

## 2020/12/29\_16位汇编\_第4课\_汇编指令的二进制格式、寻址方式

笔记本: 16位汇编

创建时间: 2020/12/29 星期二 9:48

作者: ileemi

---

- [汇编指令的二进制格式/编码](#)
  - [指令长度](#)
  - [操作码](#)
  - [操作数](#)
  - [寻址方式](#)
    - [寻址方式由两种方式决定](#)
- [立即数寻址方式](#)
  - [立即数寻址方式的流程](#)
- [寄存器寻址方式](#)
  - [寄存器寻址方式的流程](#)
- [存储器寻址方式](#)
  - [直接寻址方式](#)
    - [直接寻址方式的流程](#)
  - [寄存器间接寻址方式](#)
    - [寄存器间接寻址方式的流程](#)
  - [寄存器相对寻址方式](#)
    - [寄存器相对寻址方式的流程](#)
  - [基址变址寻址方式 \(基址+变址\)](#)
    - [基址变址寻址方式的流程](#)
  - [相对基址变址寻址方式](#)
    - [相对基址变址寻址方式的流程](#)
- [对所有寻址方式的性能做一个排序](#)

寄存器在内存中最少用三个二进制字节进行表示  $2^3 = 8$ 。

CPU厂家的汇编语法、以及对应的二进制形式都由厂家决定。

修改软件数据的时候有时候需要修改对应地址上的二进制，有时候修改二进制对应的反汇编指令不可行（汇编指令对应的二进制格式不确定）。

## 汇编指令的二进制格式/编码

不管让 CPU 干什么都需要将指令写到二进制编码中。汇编指令需要在二进制中进行描述：指令的长度（变长，定长）。

### 指令长度

变长：复杂指令集（Inter, Amd），语法长度没有限制（上限13字节）方便定义语法

定长：精简指令集（arm（安卓）现在也做变长，4字节，8字节）

## 操作码

CPU 需要获取指令对应的操作（要干什么），所以操作码一开始可以设计为 8 位（256条指令），保留一个字节（0~254）用于进行扩展指令（FF 0）。

操作码指计算机程序中所规定的要执行操作的那一部分指令或字段(通常用代码表示)，其实就是指令序列号，用来告诉CPU需要执行哪一条指令。

汇编指令：

mov ax,bx

add ax,bx

sub ax,bx

## 操作数

操作数是运算符作用于的实体，是表达式中的一个组成部分，它规定了指令中进行数字运算的量。

可以有（ $a + b = a$ ）2个，也可以没有（ret），根据CPU编码去决定（Inter 设计时只有两个操作数）。

操作数可以分为：**源操作数**，**目地操作数**。

操作数的描述称为：寻址方式（操作数从哪个地方来的，是在内存中还是在寄存器中）。

操作数的位置：1（立即数：在指令中），ax（寄存器），[2000]（在存储器中）

汇编指令：

add ax,1

add ax,bx

add ax,[2000]

所以一条指令的格式可以分为：操作码（1个字节）、寻址方式、操作数

分析操作码：

-a

0B24:0100 mov ax,bx

0B24:0102 add ax,bx

0B24:0104 sub ax,bx

-u

0B24:0100 89D8 MOV AX,BX

0B24:0102 01D8 ADD AX,BX

0B24:0104 29D8 SUB AX,BX



通过分析可以得出源和目的寄存器用 "D8" 进行表示中。尝试修改源寄存器，在  
进行分析：

汇编代码如下：

```
-a
0B24:0100 mov ax,bx
0B24:0102 mov ax,cx
0B24:0104 mov ax,dx
0B24:0106 ret
0B24:0107

-u
0B24:0100 89D8    MOV    AX,BX
0B24:0102 89C8    MOV    AX,CX
0B24:0104 89D0    MOV    AX,DX
0B24:0106 C3      RET
```

通过结果可以看出 "MOV" 指令不变的前提下，后一个字节的数据有改变，拆分为二进制再次进行分析：

源操作数（3~4位）：

d8: 11 **011** 000

c8: 11 **001** 000

d0: 11 **010** 000

尝试将 **d8** (11 **011** 000 改为 11 **000** 000) 修改为 **c0**，对应的汇编指令代码如下：

```
-e 101
0B24:0101 D8.c0
-u 100
0B24:0100 89C0    MOV    AX,AX (有改动)
0B24:0102 89C8    MOV    AX,CX
```

通过分析可以得出结论，修改**源寄存器**只需要修改中间的三位二进制数据即可。  
修改目的寄存器只需要修改后三位二进制数据即可。

对应的寄存器：

000 --> AX

011 --> BX

001 --> CX

010 --> DX

d0: 11 **010** (决定源寄存器) **000** (决定目的寄存器)

"89D8" 可以表示为：操作码（一个字节），2Bit（寻址方式）、3Bit（源操作数）、3Bit（目标操作数）。

## 寻址方式

3 (11) --> 寄存器到寄存器

2 (10) --> [bx+di+xxx] 寄存器到内存单元

1 (01) --> [bx+di-xxx] 寄存器到内存单元

0 (00) --> [bx+di] 寄存器到内存单元

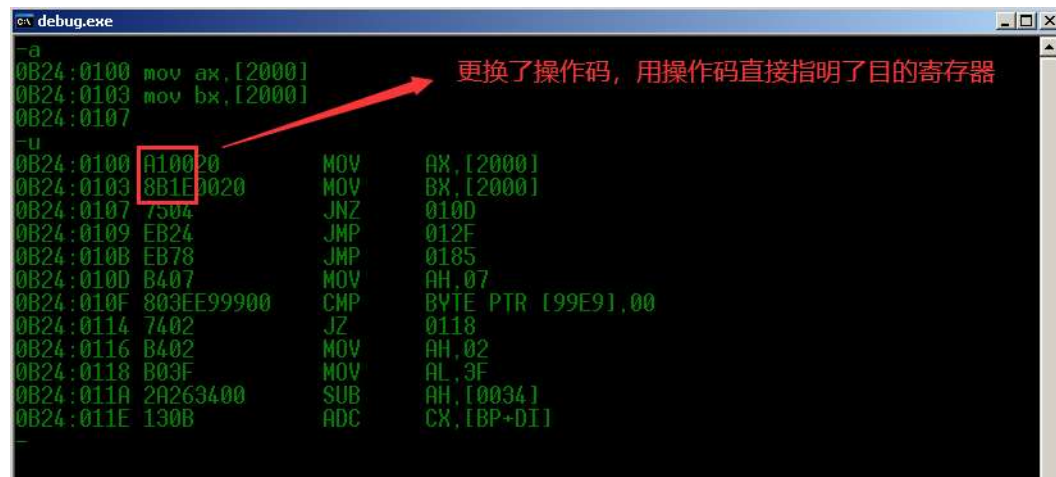
通过上面的测试可以看出 "mov" 指令只能实现 "寄存器到寄存器"、"寄存器到内存单元" 两种形式。

-a

0B24:0100 mov ax,[2000]

0B24:0103 mov bx,[2000]

0B24:0107



```
debug.exe
-a
0B24:0100 mov ax,[2000]
0B24:0103 mov bx,[2000]
0B24:0107
-u
0B24:0100 010020 MOV AX,[2000]
0B24:0103 8B1E0020 MOV BX,[2000]
0B24:0107 7504 JNZ 010D
0B24:0109 EB24 JMP 012F
0B24:010B EB78 JMP 0185
0B24:010D B407 MOV AH,07
0B24:010F 803EE9900 CMP BYTE PTR [99E9],00
0B24:0114 7402 JZ 0118
0B24:0116 B402 MOV AH,02
0B24:0118 B03F MOV AL,3F
0B24:011A 2A263400 SUB AH,[0034]
0B24:011E 130B ADC CX,[BP+DI]
```

