汇编指令集 第一部分

ADC指令
CMP命令
比较结果条件转移
DF标志
串传送指令
标志寄存器压栈 ——pushf/popf
碎片记忆

DEBUG

定义&基本命令

```
**DOS、WIn提供的实模式程序的调用工具 **
```

- 用R命令查看、改变CPU寄存器的内容
- 用D命令查看内存中的内容
- 用E命令改写内存中的内容
- 用U命令将内存中的机器指令翻译成汇编指令
- 用T命令执行一条机器指令
- 用A命令以汇编指令的格式在内存中写入一条机器指令

实例

1. 查看寄存器内容

```
-R  ;显示所有寄存器信息
-R ax  ;显示ax寄存器信息
-R ip  ;显示ip寄存器信息
```

2. 查看内存中内容

```
-D  ;查看预设地址后的内容
-d 1000:0  ;查看1000:0的128个内存单元内容
-d 1000:0 100  ;查看1000:0后的100个内存单元的内容
-d 1000:0 f  ;查看内存单元后f对应的单元内容数
```

3. 修改内存中内容

```
-E 1000:0 1 2 3 4 5  ;修改1000:0开始的单元内容
-E 1000:0 a b c d e 'a+b' "c++" "IBM"  ;修改内容可以是字符、数值、字符串等
-E 1000:0 ;可以用提问方式进行修改，回车结束、空格下一个
-E 1000:0 b8 01 00 b9 02 00 01 c8; 向内存中写入机器码
```

4. 查看内存中翻译后的汇编指令

```
-u  ;查看预设地址的16句汇编指令
-u 1000:0  ;查看1000:0的汇编指令
-u 1000:0 30  ;查看指定长度汇编指令
```

5. 写入汇编指令

```
-a  ;向预设地址中写入汇编指令
-a 1000:0  ;逐条向1000:0写入汇编指令
```

- 在8086PC机中，0-9FFF 的内存单元为主存储器的数据，A0000-BFFFF 为显存数据，C0000-FFFFF为ROM数据

编写、编译脚本

```
> edit xxx.asm      //编写脚本内容
> MASM xxx.asm      //编译命令文件
> LINK xxx.obj      //链接二进制文件
> xxx.exe           //运行脚本内容
> DEBUG xxx.exe     //逐步调试脚本
```

编写脚本注意事项：

- 在汇编程序中，数据不能以字母开头，例如A000H 需要写成 0A000H
- 在DUB中 'int 21' 该命令需要通过p命令执行，t不会执行
- 遇到大量循环确定无误后，想要一次性跳过，可以用g命令设置断点，快速执行，如 g 0012，则CS:0012前的内容均会被执行
- 同样我们可以采用p命令来将循环一次性全部执行完，直至cx=0
- 通常0:200 ~ 0:2ff 这段空间一般不使用

DEBUG 和 编译的不同

- 1. 对于'mov al,(0)' 这条命令，DEBUG将(idata)这个命令看做一个内存单元，编译器将(idata)这条命令看做一个内容为idata 的数据。编译的编译结果与我们通常采用的方式不太相同，所以通常我们有两种解决方案：

a. 将偏移地址传到bx,然后通过{bx}的方式调用

```
    ```
    mov ax 2000h
    mov ds,ax
    mov bx,0
    mov al,[bx]
    ```

    b. 通过显式调用地址：

    ```
    mov ax,2000h
    mov ds,ax
    mov al,ds:[0]
    ```
```

- 2. 程序基本框架

```
assume cs:code
code segment
    :
    数据
    :
start:
    :
    代码
    :
code ends
end start
```

- 3. 分段程序框架

```
assume cs:codesg,ds:datasg,ss:stacksg
datasg segment
    .
    .
datasg ends
stacksg segment
    .
    .
stacksg ends

codesg segment
start:  .
        .
codesg ends
end start
```

基本命令

1. 缺省数据段 —— DS

- 8086 CPU 不支持将数据直接送入段寄存器，即下面指令均为违法指令

```
mov ds,1000  ;数据段寄存器
mov cs,1000  ;代码段寄存器
mov ss,1000  ;堆栈段寄存器
mov es,1000  ;附加段寄存器
```

