计算机组成与结构
—指令系统2

计算机科学与技术学院

温故 —关于指令系统

- **CPU执行指令的4个步骤是?**

  取指令、译码、执行、存储

- **指令的格式是什么?**

  操作码Opcode + 操作数Operand (0~N)

本节学习要点

- **可执行程序的结构**
- **对Emu8086做简单使用介绍**
- **CPU是如何定位程序的? (寄存器)**
- **CPU是如何定位程序中的代码或数据的? （寻址方式）**

可执行程序的两个结构

可执行程序的两个结构

- **一个最简单的C代码：　　其实一点也不简单…**

```
int main() {
    printf("Hello world!\n");
}
```

可执行程序的两个结构

- 一个最简单的C代码：

```
int main() {
    printf("Hello world!\n");
}
```

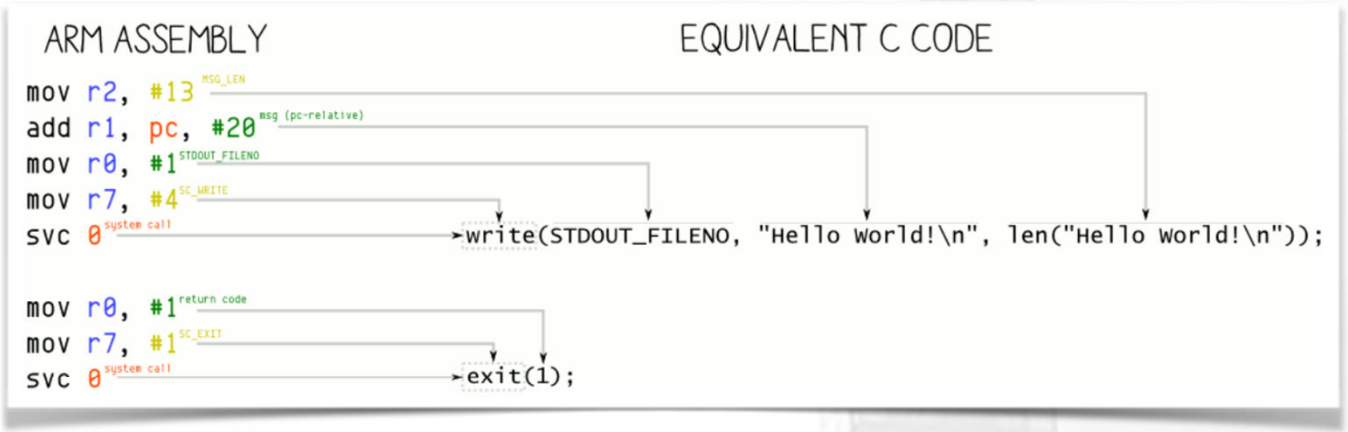
经历2个步骤变成可执行程序：

- 编译（把自己的程序逻辑变为机器代码）
- 链接（链接其它库的函数调用）

- 可执行程序的静态结构（ELF为例）
  - ELF Header
  - Sections
    - CODE
    - DATA
    - BSS
    - RODATA
    - HEAP
    - STACK
    - …
  - Section Header Table

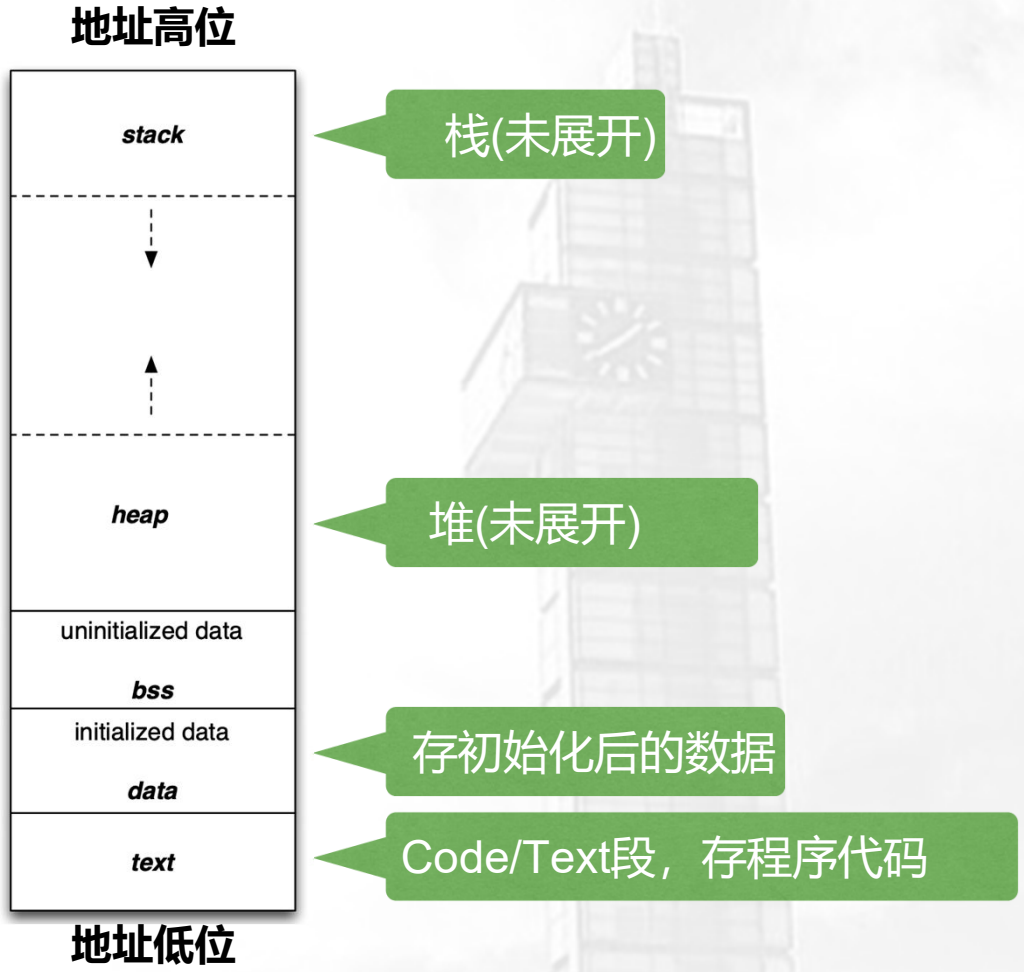
可执行程序的两个结构

- 可执行程序的**静态结构**（ELF为例）
  - ELF Header
  - Sections
    - CODE
    - DATA
    - BSS
    - RODATA
    - HEAP
    - STACK
    - …
  - Section Header Table