当汇编指令为：立即数到寄存器的时候，对应的反汇编代码为两个字节，第一个字节集成了操作码和寄存器，测试如下：

-a

0B24:0104 mov al,20

0B24:0106

-u 104

0B24:0104 B020 MOV AL,20

## 寻址方式由两种方式决定

- 操作码：mov ax,20

- 操作码后一个字节的前两位：mov ax,bx

指令的长度需要和 "8" 对齐。寻址方式最低有5中方法（操作码+后一个字节前两位的四种方法）。

## 立即数寻址方式

指令中的操作数直接存放在机器代码中，紧跟在操作码之后（操作数作为指令的一部分存放在操作码之后的主存单元中）这种操作数被称为**立即数**（imm）。

- 它可以是 8 位数值：i8(00H~FFH)
- 也可以是16 位数值：i16(0000H~FFEH)

立即数（**常量**）寻址方式常用来给寄存器赋值。数据在指令中，或者在寄存器中，在者在存储器中。立即数常写在**源操作数**中。

汇编指令：

-a

0B24:0100 mov ax,FFFF

反汇编：

0B24:0100 B8FFFF MOV AX,FFFF

**B8FFFF**（数据在指令中，不需要在从内存中获取数据）。

B8: mov ax

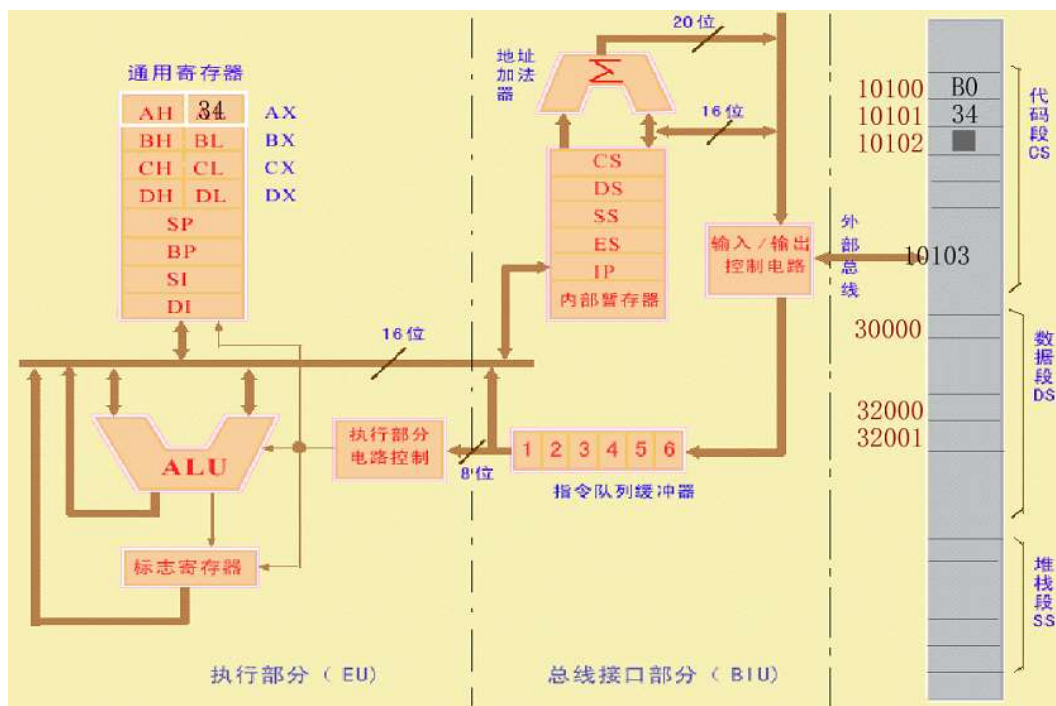
FFFF: FFFF

指令队列缓冲器：用于存放指令以及指令后面的**操作数**，提高数据的读取效率。

执行部分电路控制：用于解析指令

---

## 立即数寻址方式的流程



汇编指令：

-a

0B24:0100 mov al,34H

-u

0B24:0100 B034 MOV AL,34

代码在内存中，上面汇编指令对应的二进制代码为：B034，B034假如在内存为10100~10101的地址上：

10100	10101	10102	...
B0	34	...	...

内存访问地址由寄存器 "CS" 以及 "IP" 决定，两个寄存器上的数据通过 "地址加法器" 算出一个 20 位的内存地址，然后 CPU 通过访问这个地址，将地址上的数据按类分类走对应的路线。

地址上的值：

- 指令代码："B0" 通过"输入/输出控制电路" 到 "指令队列缓冲器"，通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后，CPU 就知道要做什么了。指令代码到达 "解释器" 后，会继续从内存中读取下一条数据，并将其放入到 "指令队列缓冲器" 中。
- 数据：通过"输入/输出控制电路" 经过 "16位数据线路"

CPU 需要 "操作数" 的时候可直接从 "指令队列缓冲器" 中获取提前准备的内存中的数据，提高数据的读取效率（不在需要从新去内存中读取数据）。

CPU 知道要将立即数 "34H" 移动到寄存器 "al" 后，由于数据不需要进行运算，所以可以直接从 "指令队列缓冲器" 中取出数据 "34" 将其送到寄存器 "al" 中。MOV 指令不影响 "标志寄存器"。

**CPU 执行流程：**

1. 初始状态：CS (1010H) ， IP (0000H) ， CPU 将从内存 1010H x 16 + 0000H 处读取指令执行；

2. 将CS、IP中的内容送入地址加法器（地址加法器完成：物理地址 = 段地址 x 16 + 编译地址）；
3. 地址加法器将物理地址送入输入输出控制电路；
4. 输入输出控制电路将物理地址 10100H 送上地址总线；
5. 从内存 10100H 单元开始存放的机器指令 B034 通过数据总线被送入 CPU；
6. 输入输出控制电路将机器指令 B034 送入指令缓冲器；
7. IP 中的值自动增加（以使 CPU 可以获取下一条指令，当前读入的指令 B034 长度为两个字节，所以 IP 中的值应加 2 个字节。此时，CS:IP 指向内存单元 1010:0002）；
8. 执行控制器执行指令 B034（即：mov al,34）；
9. 指令 B034 被执行后 寄存器 al 中的内容为：34H；

## 寄存器寻址方式

操作数都在寄存器中。操作数存放在CPU的内部寄存器 reg 中，可以是：

- 8位寄存器r8:  
AH、AL、BH、BL、CH、CL、DH、D
- 16位寄存器r16  
AX、BX、CX、DX、SI、DI、BP、SP
- 4个段寄存器seg  
CS、DS、SS、ES