- **设置段寄存器的正确方式**

```
mov bx,1000
mov ds,bx
```

同样上述指令可以对cs\ss\es进行操作赋值

- E.g,寄存器给内存单元赋值

```
mov bx,1000
mov ds,bx
mov [0],al   ;将al的数据传到1000:0的内存单元
```

2. 移动指令 —— mov

- mov指令的基本形式：

```
mov ax,8     ;mov 寄存器，数据
mov ax,bx    ;mov 寄存器，寄存器
mov ax,[0]   ;mov 寄存器，内存单元
mov [0],ax   ;mov 内存单元 寄存器
mov ds,ax    ;mov 段寄存器 寄存器
mov ax,ds    ;mov 寄存器 段寄存器
mov [0],cs   ;mov 内存单元 段寄存器
mov cs,[0]   ;mov 段寄存器 内存单元
```

上述命令反向应该皆为可行的操作

3. 加减法指令 —— ADD/SUB

- add指令的基本形式

```
add ax,8     ;add 寄存器，数据
add ax,bx    ;add 寄存器，寄存器
add ax,[0]   ;add 寄存器，内存单元
add [0],ax   ;add 内存单元，寄存器
```

- sub指令的基本形式

```
sub ax,9    ;sub 寄存器，数据
sub ax,bx   ;sub 寄存器，寄存器
sub ax,[0]  ;sub 寄存器，内存单元
sub [0],ax  ;sub 内存单元，寄存器
```

- 不能对段寄存器直接进行加减法赋值

4. 出入栈指令 —— push/pop

```
push ax  ;修改地址：220FE-220FF 5CCA
push bx  ;修改地址：220FC-220FD 6024
pop ax   ;将6024 弹出到 Ax
pop bx   ;将5CCA 弹出到 Bx
push [0004]  ; 将地址FFFF:[0004] 的内容F030进栈到 2200：00FE
push [0006]  ; 将地址为FFFF:[0006] 的内容342F进栈到 2200：00FC
```

5. 定义数据 —— db/dw/dd(dup)

- DB命令（define byte）
- 数据长度：1 字节，仅占一个地址空间

```
datasg segment
    db 'BaSiC'    //占据内存datasg:0 - datasg:4 的内存单元
    db 'iNfOrMaTiOn'    //占据内存datasg:5 - datasg:16 的内存单元
datasg ends
```

- DW命令（define word）
    - 数据长度：2字节，占两位内存单元
    - 定义后的数据默认存储在cs:2n的位置，在每次的循环的时候，需要cs:bx,(bx)=(bx)+2 如下面的例子

```
dw 0123h,0456h,0789h,0abch,0defh
    // 上面的数据分别存储在 cs:0,cs:2,cs:4,cs:6,cs:8的位置
dw 0,0,0,0,0,0,0,0
    // 若设置为零，默认为1个字节大小
```

- DD命令（double define）
    - 数据长度：4字节，占4位地址空间
- DUP 命令
    - 用于生成重复的数据，命令格式如下

```
db 重复的次数 dup (重复的字节型数据)
dw 重复的次数 dup (重复的字型数据)
dd 重复的次数 dup (重复的双字型数据)
```

- 综合

```
asume cs:codesg

data segment
    db
'1975','1976','1977','1978','1979','1980','1981','1982','1983'
    db
'1984','1985','1986','1987','1988','1989','1990','1991','1992'
    db '1993','1994','1995'

    dd
16,22,382,1356,2390,8000,16000,24486,50065,97479,140417,197514
    dd
345980,590827,803530,1183000,1843000,2759000,3753000,4649000,593
7000

    dw
3,7,9,13,28,38,130,220,476,778,1001,1442,2258,2793,4037,5635,822
6
    dw 11542,14430,15257,17800
data ends
table segment
    db 21 dup ('year summ ne ??')
table ends
```

6. 定位指令 —— bx/si/di/bp

- 上面四种寄存器(bx/si/di/dp)可以称为寻址寄存器，只有他们能够出现在(..)表示偏移地址

```
错误表达：
mov ax,[cx]
mov ax,[ax]
mov ax,[dx]
mov ax,[ds]
```

