80X86汇编语言程序设计

80X86 Assembly Language Programming

第2章 寻址方式

金良海 教授

计算机学院 医学图像信息研究中心

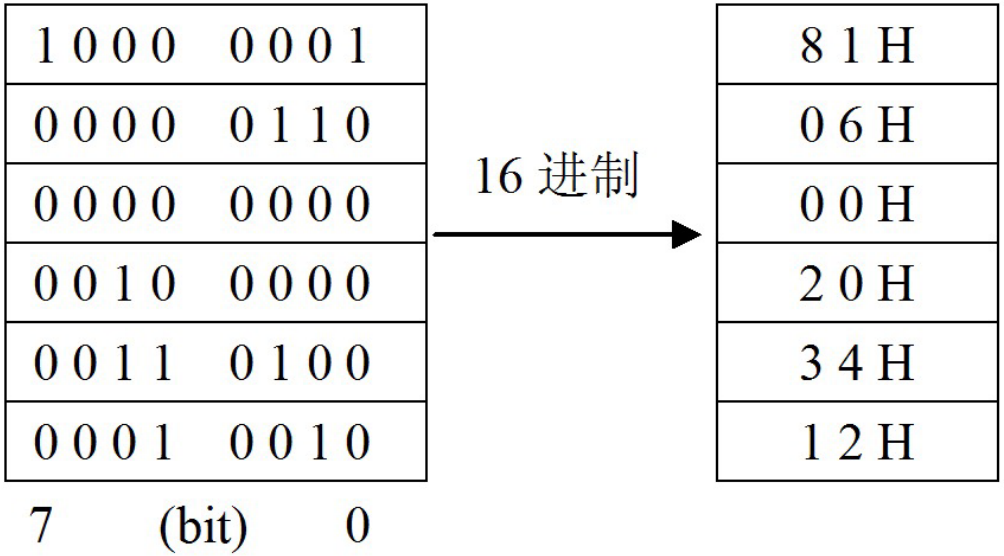
Email: lianghaijin@hust.edu.cn

每条指令由操作码和操作数组成：

操作码	操作数 (可能有多个)

操作码：表示要执行什么操作、完成什么功能

操作数：表示操作（要执行）的对象



ADD X, 1234H ;变量X的地址是数据段DS内2000h

ADD word ptr DS:[2000h], 1234h

问题

- 操作数在哪里？
- CPU如何知道操作数的地址？
- C编译器会怎样翻译C语言的变量？

问题

```
int   i;
int   A[10];
int   j;
int   B[20][10];
```

- C编译器会怎样翻译变量？

- 从程序中可分析出变量的相对位置（偏移地址）
- 是全局的统一的相对位置吗？
  程序一运行，其所有变量都要占相应空间吗？
  函数执行完，局部变量占用的空间继续保持吗？
- 指令中只会是偏移地址吗？
  A[i] 翻译成什么？B[i][j] 又翻译成什么？

第2章　寻址方式

本章的学习内容

介绍80X86汇编语言的6种寻址方式：

- 寄存器寻址
- 寄存器间接寻址
- 变址寻址
- 基址加变址寻址
- 立即寻址
- 直接寻址

寻址方式的有关问题

第2章　寻址方式

本章的学习重点

1. 6种寻址方式的使用格式及语法规定；
2. 6种寻址方式地址表示的含义及应用；
3. 寻址方式的有关问题。

本章学习的难点

直接寻址，寄存器间接寻址，变址寻址和基址加变址寻址的使用格式及功能。

概述 (1)

- 操作数在哪？

  操作数可以存放在
  - CPU内的寄存器
  - 主存贮器
  - I/O设备端口

  操作数在主存时：段址、段内偏移

- 操作数的类型？　字节/字/双字？

概述 (2)

当操作数存放在主存贮器时，怎样知道它存放在内存的哪个单元？也就是说存放操作数的内存单元的地址？

寻找操作数存放地址的方式称为寻址方式。

概述 (3)

80X86的指令有3种形式：

(1) 0操作数指令（指令中没有操作数）

这种类型的指令，操作数是隐含的。

例： POPF　将栈顶的内容弹出送入到状态寄存器EFLAGS的低16位

PUSHF　将状态寄存器EFLAGS的低16位压入堆栈

虽然这2条指令不带操作数，但其隐含的操作数是状态寄存器。

概述 (4)

(2) 1操作数指令（指令中只有1个操作数）

例： INC WORD PTR [SI]

POP AX

(3) 2操作数指令（指令中有2个操作数）

例： ADD AX, BX

MOV AX, [SI]

带操作数的指令，操作数就在指令码后面所带的参数中。但是，这些参数到底表示的是寄存器还是主存贮器，需要具体分析，也就是寻址方式。

概述 (5)

双操作数的指令格式　　操作符 OPD, OPS

ADD　AX，BX

目的操作数地址　　源操作数地址

(OPD) + (OPS) → OPD
(AX) + (BX) → AX

Question：操作结束后，运算结果保存在哪？
源操作数是否变化？

2.1 寄存器寻址 (1)

使用格式： R **(指令操作码后面的参数是寄存器名)**
功　能： 寄存器R中的内容即为操作数。
说　明： 除个别指令外，R可为任意寄存器。

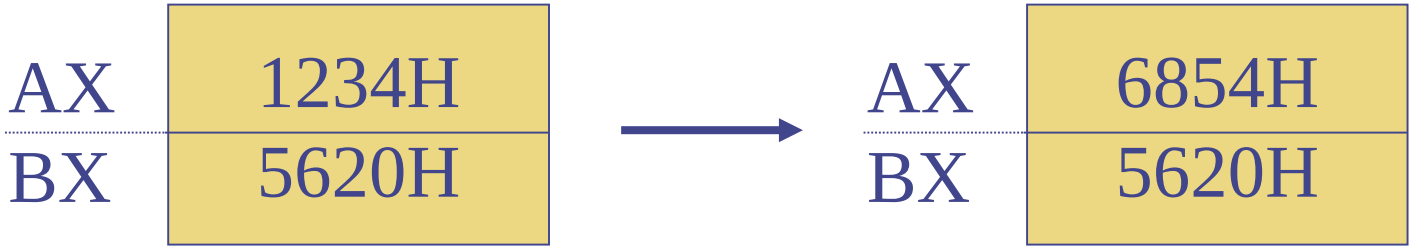
例1：DEC　BL

执行前：　(BL)=43H
执　行：　(BL) –1 = 43 H –1 = 42H → BL
执行后：　(BL)=42H

Question：　操作数在哪？操作数类型是什么？

2.1 寄存器寻址 (2)