在 CPU 手册中会说明哪些指令会影响对应的标志寄存器（例如：mov 指令不影响标志寄存器）。

汇编指令：

-a

0B24:0100 mov ds,ax

0B24:0102 mov cs,ax

0B24:0104 ret

0B24:0105

-u

0B24:0100 8ED8 MOV DS,AX

0B24:0102 8EC8 MOV CS,AX

0B24:0104 C3 RET

通过 操作码后一个字节的两位进行寻址。

inc --> 指定寄存器数值+1

dec --> 指定寄存器数值-1

"指令队列缓冲器中的数据到寄存器" 的时间效率上要比 "寄存器中的数据到另一个寄存器" 中去的时间效率要慢。原因：**1. 距离 2. 线路的制作材料**

1. mov ax,0
2. sub ax,ax
3. xor ax,ax

后两个汇编指令得出结果的时间效率上要比第一个汇编指令得出结果的时间效率高。

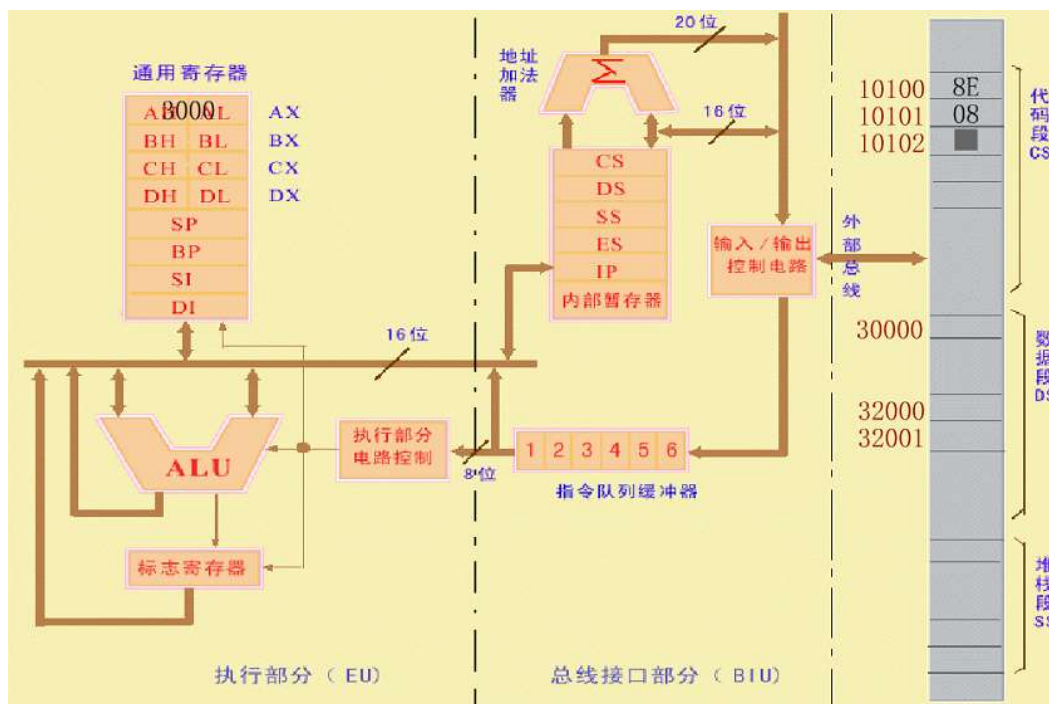
## 寄存器寻址方式的流程

汇编指令：mov ds,ax --> 将寄存器ax中的数据移动到段寄存器ds中。

对应的反汇编为：8ED8 MOV DS,AX

8E: mov

D8: ds ax



### CPU 执行流程：

1. 初始状态：CS (1010H) , IP (0000H) , CPU 将从内存 1010H x 16 + 0000H 处读取指令执行；
2. 将CS、IP中的内容送入地址加法器（地址加法器完成：物理地址 = 段地址 x 16 + 编译地址）；
3. "地址加法器" 将计算后的物理地址 "10100H" 送入 "输入输出控制电路"；
4. "输入输出控制电路" 将 物理地址 "10100H" 送上地址总线；
5. CPU 访问内存 "10100H" 单元，读取指令 "8E" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中；
6. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后，CPU 解析指令 得到操作码后会继续继续从 "CS:IP" 对应的下一个内存地址上读取下一条数据，并将其放入到 "指令队列缓冲器" 中；
7. IP 中的值自动增加（以使 CPU 可以获取下一条指令，当前读入的指令 8E 长度为1个字节，所以 IP 中的值应加 1 个字节。此时 CS:IP 指向内存单元 1010:0001）；
8. CPU 访问内存 "10101H" 单元，读取指令 "08" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中；
9. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后，CPU 解析指令 "08" 得到寻址方式以及操作数；
10. 将 "寄存器ax" 中的数据移动到 "段寄存器ds" 中。



11. IP 中的值根据当前读取的指令字节数自动增加，CPU会继续访问对应内存上的指令数据。

通过流程可以看出 "寄存器寻址方式" 相对于 "立即数寻址方式" 得出结果的时间效率要快。

## 存储器寻址方式

8086设计了五种存储器寻址方式：

- 直接寻址方式
- 寄存器间接寻址方式
- 寄存器相对寻址方式
- 基址变址寻址方式
- 相对基址变址寻址方式

多次通过内存进行数据访问会导致得出最终结果的时间效率变慢。解决效率慢的方法可以通过对代码进行优化，减少内存访问次数。

## 直接寻址方式

有效地址在指令中直接给出，汇编指令示例：

```
mov ax,[2000]
```

```
mov [2000],ax
```

```
mov [2000],[3000] --> 错误写法
```

只取一个字节：mov al,[2000] 等价于 mov al,byte ptr [2000] (byte ptr: 一个字节的指针)

只取一个字节：mov ax,[2000] 等价于 mov ax,word ptr [2000] (word ptr: 两个字节的指针)

将代码段偏移100位置上的数据送到寄存器ah中：

汇编代码：

```
mov ah, byte ptr cs:[200] --> 在xp中语法不通过，有些编译器允许这样写
```

需要使用下面的汇编语法：

```
-a
```

```
0B24:0100 cs:
```

```
0B24:0101 mov ah,byte ptr [200]
```

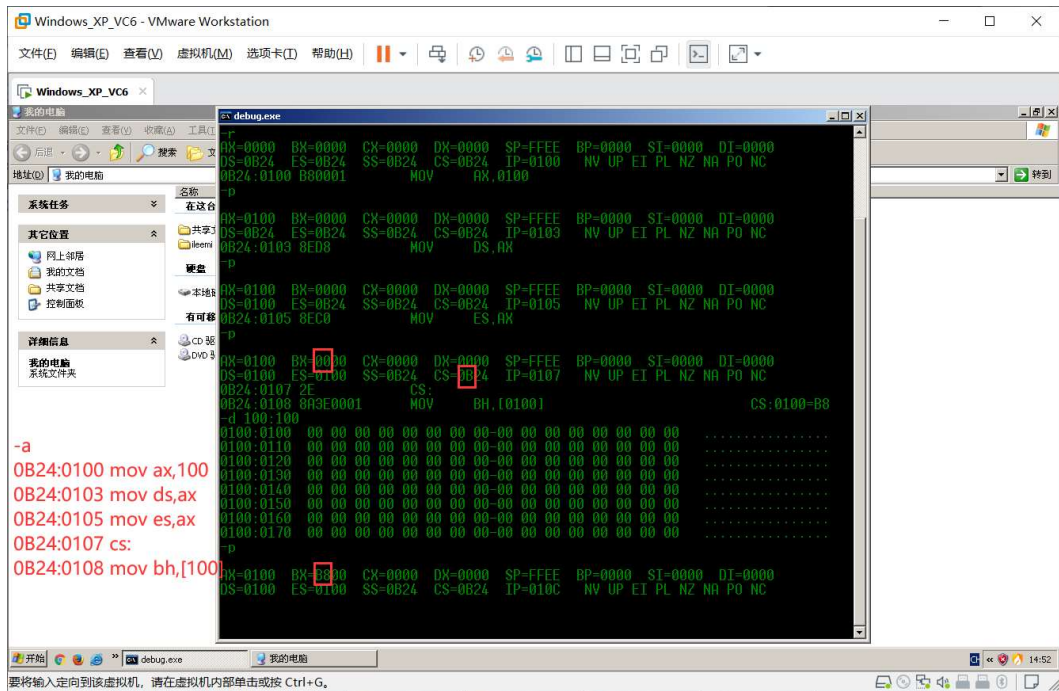
```
0B24:0105
```

```
-u
```

```
0B24:0100 2E CS:
```

```
0B24:0101 8A260002 MOV AH,[0200]
```