- 上面四种寄存器的组合方式：
    - bx/si/di/dp 单独出现
    - bx + si
    - bx + di
    - bp + si (默认段地址为ss)
    - bp + di（默认段地址为ss)

```
均遵循以下表达：
mov ax,[bx]
mov ax,[si]
mov ax,[di]
mov ax,[bp]
mov ax,[bx+si]
mov ax,[bx+di]
mov ax,[bp+si]
mov ax,[bp+di]
```

- 上面的单独表达可以表示出一维数组
- 两个寄存的组合表达可以表示出二维数组

7. 更复杂的定位 —— idata

- 在上面的单独和两个寄存器的组合方式的基础上加入idata，我们可以表示出更高结构的数据：结构体，单独使用idata是直接偏移地址寻址

```
mov ax,[idata]
结构体表达：
mov ax,[bx+idata]
mov ax,[si+idata]
mov ax,[di+idata]
mov ax,[bp+idata]
    //[bx].idata 用于结构体
    //idata[si],idata[di] 用于数组
    //[bx][idata] 用于二维数组
mov ax,[bx+si+idata]
mov ax,[bx+di+idata]
mov ax,[bp+si+idata]
mov ax,[bp+di+idata]
    //[bx].idata[si] 用于表格结构的数组项
    //idata[bx][si] 用于二维数组
```

- 复杂表达的等效形式

```
mov ax,[200+bx]
mov ax,200[bx]
mov ax,[bx].200

一般形式：
    mov ax,[a+b+c]
= = >
    mov ax,a[b].c
    mov ax,[b].a[c]
    mov ax,a[b][c]
    mov ax,[a][b][c]
注意：
    一般bx寄存器会写在[]中，通常idata项会写在括号外
```

7. 数据长度规范 —— ptr

- 在未指定寄存器，仅对内存单元和临时变量进行操作时，可以通过“word ptr”、“byte ptr"来对操作长度进行指明

```
mov word ptr ds:[0],1
inc word ptr [bx]
inc word ptr ds:[0]
add word prt [bx].2

mov byte ptr ds:[0],1
inc byte ptr [bx]
inc byte ptr ds:[0]
add byte prt [bx].2
```

8. 除法指令 —— div

- 情况1：8位除数+16被除数
  - 被除数2字节存放于AX
  - 计算得到的商存放于al，余数存放于ah

```
mov dx,1
mov ax,86A1H
mov bx,100
div bx
```

- 情况2：16位除数+32位被除数
  - 被除数4字节高字节、低字节分别放于 DX AX
  - 计算得到的商放于AX，余数放于DX

```
mov ax,1001
mov bl,100
div bl
```

- 选择哪种方式由div后面的数据大小决定

9. 转移指令 —— jmp/jcxz/loop

- 可以修改IP，或者同时修改CS、IP的指令称为转移指令
- 基本分类：
  - 修改IP的段内转移，修改CS:IP的段间转移
  - 短转移：移动范围（-128-127），近转移：移动范围（-32768 - 32767）
  - 无条件转移、条件转移、循环指令、过程、中断
- offset ——取得标号的偏移地址

```
assume cs:codesg
codesg segment
    s:  mov ax,bx
        mov si, offset s
        mov di, offset s0
        mov ax,cs:[si]
        mov cs:[di],ax
    s0: nop
        nop
codesg ends
end s
```

a. JMP 无条件转移

- short 命令
  - 命令格式

```
jmp short 标号
```

  - 命令功能

    (IP)=(IP)+8位位移

    1. 8位位移=标号处地址 - jmp指令后的第一个字节地址
    2. short 指明此处为8位位移
    3. 8位位移的范围为-128~127，补码表示
    4. 8位位移由编译程序计算得到
- near 命令
  - 格式

```
jmp near ptr 标号
```

  - 命令功能

    (IP)=(IP)+16位位移

    1. 8位位移=标号处地址 - jmp指令后的第一个字节地址
    2. 8位位移的范围为-128~127，补码表示
    3. 8位位移由编译程序计算得到
- FAR 命令
  - 格式（在机器码中显示为CS：IP型跳转）

```
jmp far ptr 标号
```