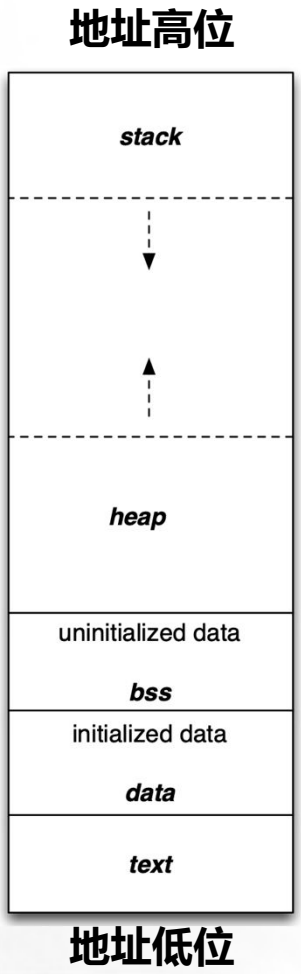


可执行程序的两个结构

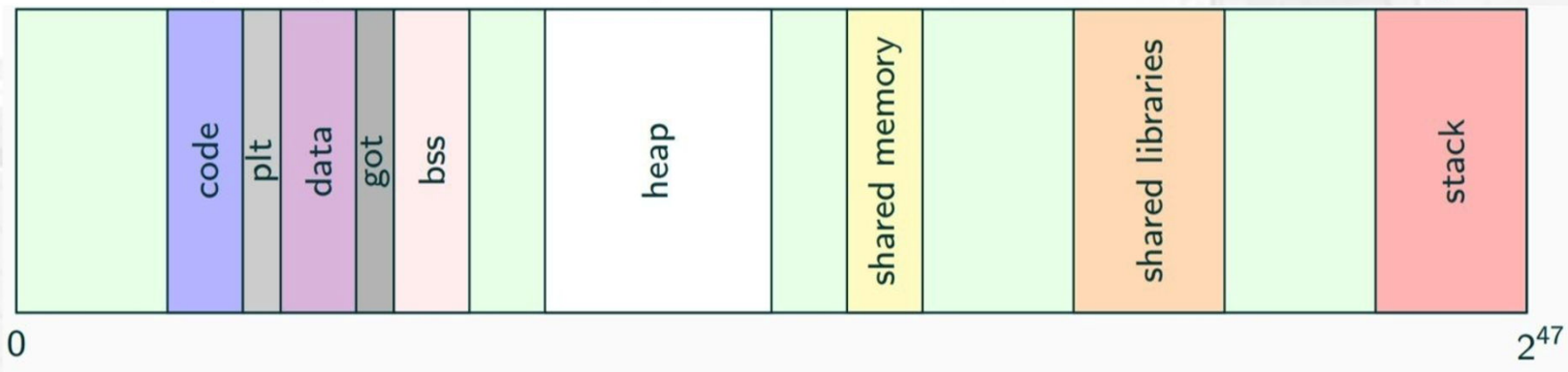
- 可执行程序的静态结构（ELF为例）
  - ELF Header
  - Sections
    - CODE
    - DATA
    - BSS
    - RODATA
    - HEAP
    - STACK
    - …
  - Section Header Table