例2：ADD AX , BX



执行前： (AX)=1234H, (BX)=5620H

执行： (AX)+(BX) = 6854H → AX

结果： (AX)=6854H , (BX)=5620H

例3：MOV AX，BX

2.1 寄存器寻址 (3)

例4：ADD EAX , EDX

执行前： (EAX)=12345678H,

(EDX)=0A004321H

执行： (EAX)+(EDX) = 1C349999H → EAX

执行后： (EAX)=1C349999H

(EDX)=0A004321H

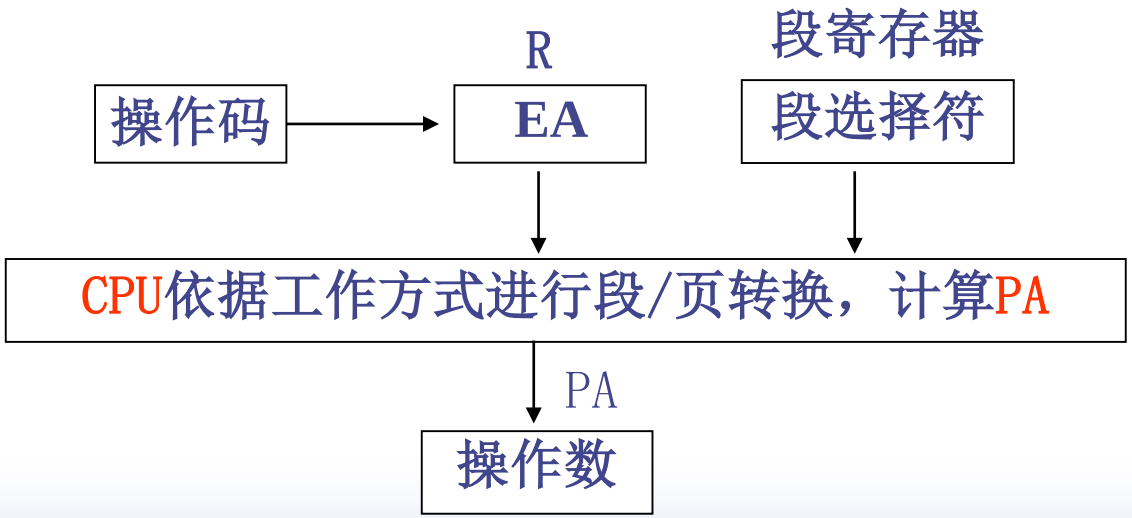
Question： 指令 MOV AX, BH 正确吗？为什么？

2.2 寄存器间接寻址 (1)

格式：[R] (指令操作码后面的参数是带中括号的寄存器名)

功能：操作数在内存中,操作数的偏移地址在寄存器R中。即（R)为操作数的偏移地址.

例如：MOV AX, [SI]



寄存器间接寻址方式的寻址过程

2.2 寄存器间接寻址 (2)

- R 可以是:
  - 8个32位通用寄存器中的任意一个
    - EAX, EBX, ECX, EDX,ESI,EDI,ESP,EBP
  - 4个16位通用寄存器中的一个
    - BX,SI,DI,BP
- 操作数的偏移地址在指令指明的寄存器中
- 操作数所在的段：

  若R为BP、EBP、ESP,则系统默认操作数在堆栈中，等同于 SS:[R];

  其它情况下，默认操作数在DS所指示的段中

- 操作数的类型：未知

2.2 寄存器间接寻址 (3)

例1：MOV AX, [SI]

执行前 (AX)=0005H

(SI) =20H

DS:(20H)=1234H

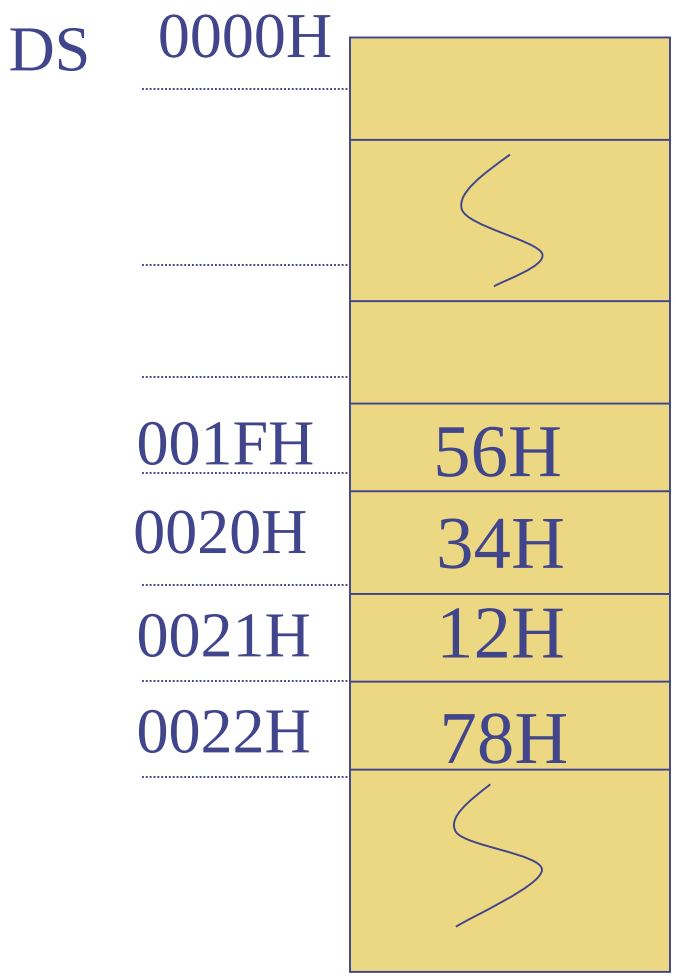
执行后 (AX)=1234H

(SI) = 20H

问：MOV CL, [SI]

(CL) = ?

