄存器保存，然后将（SP）加 2。**影响全部的标志位**
LEA 双操作数指令，将源操作数的有效地址传送至目的操作数保存，源操作数只能是内存单元，**目的操作数只能是 16 位的通用寄存器（有效地址是 16 位）**
LDS LES（装入地址指针指令）

指令格式： LDS DEST，SRC

LES DEST，SRC

指令功能：
$$(SRC) \Rightarrow DEST$$
$$(SRC+2) \Rightarrow DS或ES$$

标志位影响：无

算术运算类指令 8 条 **（其中的 INC 和 DEC 不影响 CF 标志）**
ADD 标志位影响 OF,SF,ZF,PF,AF,CF
ADC 标志位影响 OF,SF,ZF,PF,AF,CF，用于衔接高低位字节的运算，将低字或者低字节加法产生的进位反映到高字或者高字节
INC 加一指令 标志位影响除 CF 外的五个标志位 INC 并不影响 CF 标志位
SUB 标志位影响 OF,SF,ZF,PF,AF,CF
SBB 标志位影响 OF,SF,ZF,PF,AF,CF
DEC 标志位影响**除 CF 外的五个标志位**
NEG 求相反数指令 **标志位影响 OF,SF,ZF,PF,AF,CF**

例 5．3．10 假设 VA+2、VA 两个字内存单元中分别存放了一个 32 位补码的高字和低字，要求设计程序片段获取该 32 补码的相反数补码。

```
NEG   VA
MOV   AX，0
SBB   AX，VA+2
MOV   VA+2，AX
```

CMP 比较指令 **标志位影响 OF,SF,ZF,PF,AF,CF** CMP 指令除了不保存运算结果，其余的所有解释都与减法指令 SUB 相同
位操作指令 13
AND OR XOR 影响标志位 CF,OF 强制置 0，AF 不确定，SF，ZF，PF 的解释与算术运算指令保持一致
NOT 逻辑非 **不影响任何标志位**
TEST（测试指令）功能和 AND 指令一致，但不保存结果，标志位影响与 AND 一样。
SAL（算术左移指令）将目的操作数左移，从左边移出的最低一位保存到 CF 中，右边空出的补充 0。**标志位影响 OF，SF，ZF，AF，PF，CF，其中 AF 不确定**，其他与算术运算指令一致，仅当位移数为 1，OF 才有意义（SAL 解释为补码）
SAR（算术右移指令）左边空出来的数据位补充目的操作数的**符号位**
SHL（逻辑左移指令）解释为无符号数编码
SHR（逻辑右移指令）左边空出的数据位补充 0
ROL（循环左移指令）标志位影响 CF 和 OF，CF 用于保存从左边移出的最低一位，（如果位移数为 1，并且移位前后目的操作数的符号位发生改变，则溢出，OF 为

1，若符号位没有变化，则没有溢出，OF=0）
ROR（循环右移指令）标志位影响，CF，OF
RCL（带进位循环左移指令）将 CF 看做目的操作数的最高位，整体循环左移，标志位影响，CF，OF

例 5．3．21 假设一个 48 位的无符号数编码由低位到高位分别存放于 DA、DA+2、DA+4 这三个字内存单元（16 位）中。要求实现此 48 位无符号数编码乘以 2 的运算。

乘以 2 的运算即为左移 1 位，程序片段如下：

```
SHL   DA，1
RCL   DA+2，1
RCL   DA+4，1
```

RCR（带进位循环右移指令）将（CF）看做目的操作数的最低位，标志位影响 CF，OF
（若位移数大于 1，只能以（CL）给出位移数）
对于长操作数的移位只能逐位进行
处理器控制类指令 9 条
CLC（进位标志清除指令）使 CF=0
STC（进位标志置位指令）使 CF=1
CMC（进位标志取反指令）将 CF 标志取反
CLD（方向标志清除指令）使 DF=0
STD（方向标志置位指令）使 DF=1
CLI（中断使能标志清除指令）使 IF=0
STI（中断使能标志置位指令）使 IF=1
HLT（停机指令）
NOP（空操作指令）

（4）ADD [0100H]，64H
错误，目的操作数应使用PTR运算符指出类型，否则具有二义性
正确的写法：ADD BYTE PTR [0100H],64H，（或使用WORD PTR）

