



# Assembly Programming

## 第七周 指令系统

庞彦

yanpang@gzhu.edu.cn

- 8086/8088指令概述
- 数据传送指令
- 算术指令
- 逻辑指令
- 串处理指令
- 处理机控制与杂项操作指令

## **重点关注：**

- **指令的汇编格式及特点**
- **指令的基本功能**
- **指令的执行对标志位的影响**
- **指令的特殊要求**

- 8086/8088指令概述
- 数据传送指令
- 算术指令
- 逻辑指令
- 串处理指令
- 处理机控制与杂项操作指令

# 8086/8088指令概述

- 指令格式      操作码      操作数

例      ADD      AL, 10H

## (1) 操作码

指明CPU要执行什么样的操作。

是一条指令必不可少的部分，用助记符表示。

按功能  
指令分六类

数据传送  
算术运算  
逻辑运算  
串操作  
控制转移  
处理机控制

## (2) 操作数

指明参与操作的数据或数据所在的地方。

了解操作数的来源、个数、类型、执行速度。

# 8086/8088指令概述

操作数三种来源:

① 操作数在指令中, 称立即数操作数

如 MOV AL, 9

② 操作数在寄存器中, 称寄存器操作数

指令中给出用符号表示的寄存器名。

如 MOV AL, 9; MOV AL, BL

③ 操作数在内存单元中, 称存储器操作数或内存操作数

指令中给出该内存单元的地址。用[ ]表示存储器操作数

如 MOV AL, [2000H]

## 8086/8088指令概述

### □ 操作数个数

按指令格式中，操作数个数的多少分为四类：

**无操作数:** 指令只有一个操作码，没有操作数

**单操作数:** 指令中给出一个操作数

**双操作数:** 指令中给出两个操作数

**三操作数:** 指令中给出三个操作数

## 8086/8088指令概述

① **无操作数**：指令只有一个操作码，没有操作数。

有两种可能：

▲ 有些操作不需要操作数。

如 HLT（暂停），NOP（无操作）等处理机控制指令。

▲ 操作数隐含在指令中。

如 STC（CF=1），CLC（CF=0）等处理机控制命令。



## 8086/8088指令概述

② 单操作数: 指令中给出一个操作数。

有两种可能:

▲ 有些操作只需要一个操作数

如 `INC AL` ;  $(AL) \leftarrow (AL) + 1$

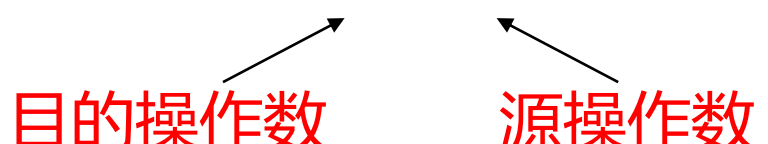
▲ 有些操作将另一个操作数隐含在指令中

如 `MUL BL` ;  $(AX) \leftarrow (AL) \times (BL)$

## 8086/8088指令概述

③ 双操作数: 指令中给出两个操作数。

如    ADD    AL , BL    ; (AL)  $\leftarrow$  (AL) + (BL)



目的操作数                  源操作数

操作后的结果通常存放在目的操作数中。

## 8086/8088指令概述

③ 三操作数: 指令中给出三个操作数。

如 IMUL BX, Array[100], 7 ;7\*(Array[100])->BX

SHLD EBX, ECX, 16 ;双精度左移

目的操作数: 1, 源操作数: 2

操作后的结果通常存放在目的操作数中。

## 8086/8088指令概述

### □ 操作数类型

8086/8088:

有的操作既可对字节操作，又可对字操作

有的操作只允许对字操作

- ◆ 指令应指明参与操作的数是字节还是字，即操作数的类型。
- ◆ 通常操作数的类型可由操作数本身隐含给出。
- ◆ 在特殊情况下需要指明。

## 8086/8088指令概述

① 指令中有寄存器操作数，由寄存器操作数决定类型。

例： MOV [BX], AL ;字节操作, [BX] ← AL

MOV [BX], AX ;字操作, [BX] ← AL, [BX+1] ← AH

MOV BX, AL ; ???