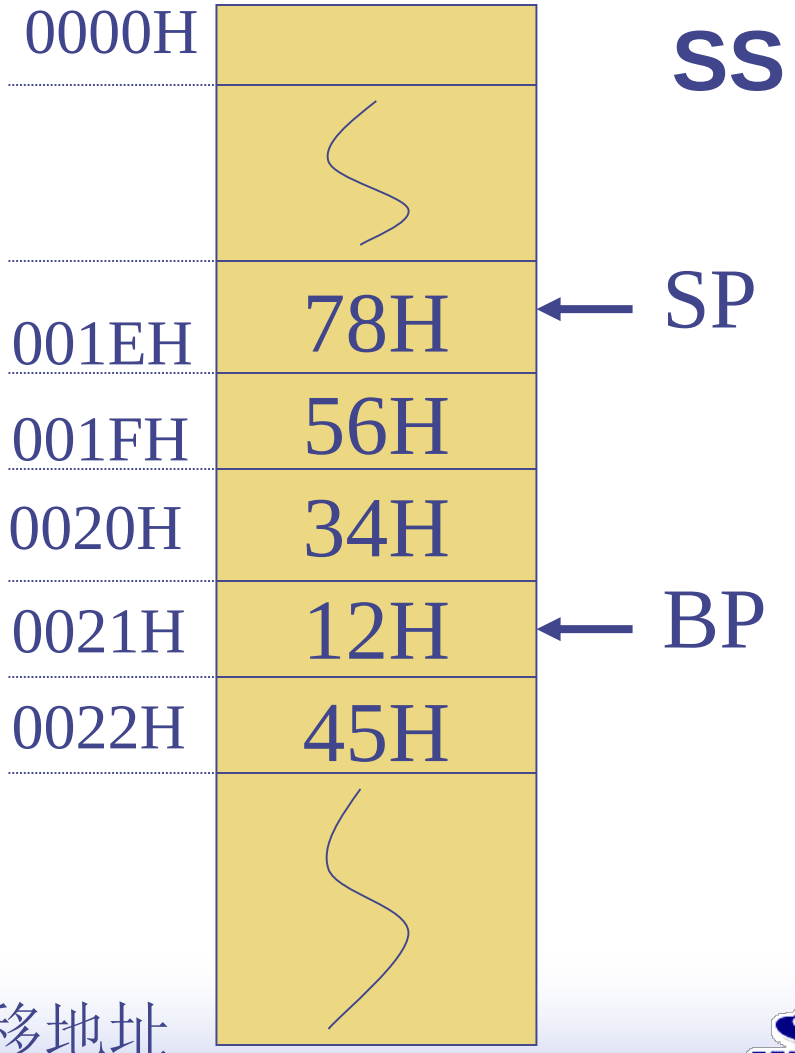
操作数的类型是如何确定的，你猜出来了吗？



2.2 寄存器间接寻址 (4)

例2：MOV AH, [BP]
执行前 (AX) = 0005H
(BP) = 21H
SS:(BP) = 12H

执行后 (AX)=1205H
(BP) = 21H



2.2 寄存器间接寻址 (5)

例3：分析在执行如下程序段后，

(EAX)=?　(BX)=?　(CX)=?

```
MOV   EAX,  -2
MOV   [ESP], EAX
POP    BX
```

2.2 寄存器间接寻址 (6)

~~MOV AX, [CX]~~

比较：

MOV AX , BX

MOV AX, [BX]

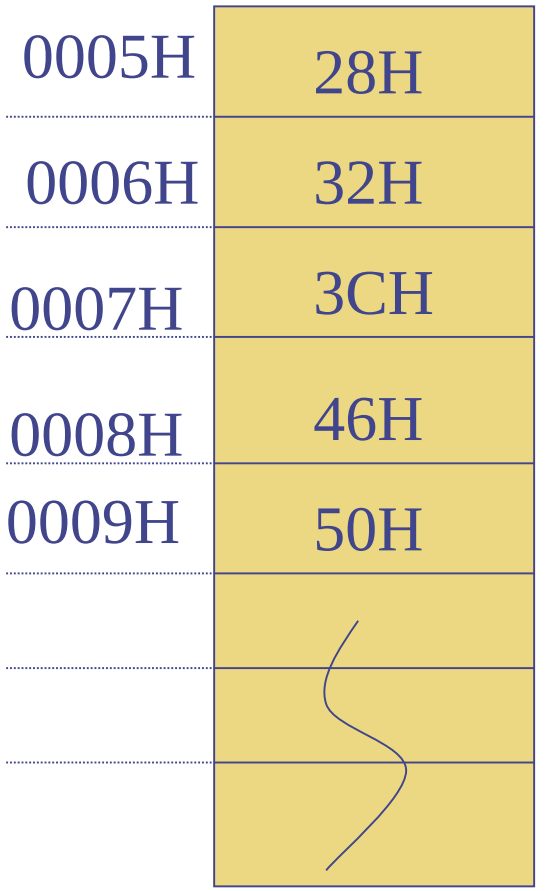
2.2 寄存器间接寻址 (7)

