汇编所涉及的所有指令

1、jmp

2、mov dst, src
3、xchg oprd1, oprd2
4、xlat 查表指令

5、push reg/mem/seg
6、pop reg/mem/seg

7、CLC (clear carry:进位):
8、CMC (complement:补 carry):
9、STC (set carry)
10、CLD (clear direction):
11、STD (set direction):
12、CLI (clear interrupt):
13、STI (set interrupt):
14、LAHF
15、SAHF
16、pushf
17、popf

18、lea reg, mem
19、lds/les reg, src
20、in CPU <- 外设
21、out CPU -> 外设

22、add reg/mem, reg/mem/imm
23、adc reg/mem, reg/mem/imm
24、inc reg/mem

25、sub reg/mem, reg/mem/imm
26、sbb reg/mem, reg/mem/imm
27、dec reg/mem
28、neg reg/mem
29、cmp reg/mem, reg/mem/imm

30、mul reg/mem
31、imul reg/mem

32、div reg/mem
33、idiv reg/mem

34、cbw
35、cwd

36、and reg/mem, reg/mem/imm
37、or reg/mem, reg/mem/imm
38、xor reg/mem, reg/mem/imm
39、test reg/mem, reg/mem/imm
40、not reg/mem

41、shl reg/mem, 1/cl
42、sal reg/mem, 1/cl
43、rol reg/mem, 1/cl
44、rcl reg/mem, 1/cl

45、shr reg/mem, 1/cl
46、sar reg/mem, 1/cl
47、ror reg/mem, 1/cl
48、rcr reg/mem, 1/cl

49、rep movs / lods / stos
50、movsb
51、movsw

52、stosb
53、stosw

54、lodsb
55、lodsw

56、repz / repe 相等/为零时重复串操作
57、repnz / repne 不相等/不为零时重复串操作
58、cmpsb
59、cmpsw

60、scasb
61、scasw

62、jo opr
63、jno opr

64、js opr
65、jns opr

66、jc  opr
67、jnc opr

68、jz / je  opr
69、jnz / jne  opr

70、jp / jpe  opr
71、jnp / jpo  opr

72、ja / jnbe  opr 大于时转移
73、jae / jnb  opr 大于等于时跳转
74、jb / jnae  opr 小于时转移
75、jbe / jna  opr 小于等于时转移

76、jg / jnle  opr 大于时转移
77、jge / jnl  opr 大于等于时转移
78、jl / jnge  opr 小于时转移
79、jle / jng  opr 小于等于时转移

80、loop opr
81、loopz / loope opr
82、loopnz / loopne opr

83、call 子程序
84、ret

85、db  define bytes
86、dw  define words
86、dd  define double words
88 、 repeat-count dup(operand, ..., operand)
89、name lable type
90、expression-name equ expression
91、=
92、$ 地址计数器的值
93、org constant expression

94、逻辑操作符
    and or xor not
95、移位操作符
    shl shr

## 关系操作符
96、eq ne lt gt le ge

97、type expression
98、length variable
99、size variable
100、offset variable/label
101、seg variable/label

102、type ptr expression
103、段操作符
104、short
105、this attribute/type
106、字节分离操作符
107、字分离操作符

## 定义数组的几种方式
1、array_1 db 1, 2, 3, 4, 6
2、array_2 db 100 dup(?)
3、array_3 db 100, 0, 100 dup(?); ??什么意思
4、array_4 label byte
    org $+100
    <=>
    array_4 db 100 dup(?)