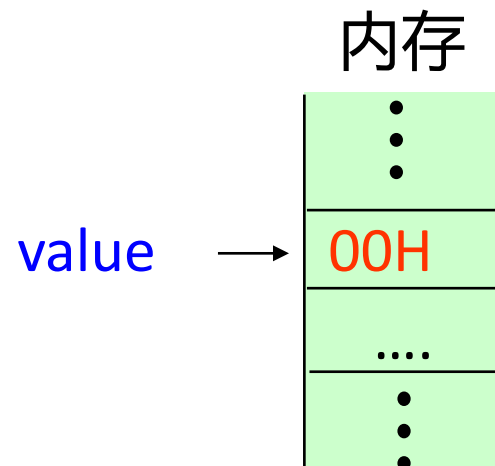
## 8086/8088指令概述

② 指令操作数中无寄存器，则由内存操作数的类型决定。

*例* value 是一个变量 (即内存操作数);

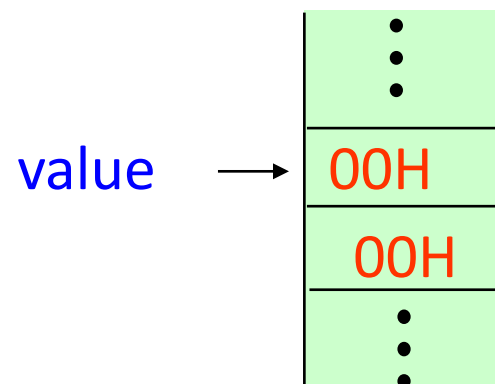
若定义value 为字节类型:

则 MOV value , 0 是字节操作。



若定义value 为字类型:

则 MOV value , 0 是一个字操作。



## 8086/8088指令概述

③ 指令中无类型的依据，需对存储器操作数加类型说明。

例

MOV [ BX ], 0



用 PTR 属性伪操作说明类型。

MOV byte PTR [BX], 0

字节操作, [ BX ]  $\leftarrow$  0

MOV word PTR [BX], 0

字操作, [ BX ]  $\leftarrow$  0, [ BX+1 ]  $\leftarrow$  0

# 8086/8088指令概述

## □ 执行速度

寄存器操作数    立即数操作数    存储器操作数

例

mov AL, BL

mov AL, 0

mov AL, [BX]