可执行程序的两个结构



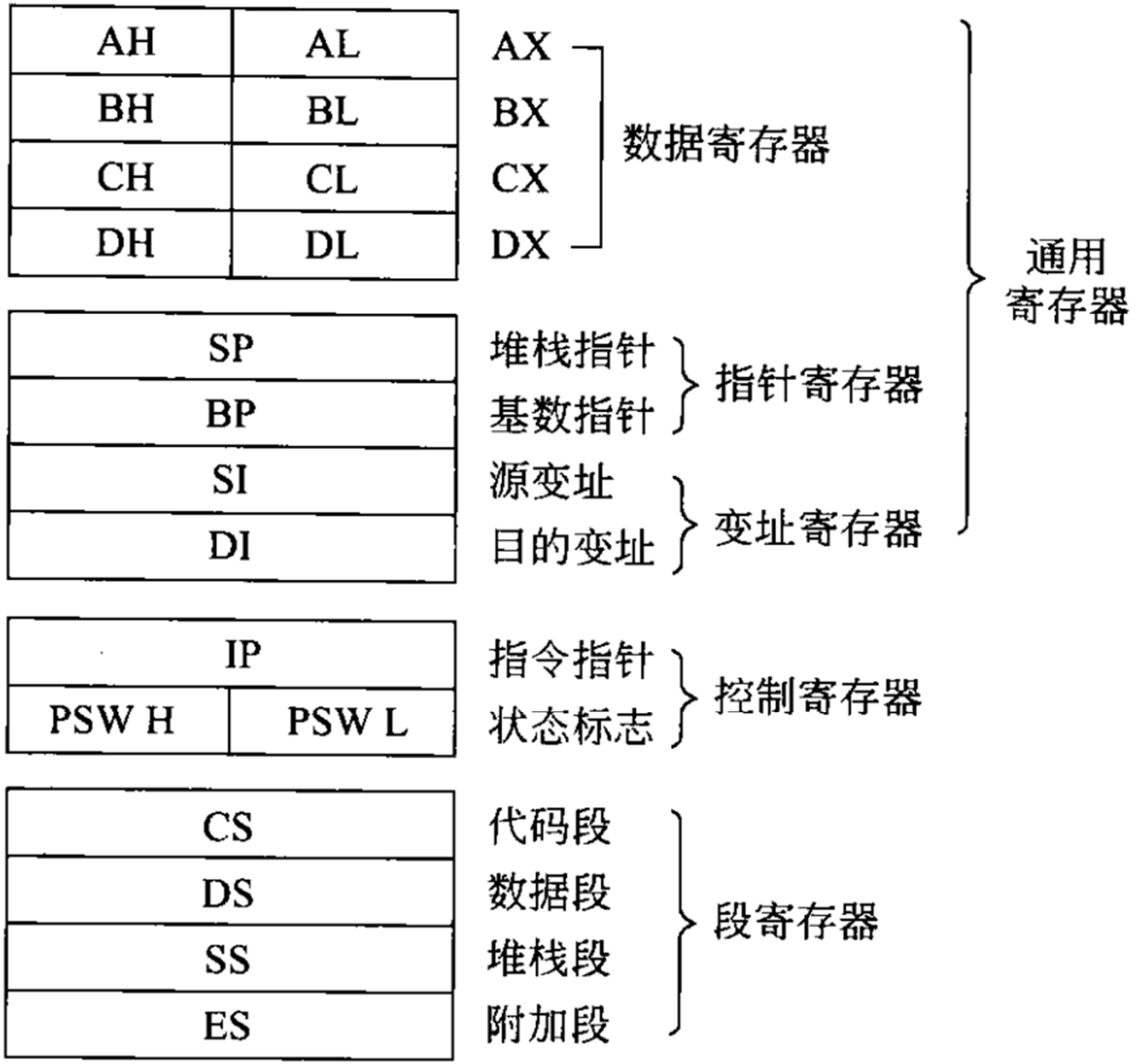
- 可执行程序的**动态结构—进程！**
  - 程序映射到4GB虚拟内存空间
    - 每个线程有自己的Stack/Heap
  - 虚拟地址被OS映射到实际物理地址



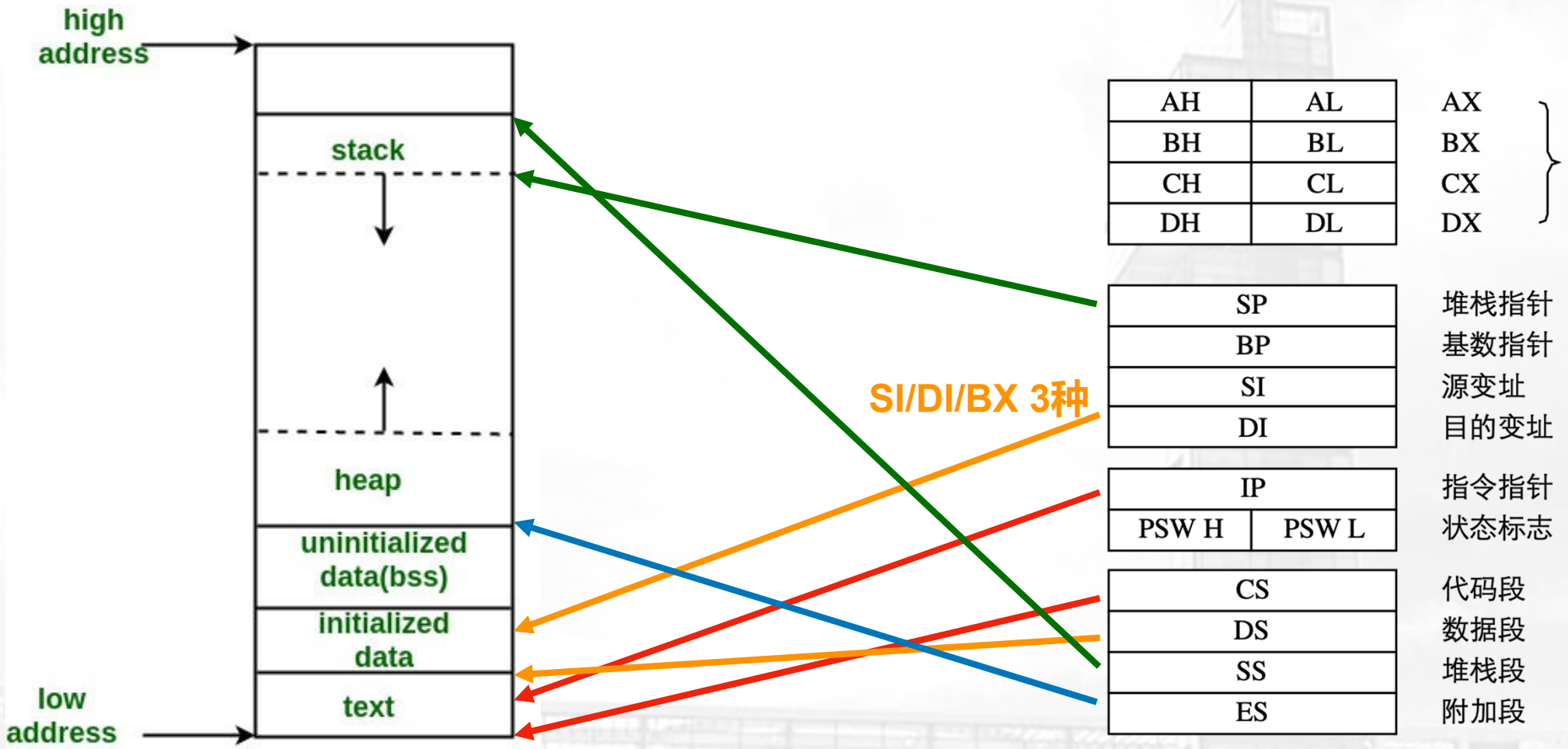
CPU的内部寄存器

8086 CPU的内部寄存器

- Registers，寄存器
  - 存储CPU运行时的各种状态，阶段，内存读写地址等
- 寄存器：
  - 数据寄存器
  - 指针寄存器
  - 变址寄存器
  - 控制寄存器
  - 段寄存器