例：设 **BUF DB 40,50,60,70,80**
即以**BUF**为首址的字节区中存放有**5**个数据，求它们的和。

算法分析：……

和？
循环次数？
数据位置？

BX **0005** **([BX])**



共同特点：单元中的内容无规律，
但单元之间的地址有规律。

c2_034j1.asm
c2_034j2.asm

2.2 寄存器间接寻址 (8)


```
.386
SEG1   SEGMENT  USE16  STACK
       DB 200  DUP(0)
SEG1   ENDS

SEG2   SEGMENT   USE16
BUF    DB  40, 50, 60, 70, 80
RES    DB  ?
SEG2   ENDS

SEG3   SEGMENT  USE16
  ASSUME  CS:SEG3,
          DS:SEG2,SS:SEG1
START:
      MOV   AX , SEG1
      MOV   DS , AX
      XOR   CX , CX
      MOV   AH , 0
      MOV   BX , OFFSET  BUF
LP:   CMP  CX,5
      JGE   EXIT
      ADD   AH, [BX]
      INC    BX
      INC    CX
      JMP    LP
EXIT: MOV   RES,AH
MOV   AX ,  4C00H
      INT   21H
SEG3  ENDS
      END   START
```

Question: ADD AH, [BX]

可否换成 MOV AH, BX

Question: 此程序有问题吗??

2.2 寄存器间接寻址 (9)

```
int   buf[5]={40,50,60,70,80} ;
int   i;
int  *p;
int   result=0;
p = buf;
for(i=0;i<5;i++)
{
    result += *p++;
 }
```

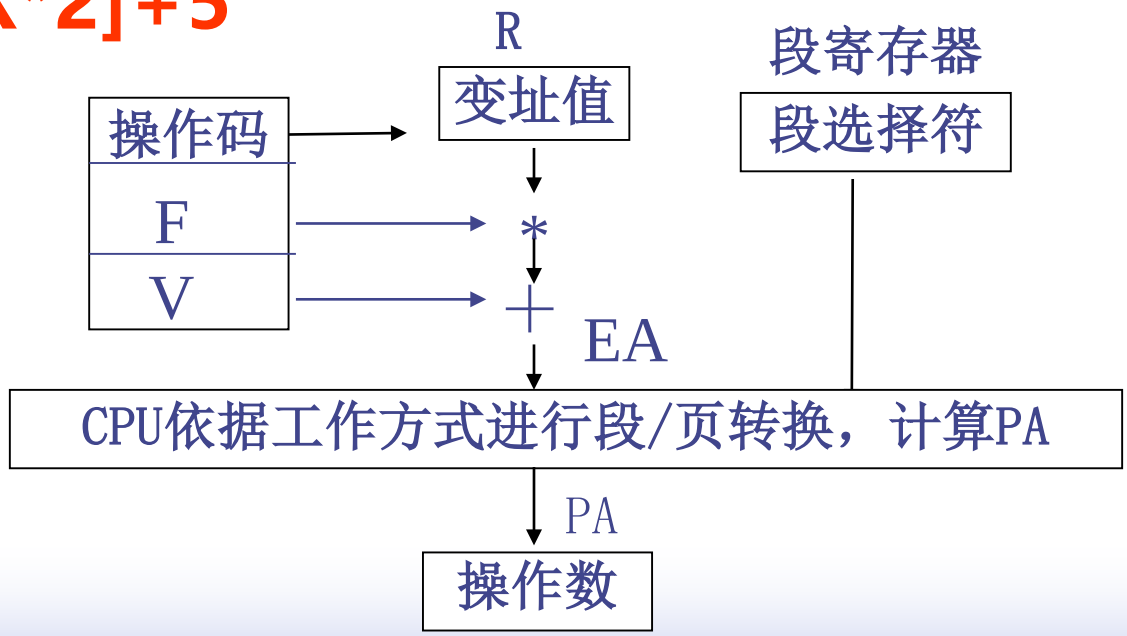
假设buf的起始地址为2000H，buf[2]的地址？

2.3 变址寻址 (1)

格式：[R*F+V] 或 [R*F]+V 或 V[R*F]

功能：R中的内容×F+V为操作数的偏移地址。

例如：MOV AL, [EBX*2]+5



2.3 变址寻址 (2)

格式：[R*F+V] 或 [R*F] +V 或 V[R*F]

- R 可以是:

  (1)8个32位通用寄存器中的任意一个
  EAX, EBX, ECX, EDX,ESI,EDI,ESP,EBP

  (2)4个16位通用寄存器中的一个
  BX,SI,DI,BP

- F 可为 1,2,4,8
  当R是16位寄存器或为ESP时，F只能为1

2.3 变址寻址 (3)

格式：[R×F+V] 或 [R×F]+V 或 V[R×F]

- V可为数值常量，也可以为一个变量。
- 当V为变量时，取该变量的偏移地址参与运算。

例：MOV AL, BUF[EBX*2]

0005H	11H	BUF
0006H	22H	
0007H	33H	

- 当R是16位寄存器时，V不超过16位；
- 当R是32位寄存器时，V不超过32位；

V是二进制补码表示的有符号数

2.3 变址寻址 (4)

- 操作数所在的段（V是常量）

  若R为BP、EBP、ESP,则系统默认操作数在堆栈中，等同于SS:[R];

  其它情况下，默认操作数在DS所指示的段中，等同于DS:[R];

2.7 寻址方式的有关问题(1)

- **操作数所在的段(V是变量或标号)**

段地址是与变量所在的段相关联的段寄存器。

过程如下：编译器在ASSUME语句中查找与定义变量的段相关联的段寄存器（若没找到则报错），然后以该段寄存器为段地址。

2.3 变址寻址 (5)

例1：MOV AL, [EBX*2]+5

执行前：(AL)=18H, (EBX)=1100H

DS: (2205H)=55H

执行：EA = (EBX)*2+5 = 2205H

DS: (2205H) → AL

执行后：(AL)=55H

(EBX)、(DS)、DS: (2205H)不变

2.3 变址寻址 (6)


```
.386
STACK  SEGMENT  USE16  STACK
          DB 200  DUP(0)
STACK  ENDS

SEG1   SEGMENT   USE16
RES     DD  ?
BUF     DD  40, 50, 60, 70, 80
SEG1   ENDS

CODE    SEGMENT  USE16
   ASSUME  CS:CODE,
                DS:SEG1,SS:STACK
START:
       MOV   AX , SEG1
       MOV   DS , AX
       MOV   EBX , 0
       MOV   EAX , 0
LP:    CMP  EBX, 5
       JGE   EXIT
       ADD   EAX,BUF [EBX*4]
       INC    EBX
       JMP    LP

EXIT:  MOV   RES, EAXH
        MOV   AX , 4C00H
        INT    21H
CODE   ENDS
        END  START
```

ADD EAX, DS:[EBX*4+4]

能否将程序中的**EBX**改成**BX**?