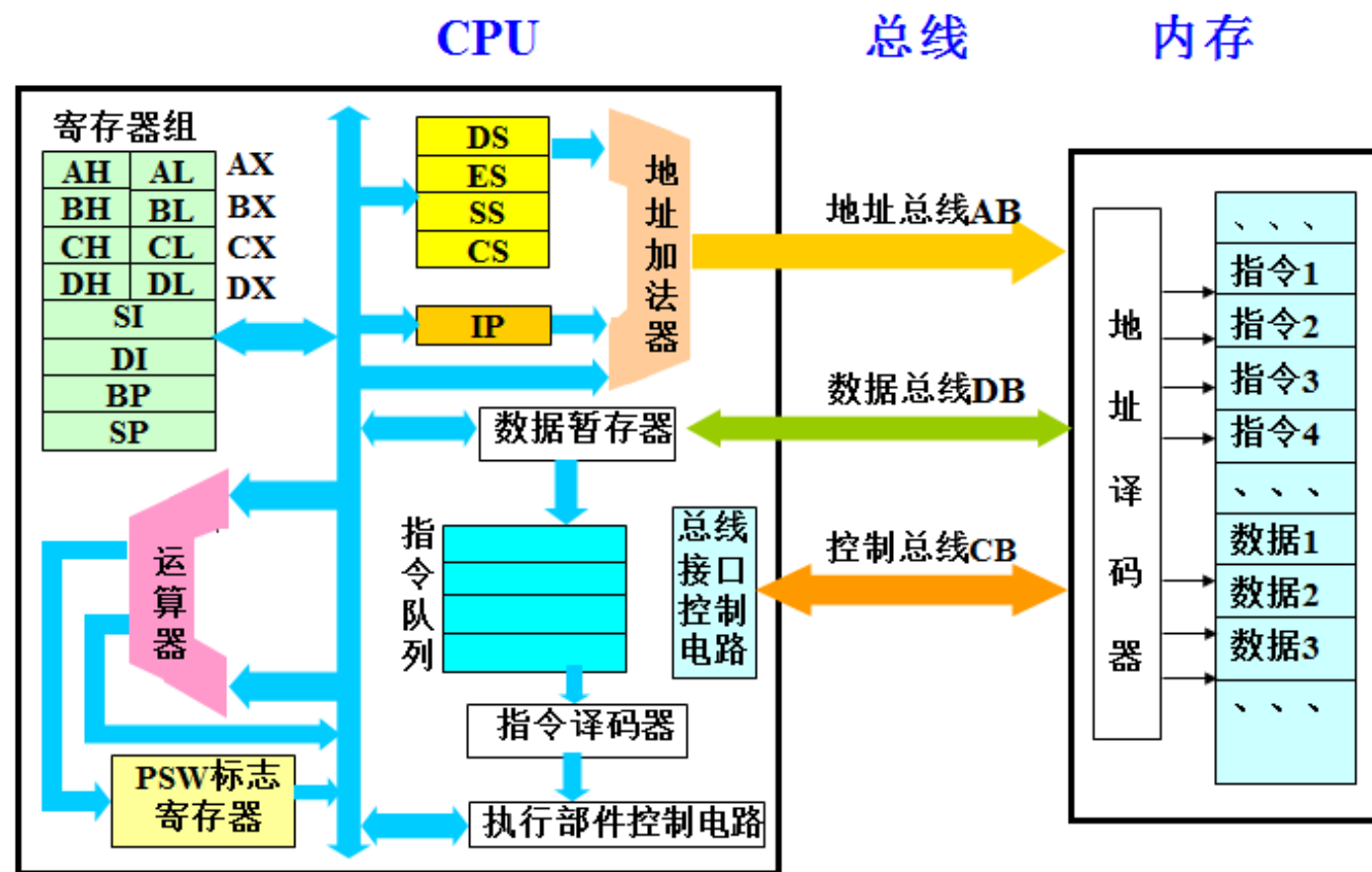
哪条指令执行速度快？

三条指令：

- ◆ 操作类型相同，都是传送指令
- ◆ 且目的操作数相同
- ◆ 仅源操作数不同



# 8086/8088指令概述



定长指令码格式：立即寻址最快，因为指令地址码即为操作数。  
变长指令码格式：因为立即寻址操作数可能很长，取指令时可能需要两次访存。而寄存器寻址取指令只需一次访存，所以寄存器寻址最快。