等值伪指令 equ　例：Pi equ 3.1415
简单变量定义
[变量名] 数据定义符 表达式 1[表达式 2，……表达式 n] [;注释]
数据定义符确定变量的数据类型，DB，DW，DD 等
定义字节变量 X DB 0　　或者 X DB ？（？表示没有给 X 赋初值，只是分配一个字节的空间）MSGHI DB ‘helloworld’
定义字变量　WVAR DW 2009H（高位存放在高地址字节，低位存档在低地址字节）
双字变量 DDVAR DD 12345678H
重复说明符 DUP

```
WVAR    DW    50   DUP (?)          ;定义了 50 个字单元变量
STRHI   DB    20   DUP  ('Good!')   ;定义了 100 个字节的字节型变量并在这
                                     ;100 个字节的存放了 20 个'Good!'。
```

段属性运算符 SEG，返回段基值
偏移量属性运算符 OFFSET，和 LEA 不同，LEA 的目标操作数只能是 16 位的通用寄存器，而 OFFSET 运算符的表达式经过编译后是一个立即数，所以目标操作数可以是 16 位的存储器单元
类型属性运算符 TYPE 返回变量的字节数或者标号的 FAR 或 NEAR 类型

标号及变量类型属性表

	类型属性	类型数字
变量	BYTE	1
	WORD	2
	DWORD	4
标号	NEAR	-1
	FAR	-2

NEAR 表示该标号智能被同段代码引用
FAR 表示该标号能被其他段代码引用

强制类型转换 数据类型 PTR 地址表达式。数据类型有 BYTE、WORD、DWORD

LABEL 伪指令用于设置或者更改变量和标号的属性
变量名 LABEL 类型 或者 标号 LABEL 类型

强制类型转换字段的格式 数据类型 PTR 地址表达式。例如变量定义为 BVAR DB 100 DUP（0）而 MOV AX，BVAR 编译会因为类型不匹配而不能通过，正确的访问方式是 MOV AX,WORD PTR BVAR

HIGH 和 LOW 操作符分别取表达式计算结果的高 8 位和低 8 位

ASSUME 伪指令指明了某段名与某段寄存器之间的关系
对 DS 赋值的语句

```
mov     ax,data
mov     ds,ax
```

通过 ax 搭桥的原因是由于所有的段寄存器都不能接收立即数；而 data 和其他段名一样是一种特殊的立即数：它们的性质一直是立即数，但它们的值是在程序装入时由操作系统确定的。

IP 和 SP 的初值
如果对于段的组合类型选择 stack，**栈指针 SP 将被设置成该段定义的字节数的总和**，如果没有一个段的段组合属性设置成 stack，就必须用类似初始化 DS 和

ES 那样让 SS 指向你设置的堆栈段并让 SP 指向栈顶。
给 IP 赋值
end main.end 是一条伪指令，表示源程序到此结束，编译程序不处理该语句之后的任何内容。End 后面可以附带一个在代码段中已定义的标号，用以说明程序启动的偏移量。程序的运行，就是程序文件被操作系统引导到内存中，然后让指令指针 CS：IP 指向程序的起点。**IP 所赋的值，就是伪指令 end 指定的**。

```
stack0  segment stack        ;堆栈段
        dw 40h dup(0)
stack0  ends

data    segment              ;数据段
        ;<变量定义>

data    ends

code    segment              ;代码段
        assume cs:code,ds:data,ss:stack0
main:   mov ax,data          ;ds指向数据段
        mov ds,ax

        ;<执行代码>


        mov ah,4ch           ;程序结束，退回DOS
        int 21h

code    ends
        end main
```

和汇编学习有关的几个 DEBUG 命令

命令字符	功　　能
?	显示 DEBUG 命令列表
q	退出 DEBUG
r	显示或设置寄存器值
d	显示默认或指定地址范围内存单元的值
e	从指定地址开始设置各字节单元的值
a	汇编命令
u	反汇编命令
g	运行指定地址开始的代码

-t 单步执行　-r **寄存器**查看/编辑命令　-d 查看指定地址开始的内存区域中**内存单元**的内容　-e **修改**指定地址**内存单元**中的内容　-g **连续执行**到断点位置