"CS:": 这里的 "cs:" 只针对下一条汇编语法有效，这样的写法叫 "段超越前缀指令"。段超越前缀指令默认为 ds 或者 es (由寄存器决定)。



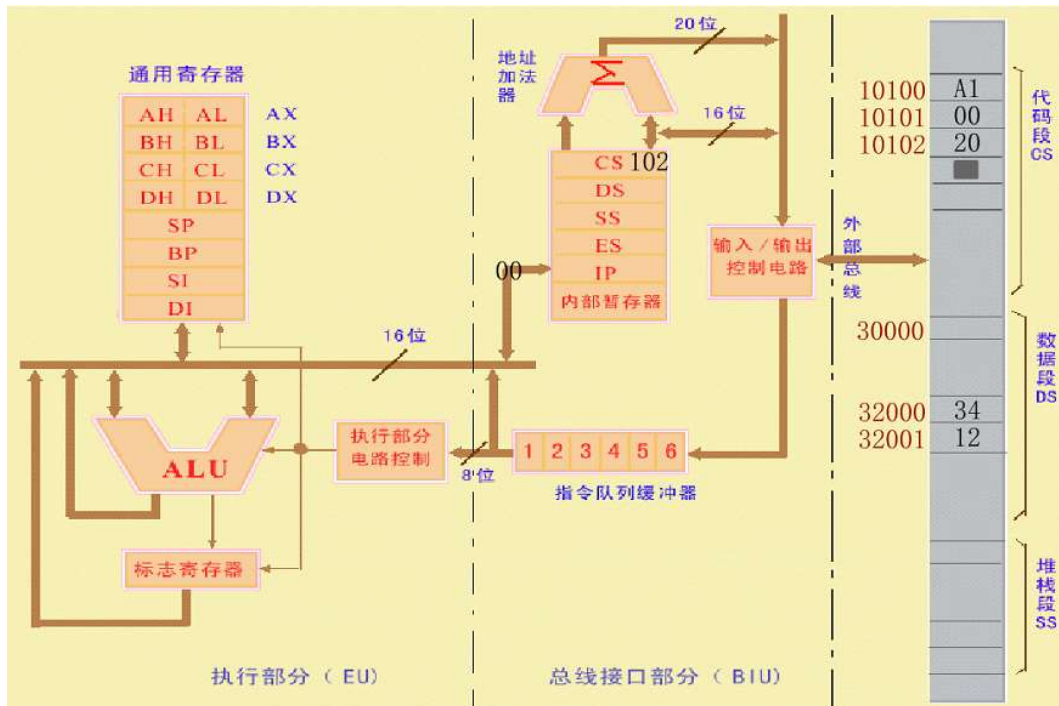
## 直接寻址方式的流程

汇编指令：mov ax,[2000] --> CPU 从当前要执行的内存地址上取出偏移数  
数据：2000，然后将偏移加上数据段寄存器的首地址，计算出对应的物理地址并  
访问，取出地址上的操作数将其移动到 "寄存器ax" 中。

对应的反汇编为：A10020 MOV AX,[2000]

A1: mov ax

0020: 2000



直接寻址使用的段超越前缀指令默认为 ds。

CPU 执行流程：

1. 初始状态: CS (1010H) , IP (0000H) , CPU 将从内存 1010H x 16 + 0000H 处读取指令执行;
2. 将CS、IP中的内容送入地址加法器 (地址加法器完成: 物理地址 = 段地址 x 16 + 编译地址) ;
3. "地址加法器" 将计算后的物理地址 "10100H" 送入 "输入输出控制电路";
4. "输入输出控制电路" 将 物理地址 "10100H" 送上地址总线;
5. CPU 访问内存 "10100H" 单元, 读取指令 "A1" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中;
6. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 得到操作码后会继续从 "CS:IP" 对应的下一个内存地址上读取下一条数据, 并将其放入到 "指令队列缓冲器" 中;
7. IP 中的值自动增加 (以使 CPU 可以获取下一条指令, 当前读入的指令 "A1" 长度为1个字节, 所以 IP 中的值应加 1 个字节。此时CS:IP 指向内存单元 1010:0001) ;
8. CPU 访问内存 10101H 单元, 读取指令 "00" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在将其送入 "段寄存器" 中的 "内部暂存器" 中;
9. "寄存器IP" 中的值自动增加当前读取的字节数;
10. CPU 继续访问 CS:IP 指向的内存 "10102H" 单元, 读取指令 "20" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在将其送入 "段寄存器" 中的 "内部暂存器" 中;
11. "内部寄存器" 将两次读取到的偏移数值拼接, 结果为 "2000H"。"寄存器IP" 中的值继续自动增加当前读取的字节数;
12. CPU 将 数据段寄存器的首地址 "3000H" 和得到的偏移地址 "2000H" 通过 "地址加法器" 得出对应的物理地址 "32000H";
13. CPU 通过 "输入输出控制电路" 访问对应的物理地址 "32000H", 取出操作数 "1234H";
14. CPU 通过 "输入输出控制电路" 将取出的操作数 "1234H" 传回段寄存器中的 "内部暂存器" 中;
15. IP 中的值根据当前读取的指令字节数自动增加, CPU 会继续访问对应内存上的指令数据;
16. CPU 将保存在 "内部暂存器" 中的操作数 "1234H" 根据对应的 "操作码" 将其移动到 "寄存器ax" 中。

相对于 "立即数寻址方式" 得出结果的时间效率上要慢。

## 寄存器间接寻址方式

有效地址**只能**存放在基址寄存器 bx、bp 或 变址寄存器 si、di 中, 默认的段地址在 DS段寄存器。

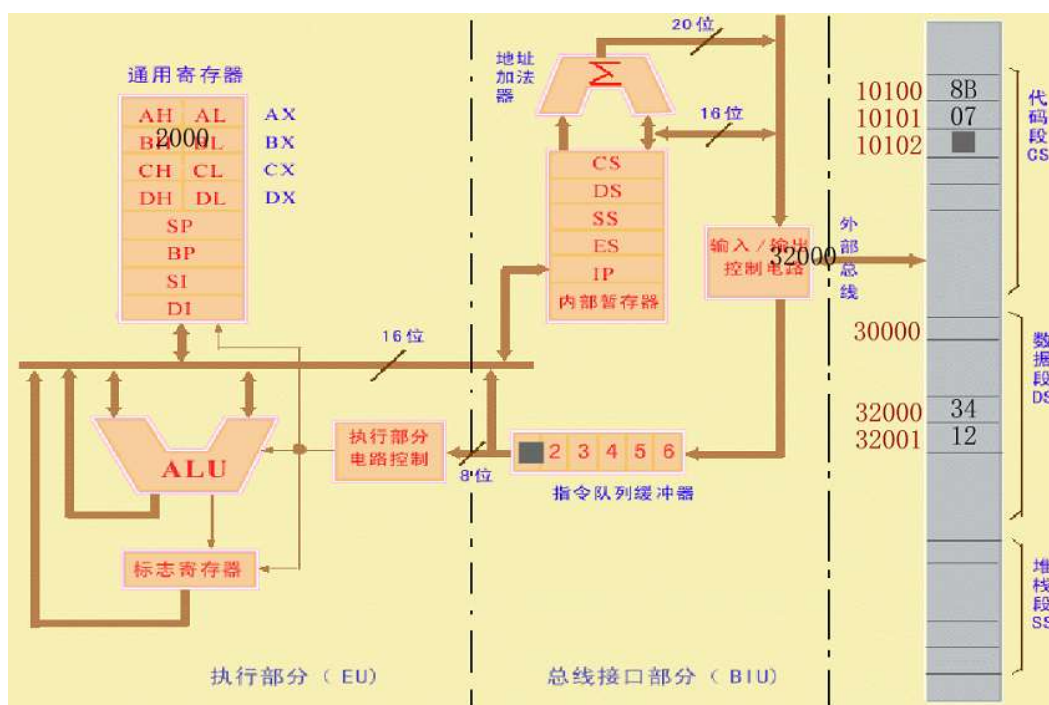
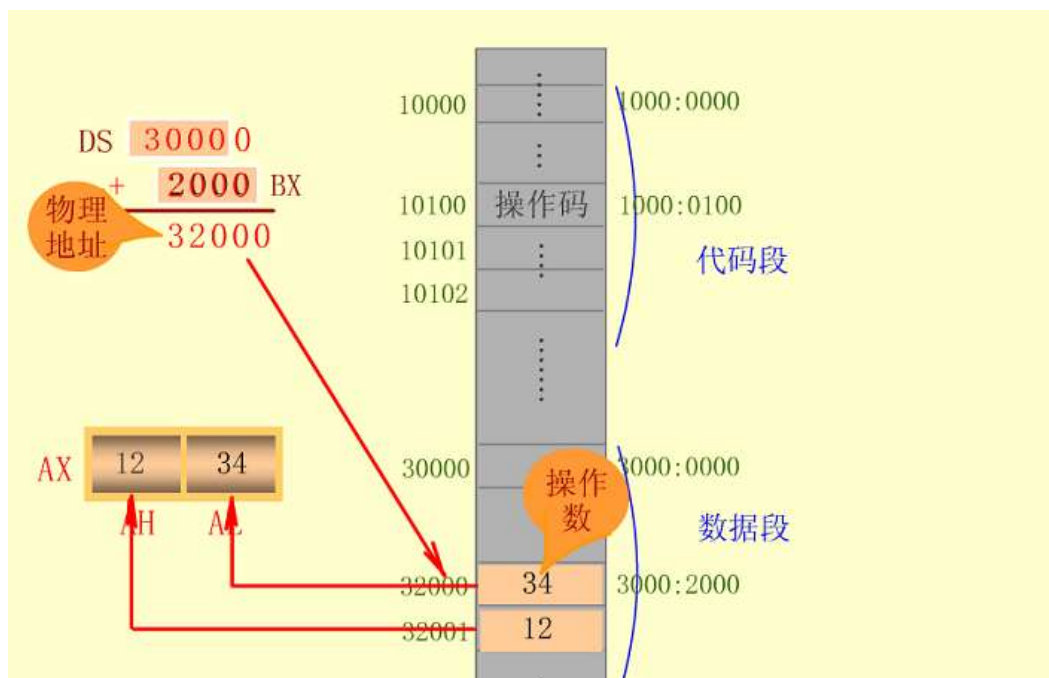
汇编指令: `mov ax,[bx] ==> mov ax,word ptr [bx]`

