



东北大学
Northeastern University

汇编语言程序设计

主讲：东北大学计算机学院 刘松冉

第五章 顺序结构程序

- 一. 程序设计的基本步骤
- 二. 流程图的应用
- 三. 程序的基本控制结构
- 四. 8086汇编指令
- 五. 简单的I/O功能调用
- 六. 顺序结构程序举例



一. 程序设计的基本步骤

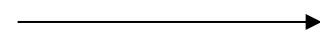
|▶ 1. 分析问题

2. 建立数学模型
3. 设计算法
4. 编制程序
5. 上机调试

二. 流程图的应用



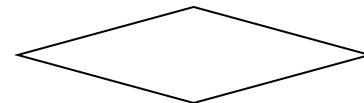
(1) 流程线



(2) 端点框



(3) 判断框



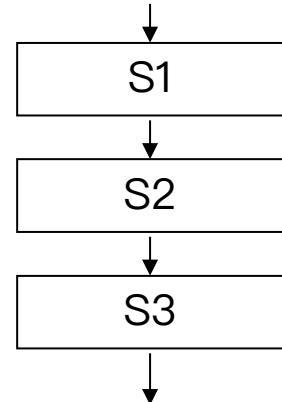
(4) 处理框



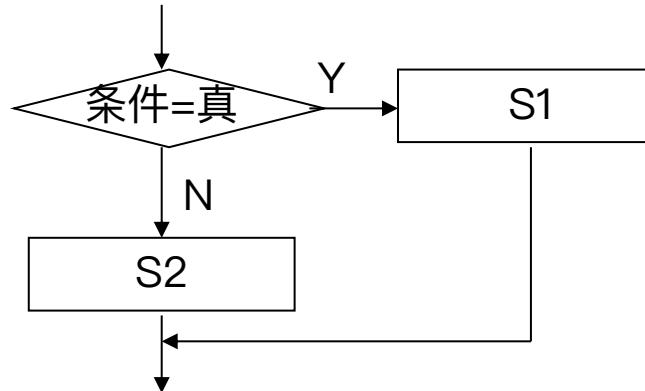
(5) 连接框



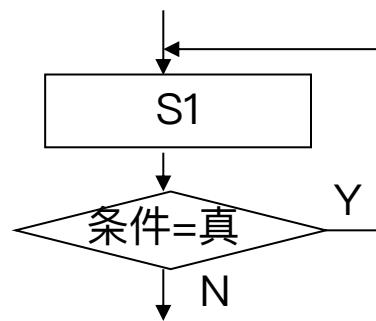
三. 程序的基本控制结构



(1) 顺序结构



(2) 分支（选择）结构



(3) 循环（迭代）结构

四. 8086汇编指令

- |▶ 1. 数据传送指令 (12)
- 2. 算数运算指令 (14)
- 3. 逻辑操作指令 (5)
- 4. 移位操作指令 (7)
- 5. 状态标志位操作指令 (4)

四. 8086汇编指令

|▶ 1. 数据传送指令 (12条)

- 1) MOV
- 2) PUSH
- 3) POP
- 4) XCHG
- 5) XLAT
- 6) LEA
- 7) LDS
- 8) LES
- 9) LAHF
- 10) SAHF
- 11) PUSHF
- 12) POPF

四. 8086汇编指令 – 1.数据传送指令

|> 1) MOV (move) 传送指令

- **指令汇编格式:** MOV dest, src
- **操作:** 将一个源操作数(字节或字) 传送到目标操作数中。
$$\text{dest} \leftarrow (\text{src})$$
- **受影响的状态标志位:** 没有
- **说明:** 指令中 dest 和 src **不能同时为存储器操作数**; CS**不能做为目标操作数使用**, 段寄存器之间不能互相传送, 立即数不能送入段寄存器。

四. 8086汇编指令 – 1.数据传送指令

|▶ 1) MOV (move) 传送指令

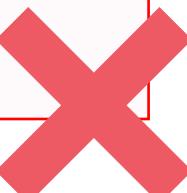
- 例1：

```
2  MOV      ALPHA_W, AX  
3  MOV      BETA_B, AL  
4  MOV      AL, ES:[BX+SI+1000H]  
5  MOV      BX, 1000H  
6  MOV      DS, BX  
7  MOV      [BX+10], AL  
8  MOV      [BX], 10H  
9  MOV      DS, 10H  
10 MOV     CS, AX
```

; ALPHA_W <- AL, ALPHA_W <- AH

BX:	34H	2000:1000H
	12H	2000:1001H

BX:	10H	2000:1000H
	??	2000:1001H



四. 8086汇编指令 – 1.数据传送指令

|▶ 1) MOV (move) 传送指令

- 例2：

```
2  MOV      ALPHA_W, AX
3  MOV      BETA_B, AL
4  MOV      AL, ES:[BX+SI+1000H]
5  MOV      BX, 1000H
6  MOV      DS, BX
7  MOV      [BX+10], AL    合法 !
8  MOV      BYTE PTR [BX], 10H
9  MOV      WORD PTR [BX], 10H
```

BX:	10H	2000:1000H
	12H	2000:1001H

BX:	10H	2000:1000H
	00H	2000:1001H

四. 8086汇编指令 – 1.数据传送指令

|▶ 2) **PUSH** (push word onto stack)进栈

- **指令汇编格式:** PUSH src
- **操作:** 堆栈指示器减 2 ; $SP \leftarrow SP - 2$
将给定的操作数存放到由SP指出的栈顶 ; $(SP+1, SP) \leftarrow (src)$
- **受影响的状态标志位:** 没有
- **说明:** PUSH指令的目标地址一定在当前堆栈中。SS内容为段基址,偏移量由堆栈指针SP指出。操作数一定是16位的寄存器或存储器操作数。

|▶ 3) POP (pop word off stack into destination)出栈

- **指令汇编格式:** POP dest
- **操作:** 将堆栈栈顶中存放的字传送到操作数中； $dest \leftarrow (SP+1, SP)$
堆栈指针加2； $SP \leftarrow SP+2$
- **受影响的状态标志位:** 没有
- **说明:**目标操作数只能是16位的存储器或寄存器操作数(CS除外)。

|▶ 4) XCHG (eXCHanGe) 交换

- **指令汇编格式:** XCHG dest, src
- **操作:** dest的内容与src的内容互换
 $(\text{dest}) \longleftrightarrow (\text{src})$
- **受影响的状态标志位:** 没有
- **说明:** dest 和 src 不能同时为存储器操作数。段寄存器、立即数不能作为操作数。

|▶ 4) XCHG (eXCHanGe) 交换

- 例：

2	XCHG	AL, BL
3	XCHG	CL, [BX]
4	XCHG	BL, [BX+SI+10]
5	XCHG	AX, AX
6	XCHG	AL, SI
7	XCHG	[SI], [BX+10]
8	XCHG	DX, DS
9	XCHG	AL, 10



|▶ 5) XLAT (translate) 转换表

- **指令汇编格式:** XLAT
- **操作:** BX和AL内容之和指出的内存字节单元的内容送到AL中
$$AL \leftarrow (BX+AL)$$
- **受影响的状态标志位:** 没有
- **说明:** XLAT指令用于查表。表的开始地址即表头地址由BX寄存器给出。
AL中的原始值是要寻址的表中元素地址的位移量，规定表中第一个字节的位移量为0。这是一种特殊的基址变址寻址方式，基址寄存器为BX，变址寄存器为AL。

|▶ 5) XLAT (translate) 转换表

- 例：

看如下的一种加密方式（替代加密）：

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
Y	Z	D	M	R	N	H	X	J	L	I	O	Q	U	W	A	C
R	S	T	U	V	W	X	Y	Z								
B	E	G	F	K	P	T	S	V								

明文：THIS IS A COMPUTER

密文：GXJE JE Y DWQAFGRB

四. 8086汇编指令 – 1.数据传送指令

5) XLAT (translate) 转换表

- 例：

2	JMTAB	DB	'YZDMRNHXJLIOQ'
3		DB	'UWACBEGFKPTSV'
4			
5		MOV	AX, SEG JMTAB
6		MOV	DS, AX
7		MOV	BX, OFFSET JMTAB
8		MOV	AL, 'T'
9		SUB	AL, 'A'
10		MOV	AH, 0
11		ADD	BX, AX
12		MOV	AL, [BX]

JMTAB	59H
+1	5AH
+2	44H
+3	4DH
+4	52H
+5	4EH
+6	48H
+7	58H
+8	4AH
+9	4CH
+10	49H
+11	4FH
+12	51H
+13	55H
+14	57H
+15	41H
+16	43H
+17	42H
+18	45H.
+19	47H
+20	46H

四. 8086汇编指令 – 1.数据传送指令

► 5) XLAT (translate) 转换表

- 例：

2	JMTAB	DB	'YZDMRNHXJLIOQ'
3		DB	'UWACBEGFKPTSV'
4		MOV	AX, SEG JMTAB
5		MOV	DS, AX
6		MOV	BX, OFFSET JMTAB
7		MOV	AL, 'T'
8		SUB	AL, 'A'
9		XLAT	