- 16位 REG
  - 格式（等效于直接修改IP）

```
jmp 16位reg
```

- word/dword 命令
  - 格式

```
jmp word ptr 内存单元地址(段内)
    ;内存单元处存放着IP
jmp dword ptr 内存单元地址(段间)
    ;(CS)=(内存单元地址+2)
    ;(IP)=(内存单元地址)
```

- 实例1

```
mov ax,0123H
mov ds:[0],ax
jmp word ptr ds:[0]
    ;执行完毕IP=0123H
```

- 实例2

```
mov ax,0123H
mov ds:[0],ax
mov word ptr ds:[2],0
jmp dword ptr ds:[0]
    ;执行完毕后CS:IP=0000:0123
```

b. JCXZ 零转移

- 命令格式

```
jcxz 标号
    ;若(cx)=0，转移到标号处执行
    ;其功能相当于if((cx)==0 jmp short 标号
    ;段内跳转
```

c. LOOP 非零转移

- 命令格式

```
loop 标号
    ;(cx)=(cx)-1.如果(cx)≠0,则转移到标号处
    ;其功能等效于(cx--);if((cx)≠0) jmp short 标号;
```

10. 转移指令 ——CALL/RET （子程序设计）

RET/RETF 命令

- ret指令用栈中的数据，修改IP的内容，实现近转移
- retf指令用栈中的数据，修改CS、IP的内容，实现远转移

```
;ret命令等效于
    POP IP
;retf命令等效于
    POP IP
    POP CS
```

CALL 指令

- 短转移

```
call 标号
    ;该命令等效于
    ;push IP
    ;jmp near ptr 标号
```

- 长转移

```
call far ptr 标号
    ;该命令等效于
    ;push CS
    ;push IP
    ;jmp far ptr 标号
```

- 寄存器转移

```
call 16位reg
    ;该命令等效于
    ;push IP
    ;jmp 16位reg
```

- 内存单元转移

```
call word ptr 内存单元地址
    ;该命令等效于
    ;push IP
    ;jmp word ptr 内存单元地址
call dword ptr 内存单元地址
    ;该命令等效于
    ;push CS
    ;push IP
    ;jmp dword ptr 内存单元地址
```

Call + Ret 子程序编写

```
    call s  ;调用子程序s

s:  .. ..
    .. ..
    ret
```

- 子程序中需要从外部传进来的参数，通常是存放在地址中而不是寄存器中，通过读取地址(di)(si)来读取数据，在子程序中通常会有寄存器冲突的问题，我们需要在子程序开始之前将使用到的寄存器数据存到栈中，然后在子程序结束的时候重新读取出来

11. 传送指令

有效地址传送 —— LEA

- 命令格式

```
LEA REG,SRC
    ;REG为16位通用寄存器
    ;SRC为内存操作数
    ;例如
LEA BX,DS:[100]
LEA BX,BUF
    ;传入DS:[100]的偏移地址
    ;上述命令执行结果为BX=100H，或BUF的偏移地址
```

交换指令 —— XCHG

- 命令格式

```
XCHG OPRD1,OPRD2
    ;源操作数：寄存器、存储器
    ;目标操作数：寄存器、存储器
    ;执行完后OPRD1与OPRD2的内容交换
    ;不允许使用段寄存器，其中必有一个寄存器
    ;例如
XCHG BX,[BP+SI]
    ; (BX) = 6F30H
    ; (BP) = 0200H
    ; (SI) = 0046H
    ; (SS) = 2F00H
    ; (2F246H) = 1234H
    ;程序执行完的结果为
    ; (BX)=1234H
    ; (2F246H)=6F30H
```

查表指令 —— XLAT

标志寄存器

ZF 标志

- 零标志位，位于flag的第6位，判断相关命令结果
- 相关命令：add，sub，mul，div，inc，or，and（运算指令）
- 若运行结果为0，那么ZF=1；若运行结果不为0，那么ZF=0

PF标志

- 奇偶标志位，位于flag的第2位，判断结果中所有bit位的1的个数
- 相关命令：（运算指令）
- 若结果的所有bit位中1的个数为偶数，pf=1，若数量是奇数，则pf=0