对应的反汇编: `8B07 MOV AX,[BX]`

8B: mov

07: ax bx

**07 (00 000 111) : 前两位决定是 寄存器间接寻址, 还是 寄存器相对寻址方式 (00 -- 寄存器间接寻址, 01/11 -- 寄存器相对寻址方式)**



## 寄存器间接寻址方式的流程

### CPU 执行流程:

1. 初始状态: CS (1010H), IP (0000H), CPU 将从内存 1010H x 16 + 0000H 处读取指令执行;
2. 将CS、IP中的内容送入地址加法器 (地址加法器完成: 物理地址 = 段地址 x 16 + 编译地址);
3. "地址加法器" 将计算后的物理地址 "10100H" 送入 "输入输出控制电路";
4. "输入输出控制电路" 将 物理地址 "10100H" 送上地址总线;
5. CPU 访问内存 "10100H" 单元, 读取指令 "8B" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中;
6. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 得到操作码后会继续从 "CS:IP" 对应的下一个内存地址上读取下一条数据, 并将其放入

- 到 "指令队列缓冲器" 中;
7. IP 中的值自动增加 (以使 CPU 可以获取下一条指令, 当前读入的指令 "8B" 长度为1个字节, 所以 IP 中的值应加 1 个字节。此时CS:IP 指向内存单元 1010:0001) ;
  8. CPU 访问内存 10101H 单元, 读取指令 "07" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令;
  9. 解析指令后, CPU 将 "基址寄存器bx" 中保存的偏移数值移动到段寄存器中的 "内部暂存器中";
  10. CPU 将 数据段寄存器的首地址 "3000H" 和得到的偏移地址 "2000H" 通过 "地址加法器" 得出对应的物理地址 "32000H";
  11. CPU 通过 "输入输出控制电路" 访问对应的物理地址 "32000H", 取出操作数 "1234H";
  12. CPU 通过 "输入输出控制电路" 将取出的操作数 "1234H" 传回段寄存器中的 "内部暂存器" 中;
  13. IP 中的值根据当前读取的指令字节数自动增加, CPU 会继续访问对应内存上的指令数据;
  14. CPU 将保存在 "内部暂存器" 中的操作数 "1234H" 根据对应的 "操作码" 将其移动到 "寄存器ax" 中。

相对于 "直接寻址方式", 减少了一次内存访问次数, 得出结果的时间效率上要快。

## 寄存器相对寻址方式

相对寻址方式, 源寄存器只能是基址寄存器或者是变址寄存器, 偏移由编码决定的。间接寻址时, 源寄存器只能是 基址寄存器 (bx、bp) 或者变址寄存器 (si、di) 。有以下四种书写方式:

MOV AX,[bx+06H]

MOV AX,[bp+06H] --> AX←SS:[bp+06H]

MOV AX,[si+06H] --> AX←DS:[si+06H]

MOV AX,[di+06H] --> AX←SS:[di+06H]

汇编指令示例: mov ax,1000[si] == mov ax,[si+1000]

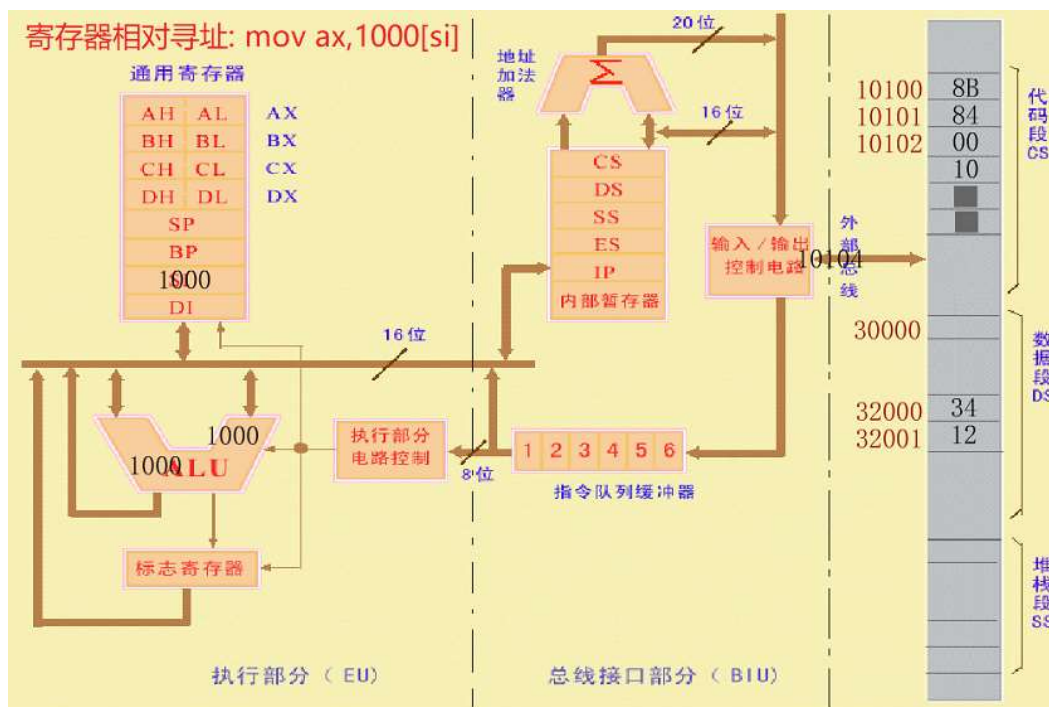
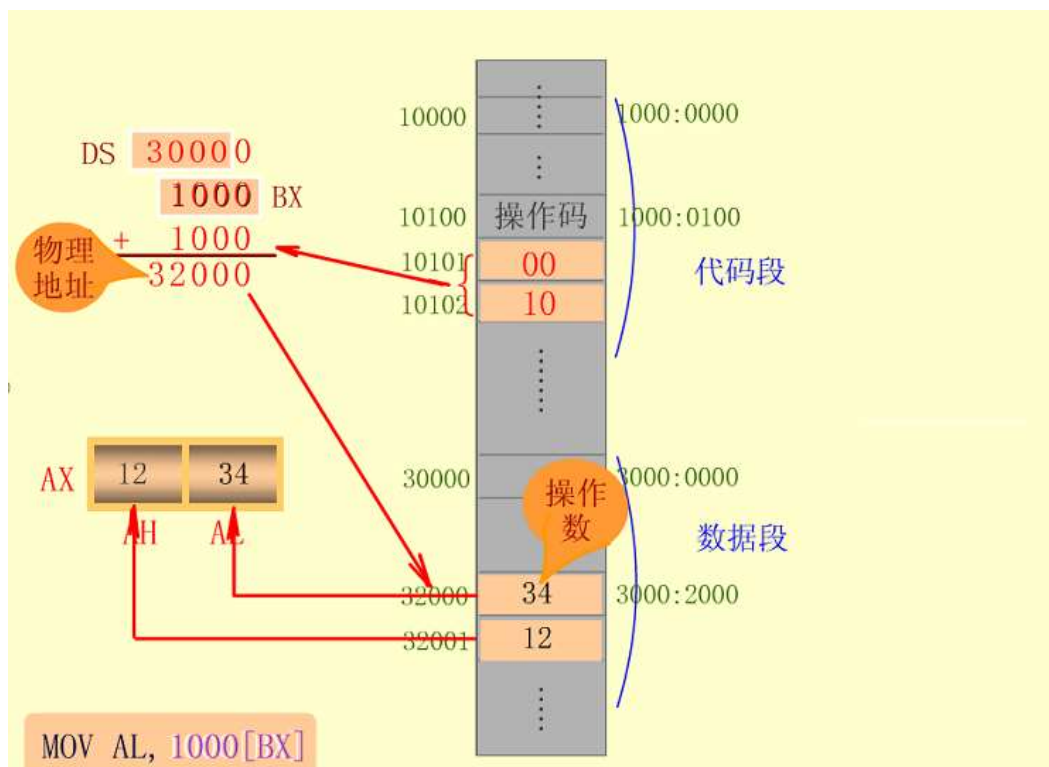
对应的反汇编为: 8B840010 MOV AX,[SI+1000]