JMTAB	59H
+1	5AH
+2	44H
+3	4DH
+4	52H
+5	4EH
+6	48H
+7	58H
+8	4AH
+9	4CH
+10	49H
+11	4FH
+12	51H
+13	55H
+14	57H
+15	41H
+16	43H
+17	42H
+18	45H.
+19	47H
+20	46H

|▶ 6) LEA (load effective address) 取有效地址

- **指令汇编格式:** LEA dest,src
- **操作:** 将指令中给出的存储器操作数的有效地址(即地址的偏移量)送到指定的寄存器中中

$$\text{dest} \leftarrow \text{EA}$$

- **受影响的状态标志位:** 没有
- **说明:** LEA指令是将源操作数地址的偏移量，即有效地址传送到目标操作数中。**源操作数**必须是一个存储器操作数，**目标操作数**可以是任一16位通用寄存器、指针寄存器或变址寄存器。

四. 8086汇编指令 – 1.数据传送指令

|▶ 6) LEA (load effective address) 取有效地址

- 例：

2	DATA1	DB	10H
3		LEA	BX, DATA1
4		MOV	BX, OFFSET DATA1
5		MOV	BX, 1000H
6		MOV	DI, 2000H
7		LEA	AX, [BX+1243H]
8		LEA	DX, [BX+SI+1234H]

► 7) LDS (load data segment register) 加载数据段寄存器

- **指令汇编格式**：LDS dest,src
- **操作**：
 - (1)将双字长存储器操作数中的低地址字传送到指定的寄存器中
 $dest \leftarrow (EA)$
 - (2)将双字长存储器操作数中的高地址字传送到DS寄存器中
 $DS \leftarrow (EA+2)$
- **受影响的状态标志位**：没有
- **说明**：LDS是将src指出的连续四个字节的内容，即一个32位的指针变量传送到一对16位的目标寄存器中。高位字为段基地址，LDS指令将其传送到数据段寄存器DS中，低位字为偏移量，传送到由dest指出的一个通用寄存器，指针寄存器或变址寄存器中，但不能是段寄存器。

四. 8086汇编指令 – 1.数据传送指令

7) LDS (load data segment register) 加载数据段寄存器

- 例：

```
1 SSEG SEGMENT STACK ;堆栈段
2 STK DB 20 DUP(0)
3 SSEG ENDS
4
5 DSEG SEGMENT ;数据段
6 DATA1 DB 35H,26H,03H ;(032635H)
7 DATA2 DB 3 DUP(0)
8 DSEG ENDS
9
10 CSEG SEGMENT ;代码段
11 ASSUME CS:CSEG, DS:DSEG
12 ASSUME SS:SSEG
13 START: MOV AX, DSEG ;段寄存器初值
14 MOV DS, AX
15 MOV AX, SSEG
16 MOV SS, AX
17 MOV SP, SIZE STK ;设置堆栈指
18 MOV AX, OFFSET DATA1
19 LDS BX, AX
```

DATA1	35H
	26H
	03H
DATA2	00H
	00H
	00H

► 8) LES (load extra segment register) 加载附加段寄存器

- **指令汇编格式**：LES dest,src
- **操作**：
 - (1)将双字长存储器操作数中的低地址字传送到指定的寄存器中
 $dest \leftarrow (EA)$
 - (2)将双字长存储器操作数中的高地址字传送到DS寄存器中
 $ES \leftarrow (EA+2)$
- **受影响的状态标志位**：没有
- **说明**：LES是将src指出的连续两个字的内容，即一个32位的指针变量传送到一对16位的目标寄存器中。高位字为段基址，LES指令将其传送到数据段寄存器ES中，低位字为偏移量，传送到由dest指出的一个通用寄存器，指针寄存器或变址寄存器中，但不能是段寄存器。

|▶ 9) LAHF (load AH from flags) 取标志

- **指令汇编格式：** LAHF
- **操作：** 标志寄存器低8位的状态标志填写在AH寄存器相应位中
 $AH \leftarrow SF:ZF:x:AF:x:PF:x:CF$
- **受影响的状态标志位：**没有
- **说明：**此指令在80x86中几乎无用，主要是为了保证与8080/8085向下兼容，才保留了该指令。

|▶ 10) SAHF (store AH into flags) 存标志

- **指令汇编格式：** SAHF
- **操作：** 将AH寄存器中的相应位传送到状态标志寄存器相应位中
 $SF:ZF:x:AF:x:PF:x:CF \leftarrow AH$
- **受影响的状态标志位：** SF， ZF， AF， PF， CF
- **说明：**此指令在80x86中几乎无用，主要是为了保证与8080/8085向下兼容，才保留了该指令。

|▶ 11) **PUSHF** (push flags onto stack) 标志进栈

- **指令汇编格式**： PUSHF
- **操作**：将堆栈指针减2，然后将标志寄存器F中的值存储到栈顶字的对应位中

$$SP \leftarrow SP - 2$$

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

- **受影响的状态标志位**：没有

|▶ 12) **POPF** (push flags onto stack) 标志出栈

- **指令汇编格式**： POPF
- **操作**：将位于堆栈栈顶字中的对应位写入标志寄存器F中，然后将堆栈指针加2

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

$$SP \leftarrow SP+2$$

- **受影响的状态标志位**：所有标志位

四. 8086汇编指令

|▶ 2. 算数运算指令 (14条)

- 1) ADD (addition)加法
- 2) ADC (addition with carry)带进位加
- 3) INC (increment by 1)增1
- 4) SUB (subtract)减法
- 5) SBB (subtract with borrow)带借位加
- 6) DEC (decrement by 1)减1
- 7) NEG (negate)取补
- 8) CMP (compare)比较
- 9) MUL (multiply,unsigned) 无符号乘法
- 10) IMUL (integer multiply,signed) 带符号乘法
- 11) DIV (division,unsigned) 无符号除法
- 12) IDIV (division,signed) 带符号除法
- 13) CBW (convert byte to word)将字节转换为字
- 14) CWD (convert word to double word)将字转换为双字

|▶ 1) ADD (addition) 加法

- **指令汇编格式** : ADD dest,src
- **操作** : 两个操作数求和，结果存目标操作数中。
$$\text{dest} \leftarrow (\text{src})$$
- **受影响的状态标志位** : OF, SF, ZF, AF, PF, CF
- **说明** : dest和src不能同时为存储器操作数和段寄存器。

|▶ 2) ADC (addition with carry) 带进位加法

- **指令汇编格式** : ADC dest,src
- **操作** : 两个操作数相加的同时,再加上CF。结果存入目标操作数中。
$$\text{dest} \leftarrow (\text{dest}) + (\text{src}) + \text{CF}$$
- **受影响的状态标志位** : OF, SF, ZF, AF, PF, CF
- **说明** : ADC指令主要用于多精度数据相加。

四. 8086汇编指令 – 2. 算数运算指令

|▶ 2) ADC (addition with carry) 带进位加法

- 例：3字节数据相加：

$$123456H + 789ABCH = 8ACF12H$$

00010010 00110100 01010110	
+ 01111000 10011010 10111100	
<hr/>	
10001010 11001111 00010010	

ADC

ADC

ADD

MOV AL, 56H
MOV AH, 34H
MOV BL, 12H
ADD AL, 0BCH
ADC AH, 9AH
ADC BL, 78H

四. 8086汇编指令 – 2. 算数运算指令

|> 2) ADC (addition with carry) 带进位加法

- 例：3字节数据相减：

$$123456H + 789ABCH = 8ACF12H$$

00010010 00110100 01010110	
+ 01111000 10011010 10111100	
<hr/>	
10001010 11001111	1
ADC	ADD

MOV AL, 56H	
MOV AX, 3456H	
MOV AH, 34H	
MOV BL, 12H	
ADD AL, 0BCH	
ADD AX, 9ABCH	
ADC AH, 9AH	
ADC BL, 78H	

|▶ 3) INC (increment by one) 增加1

- **指令汇编格式**： INC dest
- **操作**：将指定的操作数加1，并将结果回送到目标操作数中。

$$\text{dest} \leftarrow (\text{dest}) + 1$$

- **受影响的状态标志位**： OF，SF，ZF，AF，PF
- **说明**：无论操作数状态如何，INC指令执行后不影响CF。
- **例**：

2	INC	AL
3	INC	DATA1
4	INC	BX
5	INC	[BX]
6	INC	WORD PTR [BX]

四. 8086汇编指令

|▶ 2. 算数运算指令 (14条)