## 8086/8088指令概述

### 对同一类型指令，执行速度：

寄存器操作数      立即数操作数      存储器操作数

快

慢

例

```
mov    AL, BL
```

快

```
mov    AL, 0
```

```
mov    AL, [ BX ]
```

慢

## 第4&5讲：8086/8088的指令系统

- 8086/8088指令概述
- 数据传送指令
- 算术指令
- 逻辑指令
- 串处理指令
- 处理机控制与杂项操作指令

# 数据传送指令

- **通用数据传送指令**

MOV、PUSH、POP、XCHG、XLAT

- **地址传送指令**

LEA、LDS、LES

- **标志寄存器传送指令**

LAHF、SAHF、PUSHF、POPF

- **类型转换指令**

CBW、CWD

# 数据传送指令

- 通用数据传送指令

传送指令: **MOV DST, SRC**

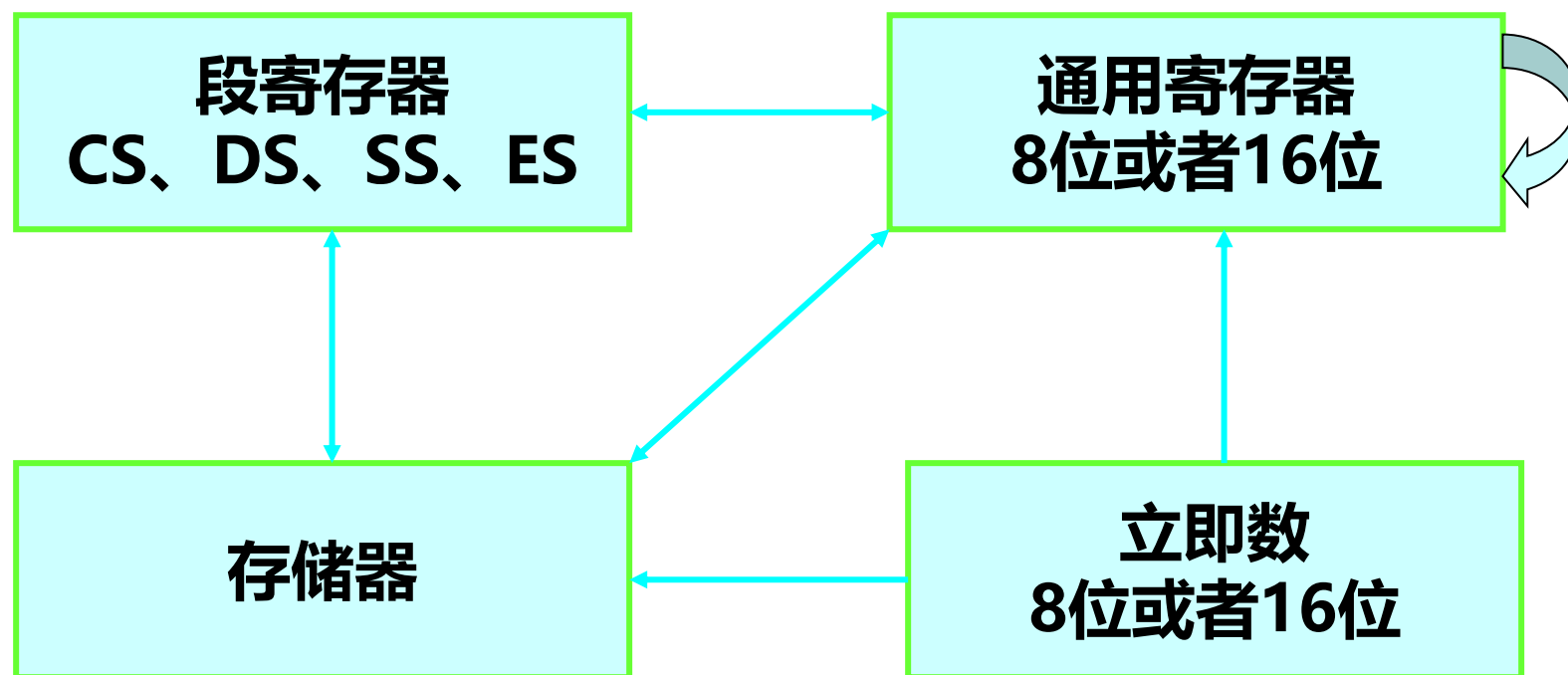
执行操作: **(DST) ← (SRC)**

**注意:**

- \* 两个操作数的数据类型要相同: **MOV BL, AX (×)**
- \* DST、SRC 不能同时为段寄存器: **MOV DS, ES (×)**
- \* 立即数不能直接送段寄存器: **MOV DS, 2000H (×)**
- \* DST 不能是立即数和CS: **MOV CS, AX; MOV 100H, AX (×)**
- \* DST、SRC 不能同时为存储器寻址: **MOV VARA, VARB (×)**
- \* 指令指针IP, 不能作为MOV指令的操作数
- \* 不影响标志位

## 数据传送指令

在汇编语言中，主要的数据传送方式如下图所示。虽然一条MOV指令能实现其中大多数的数据传送方式，但也存在MOV指令不能实现的传送方式。



## 数据传送指令

进栈指令: **PUSH SRC**

执行操作:  $(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (SRC)$

(SP)→



出栈指令: **POP DST**

执行操作:  $(DST) \leftarrow ((SP)+1, (SP))$

$(SP) \leftarrow (SP) + 2$

**堆栈: ‘先进后出’ 的存储区, 段地址存放在SS中,  
SP在任何时候都指向栈顶, 进出栈后自动修改SP。**

**注意:**

\* **堆栈操作必须以字为单位。**

\* **不影响标志位**

\* **POP不能用立即寻址方式**

\* **POP指令DST不能是CS**

**POP 1234H (×)**

**POP CS (×)**

# 数据传送指令

## PUSH指令的四种格式：

PUSH reg (寄存器)

PUSH mem (存储器)

PUSH segreg (段寄存器)

## POP指令的三种格式：

POP reg

POP mem

POP segreg (no CS)

例：

PUSH temp

PUSH AX

...