SF标志

- 符号标志位，位于flag的7位
- 相关命令：add，sub，mul，div，inc，or，and（运算指令）
- 用于提醒计算结果的第一位是否为1，若作为无符号计算，则该位无实际意义

CF标志位（无符号数）

- 进位标志位，位于flag的第0位
- 相关命令：（运算指令）
- 对于无符号数运算，该标志位记录了运算结果的最高位的借位或进位

OF标志位（有符号数）

- 溢出标志位，位于flag的11位
- 对于8位有符号数数据，机器能表示的范围是—128～127，如下是一个OF=1的例子

```
mov al, 98
add al, 99
```

如下是一个反码表示的计算结果

```
mov al, 0F0H ; 传入-16的补码
add al, 088H ; 加上-120的补码
```

- 对于16位有符号数，机器能表示的范围是 -32768 ～32767，如下OF=1

ADC指令

- 带进位加法指令

```
指令格式：adc 操作对象1，操作对象2
功能：操作对象1=操作对象1+操作对象2+CF
```

- 利用该功能实例（使得分步加法，更加便捷），计算1EF0001000H+2010001EF0H放在ax（最高16位），bx（次高16位），cx（低16位）
    - （1）先将低16位相加，完成后记录本次相加的进位
    - （2）再将次高16位和CF相加，完成后记录本次进位
    - （3）最后高16位和CF相加，完成后记录进位

```
mov ax,001EH
mov bx,0F000H
mov cx,1000H
add cx,1EF0H
adc bx,1000H
adc ax,0020H
```

CMP命令

- 其功能等效于减法指令，但是不保存结果，只是对寄存器产生影响

```
cmp ax,ax
    ;ZF=1，PF=1,SF=0,CF=0,OF=0
```

- 判断依据-1
    - ZF=1，说明(ax)=(bx)
    - ZF=0，说明(ax)≠(bx)
    - CF=1，说明(ax)<(bx)
    - CF=0，说明(ax)>=(bx)
    - CF=0且ZF=0，说明(ax)>(bx)
    - CF=1或ZF=1，说明(ax)<=(bx)

```
cmp ah,bh
```

- 判断依据-2
    - SF=1,OF=0,说明(ah)<(bh)
    - SF=1,OF=1,说明(ah)>(bh)
    - SF=0,OF=1,说明(ah)<(bh)
    - SF=0,OF=0,,说明(ah)>=(bh)

比较结果条件转移

指令	含义	检测标志位
je	等于转移	ZF=1
jne	不等于转移	ZF=0

jb	低于转移	CF=1
jnb	不低于转移	CF=0
ja	高于转移	CF=0且ZF=0
jna	不高于转移	CF=1且ZF=1

- e:equal(等于)
- b:below(低于)
- a:above(高于)
- n:not
- 当cmp命令与比较转移结合使用时，功能与高级程序中的IF语句相似

DF标志

- 方向标志位，位于第10位，用于串操作指令中si、di的增减
    - DF=0,每次操作后si,di递增
    - DF=1,每次操作后si,di递减

串传送指令

- movsb 指令
    - 格式

```
movsb
```

    - 功能(等效于)

```
mov es:[di],byte ptr ds:[si]

如果df=0:
inc si
inc di

如果df=1:
dec si
dec di
```

- movsw指令
    - 指令

```
movsw
```

    - 功能等效于

```
mov es:[di],word ptr ds:[si]

如果df=0:
add si,2
add di,2

如果df=1:
sub si,2
sub di,2
```

- 完整指令使用，串处理次数由cx决定

```
rep movsb
    ;等效于
    ;s: movsb
    ;   loop s
rep movsw
    ;s: movsw
    ;   loop s
```

- 串处理总结信息
    - 1. 传送的原始位置：ds:si
      2. 传送的目的位置：es:di
      3. 传送的长度：cx
      4. 传送的方向：df

标志寄存器压栈 ——pushf/popf

- pushf是将标志寄存器的值压栈
- popf是从栈中弹出数据送入标志寄存器

碎片记忆

1. 小端法存放

   低字节存放低地址

2.