- 1) ADD (addition)加法
- 2) ADC (addition with carry)带进位加
- 3) INC (increment by 1)增1
- 4) SUB (subtract)减法
- 5) SBB (subtract with borrow)带借位加
- 6) DEC (decrement by 1)减1
- 7) NEG (negate)取补
- 8) CMP (compare)比较
- 9) MUL (multiply,unsigned) 无符号乘法
- 10) IMUL (integer multiply,signed) 带符号乘法
- 11) DIV (division,unsigned) 无符号除法
- 12) IDIV (division,signed) 带符号除法
- 13) CBW (convert byte to word)将字节转换为字
- 14) CWD (convert word to double word)将字转换为双字

|▶ 4) **SUB** (subtract) 减法

- **指令汇编格式**： SUB dest,src
- **操作**：从目标操作数减去源操作数,结果存入目标操作数中。
$$\text{dest} \leftarrow (\text{dest}) - (\text{src})$$
- **受影响的状态标志位**： OF , SF , ZF , AF , PF , CF

|▶ 5) **SBB** (subtract with borrow) 带借位减法

- **指令汇编格式**： SBB dest,src
- **操作**：从目标操作数中减去源操作数和CF，结果存入目标操作数中。
$$\text{dest} \leftarrow (\text{dest}) - (\text{src}) - \text{CF}$$
- **受影响的状态标志位**： OF，SF，ZF，AF，PF，CF
- **说明**：SBB 指令主要用于多精度数据减法。
- **例**：

|▶ 6) DEC (decrement by one) 自减1

- **指令汇编格式**： DEC dest
- **操作**：将指定的操作数减1，并将结果送回到目标操作数中。
$$\text{dest} \leftarrow (\text{dest}) - 1$$
- **受影响的状态标志位**： OF，SF，ZF，AF，PF
- **说明**： DEC指令不影响CF。
- **例**：

2 DEC BX

3 DEC BYTE PTR [BX+SI+1000H]

4 DEC SP

|▶ 7) NEG (negate or form 2's complement) 取补(取负)

- **指令汇编格式**： NEG dest
- **操作**：从0中减去指令中给定的操作数,最后将结果送回到给定的操作数中。

$$\text{dest} \leftarrow 0 - (\text{dest})$$

- **受影响的状态标志位**： OF，SF，ZF，AF，PF，CF
- **说明**： NEG 指令对于带符号的数来说是取负，即改变操作数的符号。如果操作前操作数非零，NEG操作后，CF置1，否则置0。

|▶ 8) CMP (compare) 比较

- **指令汇编格式**： CMP dest,src
- **操作**：两个操作数求和,结果存目标操作数中。
$$\text{dest} \leftarrow (\text{dest}) + (\text{src})$$
- **受影响的状态标志位**： OF，SF，ZF，AF，PF，CF
- **说明**： CMP 将两个操作数相减，两个操作数保持原值不变，但与 SUB 指令一样影响标志。

|▶ 8) CMP (compare) 比较

- 无符号数比较与标志位取值 : CMP AX, BX

比较关系	(AX) ? (BX)	(AX) – (BX) 特点	标志寄存器
等于	(AX) = (BX)	=0	ZF= 1
不等于	(AX) ≠ (BX)	≠0	ZF= 0
小于	(AX) < (BX)	将产生借位	CF= 1, ZF=0
大于等于	(AX) ≥ (BX)	不借位	CF= 0
大于	(AX) > (BX)	不借位, ≠0	CF= 0, ZF= 0
小于等于	(AX) ≤ (BX)	或者借位, 或者为0	CF= 1 或 ZF=1

|▶ 8) CMP (compare) 比较

- 无符号数比较与标志位取值 : CMP AX, BX
- 如果执行后 :

ZF=1, 说明 $(ax) = (bx)$;

ZF=0, 说明 $(ax) \neq (bx)$;

CF=1, 说明 $(ax) < (bx)$;

CF=0, 说明 $(ax) \geq (bx)$;

CF=0 并且 ZF=0, 说明 $(ax) > (bx)$;

CF=1 或 ZF=1, 说明 $(ax) \leq (bx)$.

|▶ 8) CMP (compare) 比较

- **有符号数比较与标志位取值** : CMP AX, BX
 - 如果 $(AX)=(BX)$ 则 $(AX)-(BX)=0$, 所以 : ZF=1 ;
 - 如果 $(AX)\neq(BX)$ 则 $(AX)-(BX)\neq0$, 所以 : ZF=0 ;
 - 所以 , 我们根据cmp指令执行后ZF的值 , 就可以知道两个数据是否相等 。

|▶ 8) CMP (compare) 比较

- **有符号数比较与标志位取值** : CMP AX, BX
 - 如果 $(AX) < (BX)$, 则 $SF = 1$, 即结果为负 ;
 - 例1 : $(ah)=1$, $(bh)=2$: 则 $(ah)-(bh)=0FFH$, $0FFH$ 为 -1 的补码 , 因为结果为负 , 所以 $SF=1$ 。
 - cmp 操作对象1,操作对象2 指令执行后 , **SF=1** , 就说明操作对象1<操作对象2 ? **当然不是 !**
 - 例2 : $(ah)=22H$, $(bh)=0A0H$: 则 $(ah)-(bh)=34-(-96)=82H$, $82H$ 是 -126 的补码 , 所以 $SF=1$ 。
 - 所得到的相应结果的正负 , 并不能说明 , 运算所应该得到的结果的正负 。这是因为 在运算的过程中可能发生溢出 。

|▶ 8) CMP (compare) 比较

- **有符号数比较与标志位取值** : CMP AH, BH

➤我们应该在考察SF（得知实际结果的正负）的同时考察OF（得知有没有溢出），就可以得知逻辑上真正结果的正负，同时就可以知道比较的结果。

- 1) 如果SF=1，而OF=0。OF=0，说明没有溢出，逻辑上真正结果的正负=实际结果的正负；因SF=1，实际结果为负，所以逻辑上真正的结果为负，所以 $(ah) < (bh)$ 。
- 2) 如果SF=1，而OF=1。OF=1，说明有溢出，逻辑上真正结果的正负≠实际结果的正负；因 SF=1，实际结果为负，实际结果为负，而又有溢出，这说明是由于溢出导致了实际结果为负，简单分析一下，就可以看出，如果因为溢出导致了实际结果为负，那么逻辑上真正的结果必然为正。这样，SF=1，OF = 1，说明了 $(ah) > (bh)$ 。

|▶ 8) CMP (compare) 比较

- **有符号数比较与标志位取值** : CMP AH, BH

➤我们应该在考察SF（得知实际结果的正负）的同时考察OF（得知有没有溢出），就可以得知逻辑上真正结果的正负，同时就可以知道比较的结果。

- 3) 如果SF=0，而OF=1。OF=1，说明有溢出，逻辑上真正结果的正负≠实际结果的正负；因SF=0，实际结果非负，而OF=1说明有溢出，则结果非0，所以，实际结果为正。实际结果为正，而又有溢出，这说明是由于溢出导致了实际结果非负，简单分析一下，就可以看出，如果因为溢出导致了实际结果为正，那么逻辑上真正的结果必然为负。这样，SF=0，OF = 1，说明了 $(ah) < (bh)$ 。
- 4) 如果SF=0，而OF=0。OF=0，说明没有溢出，逻辑上真正结果的正负=实际结果的正负；因SF=0，实际结果非负，所以逻辑上真正的结果必然非负。所以 $(ah) \geq (bh)$ 。

|▶ 2. 算数运算指令 (14条)

- 1) ADD (addition)加法
- 2) ADC (addition with carry)带进位加
- 3) INC (increment by 1)增1
- 4) SUB (subtract)减法
- 5) SBB (subtract with borrow)带借位加
- 6) DEC (decrement by 1)减1
- 7) NEG (negate)取补
- 8) CMP (compare)比较
- 9) **MUL (multiply,unsigned)** 无符号乘法
- 10) **IMUL (integer multiply,signed)** 带符号乘法
- 11) DIV (division,unsigned) 无符号除法
- 12) IDIV (division,signed) 带符号除法
- 13) CBW (convert byte to word)将字节转换为字
- 14) CWD (convert word to double word)将字转换为双字

|▶ 9) **MUL** (multiply, unsigned) 无符号乘法

- **指令汇编格式**： MUL src
- **操作**：源操作数与累加器的内容相乘。如果源操作数是字节数据,就与AL中的数据相乘,乘积为字,存放在AX中。如果源操作数是字数据,就与AX中的数相乘,乘积为双字,存放在DX和AX中。

两个字节数相乘: $AX \leftarrow AL * (src)$

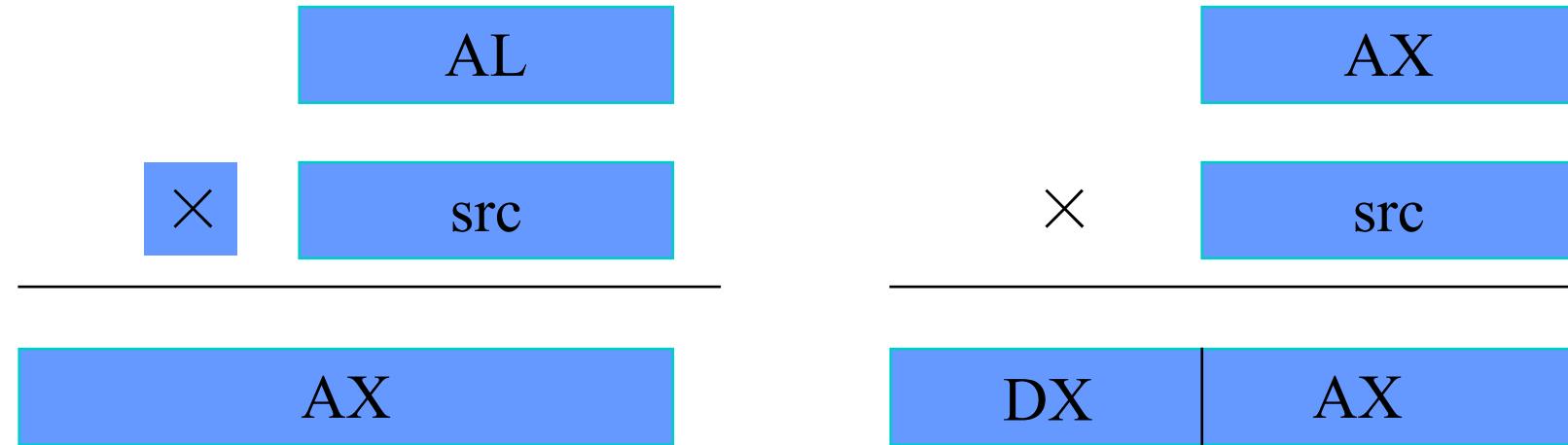
两个字数据相乘: $DX, AX \leftarrow AX * (src)$

- **受影响的状态标志位**： OF , CF
- **说明**：源操作数src不能是立即数（要指定数据类型，以防混淆）。如果乘积的高半部不为零时，CF和OF被置位，否则将被清除。