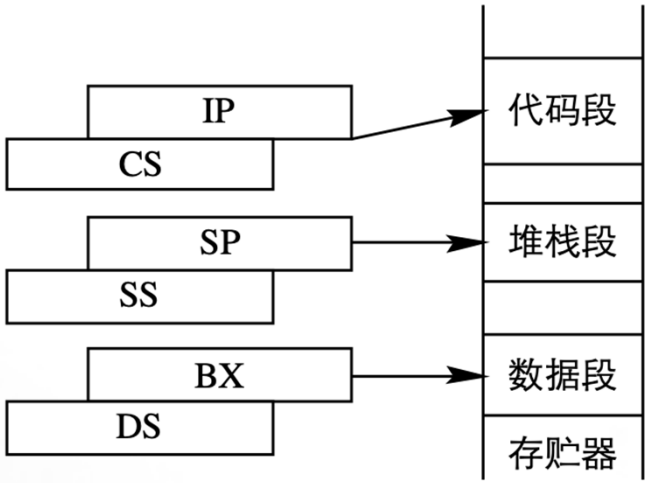


CPU的内部寄存器

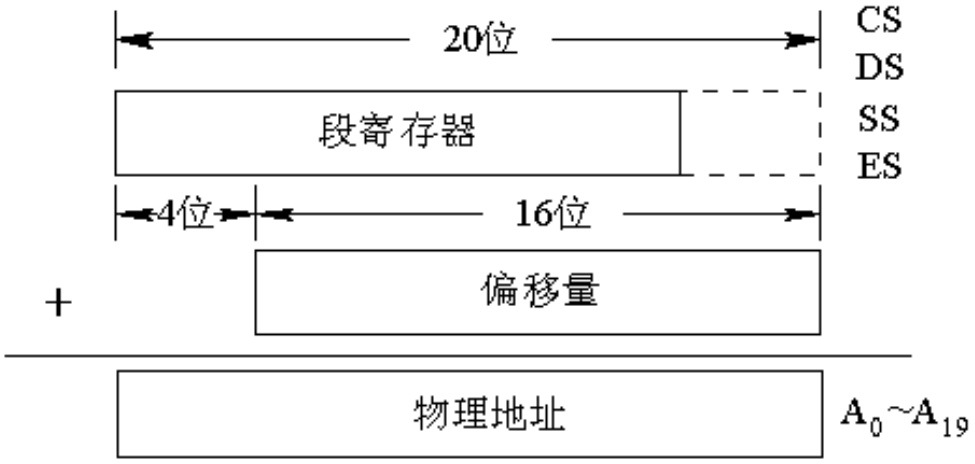

high address
stack
heap
uninitialized data(bss)
initialized data
text
low address
SI/DI/BX 3种
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
SP
堆栈指针
BP
基数指针
SI
源变址
DI
目的变址
IP
指令指针
PSW H
PSW L
状态标志
CS
代码段
DS
数据段
SS
堆栈段
ES
附加段

CPU的内部寄存器

物理地址=段寄存器x16 + 偏移地址



Then, How ???



AH	AL	AX	数据寄存器
BH	BL	BX	
CH	CL	CX	
DH	DL	DX	

SP	堆栈指针	指针寄存器
BP	基数指针	
SI	源变址	指针寄存器
DI	目的变址	

IP	指令指针	控制寄存器
PSW H / PSW L	状态标志	

CS	代码段	段寄存器
DS	数据段	
SS	堆栈段	
ES	附加段	

CPU的内部寄存器

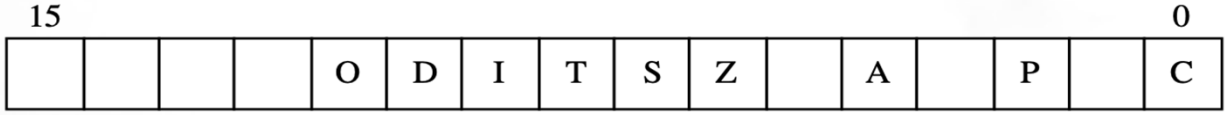
- 数据寄存器:
  - 4个16位长，AX, BX, CX, DX
  - 或8个8位，AH, AL, …
- 指针寄存器: SP和BP
  - BP是基址指针寄存器，通常用于存放基地址，以使8088的寻址更加灵活；
  - SP是堆栈指针寄存器，由它和堆栈段寄存器一起来确定堆栈内数据在内存中的位置；

CPU的内部寄存器

- **变址寄存器: SI/DI:**
  - **Source/Destination Index, 源/目的变址，用于变址寻址;**

CPU的内部寄存器

- **控制寄存器: PSW**
  - **Process State Word处理机状态字**



C —进位标志位
A —半加标志位
S —符号标志位
I —中断允许标志位
O —溢出标志位

P —奇偶标志位
Z — 零标志位
T —陷阱标志位(单步标志位)
D —方向标志位

CPU的内部寄存器

- **AMD64 / x86-64 （2023年）**

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

CPU的内部寄存器

- **ARM AArch64（2020年）**

Registers across CPU modes

usr	sys	svc	abt	und	irq	fiq
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

