1. 标志位：PF, AF, DF, IF, TF

(1) PF(Parity Flag)奇偶标志和 AF 辅助进位标志

```
mov ah, 4
add ah, 1; AH=0000 0101B, PF=1 表示有偶数个 1
mov ax, 0101h
add ax, 0004h; AX=0105h=0000 0001 0000 0101B
              ; PF=1 只统计低 8 位中 1 的个数
```

要是低 8 位中 1 的个数是奇数时，PF=0

PF 有两条相关指令：

jp(当 PF==1 时则跳)，jnp(当 PF==0 时则跳)

其中 jp 也可以写成 jpe(jump if parity even)，

jnp 也可以写成 jpo(jump if parity odd)

AF(Auxiliary Flag) 辅助进位标志

低 4 位向高 4 位产生进位或借位

例如：

```
mov ah, 1Fh; 0001 1111
add ah, 1   ; 0000 0001 +)
            ; ah=20h, AF=1
```

AF 跟 BCD(Binary Coded Decimal)码有关。

11h 表示 11

59h 表示 59

BCD 的优点是便于用以下方法快速分离出十位和个位：

59h >> 4 ➔ 5

59h & 0Fh ➔ 9

BCD 码在加法时需要做调整：总结：AF=1或AF=0但个位大于等于A，就要加6修正

```
59h + 8 = 61h; AF=1, 需要调整
需要对上述结果作调整，方法是对上述结果加 6，得 67h
59h + 2 = 5Bh ; 虽然 AF=0，但个位超过 A，所以也需要
              ; 通过加 6 作调整，得 61h
59h + 0 = 59h ; AF=0，且个位<A，所以不需要调整
```

AF 没有相关的条件跳转指令。

CF ZF SF OF AF PF：这 6 个称为状态标志

DF TF IF：这 3 个称为控制标志

DF:direction flag

TF:trace/trap flag

IF:interrupt flag

(2) DF（Direction Flag）方向标志：控制字符串的操作方向。

当 DF=0 时为正方向(低地址到高地址)，当 DF=1 是反方向。

cld 指令使 DF=0， std 指令使 DF=1

若源数据首地址>目标数据首地址，则复制时要按正方向(从低地址到高地址)；

若源数据首地址<目标数据首地址，则复制时要按反方向(从高地址到低地址)；

strcpy(target, source)；永远按正方向复制

memcpy(target, source, n)；永远按正方向复制

memmove(target, source, n)；能正确处理部分重叠

有 2 条指令可以设置 DF 的值：

cld 使 DF=0，字符串复制按正方向

std 使 DF=1，字符串复制按反方向

若源首地址<目标首地址，则复制按反方向。

1000	'A'	1002	'A'
1001	'B'	1003	'B'

1002	'C'A	1004	'C'
1003	'D'B	1005	'D'
1004	'E'C	1006	'E'

当源首地址>目标首地址时，复制时按正方向

1002	'A'C	1000	'A'
1003	'B'D	1001	'B'
1004	'C'E	1002	'C'
1005	'D'	1003	'D'
1006	'E'	1004	'E'

(3) IF（Interrupt Flag）中断标志

当 IF=1 时,允许中断;否则禁止中断。cli 指令使 IF=0 表示关/禁止硬件中断;

sti 指令使 IF=1 表示开/允许硬件中断。

```
mov ax, 0
mov bx, 1
next:
add ax, bx
;此时若用户敲键,则 CPU 会在此处插入一条 int 9h 指令并执行它
;int 9h 的功能是读键盘编码并保存到键盘缓冲区中
add bx, 1
cmp bx, 100
;若程序已运行了 1/18 秒,则 cpu 会在此处插入一条 int 8h 指令
jbe next
```

用 cli 和 sti 把一段代码包围起来可以达到该段代码在执行过程中不会被打断的效果:

```
cli; clear interrupt 禁止硬件中断
...; 重要代码
sti; set interrupt 允许硬件中断
```

(4)TF（Trace/Trap Flag）跟踪/陷阱标志

当 TF=1 时,CPU 会进入单步模式(single-step mode)。
当 TF=1 时,CPU 在每执行完一条指令后,会自动在该条指令与下条指令之间插入一条 int 1h 指令并执行它。

利用单步模式可以实现反调试:
进入单步模式
```
nop
;int 1
xxx➔mov
;int 1
yyy➔add
;int 1
zzz➔cmp
```

2．对内存变量的访问可以使用两种方式

(1) 直接寻址

设某个 8 位变量的地址为 1000h:2000h，现要取出它的值到 AL 中:

```
mov ax, 1000h
mov ds, ax
mov al, ds:[2000h]; 这个就是直接寻址
```

(2) 间接寻址

①[bx] [bp] [si] [di]就成了最简单的间接寻址方式
[ax] [cx] [dx] [sp]语法错误
②[bx+si] [bx+di] [bp+si] [bp+di]
注意[bx+bp]以及[si+di]是错误的。
③[bx+2] [bp-2] [si+1] [di-1]
④[bx+si+2] [bx+di-2] [bp+si+1] [bp+di-1]
两个寄存器相加的间接寻址方式中，bx 或 bp 通常用来表示数

组的首地址，而 si 或 di 则用来表示下标。
例如： http://10.71.45.100/bhh/arydemo.asm

两个寄存器相加再加一个常数的间接寻址通常用来访问结构数组中某个元素中的某个成员，例如：

```
struct student
{
   char name[8];
   short int score;
};
struct student a[10];
ax = a[3].score;
```

设 bx=&a[0]，si=30，则 ax=a[3].score 可转化成以下汇编代码：

```
mov ax, [bx+si+8]; /* bx+si→a[3] */
```

3．端口

CPU ←→ 端口(port) ←→ I/O 设备
端口编号就是端口地址。端口地址的范围是：
[0000h，0FFFFh]，共 65536 个端口。
对端口操作使用指令 in 与 out 实现。
通过 60h 号端口，CPU 与键盘之间可以建立通讯。
in al，60h；从端口 60h 读取一个字节并存放到 AL 中
例如： http://10.71.45.100/bhh/key.asm

70h 及 71h 端口与 cmos 内部的时钟有关。
其中 cmos 中的地址 4、2、0 中分别保存了当前的时、分、秒，并且格式均为 BCD 码。

```
mov al, 2
out 70h, al
in al, 71h; 读取 cmos 中 2 号单元的值

mov al, 4
out 70h, al
```

```
mov al, 23h
out 71h, al; 把 cmos4 号单元即小时的值改成 23 点
```

例如：http://10.71.45.100/bhh/readtime.asm

以读取键盘为例，以下为从高层到低层的编程方式排序：

```
dos       高 mov ah, 1; int 21h 功能弱,但编程简单
bios      中 mov ah, 0; int 16h
in/out    低 in al, 60h; 功能强, 但编程麻烦
```

例如：http://10.71.45.100/bhh/music.asm