5、array_5 db 50 , 0, 50 dup(0);

**有关寄存器内容**

**1、4 个通用寄存器**

① ax
② bx
③ cx
④ dx

**2、4 个段寄存器**

**CS** (code segment): 代码段寄存器
**DS** (data segment): 数据段寄存器
**SS** (stack segment): 堆栈段寄存器
**ES** (extra segment): 附加段寄存器

**3、4 个专用寄存器**

**SP** (stack pointer): 堆栈指针寄存器
**BP** (base pointer): 基址指针寄存器
**SI** (source index): 源变址寄存器
**DI** (dest index): 目的变址寄存器

**存储器寻址方式**

段寄存器与专用寄存器、通用寄存器的配合

**① 寄存器间接寻址**

物理地址 = 16 * (DS) + (BX) / (SI) / (DI)
= 16 * (SS) + (BP)

**② 寄存器相对寻址**

物理地址 = 16 * (DS) + (BX) / (SI) / (DI)
= 16 * (SS) + (BP)

**③ 基址变址寻址方式**

物理地址 = 16 * (DS) + (BX) / (BP) + (SI) / (DI)

**④ 相对基址变址寻址方式**

物理地址 = 16 * (DS) + (BX) / (BP) + (SI) / (DI) + DISP8 / DISP16

**4、16 个状态标志寄存器**

① 与**计算**有关的: CF PF AF ZF SF OF
② 与地址有关的: DF
③ 与中断有关的: TF IF

① 循环指令 loop loopz / loope loopnz / loopne 不影响标志位
② 类型转换指令 cbw cwd 不影响标志位
③ 从串中取指令 lodsb lodsw 不影响标志位
④ 处理机控制类指令 nop hlt esc wait lock 不影响标志位

**CF** (carry flag): 进位标志寄存器
CF = 0: 无 进位 或 借位
= 1: 有 进位 或 借位

**PF** (parity: 奇偶性 flag): 奇偶标志寄存器
PF = 0: 奇数
= 1: 偶数

**AF** (auxiliary: 辅助 flag): 辅助进位标志寄存器
AF = 0: 无 进位 或 借位
= 1: 有 进位 或 借位
主要用于 BCD

**ZF** (zero flag): 零标志寄存器
ZF = 0: 结果不为 0
= 1: 结果为 0

**SF** (symbol flag): 符号标志寄存器
SF = 0: 结果为 正
= 1: 结果为 负

**TF** (track flag)
TF = 0: 禁止单步中断
= 1: 允许单步中断

**IF** (interrupt flag):
IF = 0: 关中断
= 1: 开中断

**DF** (direction flag):
DF = 0: 地址指针**递增**修改
= 1: 地址指针**递减**修改
主要用于串操作

**OF** (overflow flag):
OF = 0: 无 溢出
= 1: 有 溢出

**段内直接转移**

1、**jmp short opr**: (IP) <- (IP) + 8 位偏移地址

2、**jmp near ptr opr**: (IP) <- (IP) + 16 位偏移地址

例:

```
xor bx, bx
jmp short next
add ax, bx
mov bx, ax
next:
    mov, ax, 1
```

**段内间接转移**

1、**jmp word ptr opr**: (IP) <- (EA)

2、**jmp reg**: (IP) <- (EA)

例:

```
mov ax, 1000h
jmp ax

jmp word ptr table[bx]
```

**段间直接转移**

1、**jmp far ptr opr**:

(IP) <- opr 的段内偏移地址

(CS) <- opr 所在段的段地址

例:

```
s1 segment
    jmp far ptr next
s1 ends

s2 segment
    next:
s2 ends
```

**段间间接转移**

1、**jmp dword ptr opr**:

(IP) <- (EA)

(CS) <- (EA + 2)