2.3 变址寻址 (7)


```
int   buf[5]={40,50,60,70,80};
int   i;
int   result=0;
for  (i=0;i<5;i++)
{
    result += buf[i];
}
```

思考:

(EBX) 中的值就是 变量i的值，
为何C语言中的访问方式是 buf[i]，
而汇编语言中是 buf[EBX*4] ？

2.3 变址寻址 (8)

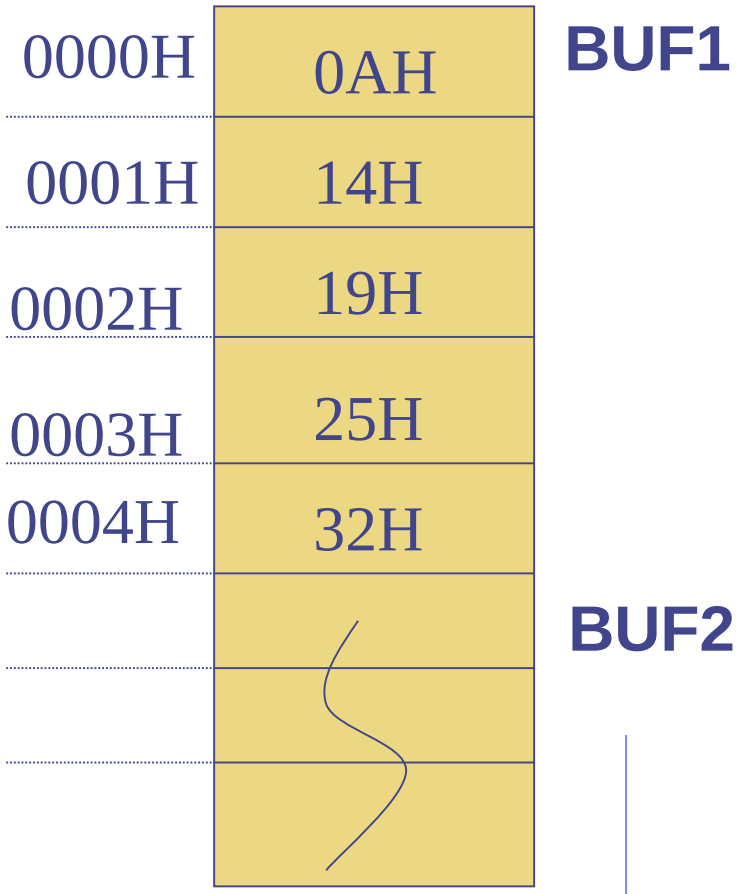
变址寻址和寄存器间接寻址有何异同？

```
        MOV   EBX , 0
        MOV   EAX , 0
LP:     CMP  EBX, 5
        JGE   EXIT
        ADD   EAX,BUF [EBX*4]
        INC    EBX
        JMP    LP
```

```
        MOV   ECX , 0
        MOV   EAX , 0
        MOV   EBX , OFFSET  BUF
LP:     CMP  ECX, 5
        JGE   EXIT
        ADD  EAX, [EBX]
        ADD  EBX,4
        INC    ECX
        JMP    LP
```

2.3 变址寻址 (9)

例2: 设 BUF1 DB 10,20,25,37,50
即以BUF1为首址的字节区中存放有5个数据，将它们拷贝到以BUF2为首址的字节区。



与数组类比：A[0], A[1], A[2],…
B[0], B[1], B[2]

I　　A[I]

BX　0000　BUF1[BX]

BUF2[BX]

2.3 变址寻址 (10)

例2：以BUF1为首址的字节区中存放有5个数据，将它们拷贝到以BUF2为首址的字节区。

```
for (i=0;i<5;i++)  BUF2[i] = BUF1[i];
```

```
        MOV  BX, 0
MAINP:  CMP  BX,5
        JGE  EXIT
        MOV  AL,  BUF1[BX]
        MOV  BUF2[BX], AL
        INC  BX
        JMP  MAINP
EXIT:
```

用变址寻址实现的程序段

2.3 变址寻址 (11)

例2：以BUF1为首址的字节区中存放有5个数据，将它们拷贝到以BUF2为首址的字节区。

```
        MOV  SI, OFFSET BUF1
        MOV  DI, OFFSET BUF2
        MOV  CX, 5

MAINP:  MOV  AL,  [SI]
        MOV  [DI], AL
        INC   SI
        INC   DI
        DEC  CX
        JNZ   MAINP
EXIT:
```

用寄存器间接寻址的程序段

2.3 变址寻址 (12)

例3：以BUF1为首址的双字区中存放有5个数据，将它们拷贝到以BUF2为首址的字节区。

```
for (int i=0;i<5;i++)  BUF2[i] = BUF1[i];
```

```
            MOV  EBX, 0
MAINP:      CMP  EBX,5
            JGE   EXIT
            MOV  EAX,  BUF1[EBX*4]
            MOV  BUF2[EBX*4], EAX
            INC    EBX
            JMP   MAINP
EXIT:
```