POP AX

POP BX

错误：

PUSH data (立即数)

POP data (立即数)



# 数据传送指令

例：假设  $(AX) = 2107H$ ，执行 **PUSH AX**



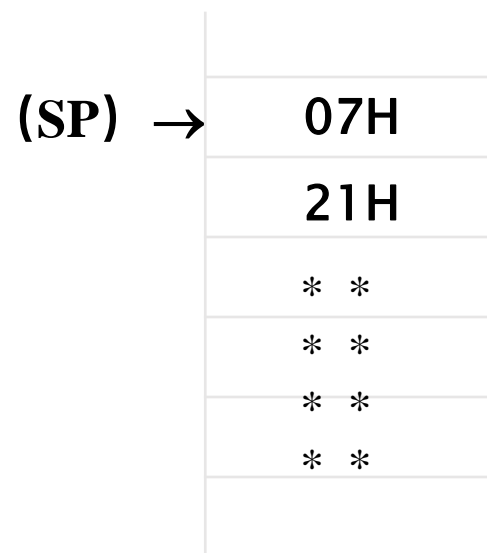
**PUSH AX 执行前**



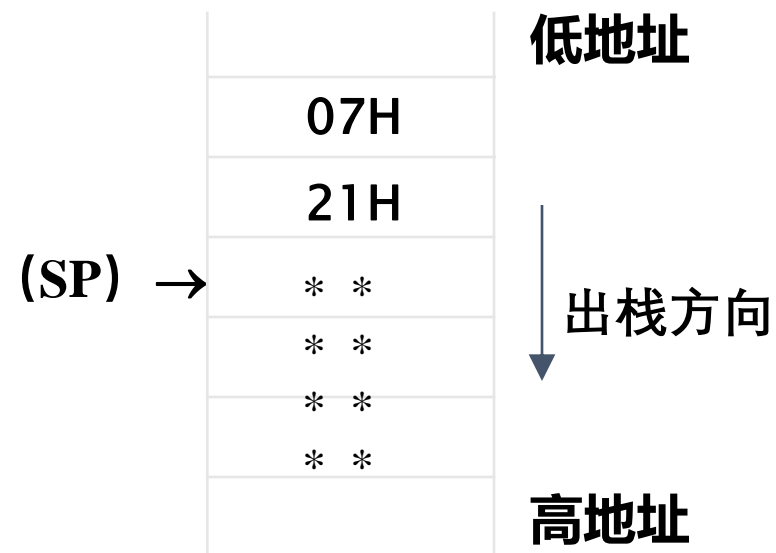