(EA): 存储器寻址方式

例:

```
jmp dword ptr table[bx]
```

**注**:

1、双操作数指令除串操作外

src、dst 不能同时为 seg 或 mem

2、cs、imm 不能做 dst

3、imm 不能传 seg

4、src 与 dst 类型要匹配

**通用数据传送指令**

1、**mov dst, src**

dst <- src

dst: reg/mem/seg

src: reg/mem/seg/imm

例:

mov cl, 4

mov ax, 03ffh

mov al, bl

mov ax, es

mov ds, ax

mov ax, [si]: 寄存器间接寻址
把 si 存放的地址
所对应的地址单元
的数据放入 ax

mov [di], cx: 寄存器间接寻址
把 cx 的数值
赋给 di 存放的地址
所对应的地址单元
中

mov [1000h], al: 内存直接寻址

mov dest[bp + si], es: 相对基址变址寻
址

**2、xchg oprd1, oprd2**

oprd1 <-> oprd2

oprd1: reg/mem

oprd2: reg/mem

例:

mov bx, [bp + si]: 基址变址寻址

**3、xlat 查表指令**

(AL) <- ((BX) + (AL))

堆栈指令

1、**push reg/mem/seg**

① (SP) <- (sp) – 2

② 放入数据

例:

ax = 1234h

push ax

**SP：**只配合 PUSH，POP 使用，不做寻址使用

**BP：**可用于寄存器寻址

2、**pop reg/mem/seg**

① 取出数据

② (SP) <- (SP) + 2

**标志类指令**

**CF**

1、**CLC** (clear carry:进位): 进位位置 0 指令
2、**CMC** (complement carry): 进位位求反指令
3、**STC** (set carry) 进位位置 1 指令

**DF**

4、**CLD** (clear direction): 方向标志置 0 指令，递增
5、**STD** (set direction): 方向标志置 1 指令，递减

**IF**

6、**CLI** (clear interrupt): 中断标志置 0 指令，关中断
7、**STI** (set interrupt): 中断标志置 1 指令，开中断

8、**LAHF** load ah with flags 存标志寄存器指令
(AH) <- (flags 的低字节)

9、**SAHF** store ah in flags 取标志寄存器指令
(flags 的低字节) <- (AH)

10、**pushf** 标志寄存器进栈指令
sp = sp - 2

11、**popf** 标志寄存器出栈指令
sp = sp + 2

可以修改追踪标志 TF

**地址传送指令**

1、**lea reg, mem**

load effective address

取有效地址指令，主存按 src 的寻址方式计算偏移地址，送入指定寄存器

dst 不能为 seg

src 可使用除 立即数、寄存器之外的任一种存储器寻址方式

例:

lea si, buff <==> mov si, offset buff

lea si, [bx + si + 1000h]

offset: 只能用于符号地址

2、**lds/les reg, src**

段寄存器装入指令

src 要求为存储器寻址方式

**lds reg, src**:

(reg) <- src

(ds) <- (src + 2)

**les reg, src:**

(reg) <- src

(ds) <- (src + 2)

例:

lds bx, [2000h]

(bx) <- [2000h]、[2001h] 内容

(ds) <- [2002h]、[2003h] 内容

**输入输出指令 CPU 与 I/O 端口**

1、**in CPU <- 外设**

**in al, port**

(al) <- port 8bit

**in ax, port**

(ax) <- port 16bit

**in al, dx dx** 中装的是地址

(al) <- ((dx)) 8bit

**in ax, dx**

(ax) <- ((dx) + 1, (dx))

2、**out CPU -> 外设**

**out port, al**

**out port, ax**

**out dx, al**

**out dx, ax**

例:

in ax, 28h

mov data_word / [data_word], ax

mov dx, 3fch

in ax, dx

out 5, al

**算术运算指令**

**加法**

1、**add reg/mem, reg/mem/imm**

(dst) <- (src) + (dst)

修改: CF PF

2、**adc reg/mem, reg/mem/imm**

(dst) <- (dst) + (src) + CF

修改: CF OS SF PF ZF AF

3、**inc reg/mem**

(opr) <- (opr) + 1

修改: PF AF ZF SF OF

**减法**

1、**sub reg/mem, reg/mem/imm**

(dst) <- (dst) - (src)