2.3 变址寻址 (13)


```
.386
data  segment  use16
 x      db  10H, 20H, 30H
x2      db  11H, 22H
data  ends
stack  segment  use16 stack
y       db    40H, 50H
        db    200 dup(0)
stack ends
code segment  use16
        assume cs:code, ds:data, ss:stack
z       db 60H,70H,80H
Begin :  ……
```

```
mov  BX, 0
mov  ah, x[BX]
mov  al,  y[BX]
mov  cl,  z[BX]
```

汇编后的结果：

```
MOV  ah, [BX]
MOV  al, SS:[BX]
MOV  cl, CS:[BX]
```

C2_035j1.asm

操作数的段属性

2.3 变址寻址 (14)


```
data  segment
 x      db  10h, 20h, 30h
x2      db  11h, 22h
data  ends
stack  segment  stack
y       db    40h, 50h
        db    200  dup(0)
stack ends
code segment
        assume cs:code, es:data, ss:stack
z       db 60h, 70h, 80h, 0A0h, 0, 0B0h
Begin :
```

```
mov  ax, data
mov  ds, ax
mov  ax, code
mov  es, ax

mov  BX,  1
mov  ch,  [BX]
mov  ah, x[BX]
mov  al,  y[BX]
mov  cl,  z[BX]
```

操作数的段属性

```
MOV AH,  SS:x[BX]
MOV AL,  x2[BX]
MOV CL,  CS:x2[BX]
```

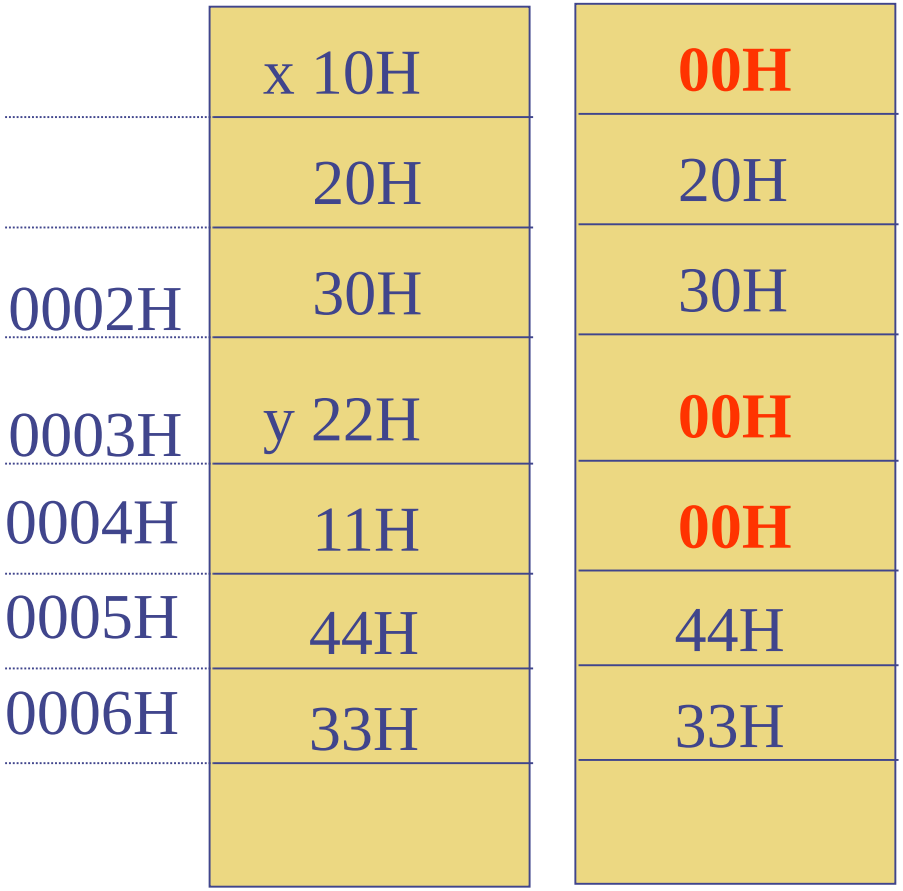
结果是什么？

2.3 变址寻址(15)

```
Data  segment
 x      DB   10H, 20H, 30H
 y      DW  1122H, 3344H
Data  ends
Code segment
        assume cs:code, ds:data
Begin :……
        MOV  BX, 0
        MOV  x[BX], 0
        MOV  y[BX], 0
        ……
```

C2_035j2.asm

应用 : C2_036j.asm



为什么两条指令执行后的结果不同？
翻译出的机器指令各是什么？

```
mov   byte ptr [BX], 0
mov  word ptr [BX+0003], 0
```

操作数的类型

2.3 变址寻址(16)

Problem: 以BUF1为首址的字缓冲区中存放有5个数据，将它们拷贝到以BUF2为首址的字缓冲区：

```
short  int  *p = BUF2;
for (int i=0;i<5;i++)  *p++  = BUF1[i];
```

请将这个C程序段翻译成汇编语言程序。

```
                MOV  SI, OFFSET BUF2
                MOV  EBX, 0
MAINP:          CMP  EBX, 5
                JGE   EXIT
                MOV  AX,  BUF1[EBX*2]
                MOV  [SI], AX
                ADD   SI, 2
                INC    EBX
                JMP   MAINP
EXIT:
```

2.4 基址加变址寻址 (1)

格式： [BR＋IR×F+V]

或 V[BR][IR×F]　或　V [IR×F] [BR]

或 V[BR+IR×F]

功 能：操作数的偏移 ＝变址寄存器IR中的内容×比例因子F +位移量V+基址寄存器BR中的内容 。

EA = (IR)*F + V + (BR)

例如： MOV EAX, -6[EDI*2][EBP]

2.4 基址加变址寻址 (2)

格式：[BR＋IR×F+V]

- F 可为 1，2，4，8
- 当使用16位寄存器时

  BR 是 BX、BP 之一；

  IR 是 SI、DI 之一，F 只能为1

当 BR=BX时，默认操作数在DS所指示的段中；

当 BR=BP时，默认操作数在SS所指示的段中。

2.4 基址加变址寻址 (3)

◆当使用32位寄存器时

BR可以是 EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP 之一；

IR 可以是除ESP外的任一32位寄存器；

未带比例因子的寄存器是 BR;

当没有比例因子的时候，写在前面的寄存器是BR.