四. 8086汇编指令 – 2. 算数运算指令

|▶ 9) MUL (multiply, unsigned) 无符号乘法

- 例：



2 MUL BL

3 MUL DX

4 MUL BYTE PTR [SI+BX+1000H]

|▶ 10) IMUL (integer multiply, signed) 无符号乘法

- **指令汇编格式**： IMUL src
- **操作**：源操作数与累加器的内容相乘。如果源操作数是字节数据,就与AL中的数据相乘,乘积为字,存放在AX中。如果源操作数是字数据,就与AX中的数相乘,乘积为双字,存放在DX和AX中。
 - 两个字节数相乘: $AX \leftarrow AL * (src)$
 - 两个字数据相乘: $DX, AX \leftarrow AX * (src)$
- **受影响的状态标志位**： OF , CF
- **说明**：源操作数src不能是立即数 (要指定数据类型,以防混淆) 。 IMUL 指令视操作数为带符号的数。如果乘积的高半部不是符号位的扩展时, CF和OF被置位,否则将被清除。

|▶ 2. 算数运算指令 (14条)

- 1) ADD (addition)加法
- 2) ADC (addition with carry)带进位加
- 3) INC (increment by 1)增1
- 4) SUB (subtract)减法
- 5) SBB (subtract with borrow)带借位加
- 6) DEC (decrement by 1)减1
- 7) NEG (negate)取补
- 8) CMP (compare)比较
- 9) MUL (multiply,unsigned) 无符号乘法
- 10) IMUL (integer multiply,signed) 带符号乘法
- 11) DIV (division,unsigned) 无符号除法
- 12) IDIV (division,signed) 带符号除法
- 13) CBW (convert byte to word)将字节转换为字
- 14) CWD (convert word to double word)将字转换为双字

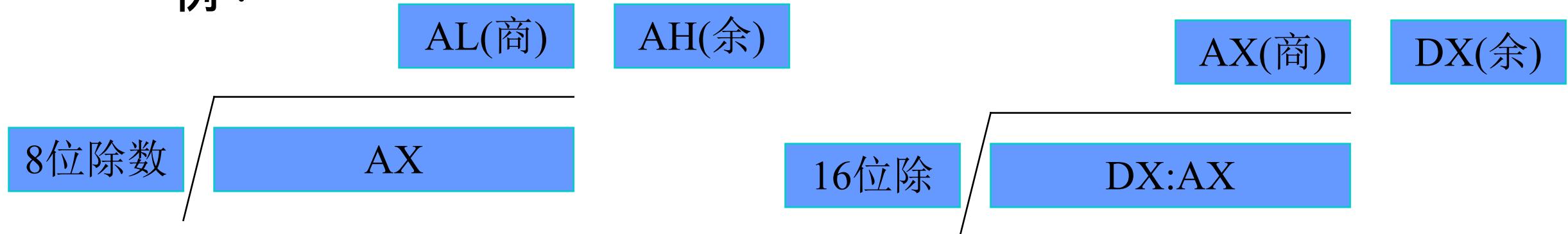
► 11) DIV (division,unsigned) 无符号除法

- **指令汇编格式**： DIV src
- **操作**： AX(或DX,AX)的内容除以src的内容。商存放在AL(字节时)或AX(字时)，并将余数存放在AH(字节时)或DX(字时) 中。
 - 字除以字节: $AX / (src); AL \leftarrow \text{商}, AH \leftarrow \text{余数}$
 - 双字除以字: $DX, AX / (src); AX \leftarrow \text{商}, DX \leftarrow \text{余数}$
- **受影响的状态标志位**：不产生有效的状态标志
- **说明**： src不能是立即数；如果商数超过了允许的最大值（字节时为OFFH，字时为0FFFFH）时就产生一个方式0的中断，并且商和余数都不确定。

四. 8086汇编指令 – 2. 算数运算指令

|> 11) DIV (division,unsigned) 无符号除法

- 例：



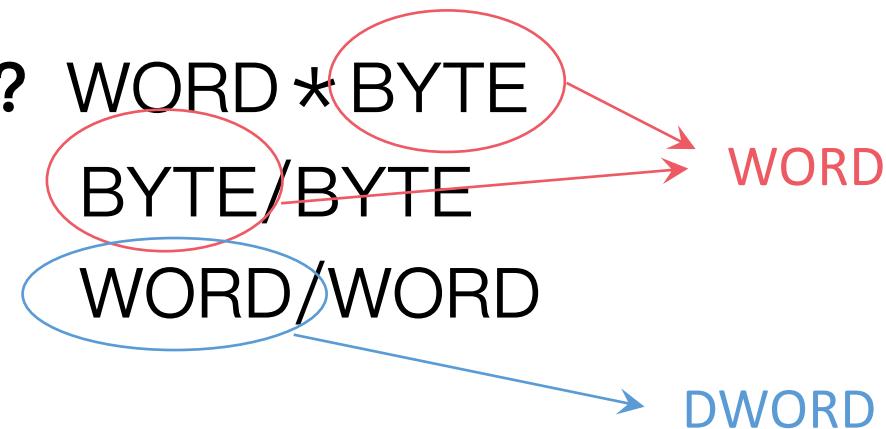
2	DIV	BL
3	DIV	WORD PTR [BX+DI+1000H]
4	MOV	AX, 1000H
5	MOV	CL, 08H
6	DIV	CL

► 12) IDIV (integer division,signed) 有符号除法

- **指令汇编格式**： IDIV src
- **操作**： AX(或DX,AX)的内容除以src的内容。商存放在AL(字节时)或AX(字时)，并将余数存放在AH(字节时)或DX(字时) 中。
 - 字除以字节: $AX / (src); AL \leftarrow \text{商}, AH \leftarrow \text{余数}$
 - 双字除以字: $DX, AX / (src); AX \leftarrow \text{商}, DX \leftarrow \text{余数}$
- **受影响的状态标志位**：不产生有效的状态标志
- **说明**： src不能是立即数；如果商数超过了允许的最大值(字节时为-128~127，字时为-32768~32767)时就产生一个方式0的中断，并且商和余数都不确定。

► 补充：乘除法指令的应用

- 乘法指令能实现： $\text{BYTE} * \text{BYTE} = \text{WORD}$
 $\text{WORD} * \text{WORD} = \text{DWORD}$
- 除法指令能实现： $\text{WORD} / \text{BYTE} = \text{BYTE}$
 $\text{DWORD} / \text{WORD} = \text{WORD}$
- 如何实现下述操作？
 $\text{WORD} * \text{BYTE}$
 $\text{BYTE} / \text{BYTE}$
 $\text{WORD} / \text{WORD}$



► 补充：字节/字转换为字/双字

- **无符号：**

- 字节转换为字(AL→AX): MOV AH, 0
- 字转换为双字(AX→DX:AX): MOV DX, 0

- **带符号：**

- 字节转换为字(AL→AX)

正数: MOV AH, 0

负数: MOV AH, 0FFH

- 字转换为双字(AX→DX:AX)

正数 : MOV DX, 0

负数 : MOV DX, 0FFFFH

|▶ 13) CBW (convert byte to word) 字节转换为字

- **指令汇编格式** : CBW
- **操作** : 将AL中第7位的值扩展到整个AH中。
 - 如果AL为正,那么 $AH \leftarrow 00H$
 - 否则 $AH \leftarrow OFFH$
- **受影响的状态标志位** : 没有
- **说明** : CBW是将AL寄存器中数的符号位扩展到整个AH寄存器中。

► 14) CBD (convert byte to double word) 字节转换为双字

- 指令汇编格式： CBD

- 操作：将AX中的最高位扩展到整个DX中。

如果 AX 为正，那么 $DX \leftarrow 0000H$

否则 $DX \leftarrow FFFFH$

- 受影响的状态标志位：没有

- 说明： CWD将寄存器AX的符号位扩展到整个寄存器DX中。

四. 8086汇编指令

|▶ 3. 逻辑操作指令 (5条)