(dst) < (src): CF = 1

2、**sbb reg/mem, reg/mem/imm**

SuBtract with Borrow

(dst) <- (dst) - (src) - CF

3、**dec reg/mem**

(opr) <- (opr) - 1

修改: PF AF ZF SF OF

4、**neg reg/mem**

negate: 否定

求补

(opr) <- -1 * (opr)

按位求反末位加一

(opr) <- 0ffffh - (opr) + 1

5、**cmp reg/mem, reg/mem/imm 常与后面的 ja / jg 等指令连用**

(dst) - (src)

**ZF** = 1:

dst = src

ZF = 0 && 无符号数 && **CF** = 1:

dst < src

ZF = 0 && 有符号数 && **OF** != **SF**:

dst < src

OF xor SF = 1

此处可以考虑 OF = 0，即没有溢出的情况下，dst - src 对 SF 的值的影响

OF = 0: 无溢出

SF = 1: dst - src < 0 dst < src

例:

X、Y、Z 为 32bit 双精度数，存放在 x, x+2, y, y+2, z, z+2，中，实现加法 w <- x + y + 24 - z

```
datas segment
datas ends

codes segment
    assume ds: datas, cs: codes
    main proc far
        start:
            mov ax, datas
            mov ds, ax
            sub ax, ax

            ; X + Y
            mov ax, x
            mov dx, x + 2
            add ax, [y]
            adc dx, [y + 2]

            ; X + Y + 24
            add ax, 24
            adc dx, 0

            ; X + Y + 24 - Z
            sub ax, [z]
            sbb dx, [z + 2]

            ; 高位数放高地址
            ; 低位数放低地址
            mov [w], ax
            mov [w + 2], dx

            mov ah, 4ch
            int 21h
    main endp
codes ends
end start
```

古今成大事者，不惟有**超世之才**，亦必有坚韧不拔之志

**乘法指令**

1、**mul reg/mem**

无符号数乘法指令

(ax) <- (al) * (src)

(dx, ax) <- (ax) * (src)

例:

mul dl

(ax) <- (al) * (dl)

mul word ptr [2000h]

(dx, ax) <- (al) * ([2001h], [2000h])

2、**imul reg/mem**

带符号乘法指令

integer multiplication

修改: CF OF

例:

计算 (v - (x * y + z - 540)) / x

其中变量均为 16 位带符号数,

结果商存入 ax, 余数存入 dx

```
datas segment
    datas ends

    codes segment
        main proc far
            assume cs: codes, ds: datas
            start:
                mov ax, datas
                mov dx, ax
                sub ax, ax

                mov ax, x
                imul ax, y

                mov cx, ax
                mov bx, dx

                mov ax, z
                cwd

                add cx, ax
                adc bx, dx

                sub cx, 540
                sbb bx, 0

                mov ax, v
                cwd

                sub ax, cx
                sbb dx, bx

                idiv x
        main endp
    codes ends
    end start
```

在立即数的算术运算时，无需考虑位扩展的问题，可以直接进行算术运算;

当把数据放入 ax 时，做算术运算时需要对其进行扩展，使得两个数长度相一致

**除法指令**

1、**div reg/mem**

无符号除法指令

8bits 字节操作

(al) <- (ax) // (src)　即 商

(ah) <- (ax) mod (src)　即 余数\

16bits 字操作

(ax) <- (dx, ax) // (src)

(dx) <- (dx, ax) mod (src)

2、**idiv reg/mem**

带符号除法指令

修改: 对所有条件码位无意义

除法运算和 CBW、CWD 配合使用

**类型转换指令 不影响标志位**

1、**cbw**

change byte to word

将 al 中数据的 符号位 扩展到 ah 中

al 符号位 为 1 --> ah = 0ffh 全 1

al 符号位 为 0 --> ah = 00h 全 0

字节转换成字

2、**cwd**

change word to double word

把 ax 中数据的 符号位 扩展到 dx 中

**位操作类指令**

**逻辑运算指令**

1、**and reg/mem, reg/mem/imm**

逻辑与指令 按位与

(dst) <- (dst) ^ (src)