当BR为ESP,EBP时，默认段是SS.

5[EBX][ESP]　　5[ESP][EBX]

5[EBX][ESP*2]　　5[ESP*2][EBX]

?

2.4 基址加变址寻址 (4)

说明：当V是变量或标号时，系统默认操作数的段地址是与变量所在的段相关联的段寄存器，除非显式地指出了操作数的段。

- 操作数的类型：

  若V为变量，则操作数类型为变量的类型；

  若V为常量，类型未知。

2.4 基址加变址寻址 (5)

右边是一段内存分布图
执行下面2条语句后，
(EAX)=?，(EBX)=?

```
MOV  AX, 8[BX][SI]
MOV  EBX, -8[EDI*2][EBP]
```

执行前：(DS) = 1234H
(ES) = 1200H
(SS) = 1200H
(EAX) = 89ABCDEFH
(EBX) =10H
(ESI) = 20H
(EBP) = 300H
(EDI) = 40H

物理地址	
12375H	02 H
12376H	05 H
12377H	1E H
12378H	28 H
12379H	32 H
1237AH	12 H
1237BH	34 H
1237CH	52H

2.5 立即寻址

- 操作数直接放在指令中，在指令的操作码后；
- 操作数是指令的一部分，位于代码段中；
- 指令中的操作数是8位、16位或32位二进制数。

使用格式： n

操作码及目的操作数寻址方式码
立即操作数 n

立即操作数只能作为源操作数。

例： MOV AX , 10

2.6 直接寻址 (1)

- 操作数在内存中；
- 操作数的偏移地址EA紧跟在指令操作码后面。

格式： **段寄存器名：[n]**
或 变量 或 变量 + 常量

功能：操作码的下一个字（或双字）单元的内容为操作数的偏移地址EA。

- 操作数所在的段：由段寄存器名指示，或者与变量所在的段相关联的段寄存器
- 操作数的类型：若有变量，则是定义变量的类型；否则，未知。

2.6 直接寻址 (2)

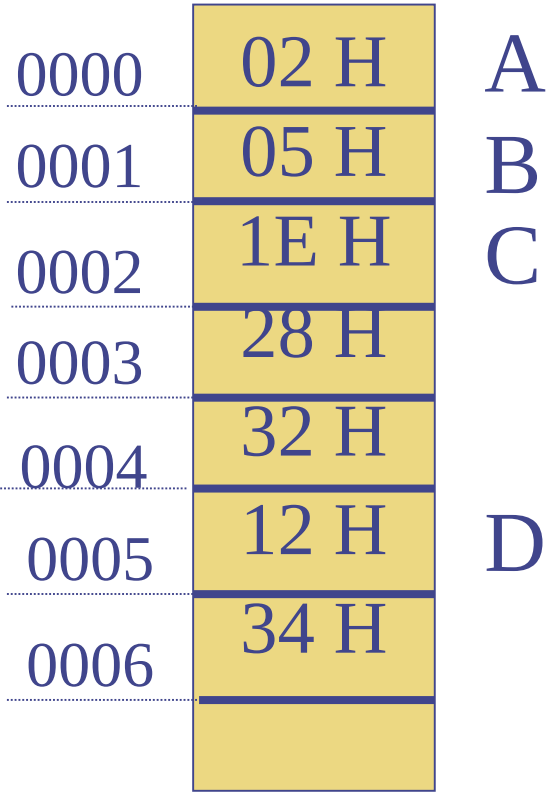
例1： MOV AX，
DS:[2000H]

执行前：(AX)=1,
DS:(2000H)=976

执行后：(AX)=976

2.6 直接寻址 (3)


```
DATA  SEGMENT
A       DB   2
B       DB   5
C       DB   30,40,50
D       DW   3412H
DATA  ENDS
CODE  SEGMENT
        ASSUME  CS:CODE, ES:DATA
START:  MOV   AX, DATA
        MOV   ES, AX
        MOV   AH, B
        MOV   CX, D
        MOV   AL, C+1
        MOV   BX, 1
        MOV   AL, C [BX]
        MOV   AX, 4C00H
        INT   21H
CODE    ENDS
        END  START
```



在变址寻址、基址加变址寻址中用变量的有效地址作为位移量

2.7 寻址方式的有关问题(1)

- 关于带变量名的寻址方式：
  变址、基址+变址、直接

(1) 在机器指令中，变量名消失，以相应的偏移地址替换之。即：凡是出现变量名的地方，都会以相应的偏移地址替换之；

(2) 段地址。若寻址方式中没有出现段强制说明（如ES:），则段地址是与变量所在的段相关联的段寄存器。过程如下：编译器在ASSUME语句中查找与定义变量的段相关联的段寄存器（若没找到则报错），然后以该段寄存器为段地址。

2.7 寻址方式的有关问题(2)

关于带变量名的寻址方式：
变址、基址+变址、直接

(3) 当多个段寄存器与同一个段名相关联时（如ASSUME DS:DATA, ES:DATA），变量的段关联优先次序从高到低为：DS、SS、ES、CS。

2.7 寻址方式的有关问题(3)


```
ASEG   SEGMENT USE16
A      DW  1234H
ASEG   ENDS
BSEG   SEGMENT USE16 STACK
B      DW   3456H
       ASSUME   CS: BSEG, SS: BSEG,
START:MOV  AX, ASEG
       MOV  DS, AX
       MOV  ES, AX
       MOV  AX, A              ;错误，why?
       MOV  BX, B              ;机器码及（BX）
       ASSUME  ES: ASEG
       MOV  AX, A              ;机器码及（AX）
       ASSUME  DS: ASEG
       MOV  AX, A              ;机器码及（AX）
       MOV  AX, 4C00H
       INT   21H
       DB    200 DUP(0)
BSEG   ENDS
       END START
```