- 1) NOT (not,or form 1's complement) 取反
- 2) AND (and,logical conjunction) 逻辑与
- 3) OR (or,inclusive) 逻辑或
- 4) XOR (exclusive or) 异或
- 5) TEST (test,or logical compare) 测试

四. 8086汇编指令 – 3.逻辑操作指令

|▶ 1) NOT (not, or form 1's complement) 取反

- **指令汇编格式**： NOT dest
- **操作**：将操作数的每一位求反,然后将结果回送到对应位中。
$$\text{dest} \leftarrow \overline{\text{dest}}$$
- **受影响的状态标志位**：没有

|▶ 2) AND (and, logical conjunction) 逻辑与

- **指令汇编格式**： AND dest,src
- **操作**：两个操作数进行逻辑“与”，即如果两个操作数的对应位都为1时，结果的对应位才为1，否则结果的对应位为0。
$$\text{dest} \leftarrow (\text{dest}) \wedge (\text{src}), \quad \text{CF} \leftarrow 0, \quad \text{OF} \leftarrow 0$$
- **受影响的状态标志位**： OF, SF, ZF, AF, PF, CF
- **说明**：AND指令可借助某个给定的操作数将另一个操作数中的某些位清除(这种操作也称设置屏蔽)，使某些位保持不变，这个数通常称为掩码。例如，将一个源操作数与00001111B相“与”，可将这个源操作数的高四位变为0，而低四位保持不变。

|▶ 3) OR (or, inclusive)逻辑或

- **指令汇编格式**： OR dest,src
- **操作**：两个操作数进行逻辑“或”操作,即当两个操作数的对应位都为0时结果的对应位为0,否则结果的对应位为1,结果存入目标操作数中,CF和OF位置0。

$$\text{dest} \leftarrow (\text{dest}) \vee (\text{src}), \quad \text{CF} \leftarrow 0, \quad \text{OF} \leftarrow 0$$

- **受影响的状态标志位**： OF，SF，ZF，AF，PF，CF
- **说明**： OR指令可用来使操作数中的某些位为1，某些位保持不变。例如，用10000000B与一个字节操作数相“或”，可使这个字节操作数的最高有效位为1。

|▶ 4) XOR (exclusive or) 异或

- **指令汇编格式**： XOR dest,src
- **操作**：两个操作数执行逻辑“异或”操作,即当两个操作数的对应位相同时,结果的对应位为0; 否则结果的对应位为1。结果存入目标操作数中,CF和OF位置0 。

$$\text{dest} \leftarrow (\text{dest}) \otimes (\text{src}), \quad \text{CF} \leftarrow 0, \quad \text{OF} \leftarrow 0$$

- **受影响的状态标志位**： OF , SF , ZF , AF , PF , CF
- **说明**： XOR指令可用来使操作数中的某些位取反，某些位不变。

|▶ 5) TEST (test,or logical compare) 测试

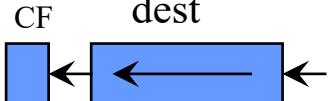
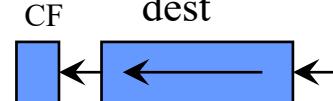
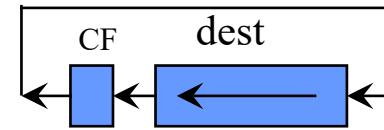
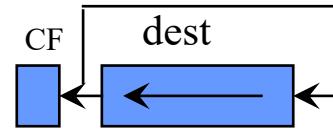
- **指令汇编格式**： TEST dest,src
- **操作**：将两个操作数进行逻辑“与”,根据结果设置状态标志位,但不改变两个操作数的原始值,并将CF和OF清除。
 $(dest) \wedge (src)$, CF $\leftarrow 0$, OF $\leftarrow 0$
- **受影响的状态标志位**： OF , SF , ZF , AF , PF , CF
- **说明**：TEST指令通过“与”的方式对两个源操作数进行比较,但不留“与”的结果。

|▶ 4. 移位操作指令 (7条)

- 1) SHL (SHift Left) 逻辑左移
- 2) SHR (SHift Right) 逻辑右移
- 3) SAL (Arithmetic Shift Left) 算术左移
- 4) SAR (Arithmetic Shift Right) 算术右移
- 5) ROL (ROtate Left) 不带进位循环左移
- 6) ROR (ROtate Right) 不带进位循环右移
- 7) RCL (Rotate Left with Carry) 带进位循环左移
- 8) RCR (Rotate Right with Carry) 带进位循环左移

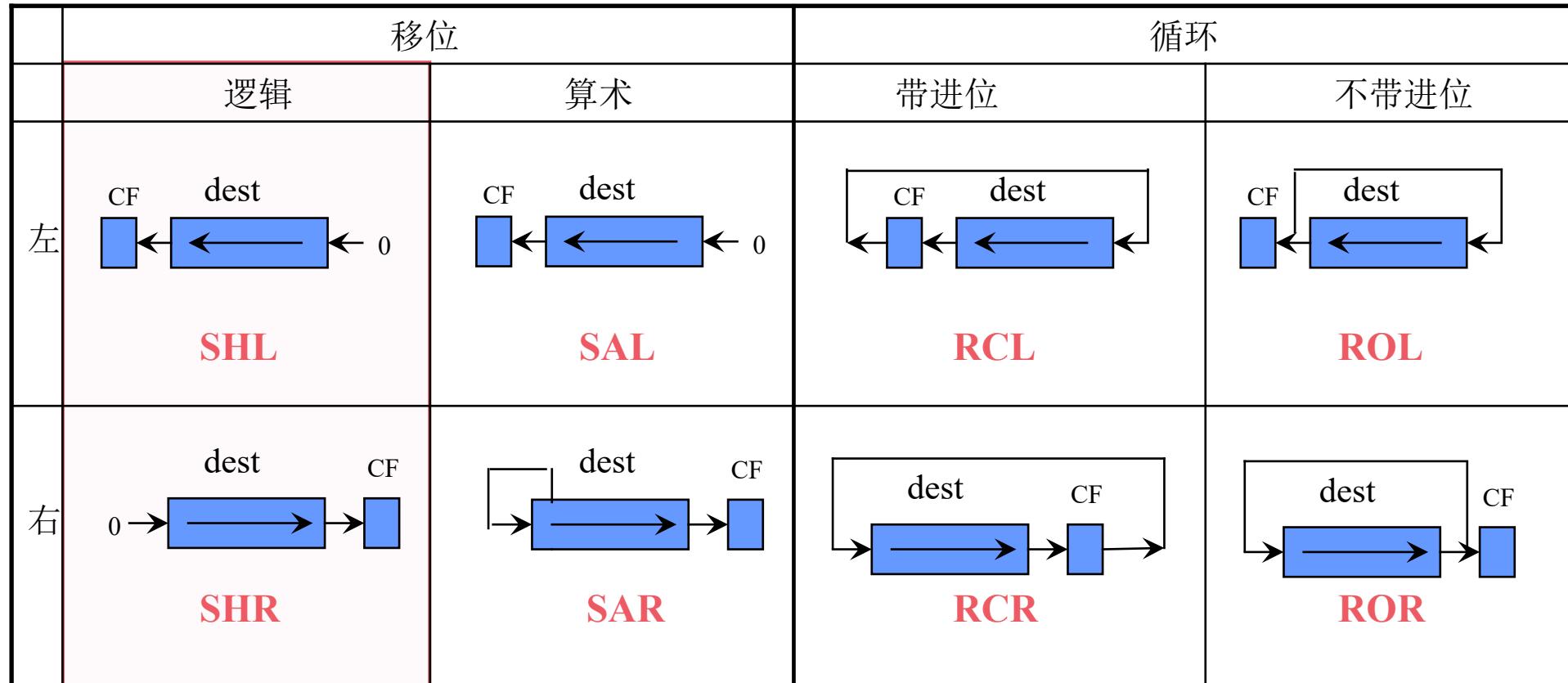
四. 8086汇编指令

4. 移位操作指令 (7条)

移位		循环		
	逻辑	算术	带进位	
左	 SHL	 SAL	 RCL	 ROL
	 SHR	 SAR	 RCR	 ROR

四. 8086汇编指令

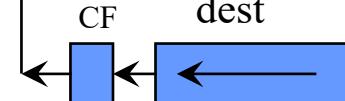
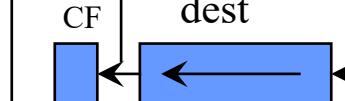
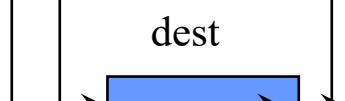
4. 移位操作指令 (7条)



- 逻辑左移：最高位移向CF，最低位补0。
- 逻辑右移：最高位补0

四. 8086汇编指令

4. 移位操作指令 (7条)