2、**or reg/mem, reg/mem/imm**

逻辑或指令 按位或

(dst) <- (dst) || (src)

3、**xor reg/mem, reg/mem/imm**

逻辑异或指令 按位异或

(dst) <- (dst) (+) (src)

4、**test reg/mem, reg/mem/imm**

(opr1) ^ (opr2)

结果不回送

5、**not reg/mem**

逻辑非指令 按位取反

neg reg/mem

求补

例:

屏蔽(置零)第 0、1 位

```
mov al, 0bfh; 1011 1111
add al, 0fch; 1111 1100
```

第 5 位置 1

```
mov al, 43h; 0100 0011
or al, 20h;   0010 0000'
```

测试 0, 1, 2, 3, 5, 7 位是否全为 0

```
mov al, 40h
test al, 10100111b
```

寄存器 ax 清零

```
xor ax, ax
sub ax, ax
mov ax, 0
```

测试两个数是否相等

```
mov ax, a
mov bx, b
xor ax, bx
jz match
```

**移位指令**

**cnt**: 移动次数

cf = 移出的数字
cnt = 1:
    of = 0: 最高有效位的值不变
    of = 1: 最高有效位的值改变

一般移位指令 修改: SF ZF PF; AF 无意义
循环移位指令 不修改以上标志位

**左移**

1、**shl reg/mem, 1/cl**
    shift left 逻辑左移

    无符号数 最高位 -> CF 最低位补 0

2、**sal reg/mem, 1/cl**
    arithmetic: 算术 shift left

    有符号数 最高位 -> CF 最低位补 0

3、**rol reg/mem, 1/cl**
    rotate left 循环左移

    移出位 -> CF && -> 尾

4、**rcl reg/mem, 1/cl**
    rotate left with carry 带 CF 循环左移

**右移**

1、**shr reg/mem, 1/cl**
    shift right 逻辑右移

    最低位 -> CF 最高位补 0

2、**sar reg/mem, 1/cl**
    arithmetic shift right

    最低位 -> CF 最高位补符号位

3、**ror reg/mem, 1/cl**
    rotate right

    移出位 --> CF && --> 尾

4、**rcr reg/mem, 1/cl**
    rotate right with carry 带 cf 循环右移

例:

    左移 1 次
    mov al, x
    shr al, 1

    左移 2 次
    mov al, x
    mov cl, 2
    shr al, cl

**串处理指令**

**rep** 重复串操作直到(cx) = 0
rep movs / lods / stos

cld: DF = 0 地址递增
std: DF = 1 地址递减

**movs 把位于 DS 的某个串移到 ES**
1、**movsb**
cld: DF = 0
、((ES) : (DI)) <- ((DS) : (SI))
、(SI) <- (SI) + 1
、(DI) <- (DI) + 1

**2、movsw**
cld: DF = 0
、((ES) : (DI)) <- ((DS) : (SI))
、(SI) <- (SI) + 2
、(DI) <- (DI) + 2

例:
DS 4000h 有一 80h 字节的字符串
将其传送到 ES 4200h
开始的存储区

```
mov si, 4000h
mov di, 4200h
mov cx, 80h
cld

rep movsb
```

**stos 存入串指令 把 al 中的数赋给串，常与 rep 搭配**
用于串的初始化
将 al/ax 中的数存入(ES) : (DI) 所指向的位置

1、**stosb**
cld
((ES) : (DI)) <- (AL)
(DI) <- (DI) + 1
例:

```
mov cx, 10h
mov al, 20h
```