寻址方式

运算，
需要数据，
需要取数据，
需要知道用哪种方式去取数据，
还需要知道从哪里取数据，
—— 寻址方式
Addressing Mode！

但在此之前，
必须仔细学习一个汇编指令…

- **MOV指令**

格式：　**MOV　OPRD1，　OPRD2**

目的　　源

功能：　（**OPRD2**）→（**OPRD1**）

其中：　目的操作**OPRD1**

存储器	**mem**
通用寄存器	**r**
段寄存器	**SEG**（除**CS**）

源操作数　**OPRD2**

立即数	**im**
存储器	**mem**
通用寄存器	**r**
段寄存器	**SEG**（含**CS**）

有4种例外情况：

- 目的操作数不能为立即数，或 CS、IP；
- 内存单元之间不能进行数据直接传送；
- 立即数不能直接传送到段寄存器中；
- 段寄存器之间的不能进行数据直接传送；

Done ！

寻址方式Addressing Mode

- Immediate 立即
- Direct 直接
- Register 寄存器
- Register + Indirect 寄存器间接 (2种)
- Register + Indexing 寄存器相对
- Base + Displacement 基址变址 (2种)
- Base + Displacement + Indexing 基址+变址+相对
- Implicit 隐式

寻址方式 — 立即寻址

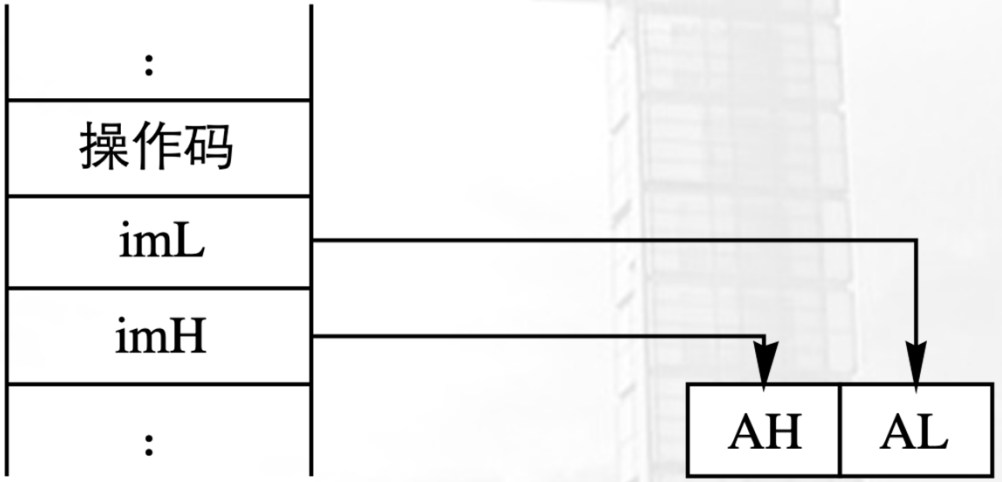
- **操作数直接包含在指令中！ (跟在操作码后面)**
- **操作数是在代码段！！**

例如:

MOV AL, 27H

ADD AL, 35H

MOV AX, 2000H



注意！！ 地址低位去AL，高位去AH !!!!

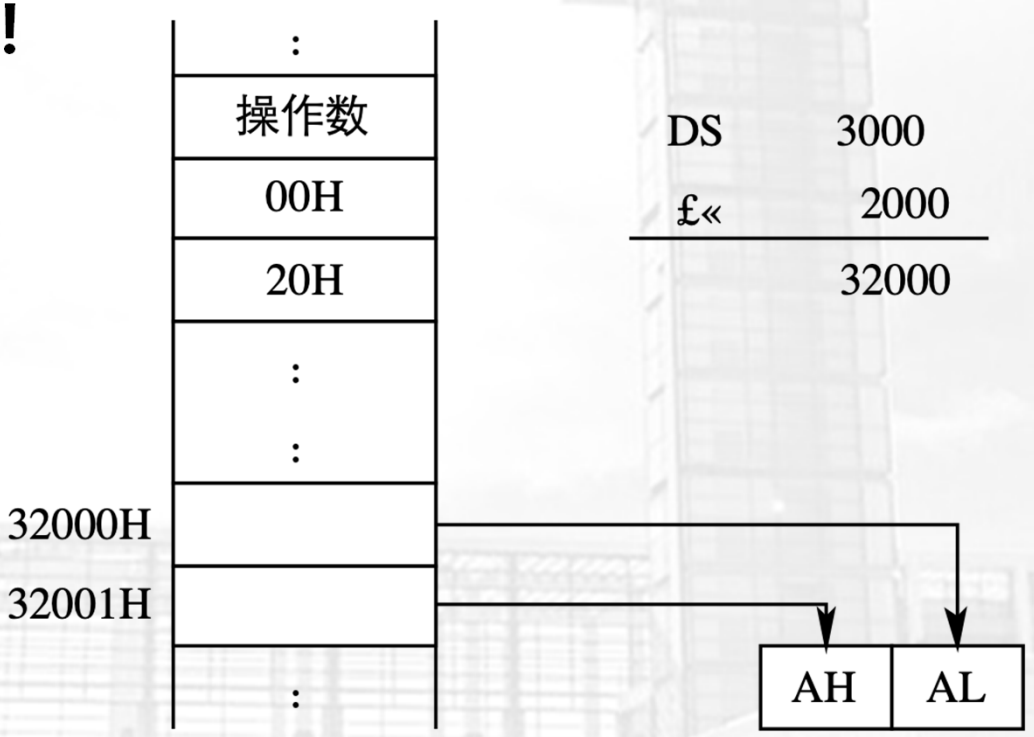
寻址方式 — 直接寻址

- **操作数的段内偏移地址直接包含在指令中！（跟在操作码后面）**
- **操作数是在数据段！！**

例如：

MOV AX, DS: [2000H]

设DS=3000H



寻址方式 — 寄存器寻址

- **操作数在某个寄存器中！**

例如:

MOV DS, AX

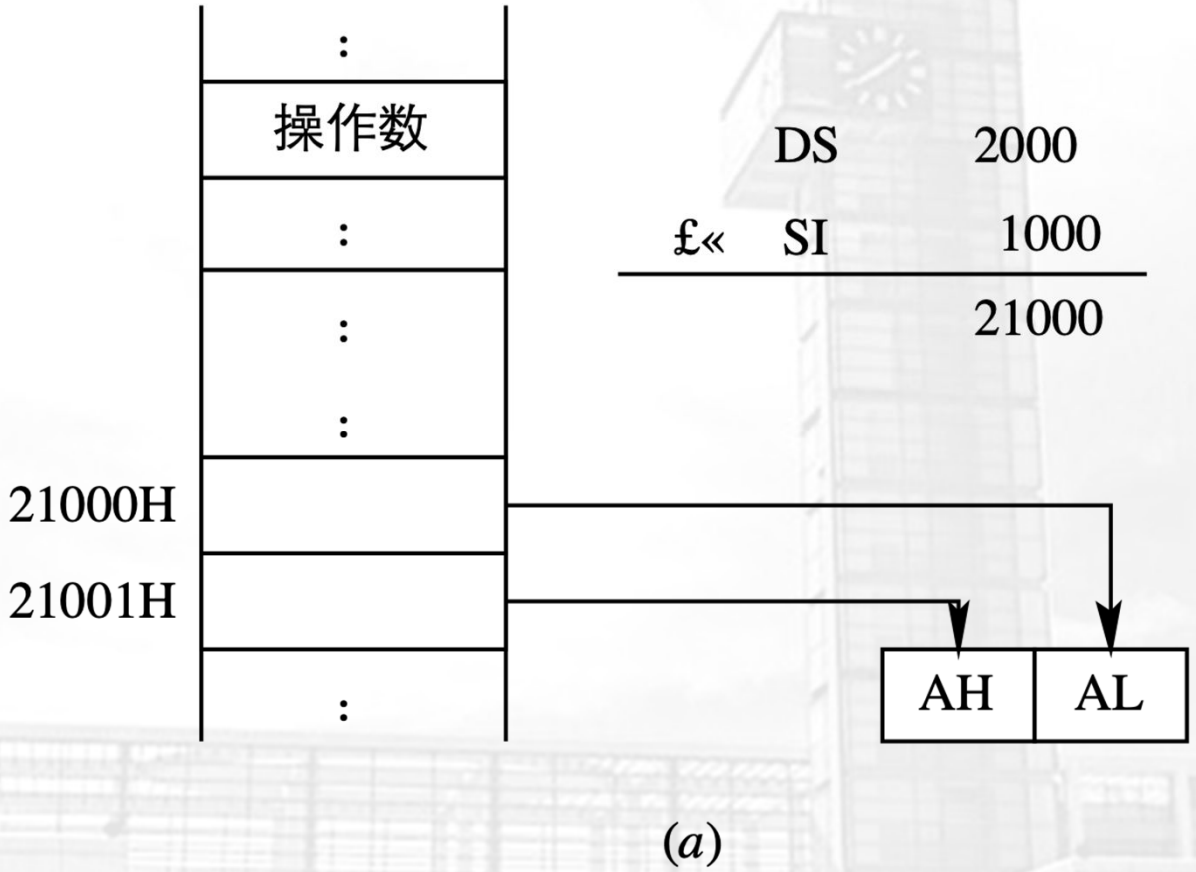
寻址方式 — 寄存器间接寻址

- 操作数**的段内偏移地址**放在SI/DI/BP/BX其中一个寄存器中!
- 分两种情况:
  - 如果是SI/DI/BX　<—搭配DS在数据段找操作数
  - 如果是BP　<—搭配SS在堆栈段找操作数

寻址方式 — 寄存器间接寻址

- **例如：**

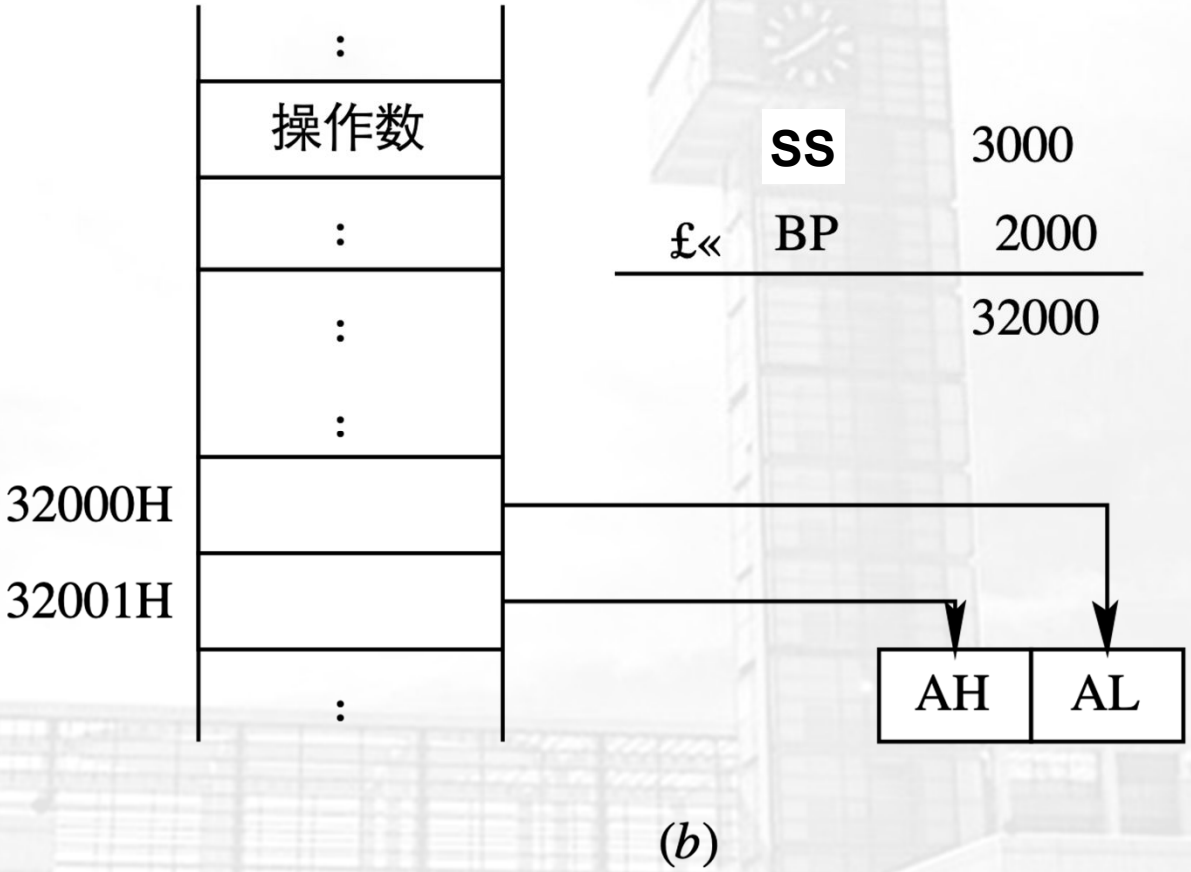
```
MOV AX, 2000H
MOV DS,  AX
MOV SI,  1000H
MOV AX,  [SI]
```