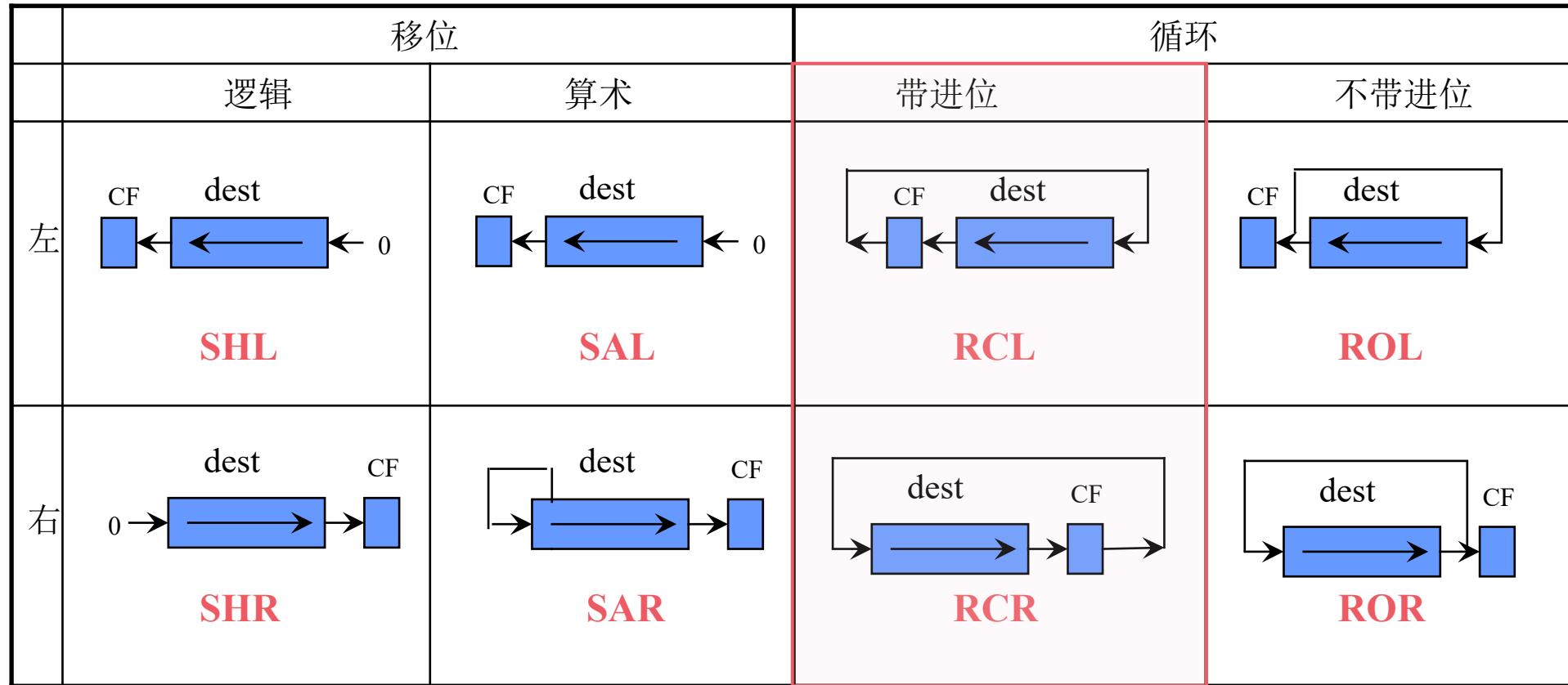
移位		循环	
逻辑	算术	带进位	不带进位
左  SHL	 SAL	 RCL	 ROL
右  SHR	 SAR	 RCR	 ROR

将 10000101 算术右移三次
每次一位。
第一次：11000010 CF=1
第二次：11100001 CF=0
第三次：11110000 CF=1

- 算数左移：最高位移向CF，最低位补0。
- 算数右移：最低位移向CF，最高位补没移前最高位的值。

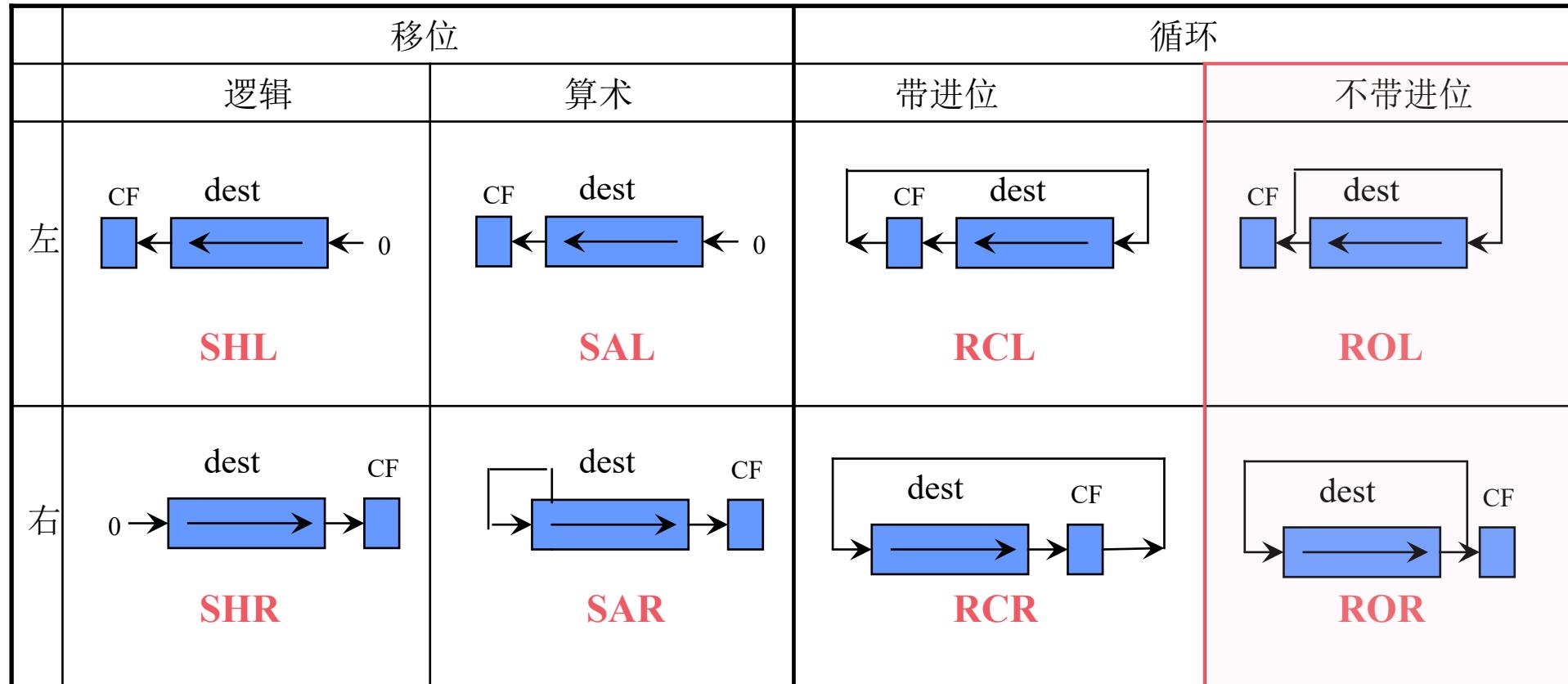
四. 8086汇编指令

4. 移位操作指令 (7条)



四. 8086汇编指令

4. 移位操作指令 (7条)



- 不带进位循环左移：最高位移向CF，同时最高位补向最低位。
- 不带进位循环右移：最低位移向CF，同时最低位移向最高位。

▶ 4. 移位操作指令 (7条)

- **指令汇编格式**： *instruction dest, cnt*
- **操作**：将dest按照instruction指定的方式移动cnt位。
- **受影响的状态标志位**：
 - SHL/SAL，SHR，SAR影响CF，OF，PF，SF，ZF；
 - ROL，ROR，RCL，RCR仅影响CF和OF位
- **说明**：
 - dest可以是任一通用寄存器（字节或字），也可以是各种寻址方式给出的内存单元，cnt为移位次数。移位操作是将给定的目标操作数移动cnt次。cnt为1时，使用常数1或CL；cnt大于1，则必须使用CL。
 - CF的值总是最后一次被移入的值。在移动1位的移位中,如果源操作数的最高位(符号位)改变了，那么OF就被置1，否则，OF就被置0。移位次数大于1，OF不确定。

四. 8086汇编指令

▶ 5. 状态标志位操作指令 (7条)

- 说明：8086/8088有7条状态标志位控制指令，它们可以直接和独立地对8086 CPU中的某一状态标志位进行控制，用来设置或改变状态标志位的状态。

指令名称	说明	助记
CLC	清进位标志，CF = 0	Clear
STC	置进位标志，CF = 1	Setup
CLD	清方向标志，DF = 0	
STD	置方向标志，DF = 1	
CLI	清中断标志，IF = 0	
STI	置中断标志，IF = 1	
CMC	进位取反	Complement Carry Flag

五. 简单I/O功能调试

- |▶ 1. 键盘输入
- 2. 显示器输出
- 3. 程序结束退出
- 4. 设置断点

|▶ 1. 键盘输入

- **指令调用方法 :** MOV AH,01H
 INT 21H
- **操作 :** 从键盘上接收一个字符，并在屏幕上显示出来。
- **入口条件 :** 无
- **出口条件 :** 键入字符的ASCII码在AL中