2、**stosw**
cld
((ES) : (DI)) <- (AX)
(DI) <- (DI) + 2
cld

```
rep stosb
```

**从串中取指令 lods**

一般不与 rep 连用 不影响标志位 从 DS 串中取某个字或字节赋给 ax 或 al

1、**lodsb**

cld
(AL) <- ((DS) : (SI))
(SI) <- (SI) + 1

2、**lodsw**

cld
(AX) <- ((DS) : (SI))
(SI) <- (SI) +

**串比较指令 cmps**

两个串(分别位于 **DS**、**ES**)的逐位比较

**repz / repe** 相等/为零时重复串操作

(CX) == 0 || (ZF) == 0 时 退出
(ZF) == 0: 上一个运算结果不相等
(CX) <- (CX) - 1

repeat when equal

**repnz / repne** 不相等/不为零时重复串操作

repeat when not equal

1、**cmpsb**

cld
((DS) : (SI)) - ((ES) : (DI))
(SI) <- (SI) + 1
(DI) <- (DI) + 1

2、**cmpsw**

cld
((DS) : (SI)) - ((ES) : (DI))
(SI) <- (SI) + 2
(DI) <- (DI) + 2

**串扫描指令 scas**
**查找一个符号在一个串(ES)中是否出现**

1、**scasb**

cld
(AL) - ((ES) : (DI))
(DI) = (DI) + 1

2、**scasw**

cld
(AX) - ((ES) : (DI))
(DI) = (DI) + 2

例:

串向前移动一个字
std
先移动大地址

es：2000h 有 100 个字节的字符串，查找是否含有'#'字符；有，将'#'所在单元地址送 BX；没有，将 9000h 送 bx

```
datas segment
    datas ends

    codes segment
        main proc far
            start:
                mov ax, datas
                mov ds, ax
                sub ax, ax

                mov di, 2000h
                mov al, '#'
                mov cx 100
                cld
                repnz scasb
                jz find
                mov bx, 9000h
                jmp exit

            find:
                dec di
                mov bx, di
            exit:
                mov ah, 4ch
                int 21h
        main endp
    codes ends
    end start
```

**控制转移指令**

无条件转移指令 **jmp**

**有条件转移指令**

**单个标志位的**

**OF**
1、**jo opr**
2、**jno opr**

**SF**
3、**js opr**
4、**jns opr**

**CF**
5、**jc opr**
6、**jnc opr**

**ZF**
7、**jz / je opr**
8、**jnz / jne opr**

**PF**
9、**jp / jpe opr**
10、**jnp / jpo opr**

**无符号数比较**
**ZF** 和 **CF** 的值

1、**ja / jnbe opr** 大于时转移
cf = 0 && zf = 0
jmp when above

2、**jae / jnb opr** 大于等于时跳转
cf = 0 || zf = 1
jmp when above or equal

3、**jb / jnae opr** 小于时转移
jmp when blow

4、j**be / jna opr** 小于等于时转移
jmp when blow or equal

**有符号数比较**
**ZF** 和 (**OF xor SF**)

1、**jg / jnle** opr 大于时转移
zf = 0 && (OF == SF)
jmp when greater

2、**jge / jnl opr** 大于等于时转移
zf = 1 && (OF == SF)
jmp when greater or equal

3、**jl / jnge opr** 小于时转移
zf = 0 && (OF != SF)
jmp when less

4、**jle / jng opr** 小于等于时转移
zf = 1 && (OF != SF)
jmp when less or equal

例:

x, y 为 16 位数,
判断 x > 50，是则转移 TO_HIGH
否则 x - y
溢出 转 OVERFLOW;
否则求 |x - y|,结果送 result

```
datas segment
    datas ends

    codes segment
        main proc far
            assume cs: codes, ds: datas
            main proc far
                start:
                    mov ax, datas
                    mov ds, ax
                    sub ax, ax

                    mov ax, x

                    cmp ax, 50
                    ja TO_HIGH
                    sub ax, y
                    jo OVERFLOW

                    neg ax
                    mov [result], ax
                    jmp exit
                TO_HIGH:
                    ...
                    jmp exit
                OVERFLOW:
                    ...
                exit:
                    mov ah, 4ch
                    int 21h
            main endp
codes ends
end start
```