8B: mov

84 (10 000 100) : ax si

0010: 1000

**84 (10 000 100) : 前两位决定是 寄存器间接寻址, 还是 寄存器相对寻址方式 (00 -- 寄存器间接寻址, 01/11 -- 寄存器相对寻址方式)**



## 寄存器相对寻址方式的流程

### CPU 执行流程:

1. 初始状态: CS (1010H) , IP (0000H) , CPU 将从内存 1010H x 16 + 0000H 处读取指令执行;
2. 将CS、IP中的内容送入地址加法器 (地址加法器完成: 物理地址 = 段地址 x 16 + 编译地址) ;
3. "地址加法器" 将计算后的物理地址 "10100H" 送入 "输入输出控制电路";
4. "输入输出控制电路" 将 物理地址 "10100H" 送上地址总线;

5. CPU 访问内存 "10100H" 单元, 读取指令 "8B" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中;
6. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 得到操作码后会继续从 "CS:IP" 对应的下一个内存地址上读取下一条数据, 并将其放入到 "指令队列缓冲器" 中;
7. IP 中的值自动增加 (以使 CPU 可以获取下一条指令, 当前读入的指令 "8B" 长度为1个字节, 所以 IP 中的值应加 1 个字节。此时CS:IP 指向内存单元 1010:0001) ;
8. CPU 访问内存 10101H 单元, 读取指令 "84" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 (IP值继续增加) ;
9. 解析指令后, CPU 将 "通用寄存器si" 中保存的数值移动到 CPU 的算数逻辑单元 "ALU"中;
10. CPU 通过 CS:IP 继续访问内存单元 "10102H", 读取指令 "00" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后 ("寄存器IP" 中的值自动增加当前读取的字节数), 在将其送入 "段寄存器" 中的 "内部暂存器" 中;
11. CPU 继续访问 CS:IP 指向的内存 "10103H" 单元, 读取指令 "10" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在将其送入 "段寄存器" 中的 "内部暂存器" 中;
12. "内部寄存器" 将两次读取到的偏移数值拼接, 结果为 "1000H" ("寄存器IP" 中的值继续自动增加当前读取的字节数; ) 。
13. "内部暂存器" 也将结果送入到 CPU 的算数逻辑单元 "ALU"中;
14. CPU 的算数逻辑单元 "ALU", 将接收到的两个结果进行相加, 将相加后的结果送回到 "内部暂存器" 中。
15. CPU 将 数据段寄存器的首地址 "3000H" 和得到的偏移地址 "2000H" 通过 "地址加法器" 得出对应的物理地址 "32000H";
16. CPU 通过 "输入输出控制电路" 访问对应的物理地址 "32000H", 取出操作数 "1234H";
17. CPU 通过 "输入输出控制电路" 将取出的操作数 "1234H" 传回段寄存器中的 "内部暂存器" 中;
18. IP 中的值根据当前读取的指令字节数自动增加, CPU 会继续访问对应内存上的指令数据;
19. CPU 将保存在 "内部暂存器" 中的操作数 "1234H" 根据对应的 "操作码" 将其移动到 "寄存器ax" 中。