五. 简单I/O功能调试

▶ 2. 显示器输出

- 指令调用方法 : $MOV\ AH,\ 02H$
 $INT\ 21H$
- 操作 : 在屏幕上光标处显示一个字符。注意,使用这个功能调用后,AL的内容将被改变。
- 入口条件 : 要显示的字符的ASCII码在DL中
- 出口条件 : 字符显示在屏幕上
- 例子 : 在屏幕上显示A,用下述程序段实现

```
2  MOV      DL, "A"  
3  MOV      AH, 02H  
4  INT      21H
```

|▶ 3. 程序结束退出

- **指令调用方法 :** MOV AH, 4CH
 MOV AL, 0
 INT 21H
- **操作 :** 结束程序运行 , 返回操作系统。欲使程序结束运行时, 使用本功能
- **入口条件 :** AL=返回码
- **出口条件 :** 无

► 4. 设置断点

- **指令调用方法：** INT 3
- **操作：**停止程序运行，返回DEBUG。欲使程序结束运行而想检查运行结果时，则不应退出DEBUG，这时使用本功能。在以后上机实验时，多数是使用本功能来结束程序运行
- **入口条件：**无
- **出口条件：**无

六. 顺序结构程序举例

例1：

- **题目**：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

- 一个数据取补，就是将这个数据包括符号位在内取反加1，或者用0减去这个数据。带符号数据在机内用补码形式表示时，若原数据为负数，经取补操作后变为其绝对值，若原数据为正数，经取补操作后变为绝对值与其相等的负数。
- 取补与补码是两个不同的概念，一个数据的补码，是用补码形式表示这个数据，当数据为正时，它的补码就是数据本身；当数据为负时，把其原码表示形式除符号位外取反加1或者将其绝对值进行取补所得结果为其补码。
- 例子：已知 $N1 = 00000101$ ，取补后 $N1 = 11111011$
 $N2 = 11111011$ ，取补后 $N2 = 00000101$

六. 顺序结构程序举例

例1：

- **题目**：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

- **分析**：

➤ 思路1：根据取补的定义。

- 包括符号位在内取反加1，或 (用到的指令：NOT，ADC，ADD)
- 用0减去这个数据 (用到的指令：SUB，SBB)

寄存器： BL AH AL

数据： ?? ?? ??H

六. 顺序结构程序举例

例1：

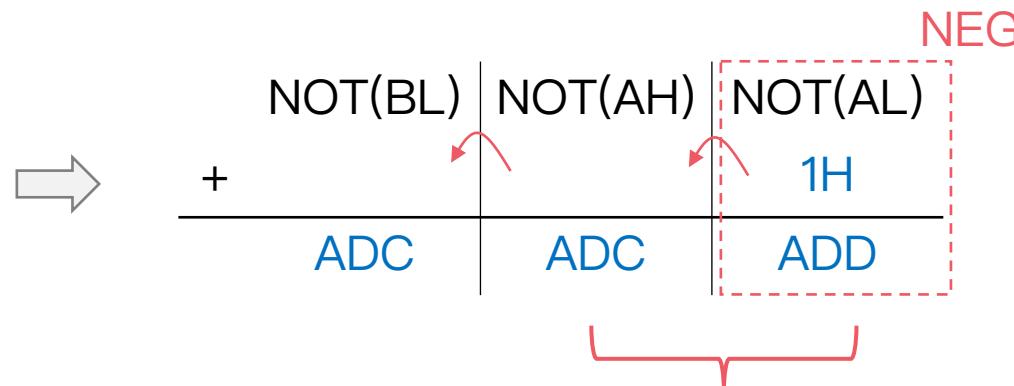
- 题目：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

- 分析：

- 思路1：根据取补的定义。

- 包括符号位在内取反加1，或（用到的指令：NOT，ADC，ADD）
 - 用0减去这个数据（用到的指令：SUB，SBB）

寄存器：	BL	AH	AL
数据：	NOT	(?? ?? ??H)	
	+		1H
	对 (?? ?? ??H) 取补		



什么时候会产生进位？

- $0FFH+1$, $NOT(AL)=0FFH$
- $AL = 0H$
- 当 AL 值为0时产生进位，非0时不进位

六. 顺序结构程序举例

例1：

- **题目**：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。
- **分析**：
 - 思路1：根据取补的定义。
 - 包括符号位在内取反加1，或（用到的指令：NOT，ADC，ADD）
 - 用0减去这个数据（用到的指令：SUB，SBB）
 - 思路2：利用NEG指令。
 - **局限**：NEG最多可对1个字数据取补
 - 三个字节长数据如何处理？

► 7) NEG (negate or form 2's complement) 取补(取负)

- **指令汇编格式**： NEG dest
- **操作**：从0中减去指令中给定的操作数,最后将结果送回到给定的操作数中。
$$\text{dest} \leftarrow 0 - (\text{dest})$$
- **受影响的状态标志位**： OF，SF，ZF，AF，PF，CF
- **说明**： NEG 指令对于带符号的数来说是取负，即改变操作数的符号。如果操作前操作数非零，NEG操作后，CF置1，否则置0。

六. 顺序结构程序举例

例1：

- 题目：计算 $y = -x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

• 算法 :

- 1) 最低字节取补;
 - 2) 进位取反;
 - 3) 中间字节取反，取反的结果加进位;
 - 4) 高字节取反，取反结果加进位。

```

    graph LR
        A[NOT(BL)] --- B[ADC]
        B --- C[ADD]
        C --- D["NOT(AH)"]
        D --- E["NOT(AL)"]
        E --- F["1H"]
    
```

	低字节 不为0	低字节为0, 中字节不为0	中低字节为0, 高字节不为0
(1) 最低字节 取补;	NEG , CF=1	NEG , CF=0	NEG , CF=0
(2) 进位取反;	CF=0	CF=1	CF=1
(3) 中间字节 取反,取反的 结果加进位;	NOT+0 CF=0	NOT+1=NEG CF=0	NOT+1=NEG CF=1
(4) 高字节取 反,取反结果 加进位。	NOT+0	NOT+0	NOT+1=NEG

六. 顺序结构程序举例

例1：

- 题目：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

```
1 ;*****EXAM 5.1.1*****
2 SSEG SEGMENT STACK ;堆栈段
3 STK DB 20 DUP(0)
4 SSEG ENDS
5
6 DSEG SEGMENT ;数据段
7 DATA1 DB 35H, 26H, 03H ;(032635H)
8 DATA2 DB 3 DUP(0)
9 DSEG ENDS
10
11 CSEG SEGMENT ;代码段
12 ASSUME CS:CSEG, DS:DSEG
13 ASSUME SS:SSEG
14 MBNEG: MOV AX, DSEG ;段寄存器初值
15 MOV DS, AX
16 MOV AX, SSEG
17 MOV SS, AX
18 MOV SP, SIZE STK ;设置堆栈指
19 MOV AL, DATA1 ;读入数据低字节
20 MOV AH, DATA1+1 ;读入数据中字节
21 MOV BL, DATA1+2 ;读入数据高字节
22 NEG AL ;取补低字节
23 CMC ;进位取反
24 NOT AH ;中字节取反
25 ADC AH, 0 ;加进位
26 NOT BL ;高字节取反
27 ADC BL, 0 ;加进位
28 MOV DATA2, AL ;保存结果
29 MOV DATA2+1, AH
30 MOV DATA2+2, BL
31 MOV AX, 4C00H
32 INT 21H
33 CSEG ENDS
34 END MBNEG
```

0FH	
10H	
11H	
12H	
13H	
14H	

←

DATA1	35H
	26H
	03H
DATA2	00H
	00H
	00H

六. 顺序结构程序举例

例1：

- 题目：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

```
1 ;*****EXAM 5.1.1*****
2 SSEG SEGMENT STACK ;堆栈段
3 STK DB 20 DUP(0)
4 SSEG ENDS
5
6 DSEG SEGMENT ;数据段
7 DATA1 DB 35H, 26H, 03H ;(032635H)
8 DATA2 DB 3 DUP(0)
9 DSEG ENDS
10
11 CSEG SEGMENT ;代码段
12 ASSUME CS:CSEG, DS:DSEG
13 ASSUME SS:SSEG
14 MBNEG: MOV AX, DSEG ;段寄存器初值
15 MOV DS, AX
16 MOV AX, SSEG
17 MOV SS, AX
18 MOV SP, SIZE STK ;设置堆栈指
19 MOV AL, DATA1 ;读入数据低字节
20 MOV AH, DATA1+1 ;读入数据中字节
21
22
23
24
25
26
27
28
29
30
31
32
33
34
```

MOV	BL, DATA1+2	;读入数据高字节
NEG	AL	;取补低字节
CMC		;进位取反
NOT	AH	;中字节取反
ADC	AH, 0	;加进位
NOT	BL	;高字节取反
ADC	BL, 0	;加进位
MOV	DATA2, AL	;保存结果
MOV	DATA2+1, AH	
MOV	DATA2+2, BL	
MOV	AX, 4C00H	
INT	21H	
CSEG	ENDS	
END MBNEG		