loop 循环指令

1、**loop opr**

、(CX) <- (CX) - 1
、(CX) != 0: 转至 opr

2、**loopz / loope opr** 相等则继续跳转

(CX) <- (CX) - 1
(CX) != 0 && (ZF) == 1: 转至 opr

3、**loopnz / loopne opr** 不等则继续跳转

、(CX) <- (CX) - 1
、(CX) != 0 && (ZF) == 0: 转至 opr

不影响标志位

**子程序**

1、**call** 子程序

① call near ptr subp： 段内调用，（IP）入栈

② call far ptr subp：段家调用，（CS）入栈，（IP）入栈

2、**ret**

子程序返回

调用 call 命令时

若为段内调用，则 IP 入堆栈;

若为段间调用，则 CS 先入栈，IP 后入栈，IP 位于低地址，CS 位于高地址.

**ret 段内返回**

(IP) <- ((SP) + 1, (SP))

(SP) <- (SP) + 2

**ret exp 段内带立即数返回**

(IP) <- ((SP) + 1, (SP))

(SP) <- (SP) + 2

(SP) <- (SP) + 16bit 位移量

<- (SP) + imm

**ret 段间返回**

(IP) <- ((SP) + 1, (SP))

(SP) <- (SP) + 2

(CS) <- ((SP) + 1, (SP))

(SP) <- (SP) + 2

**ret 段间带立即数返回**

(IP) <- ((SP) + 1, (SP))

(SP) <- (SP) + 2

(CS) <- ((SP) + 1, (SP))

(SP) <- (SP) + 2

(SP) <- (SP) + 16bit 位移量

**中断**

iret 中断返回指令

1、**01h 单字符输入**，送 **al**

2、**02h 单字符输出**，显示 dl 中字符

3、**09h 字符串输出**，待显示字符串地址送 **dx**

4、**0ah 字符串输入**，输入字符串首地址送 **dx**

字符串在 datas 中的定义必须为:

```
datas segment
    input_string db 50, 0, 50 dup (?)
    ; 第一个 50 表示数据区最大长度为 50
    ; 0 表示实际输入个数为 0，可用'?'代替

datas ends
```

**处理机控制类指令**

1、**nop** 无操作指令

2、**hlt** 停机指令

**伪操作**

内存根据响应指令分配空间

[variable] mnemonic
operand, ..., operand [;comments]

variable: 变量
mnemonic: 助记符 常数 || 表达式
comments: 注释

**mnemonic 助记符**

1、**db** define bytes
定义字节
2、**dw** define words
定义字
3、**dd** define double words
定义双字

4、**dup**
repeat-count **dup**(operand, ..., operand)
操作复制符 可嵌套

5、**label**
**name lable type**
数据项
varibale-name label byte/word/dword

可执行代码
label-name label near/far

6、**equ**
**expression-name equ expression**
表达式赋值伪操作
可用表达式名替代该表达式

例:

byte_array **label** byte

cons **equ** 256; 定义 cons = 256
data **equ** height + 12; 地址表达式赋以符号名
b **equ** [bp + 8]; 变址引用赋以符号名