**PUSH AX 执行后**

# 数据传送指令

例： POP BX



POP BX 执行前

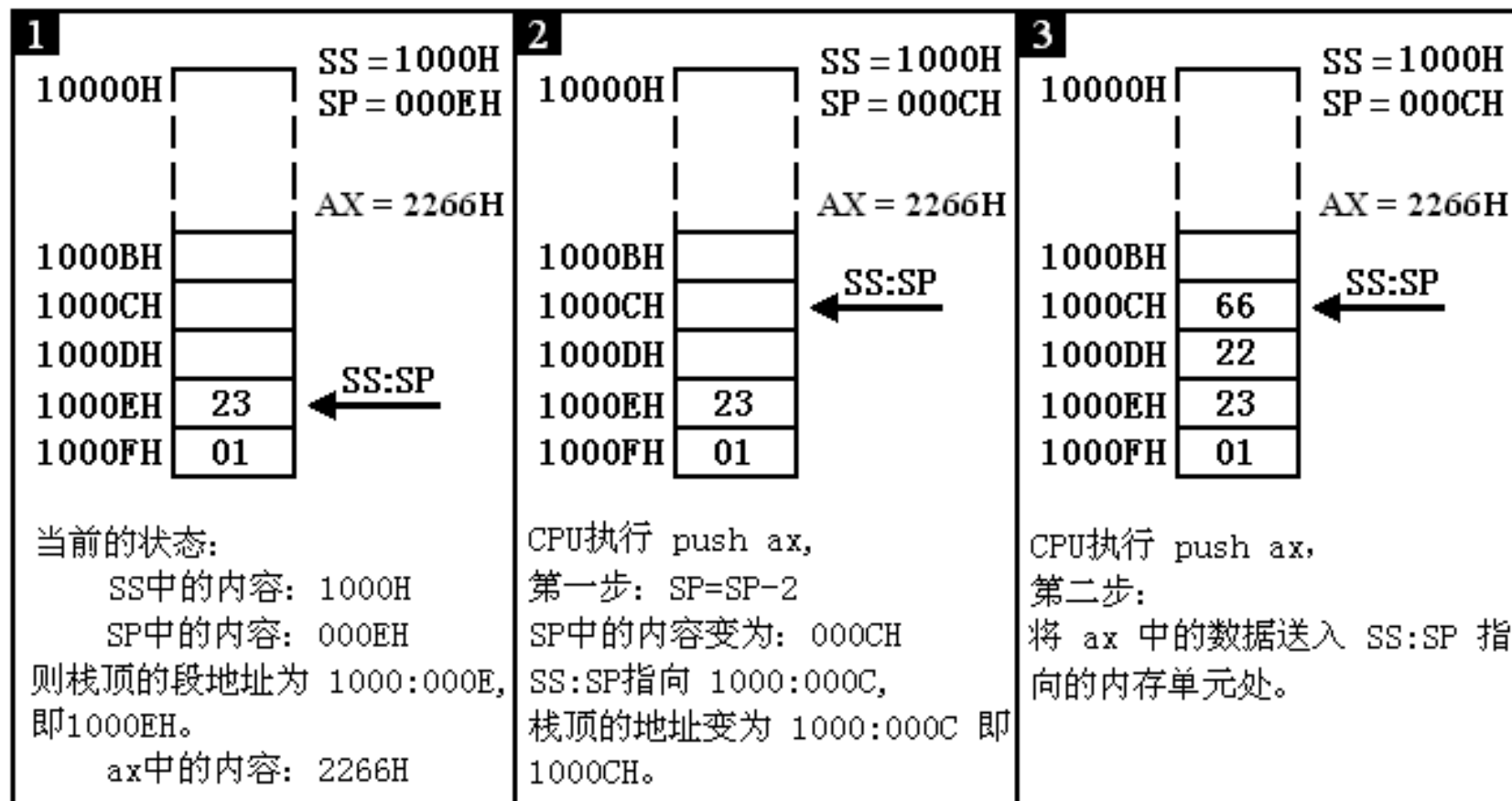


POP BX 执行后

(BX) = 2107H

## 数据传送指令

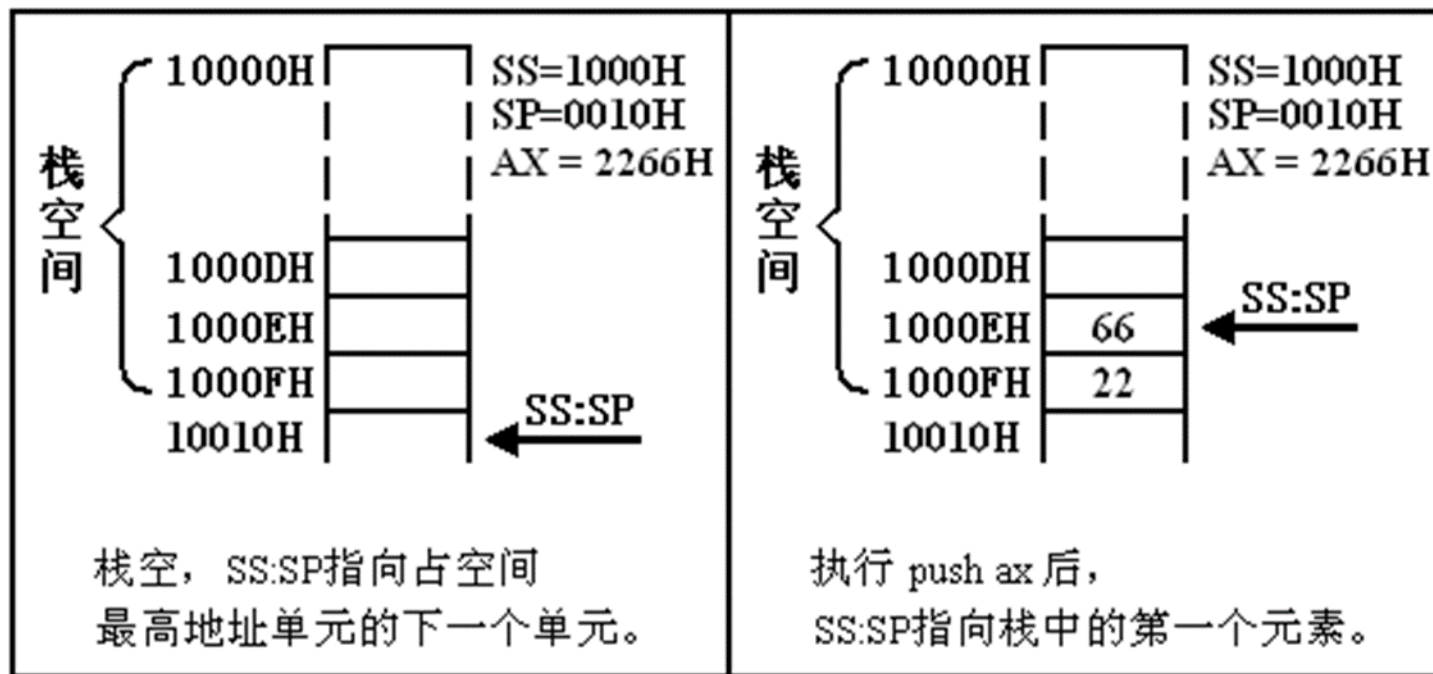
例：开辟一个16字节的堆栈，其SS=1000H，SP指向栈顶。即堆栈的内存地址范围为：10000H-1000FH。