0FH	
10H	
11H	
12H	
13H	
14H	
DATA1	35H
	26H
	03H
DATA2	00H
	00H
	00H

六. 顺序结构程序举例

例1：

- 题目：计算 $y=-x$ 。设 x 为三个字节长的数据，存于DATA1开始的单元。结果存入DATA2开始的单元。计算 $-x$ ，就是对 x 取补。

```
1 ;*****EXAM 5.1.1*****
2 SSEG SEGMENT STACK ;堆栈段
3 STK DB 20 DUP(0)
4 SSEG ENDS
5
6 DSEG SEGMENT ;数据段
7 DATA1 DB 35H, 26H, 03H ;(032635H)
8 DATA2 DB 3 DUP(0)
9 DSEG ENDS
10
11 CSEG SEGMENT ;代码段
12 ASSUME CS:CSEG, DS:DSEG
13 ASSUME SS:SSEG
14 MBNEG: MOV AX, DSEG ;段寄存器初值
15 MOV DS, AX
16 MOV AX, SSEG
17 MOV SS, AX
18 MOV SP, SIZE STK ;设置堆栈指
19 MOV AX, WORD PTR DATA1 ;读入数据低字节
20
21 MOV BL, DATA1+2 ;读入数据高字节
22 NEG AL ;取补低字节
23 CMC ;进位取反
24 NOT AH ;中字节取反
25 ADC AH, 0 ;加进位
26 NOT BL ;高字节取反
27 ADC BL, 0 ;加进位
28 MOV WORD PTR DATA2, AX ;保存结果
29
30 MOV DATA2+2, BL
31 MOV AX, 4C00H
32 INT 21H
33 CSEG ENDS
34 END MBNEG
```

0FH	
10H	
11H	
12H	
13H	
14H	

←

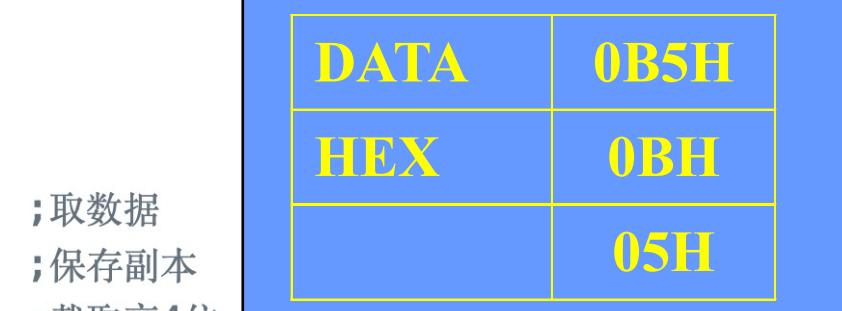
DATA1	35H
	26H
	03H
DATA2	00H
	00H
	00H

六. 顺序结构程序举例

例2：

- 题目：设内存DATA单元存放一个无符号字节数据，编制程序将其拆成两位十六进制数，并存入HEX和HEX+1单元的低4位，HEX存放高位十六进制数，HEX+1单元存放低位十六进制数。

```
1 ;*****EXAM 5.2*****
2 SSEG SEGMENT STACK
3 STK DB 20 DUP(0)
4 SSEG ENDS
5
6 DSEG SEGMENT
7 DATA DB 0B5H
8 HEX DB 0,0
9 DSEG ENDS
10
11 CSEG SEGMENT
12 ASSUME CS:CSEG, DS:DSEG
13 ASSUME SS:SSEG
14 DISC: MOV AX, DSEG
15 MOV DS, AX
16
17 MOV AX, SSEG
18 MOV SS, AX
19 MOV SP, LENGTH STK
20 MOV AL, DATA
21 AND AL, 0F0H ;截取高4位
22 MOV CL, 04
23 SHR AL, CL ;移至低4位
24 MOV HEX, AL
25 AND AH, 0FH ;截取低4位
26 MOV HEX+1, AH
27 MOV AX, 4C00H
28 INT 21H
29 CSEG ENDS
30 END DISC
```



六. 顺序结构程序举例

例3：

- 题目：设HEX，HEX+1单元的低4位分别存放一位十六进制数，编制程序将其装配在一个字节中并存入DATA单元。HEX单元中数据做为高位部分。

```
1 ;*****EXAM 5.3*****  
2 SSEG SEGMENT STACK  
3 STK DB 20 DUP(0)  
4 SSEG ENDS  
5  
6 DSEG SEGMENT  
7 HEX DB 0AH,06H  
8 DATA DB 0  
9 DSEG ENDS  
10  
11 CSEG SEGMENT  
12 ASSUME CS:CSEG, DS:DSEG  
13 ASSUME SS:SSEG  
14 PACK: MOV AX, DSEG
```

15	MOV	DS, AX
16	MOV	AX, SSEG
17	MOV	SS, AX
18	MOV	SP, SIZE STK
19	MOV	AL, HEX ;取数据高序位
20	MOV	CL, 04
21	SHL	AL, CL ;左移4位
22	OR	AL, HEX+1 ;与低4位或
23	MOV	DATA, AL ;保存结果
24	MOV	AX, 4C00H
25	INT	21H
26	CSEG	ENDS
27	END	PACK

HEX	0BH
	05H
DATA	0B5H

六. 顺序结构程序举例

例4：

- 题目：计算 $Y=5X+8$ ，设 X 为无符号字节数据，且在 $ARGX$ 单元存放。
计算结果，存入 $RLTY$ 单元。

1	;	*****EXAM 5.4*****	15	MOV DS, AX	
2	SSEG	SEGMENT STACK	16	MOV AX, SSEG	
3	STK	DB 20 DUP(0)	17	MOV SS, AX	
4	SSEG	ENDS	18	MOV SP, LENGTH STK	
5			19	MOV AL, ARGX ;取原始数	
6	DSEG	SEGMENT	20	MOV BL, 05	
7	ARGX	DB 15	21	MUL BL	;计算5x
8	RLTY	DW 0	22	MOV BX, 08	
9	DSEG	ENDS	23	ADD AX, BX	;再加上8
10			24	MOV RLTY, AX	;保存结果
11	CSEG	SEGMENT	25	MOV AX, 4C00H	
12		ASSUME CS:CSEG, DS:DSEG	26	INT 21H	
13		ASSUME SS:SSEG	27	CSEG ENDS	
14	CALCU:	MOV AX, DSEG	28	END CALCU	

ARGX	15H
RLTY	00H
	00H

六. 顺序结构程序举例

例5：

- 题目：用查表的方法将HEX单元中低4位十六进制数转换为对应的ASCII码，并显示出来。

十六进制	ASCII	十六进制	ASCII	十六进制	ASCII	十六进制	ASCII
0	30H	4	34H	8	38H	C	43H
1	31H	5	35H	9	39H	D	44H
2	32H	6	36H	A	41H	E	45H
3	33H	7	37H	B	42H	F	46H

0~9: +30H

A~F: +37H

六. 顺序结构程序举例

例5：

```
1 ;*****EXAM 5.5.1*****
2 SSEG SEGMENT STACK
3 STK DB 20 DUP (0)
4 SSEG ENDS
5
6 DSEG SEGMENT
7 HATAB DB 30H,31H,32H,33H,34H
8   DB 35H,36H,37H,38H,39H
9   DB 41H,42H,43H,44H,45H,46H
10 HEX DB 0CH
11 DSEG ENDS
12
13 CSEG SEGMENT
14   ASSUME CS:CSEG,DS:DSEG
15   ASSUME ES:DSEG,SS:SSEG
```

16 HTOA: MOV AX, DSEG
17 MOV DS, AX
18 MOV AX, SSEG
19 MOV SS, AX
20 MOV SP, SIZE STK
21 MOV AL, HEX
22 LEA BX, HATAB
23 XLAT
24 MOV DL, AL
25 MOV AH, 02
26 INT 21H
27 MOV AX, 4C00H
28 INT 21H
29 CSEG ENDS
30 END HTOA

HATAB+0	30H
+1	31H
+2	32H
+3	33H
+4	34H
+5	35H
+6	36H
+7	37H
+8	38H
+9	39H
+0A	41H
+0B	42H
+0C	43H
+0D	44H
+0E	45H
+0F	46H



1. 编写完成下列功能的小程序

(1) 将寄存器AL的第3, 4位变反, 其余位保持不变。

(2) 将寄存器AL的第3, 4位清零, 其余位保持不变。

(3) 将寄存器AL的第3, 4位置1, 其余位保持不变。

(4) DX和AX联合存放一个32位整数(DX存放高16位),
将该数据逻辑左移一位。

(5) 内存单元MEMB存放着一个8位的数据, 将它的高4位
和低4位互换。

- |▶ 2. 编写程序将AL中的第7位和第0位，第6位和第1位，第5位和第2位，第4位和第3位互换。
3. 编程序将AL中的第*i*, *i*+1位写入MEM单元的第*i*, *i*+1位，其他内容不变。
4. 对于下述指令，当AX=8573H，BX=8032H时，写出各指令执行后标志位的状态变化及目标地址内容。

指 令	OF	SF	ZF	PF	CF	目标地址内容
(1) ADD AL, BL						
(2) SUB AH, BL						
(3) NEG BH						
(4) SAR AH, 1						
(5) AND AL, BL						