ab **equ** cons + 2; **cons** 需先被定义

7、**=**
= 可重复定义
equ 不可重复定义

地址计数器与地址伪操作

1、**$** 地址计数器的值

在**指令**中，本条指令第一个字节的值

在**伪操作**中，地址计数器的当前的值

2、**org**

org constant expression

设置当前地址计数器的值，使下一个字节的地址成为常数表达式的值 constant

**操作数项**

**算术操作符**

**+ - * / mod**

**逻辑与移位操作符**

按位操作，只能用在数字表达式中

1、**逻辑操作符**

① **and**

② **or**

③ **xor**

④ **not**

2、**移位操作符**

① **shl**

② **shr**

expression shl / shr num-shift

将 expression 左/右移 num-shift 位

例:

```
opr1 equ 25
opr2 equ 24
and ax, opr1 and opr2

mov ax, 0ffffh shr 2
```

**关系操作符**

① **eq**：等于

② **ne**：不等于

③ **LT**：小于

④ **gt**：大于

⑤ **le**：小于等于

⑥ **ge**：大于等于

两个操作数必须是

1、数字

2、同一段内的两个存储器地址

计算结果为真: return 0ffffh

假: return 0

例:

```
mov bx, a lt b
a < b:   bx <- 0ffffh = -1
a !< b: bx <= 0

mov bx, ((a lt 5) and 20)
          or
          ((a ge 5) and 30)
a < 5:   bx <- 20
a >= 5: bx <- 30
```

## 数值回送操作符

1、**type**

**type expression**

Expression 是:

① 变量: 回送该变量字节数

② 标号: near ret -1; far ret -2

③ 常数: return 0

2、**length**

**length variable**

variable

① 使用 dup() return repeat-count

② return 1

3、**size**

**size variable**

return **分配给该变量的字节数**

= length variable * type variable

4、**offset**

offset variable/label

return 变量 或 标号 **偏移地址**

5、**seg**

seg variable/label

return 变量 或 标号 **段地址值**

## 属性操作符

1、**type ptr expression**

用来建立一个符号地址，为给已经分配的存储地址类型赋予另一种属性

**type**: 所赋予的新的属性类型

**exp**: 被取代的符号地址

例:

```
two_byte dw
a equ byte ptr two_byte
b equ byte ptr (two_byte + 1)

mov byte ptr [bx], 5
mov word ptr [bx], 5
```

2、**段操作符**

mov ax, **es**:[bx + si]

3、**short**

修饰 jmp 转向地址的属性

4、**this**

**this attribute/type**

详见 PPT

该操作数的段地址和偏移地址与下一个地址单元相同

例:

```
byte_type equ this byte
word_type dw 100 dup(?)
```

byte_type 和 word_type 偏移地址相同,

但 byte_type 为 byte 类型，...

```
start equ this far
mov ax, 100
```

、mov 有一个 far 类型的地址 start

、start: mov ax, 100 类似

5、**字节分离操作符**

**high** 取数/地址表达式 高位 字节

**low** 取数/地址表达式 高位 字节

mov ah, high a_number

6、**字分离操作符**

**highword**

**lowword**

一些较为特殊的指令，如隐含用到某类寄存器的指令，或者说是必须用某些寄存器

**1、xlat 查表指令**

(AL) <- ((BX) + (AL))

**2、I/O 设备输入输出指令** in 和 out 相对与 CPU； in：传入 CPU；out：传出 CPU

① in al, port
② in ax, port
③ in al, dx
④ in ax, dx
⑤ out port, al
⑥ out port ax
⑦ out dx, ax
⑧ out dx, ax

**3、移位指令**

一次移动多位:

mov cl, shift_count

shl dst, cl

**4、rep 配合 C X**

**5、存入串指令 stos**

stosb 与 al 搭配；stosw 与 ax 搭配

**6、从串中取指令 lods**

lodsb 与 al 搭配；lodsw 与 ax 搭配

**7、串扫描指令 scas**

比较串中的数据与 al/ax 中的数据是否相等

scasb 与 al 搭配；scasw 与 ax 搭配

**8、中断**

① **01h 单字符输入**，送 **al**
② **02h 单字符输出**，显示 dl 中字符
③ **09h 字符串输出**，待显示字符串地址送 **dx**
④ **0ah 字符串输入**，输入字符串首地址送 **dx**

**8、mul / imul**

① (ax) <- (al) * src 8 位
② (dx, ax) <- (ax) * src 16 位

**9、div / idiv**

① (al) <- (ax) / src
② (ax) <- (dx, ax) / src