通过流程可以看出 "寄存器相对寻址方式" 相对于 "直接寻址方式" 得出结果的时间效率上还要慢 (访问内存较多) 。

## 基址变址寻址方式 (基址+变址)

**有效地址** 由 基址寄存器 (BX或BP) 的内容加上 变址寄存器 (SI或DI) 的内容构成 (**没有相对就没有偏移**) :

有效地址 = BX/BP + SI/DI

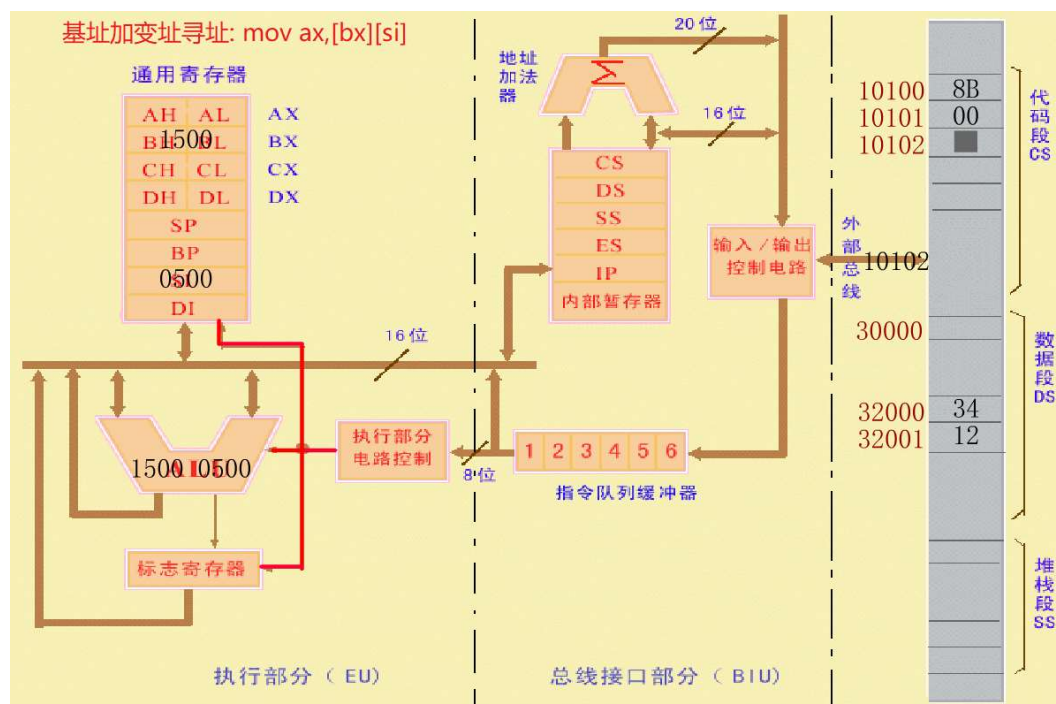
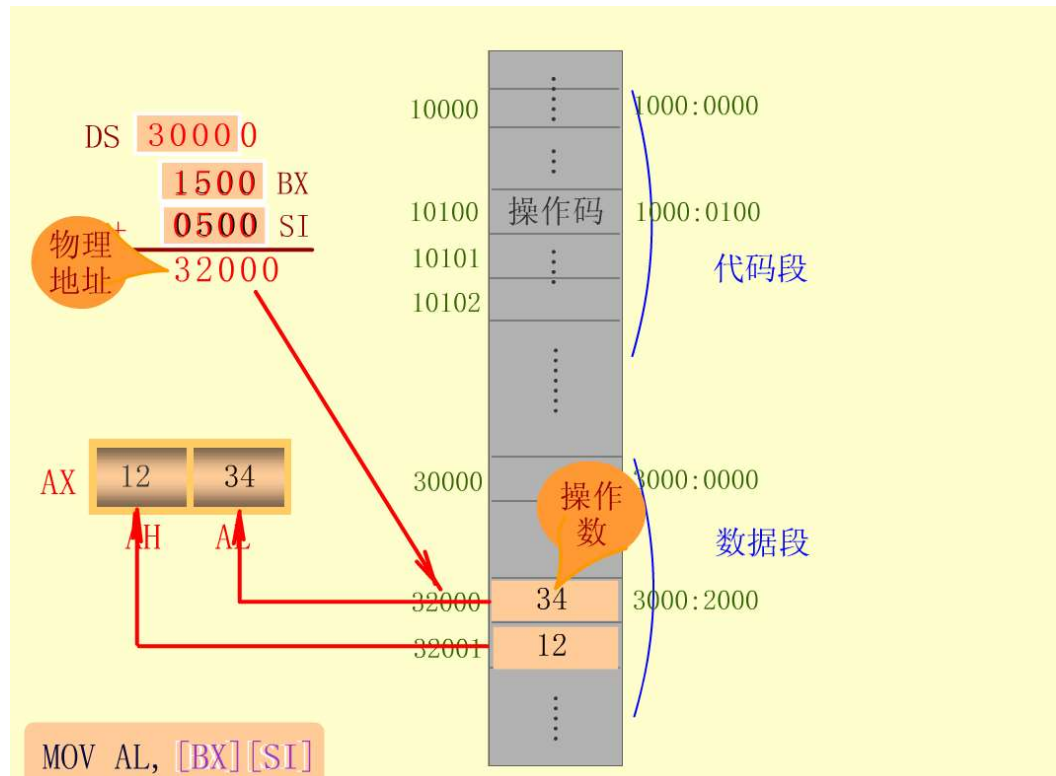
段地址对应BX基址寄存器默认是DS, 对应BP基址寄存器默认是SS, 可用段超越前缀改变。



`mov ax,[bx+si] ds段`  
`mov ax,[bp+si] ss段`  
`mov ax,[bp+di] ss段`  
`mov ax,[bx-1] --> -1 == +FF`

汇编指令示例: `mov ax,[bx][si]`

对应的反汇编为: `8B00 MOV AX,[BX+SI]`



## 基址变址寻址方式的流程

### CPU 执行流程:



1. 初始状态: CS (1010H) , IP (0000H) , CPU 将从内存 1010H x 16 + 0000H 处读取指令执行;
2. 将CS、IP中的内容送入地址加法器 (地址加法器完成: 物理地址 = 段地址 x 16 + 编译地址) ;
3. "地址加法器" 将计算后的物理地址 "10100H" 送入 "输入输出控制电路";
4. "输入输出控制电路" 将 物理地址 "10100H" 送上地址总线;
5. CPU 访问内存 "10100H" 单元, 读取指令 "8B" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中;
6. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 得到操作码后会继续从 "CS:IP" 对应的下一个内存地址上读取下一条数据, 并将其放入到 "指令队列缓冲器" 中;
7. IP 中的值自动增加 (以使 CPU 可以获取下一条指令, 当前读入的指令 "8B" 长度为1个字节, 所以 IP 中的值应加 1 个字节。此时CS:IP 指向内存单元 1010:0001) ;
8. CPU 访问内存 10101H 单元, 读取指令 "00" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 (IP值继续增加) ;
9. 解析指令后, CPU 将 "通用寄存器bx" 以及 "通用寄存器si" 中保存的数值 "1500H" "0500H" 分别移动到 CPU 的算数逻辑单元 "ALU"中;
10. CPU 的算数逻辑单元 "ALU", 将接收到的两个结果进行相加, 将相加后的结果送回到 "内部暂存器" 中。
11. CPU 将 "数据段寄存器ds" 对应的的首地址 "3000H" 和得到的偏移地址 "2000H" 通过 "地址加法器" 得出对应的物理地址 "32000H";
12. CPU 通过 "输入输出控制电路" 访问对应的物理地址 "32000H", 取出操作数 "1234H";
13. CPU 通过 "输入输出控制电路" 将取出的操作数 "1234H" 传回段寄存器中的 "内部暂存器" 中;
14. IP 中的值根据当前读取的指令字节数自动增加, CPU 会继续访问对应内存上的指令数据;
15. CPU 将保存在 "内部暂存器" 中的操作数 "1234H" 根据对应的 "操作码" 将其移动到 "寄存器ax" 中。

通过流程可以看出 "基址变址寻址方式" 相对于 "寄存器相对寻址方式" 得出结果的时间效率上要快, 但是相对于 "寄存器间接寻址方式" 得出结果的时间效率上要慢 (多了一步运算操作) 。

## 相对基址变址寻址方式

有效地址是基址寄存器 (BX/BP)、变址寄存器 (SI/DI) 与一个 8位 或 16位 位移量之和:

**有效地址 = BX/BP + SI/DI + 8/16位位移量 (位移量可用符号表示)**

段地址对应BX基址寄存器默认是DS, 对应BP基址寄存器默认是SS。

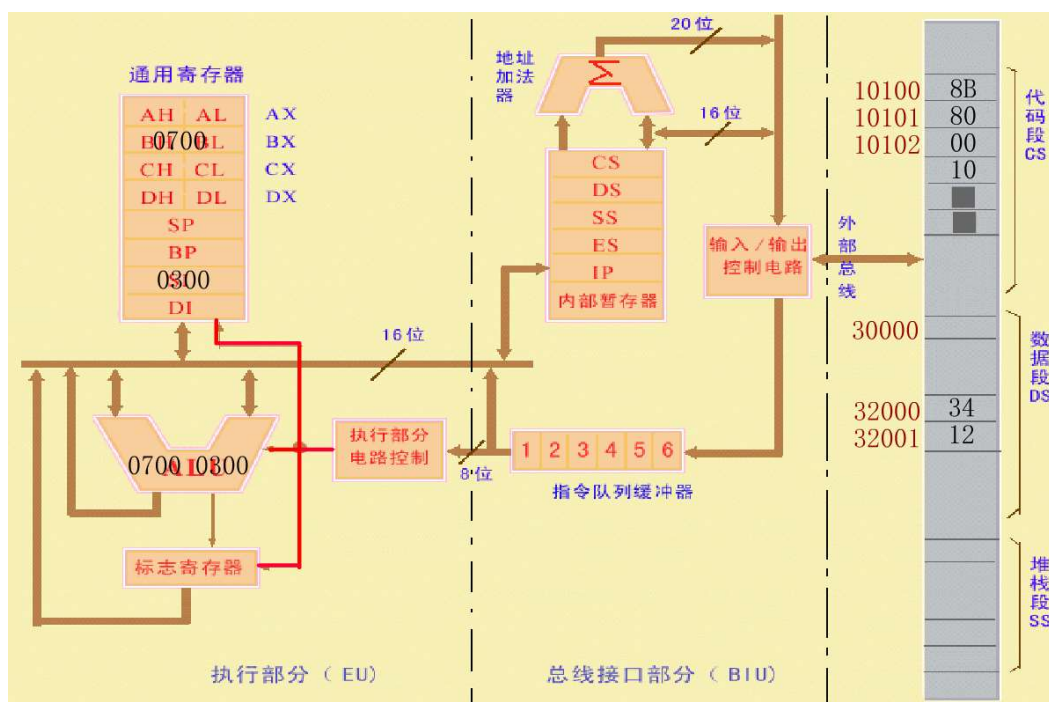
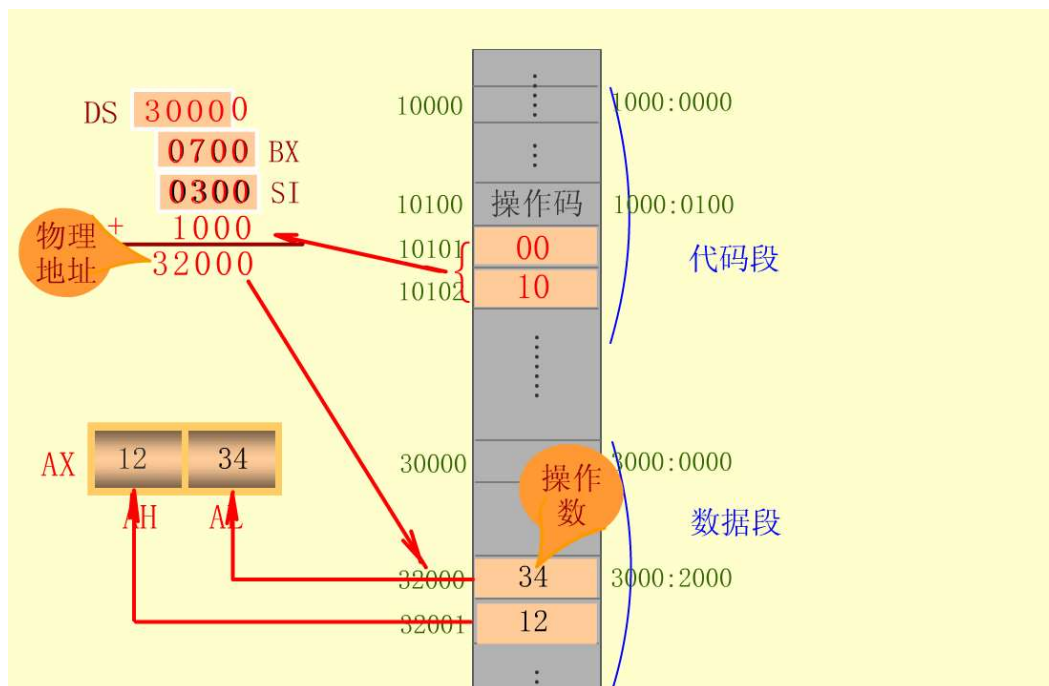
汇编指令示例: `mov ax,1000[bx][si]` 等价 `mov ax,[bx+si+1000]`

对应的反汇编为: `8B800010 MOV AX,[BX+SI+1000]`

```

debug.exe
-a
0B24:0100 mov ax,[bx+di+7f]
0B24:0103 mov ax,[bx+di+ff]
0B24:0107 mov ax,[bx+di-1]
0B24:010A
-u
0B24:0100 8B417F      MOV     AX,[BX+DI+7F]
0B24:0103 8B41FF00     MOV     AX,[BX+DI+00FF]
0B24:0107 8B41FF      MOV     AX,[BX+DI-01]
0B24:010A 24EB          AND     AL,EB
0B24:010C 78B4          JS      00C2
0B24:010E 07            POP     ES
0B24:010F 803EE99900    CMP     BYTE PTR [99E9],00
0B24:0114 7402          JZ      0118
0B24:0116 B402          MOV     AH,02
0B24:0118 B09F          MOV     AL,9F
0B24:011A 2A263400     SUB     AH,[0034]
0B24:011E 130B          ADC     CX,[BP+DI]

```



## 相对基址变址寻址方式的流程

CPU 执行流程:

1. 初始状态: CS (1010H) , IP (0000H) , CPU 将从内存 1010H x 16 + 0000H 处读取指令执行;
2. 将CS、IP中的内容送入地址加法器 (地址加法器完成: 物理地址 = 段地址 x 16 + 编译地址) ;
3. "地址加法器" 将计算后的物理地址 "10100H" 送入 "输入输出控制电路";
4. "输入输出控制电路" 将 物理地址 "10100H" 送上地址总线;
5. CPU 访问内存 "10100H" 单元, 读取指令 "8B" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中;
6. 通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 得到操作码后会继续从 "CS:IP" 对应的下一个内存地址上读取下一条数据, 并将其放入到 "指令队列缓冲器" 中;
7. IP 中的值自动增加 (以使 CPU 可以获取下一条指令, 当前读入的指令 "8B" 长度为1个字节, 所以 IP 中的值应加 1 个字节。此时CS:IP 指向内存单元 1010:0001) ;
8. CPU 访问内存 10101H 单元, 读取指令 "80" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在通过 "指令队列缓冲器" 到达 "执行部分电路控制" 中后, CPU 解析指令 (IP值继续增加) ;
9. 解析指令后, CPU 将 "通用寄存器bx" 以及 "通用寄存器si" 中保存的数值 "0700H" "0300H" 分别移动到 CPU 的算数逻辑单元 "ALU"中;
10. CPU 通过 CS:IP 继续访问内存单元 "10102H", 读取指令 "00" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后 ("寄存器IP" 中的值自动增加当前读取的字节数) , 在将其送入 "段寄存器" 中的 "内部暂存器" 中;
11. CPU 继续访问 CS:IP 指向的内存 "10103H" 单元, 读取指令 "10" 通过 "输入输出控制电路" 送入 "指令队列缓冲器" 中后, 在将其送入 "段寄存器" 中的 "内部暂存器" 中;
12. "内部寄存器" 将两次读取到的偏移数值拼接, 结果为 "1000H" ("寄存器IP" 中的值继续自动增加当前读取的字节数; ) 。
13. "内部暂存器" 将结果 "1000H" 送入到 CPU 的算数逻辑单元 "ALU"中;
14. CPU 的算数逻辑单元 "ALU", 将之前接收到的两个结果的和和这个 "1000H" 进行相加, 将相加后的结果 "2000H" 送回到 "内部暂存器" 中。
15. CPU 将 "数据段寄存器ds" 对应的的首地址 "3000H" 和得到的偏移地址 "2000H" 通过 "地址加法器" 得出对应的物理地址 "32000H";
16. CPU 通过 "输入输出控制电路" 访问对应的物理地址 "32000H", 取出操作数 "1234H";
17. CPU 通过 "输入输出控制电路" 将取出的操作数 "1234H" 传回段寄存器中的 "内部暂存器" 中;
18. IP 中的值根据当前读取的指令字节数自动增加, CPU 会继续访问对应内存上的指令数据;
19. CPU 将保存在 "内部暂存器" 中的操作数 "1234H" 根据对应的 "操作码" 将其移动到 "寄存器ax" 中。

通过流程可以看出 "相对基址变址寻址方式" 相对于 "寄存器相对寻址方式" 得出结果的时间效率还要慢 (多了一步运算操作, 都是访问了5次内存) 。

## 对所有寻址方式的性能做一个排序

由快到慢：

1. 寄存器寻址方式
2. 立即数寻址方式
3. 寄存器间接寻址方式
4. 直接寻址方式（理论上比基址变址寻址方式快一点，有缓存的有原因，mov指令执行完后，偏移值就已经在缓存中了）
5. 基址变址寻址方式
6. 寄存器相对寻址方式
7. 相对基址变址寻址方式