2.7 寻址方式的有关问题(4)

●寻址方式小结

根据操作数的存放位置，寻址方式归为3类：

- 寄存器方式
- 立即方式
- 存储器方式
  - 寄存器间接寻址方式
  - 变址寻址方式
  - 基址加变址寻址方式
  - 直接寻址方式

2.7 寻址方式的有关问题(5)

1. 双操作数寻址方式的规定

一条指令的源操作数和目的操作数不能同时用存储器方式。

```
MOV   AX,  BX          MOV   BUF, 0
MOV   BUF, BX          MOV   BX,   0
MOV   BX,   BUF
```

~~MOV BUF, MSG~~

~~MOV BUF, [BX]~~

~~MOV BUF, 5[BX]~~

~~MOV BUF, 5 [EAX + EDI *4]~~

2.7 寻址方式的有关问题(6)

2. 操作数的类型

- 寄存器寻址方式中，操作数的类型由寄存器决定
- 立即数没有类型
- 不含变量的存储器方式所表示的操作数类型未知
- 含变量的寻址方式对应的操作数类型是变量的类型

2.7 寻址方式的有关问题(7)

3. 双操作数的类型规定

- 双操作数中至少应有一个的类型是明确的；
- 若两个操作数的类型都明确，则两个的类型应相同。

(1) MOV BX, AX

(2) MOV BX, AL ✗ 两个操作数的类型不匹配

(3) MOV [BX], 0 ✗ 两个操作数的类型不明确

属性定义算符 PTR

```
MOV   BYTE  PTR [BX],   0
MOV   WORD PTR [BX],   0
MOV   DWORD PTR[BX], 0
```

2.7 寻址方式的有关问题 (8)

4. 寻址方式中的段地址

对于存储器方式：

(1) 标明了段寄存器，OP在段寄存器指明的段中。否则：

(2) 若有变量，则是变量所在的段；否则

(3) 系统默认 (V为常量时)

对于 [R], R为BP, EBP, ESP时，是SS段。

对于 V[R*F], R为BP, EBP, ESP时，是SS段。

对于V[BR+IR*F], BR为BP, EBP, ESP时是SS段。

R： EAX,EBX,ECX,EDX,ESI,EDI,EBP,ESP， BX, BP, SI, DI

BR： EAX,EBX,ECX,EDX,ESI,EDI,EBP,ESP， BX, BP

IR： EAX,EBX,ECX,EDX,ESI,EDI,EBP SI, DI

2.7 寻址方式的有关问题(9)

除了隐式规定数据所在的段外，还可以显式指出段的位置。

使用格式：段寄存器名：

例 (1) MOV AX, DS：[BP]
(2) MOV BX, ES：[BX]
(3) MOV CX, SS：[SI]
(4) MOV DX, SS：[DI]
(5) MOV CX, SS：[EAX]
(6) MOV AX, CS: SUM[BP+SI]

第2章 复习

1. 6种寻址方式

寄存器寻址	R
寄存器间接寻址	[R]
变址寻址	[RxF+V]　V[RxF]
基址加变址寻址	[BR+IRxF+V]　V[BR][IRxF]
立即寻址	n
直接寻址	段R:[n]　V+n

2. 翻译带变量名的汇编指令（2.3-14, 综合练习-3）

表2.1 指令分析表(前后指令间没有关系)　　P45, 2.2

指令	地址	执行前	执行后
MOV AX, 3	AX	00AAH	0003H
SUB WORD PTR[AX] , 3	错		
ADD WORD PTR[EAX] , 3	100AAH	0002H	0005H
ADD BH, 2	BH	00H	02H
SUB EBP, 2	EBP	5000H	4FFEH
SUB [CX], DX	错		
ADD DI, 2	DI	6000H	6002H
SUB SI, 10	SI	4000H	3FF6H
MOV [BX], BX	100BBH	0200H	00BBH
MOV [DI], DX	16000H	0300H	00DDH
MOV ES:[DX], BX	错		
MOV ES:[SI], DS	24000H	0400H	1000H

指令	目的操作数地址	执行前	执行后
MOV DX, SS	DX	00DDH	3000H
MOV EAX , EIP	错		
SUB 2[DX] , AX	错		
ADD 500H[BP], CX	35500H	355H	0421H
SUB [SI-300H], AX	13D00H	0F13DH	0F093H
MOV [AX+2], BX	错		
MOV [DI+1000H],SI	17000H	0F13DH	4000H
MOV [CX-100H], AX	错		
MOV [DX+60], AX	错		
MOV –8[BX], CX	100B3H	0F8BBH	00CCH
MOV ES:1000H[DI], BP	27000H	0270H	5000H
MOV [BP+SI], DX	39000H	0039H	00DDH

P45, 2.2

指令	目的操作数地址	执行前	执行后
MOV [DI+SI], DX	错		
MOV [EDI+ESI] , DX	1A000H	001AH	00DDH
MOV [BX+DI] , 10H	错		
MOV [BX+DI], DX	160BBH	0H	00DDH
MOV ES:[7000H], DX	27000H	270H	00DDH

```
data    segment
BUF1  db  20, 39, 50, -20, 0, -12
BUF2  db  6 dup(0)
data    ends

code   segment
       assume  cs:code, ds:data
start:  mov  ax, data
       mov  ds, ax
       mov  cx, 6
       mov   bx, 0
LP:   mov   al, buf1[bx]
       mov   buf2[bx], al
       inc     bx
       dec    cx
       jnz     LP
       mov   ah, 4CH
       int     21H
code  ends
       end  start
```

将BUF1中的6个字节依次拷贝到以BUF2为首址的字节单元中。

算法思想？

采用什么寻址方式？

空中填什么语句？