分支与循环程序设计
无条件跳转指令 JMP
JMP 后面可以是标号或者 16 位的通用寄存器名称或者字类型的内存单元（段内间接转移）

条件跳转指令
JX NEAR 类型的**段内标号**
（1） 单标志位转移指令

指令助记符	指令功能
JC	CF=1，转移至目标地址 CF=0，顺序执行
JNC	CF=1，顺序执行 CF=0，转移至目标地址
JO	OF=1，转移至目标地址 OF=0，顺序执行
JNO	OF=1，顺序执行 OF=0，转移至目标地址
JS	SF=1，转移至目标地址 SF=0，顺序执行
JNS	SF=1，顺序执行 SF=0，转移至目标地址
JZ / JE	ZF=1，转移至目标地址 ZF=0，顺序执行
JNZ / JNE	ZF=1，顺序执行 ZF=0，转移至目标地址
JP / JPE	PF=1，转移至目标地址 PF=0，顺序执行
JNP / JPO	PF=1，顺序执行 PF=0，转移至目标地址

图 7.2.1 单标志位指令的助记符与功能

（2） JCXZ 指令 若（CX）=0，则转移到目标地址，（CX）不为 0，则顺序执行。**在 8086/8088 指令系统中，该指令是唯一不以标志位为判断条件的条件转移指令。**

（3） 无符号数条件跳转指令 将最近一条执行的 CMP（或者 SUB）指令中的两个操作数解释为无符号数编码，以 CMP 指令所影响的 CF、ZF 标志位来综合判断被减数与减数的大小关系，从而根据判断结果来决定是否实施流程转移。**将 CMP 指令的操作数解释为无符号数**

指令助记符	指令功能
JA / JNBE	若 $A > B$，转移至目标地址 若 $A \leq B$，顺序执行
JAE / JNB	若 $A \geq B$，转移至目标地址 若 $A < B$，顺序执行
JB / JNAE	若 $A < B$，转移至目标地址 若 $A \geq B$，顺序执行
JBE / JNA	若 $A \leq B$，转移至目标地址 若 $A > B$，顺序执行

图 7.2.2 无符号数条件转移指令的助记符与功能

A 表示被减数，B 表示减数

（4） 带符号数条件跳转指令

将 CMP 指令的操作数解释为补码

指令助记符	指令功能
JG / JNLE	若 $A > B$，转移至目标地址 若 $A \le B$，顺序执行
JGE / JNL	若 $A \ge B$，转移至目标地址 若 $A < B$，顺序执行
JL / JNGE	若 $A < B$，转移至目标地址 若 $A \ge B$，顺序执行
JLE / JNG	若 $A \le B$，转移至目标地址 若 $A > B$，顺序执行

图 7.2.3 带符号数条件转移指令的助记符与功能

循环控制指令

（1） LOOP 指令 指令功能：首先（CX）-1=>CX，这个减法不影响标志位，然后如果（CX）不为 0，则跳转至目标地址，否则顺序执行

（2） LOOPZ/LOOPE (CX)!=0 并且 ZF=1，则转移至目标地址

（3） LOOPNZ/LOOPNE (CX)!=0 并且 ZF=0，则转移至目标地址

子程序调用与返回指令不影响标志位

段内直接调用 CALL。CALL 加子程序名称 指令功能，首先 SP 减 2，指向一个空的栈顶，然后将 CALL 指令后面一条指令的偏移量（IP）送入栈顶保存。再将子程序名称的偏移量保存到 IP，使程序流程想子程序入口转移

段内间接调用指令 CALL 后面为 16 位的通用寄存器名称或者字内存单元的逻辑地址

不带参数返回的**段内返回指令 RET** 栈顶单元内容送 IP 保存，使程序流程转向主程序返回点，然后 SP+2，栈顶内容出栈

段间直接调用指令 指令功能 先 SP-2，指向空的栈顶，然后保存 CALL 指令后一条指令的偏移量，SP-2，保存 call 指令后一条指令的段基值，然后子程序名称的偏移量保存到 IP，子程序名称的段基值保存到 CS

段间间接调用指令 双字内存单元的逻辑地址

（4）带参数的段内返回指令

指令格式：RET N

RETN N

第一种格式在 NEAR 类型的子程序中使用，在汇编过程中汇编程序会自动将它解释为第二种格式；第二种格式明确指出返回类型为段内返回。两种格式中的参数 N 都为整数，并且必须是偶数。