## 数据传送指令

**思考：**如果我们将10000H~1000FH 这段空间当作栈，初始状态栈是空的，此时，SS=1000H，SP=?

- ◆ 任意时刻，SS:SP指向栈顶，当栈中只有一个元素的时候，SS = 1000H，SP=000EH。
- ◆ 栈为空，就相当于栈中唯一的元素出栈，出栈后，SP=SP+2，所以当栈为空的时候，SS=1000H，SP=10H。



# 数据传送指令

## 所有寄存器进栈指令：PUSH A

指令的格式为：PUSH A

功能：16位通用寄存器依次进栈，次序为AX、CX、DX、BX，指令执行前的SP、BP、SI、DI。指令执行后 $(SP)-16 \rightarrow (SP)$ 仍指向栈顶。

## 所有寄存器出栈指令：POP A

指令的格式为：POP A

功能：16位通用寄存器依次出栈，次序为DI、SI、BP、SP，指令执行前的BX、DX、CX、AX。指令执行后 $(SP)+16 \rightarrow (SP)$  仍指向栈顶。

。

**需要说明的是：**SP出栈只是修改了指针使其后的BX能够出栈，而堆栈中原先由PUSH A指令存入的SP的原始内容被丢弃，并未真正送到SP寄存器中。

上述两条堆栈指令均不影响标志位。

## 数据传送指令

**例：**在有子程序或中断调用的程序中，若有些寄存器的内容在子程序或中断调用后还要用到，则可以用堆栈来保存。

**PUSH AX**

**PUSH BX**

**PUSH CX**

**PUSH DX**

**.....**

**POP DX**

**POP CX**

**POP BX**

**POP AX**

**；其间用到AX、BX、CX、DX**

**；后进先出**

**PUSHA**

**...**

**POPA**

## 数据传送指令

**思考：**SS和SP只记录了栈顶的地址，依靠SS和SP可以保证在入栈和出栈时找到栈顶。可是，如何能够保证在入栈、出栈时，栈顶不会超出栈空间？

- ◆ 当栈满的时候再使用push指令入栈，栈空的时候再使用pop指令出栈，都将发生栈顶超界问题。
- ◆ 因为栈空间之外的空间里很可能存放了具有其他用途的数据、代码等，这些数据、代码可能是我们自己的程序中的，也可能是别的程序中的。
- ◆ 但是由于我们在入栈出栈时的不小心，而将这些数据、代码意外地改写，将会引发一连串的错误。
- ◆ **栈顶超界是危险的。8086CPU不保证对栈的操作不会超界。**
- ◆ 这就是说，8086CPU只知道栈顶在何处（由SS:SP指示），而不知道安排的栈空间有多大。这点就好像，CPU只知道当前要执行的指令在何处（由CS:SP指示）而不知道读者要执行的指令有多少。