寻址方式 — 寄存器间接寻址

- **例如：**

```
MOV AX, 3000H
MOV SS,  AX
MOV BP,   2000H
MOV AX,  [BP]
```



寻址方式 — 寄存器间接寻址

- **例:假设有指令：MOV BX,[DI]，在执行时，(DS)=1000H，(DI)=2345H，存储单元12345H的内容是4354H。问执行指令后，BX的值是什么?**

- **解:**

根据寄存器间接寻址方式的规则，在执行本例指令时，寄存器DI的值不是操作数，而是操作数的地址。该操作数的物理地址(Physical Address，PA)应由DS和DI的值形成，即:

PA=(DS)*16+DI=1000H*16+2345H=12345H。

所以，该指令的执行效果是：把从物理地址为12345H开始的一个字的值传送给BX。

寻址方式 — 寄存器间接寻址

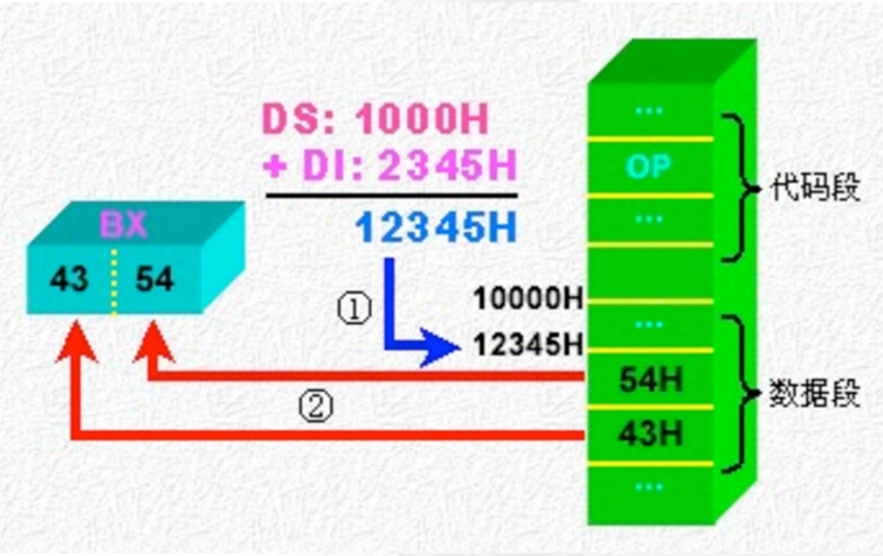
- **例:假设有指令：MOV BX,[DI]，在执行时，(DS)=1000H，(DI)=2345H，存储单元12345H的内容是4354H。问执行指令后，BX的值是什么?**

- **解:**

根据寄存器间接寻址方式的规则，在执行本例指令时，寄存器DI的值不是操作数，而是操作数的地址。该操作数的物理地址(Physical Address, PA)应由DS和DI的值形成，即:

PA=(DS)*16+DI=1000H*16+2345H=12345H。

所以，该指令的执行效果是：把从物理地址为12345H开始的一个字的值传送给BX。



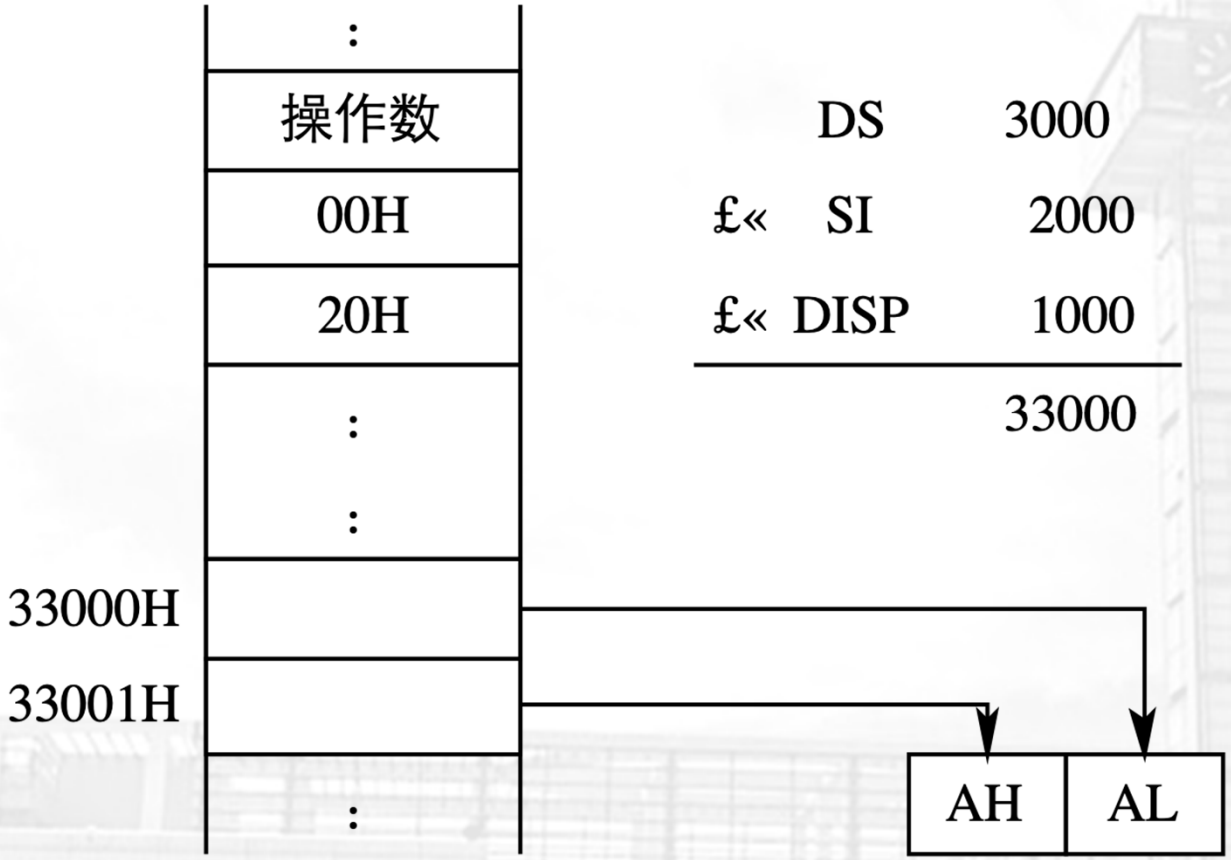
寻址方式 — 寄存器相对寻址

- 本质是寄存器间接 + 相对寻址：在寄存器间接的基础上再加上地址偏移值DISP；
- DISP跟着操作码后面，在代码段！！！
- 同样分两种情况：
  - 如果是SI/DI/BX　<—搭配DS在数据段找操作数
  - 如果是BP　<—搭配SS在堆栈段找操作数

例：MOV AX，DISP［SI］

寻址方式 — 寄存器相对寻址

例：MOV AX，DISP［SI］； DISP=1000H ; DS = 3000H; SI = 2000H



寻址方式 — 寄存器相对寻址

例: 假设指令：MOV BX, [SI+100H]，在执行它时，(DS)=1000H，(SI)=2345H，内存单元12445H的内容为2715H，问该指令执行后，BX的值是什么?

解：根据寄存器相对寻址方式的规则，在执行本例指令时，源操作数的有效地址EA为：

EA=(SI)+100H=2345H+100H=2445H

该操作数的物理地址应由DS和EA的值形成，即：

PA=(DS)*16+EA=1000H*16+2445H=12445H。

所以，该指令的执行效果是：把从物理地址为12445H开始的一个字的值传送给BX。

寻址方式 — 寄存器相对寻址

例: 假设指令：MOV BX, [SI+100H]，在执行它时，(DS)=1000H，(SI)=2345H，内存单元12445H的内容为2715H，问该指令执行后，BX的值是什么？

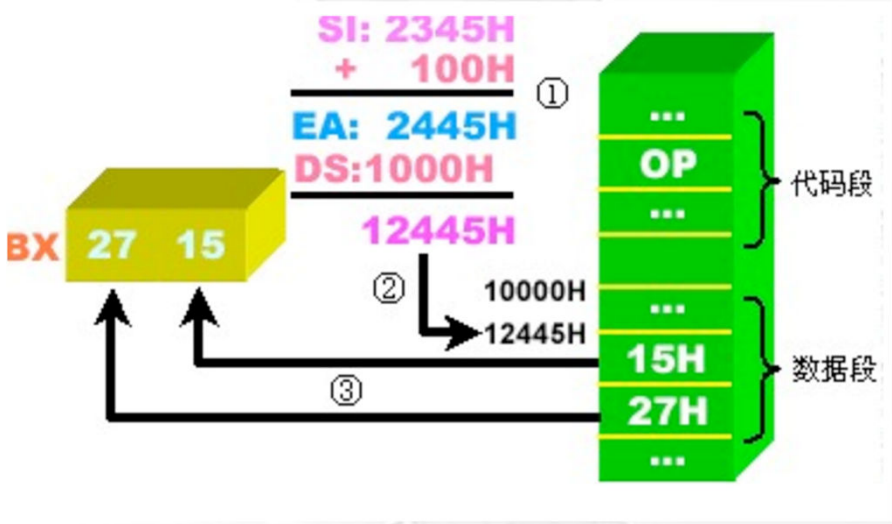
解：根据寄存器相对寻址方式的规则，在执行本例指令时，源操作数的有效地址EA为：

EA=(SI)+100H=2345H+100H=2445H

该操作数的物理地址应由DS和EA的值形成，即：

PA=(DS)*16+EA=1000H*16+2445H=12445H。

所以，该指令的执行效果是：把从物理地址为12445H开始的一个字的值传送给BX。



数据寻址方式 — 总结

- **加起来！！！ 把右边的地址值全加起来！！！**
- **BX找DS！！！！**
- **BP找SS！！！！**
- **“直接” v.s “间接”**
- **“相对” v.s “变址”**

源操作数	指令的变形	源操作数的寻址方式
只有偏移量	MOV AX, [100H]	直接寻址方式
只有一个寄存器	MOV AX, [BX] 或 MOV AX, [SI]	寄存器间接寻址方式
有一个寄存器和偏移量	MOV AX, [BX+100H] 或 MOV AX, [SI+100H]	寄存器相对寻址方式
有二个寄存器	MOV AX, [BX+SI]	基址加变址寻址方式
有二个寄存器和偏移量	MOV AX, [BX+SI+100H]	相对基址加变址寻址方式