指令功能：

1）$((SP)) \Rightarrow IP$，栈顶单元内容送 IP 保存，使程序流程转向主程序返回点；

2）$(SP)+2 \Rightarrow SP$，栈顶单元出栈

3）$(SP)+N \Rightarrow SP$，从栈顶直接出栈 N / 2 个字，此功能用于清除主程序在调用子程序前通过堆栈传递的 N / 2 个入口参数（读者可在后面介绍的子程序设计中深刻体会这一功能的用途）。

Pay attention 段间调用入栈的有段基值和偏移量两个字，相应的段间返回出栈也是两个字，段内调用和返回出栈入栈的只有 IP 一个字

参数传递方式，寄存器传递方式，**堆栈传递方式**，数据区传递方式

系统调用
（1） 单个字符输入
MOV AH,01H
INT 21H
功能，接受一个键盘输入字符，并保存到 AL 寄存器中。
入口参数：无 出口参数：（AL）=输入字符的 ASCII 码

（2） 单个字符输出
MOV AH，02H
INT 21H
功能，在屏幕光标位置显示（DL）指示的字符
入口参数：（DL）=输出字符的 ASCII 码 出口参数：无

（3） 字符串输入
MOV AH，0AH
MOV DX,缓冲区起始偏移量
INT 21H
功能，接受接盘字符串输入，输入的字符串保存在（DS）:（DX）+2 起始的缓冲区，（DS）:（DX）指示的字节单元为入口参数，含义为字符串的最大字符数，（DS）:（DX）+1 为实际从键盘接收的字符数。
入口参数 （DS）:（DX）=缓冲区首字节地址，（DS）:（DX）=最大字符数
出口参数：（（DS）:（DX）+1）=实际输入字符数

（4） 字符串输出
MOV AH,09H
MOV DX,字符串起始偏移量
INT 21H
功能：在屏幕显示逻辑地址（DS）:（DX）指向的字符串，直到遇见“$”结束。
字符串必须以**“$”结尾**。

实现回车换行的子程序

```
NEWLINE PROC
        PUSH AX;CPU 现场保护
        PUSH DX
        MOV DL,0DH
        MOV AH,02H
        INT 21H
        MOV DL,0AH
        MOV AH,02H
```

```
        INT 21H
        POP DX
        POP AX
        RET
NEWLINE ENDP
```

0DH 和 0AH 分别为回车字符和换行字符。
回车字符不改变光标所在行，将光标置于第 0 列（首列）换行字符不改变光标所在列，只是将光标置于下一行同列位置。

- 数字字符：30H~39H
- 大写字母：41H开始
- 小写字母：61H开始
- 回车字符：0DH
- 换行字符：0AH

小写变大写 SUB AL，20H（假设字符保存在 AL 中）

乘法

指令格式：

MUL 寄存器/内存操作数 ;; 无符号整数乘法指令

IMUL 寄存器/ 内存操作数 ;; 有符号整数乘法指令

乘法类型	隐含（目的）操作数	源操作数	乘积
字节乘（Byte）	AL	8 位寄存器名 8 位数据地址	16 位数 存于 AX 中
字乘（Word）	AX	16 位寄存器名 16 位数据地址	32 位数 存于 DX：AX 中
双字乘（DWord）	EAX	32 位寄存器名 32 位数据地址	64 位数 存于 EDX：EAX 中

除法运算指令

DIV Regs/Mem

IDIV Regs/Mem

除法类型	被除数	除数	商和余数
字节除法	AX	8 位寄存器或内存操作数	商：AL ， 余数：AH
字除法	DX：AX	16 位寄存器或内存操作数	商：AX ， 余数：DX
双字除法	EDX：EAX	32 位寄存器或内存操作数	商：EAX ， 余数：EDX

BCD 码，用 4 个二进制位表示一个十进制数

非组合型 BCD 码，占用一个字节的低 4 位，高四位用 0 填充
组合型 BCD 码，由两位 BCD 码组成，占用一个字节，一个 BCD 码占用字节的低四位，一个 BCD 码占用高四位。

BCD 码的加减法指令有：

非组合型 BCD 码加法调整指令 AAA（ASCII Adjust after Adition）

非组合型 BCD 码减法调整指令 AAS（ASCII Adjust after Subtraction ）

组合型 BCD 码加法调整指令 DAA（Decimal Adjust after Adition）

组合型 BCD 码减法调整指令 DAS（Decimal Adjust after Subtraction）

都是在加法指令（ADD，ADC）和减法指令（SUB，SBB）之后，对存放在 AL 中的加法减法结果进行调整