## 数据传送指令

**交换指令：XCHG OPR1, OPR2**

**执行操作：(OPR1)  $\leftrightarrow$  (OPR2)**

**注意：**

- \* 不影响标志位**
- \* 两个操作数中必须有一个是寄存器**
- \* 不允许使用段寄存器**

**例：XCHG BX, [ BP+SI ]**

**XCHG AL, BH**

**例 假设(AL)=2AH, (204DH)=5BH, 则：**

**指令XCHG AL, [204DH]**

**执行后：(AL)=5BH, (204DH)=2AH**



# 数据传送指令

换码指令：XLAT 或 XLAT **OPR** (OPR只是为了增加可读性，首地址需要预先送到BX)

执行操作： $(AL) \leftarrow ((BX) + (AL))$

用于代码转换，例如：把字符扫描码转换成ASCII码，或者把数字0-9转换成7段数码管所需要的相应代码等。

在使用这条指令之前，应先建立一个字节表格：

表格的首地址→ (BX)

需转换的代码相对于首地址的位移量 → (AL)

指令执行完后，AL中即为转换后的代码。

Esc 110	F1 112	F2 113	F3 114	F4 115	F5 116	F6 117	F7 118	F8 119	F9 120	F10 121	F11 122	F12 123	Print Screen 124	Scroll Lock 125	Pause 126	Num Lock 127	Cap Lock 128	Scroll Lock 129	
- 1	! 2	@ 3	# 4	\$ 5	% 6	& 7	* 8	~ 9	^ 10	> 11	- 12	= 13	_ 14	Bk.Spc 15					
Tab 16	Q 17	W 18	E 19	R 20	T 21	Y 22	U 23	I 24	O 25	P 26	[ 27	] 28	Enter 29						
CapLock 30	A 31	S 32	D 33	F 34	G 35	H 36	J 37	K 38	L 39	; 40	' 41	Enter 42							
Shift 44	~ 45	Z 46	X 47	C 48	V 49	B 50	N 51	M 52	. 53	/ 54	~ 55	Shift 56							
Ctrl 58			Alt 60							Alt Gr 61			Ctrl 64						

英式102键键盘及其扫描码

# 数据传送指令

换码指令: XLAT 或 XLAT OPR

执行操作:  $(AL) \leftarrow ((BX) + (AL))$

例: MOV BX, OFFSET **TABLE**; (BX)=0040H

MOV AL, 3

XLAT **TABLE**

指令执行后 (AL)=33H

注意:

\*该指令不影响标志位

\*字节表格长度不能超过256个字节

(DS)=F000H		
<b>TABLE</b>		
(BX) →	30 H	F0040
(AL) = 3	31 H	F0041
	32 H	F0042
	33 H	F0043

# 数据传送指令

- 地址传送指令

有效地址送寄存器指令： **LEA REG, SRC**

执行操作： **(REG) ← SRC的有效地址**

例： **LEA BX, [BX+SI+0F62H]**

若指令执行前 **(BX) =0400H, (SI) =003CH**

则指令执行后 **(BX) =0400+003C+0F62=139EH**

思考：

(1) **MOV BX, [BX+SI+0F62H]** ； 该指令与上述指令有何区别？

(2) **LEA BX, LIST**

**MOV BX, OFFSET LIST** ； 这两条指令的功能是否相同？

## 数据传送指令

- 地址传送指令

指针送寄存器和DS指令: `LDS REG, SRC`

执行操作:  $(REG) \leftarrow (SRC)$   
 $(DS) \leftarrow (SRC+2)$

相继二字 → 寄存器、DS

指针送寄存器和ES指令: `LES REG, SRC`

执行操作:  $(REG) \leftarrow (SRC)$   
 $(ES) \leftarrow (SRC+2)$

相继二字 → 寄存器、ES

注意:

- \* 不影响标志位
- \* REG 不能是段寄存器
- \* SRC 必须为存储器寻址方式

## 数据传送指令

例: **LDS DI, [BX]**

如指令执行前 (DS) = B000H, (BX) = 080AH,  
(B080AH) = 05AEH, (B080CH) = 4000H

则指令执行后 (DI) = , (DS) =

## 数据传送指令

**例: LDS DI, [BX]**

如指令执行前 (DS) = B000H, (BX) = 080AH,  
(B080AH) = 05AEH, (B080CH) = 4000H

则指令执行后 (DI) = 05AEH, (DS) = 4000H

# Q&A



Fall 2023