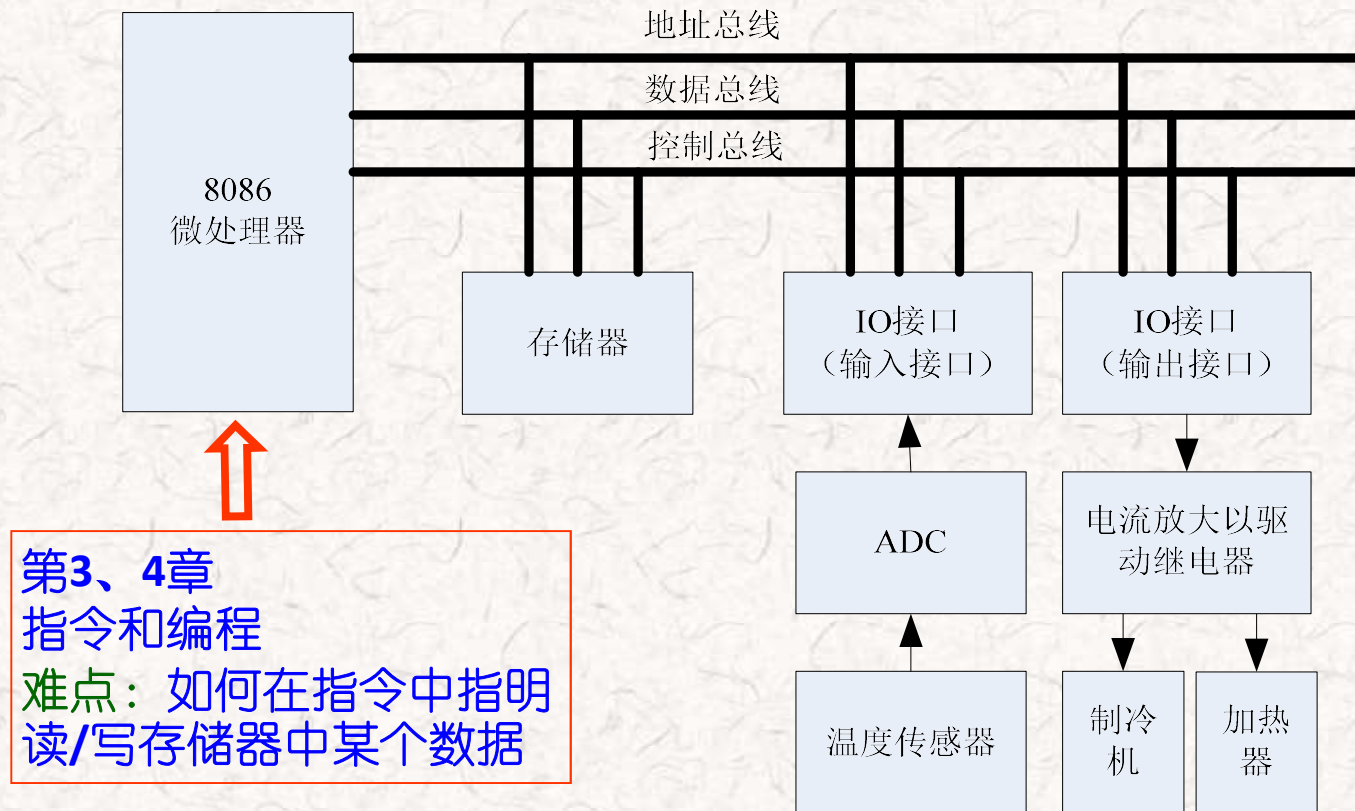




# 微机原理与接口技术

第三章和第四章 指令系统与编程

# 课程内容之8086指令系统与编程



## 第三章和第四章的学习内容

- 指令与编程分为7次课堂练习，每次2~3个练习
- 练习17是较为复杂的综合练习，课后完成
- 预习第三章和第四章大纲中内容，同时在编程练习中自学指令的用法
- 学习指令应注意：
  - 1. 根据指令助记符记住其对应的操作
  - 2. 指令所允许使用的寻址方式
  - 3. 指令对各标志位的影响情况：影响、不影响、不确定
- 重点：
  - 各种寻址方式--说明从哪里得到操作数
  - 地址指针寄存器的用法

# 入门1—指令、操作数、寻址方式

- **MOV AL, 12H ;**
  - 传送类指令，源操作数--立即数寻址，目的操作数--寄存器寻址，数字**12H**，也可以表示为**18**、**10+8**、**3\*6**等其它进制或者表达式（必须是静态的，即已知的）
  - 注意：立即数是没有位宽信息的，传送类指令两个操作数位宽需一致
- **MOV AH, AL**
  - 传送类指令，源操作数和目的操作数-寄存器寻址，将**AL**中的字节赋给**AH**
- **MOV BX, AX**
  - 传送类指令，源操作数和目的操作数-寄存器寻址，将**AX**中的字赋给**BX**



## 入门2—存储器寻址方式

- **MOV AL, ver1**
  - 传送类指令，源操作数—直接寻址，目的操作数--寄存器寻址
  - **Ver1**是定义在存储器中的变量（代表其偏移地址），可以为字节、字（位宽由寄存器操作数决定），也可以写为：**MOV AL, [2000H]**
- **MOV DH, [BX]**
  - 源操作数：寄存器间接寻址，将**BX**的内容作为存储器偏移地址，从数据段内该地址的存储单元读取一个字节到**DH**
- **MOV 2[SI], AX**
  - 目的操作数：寄存器相对寻址，将**SI**的内容+2作为存储器偏移地址，向数据段内该地址和该地址+1的存储单元写入**AX**中的字，**AL**写入偏移地址为**SI+2**的单元，**AH**写入偏移地址为**SI+3**的单元

## 入门3—存储器寻址方式

- **MOV AL, [BX][SI]**
  - 源操作数—基址变址寻址
  - 从数据段内某存储单元读一个字节到**AL**，存储单元的段内偏移地址为**BX+SI+1**
  - 基址寄存器：**BX、BP**，变址寄存器：**SI、DI**
- **MOV [BP][DI]5,AX**
  - 目的操作数：相对基址变址寻址，将**AX**的内容写入堆栈段连续两个存储单元，**AL**写入偏移地址为**BP+DI+5**的单元，**AH**写入偏移地址为**BP+DI+6**的单元
  - **BP**为基址寄存器时缺省指向堆栈段，通过**MOV DS:[BP][DI]5,AX**可以指向数据段

## 入门4—其它寻址方式

- **MUL BL**

- 执行  $AX = AL * BL$ ，源操作是BL 和 AL，目的操作数是AX，AL与AX就是隐含寻址

- **IN AL, 80H**

- 源操作数 AL为寄存器寻址，目的操作数80H表示端口地址，为直接IO寻址

- **MOV DX, 123H**（寄存器寻址，立即寻址）

- **OUT DX, AL**

- 源操作数 AL为寄存器寻址，目的操作数DX表示端口地址，即向地址为123H的端口执行写操作，DX为间接IO寻址



# 学习编程的小帮手

- 8086虚拟机
- 编写汇编指令，观察单步或者连续执行结果：存储器、寄存器、堆栈、标志寄存器
- 使用英文标点符号
- 注意虚拟机不完善的地方：
- 很多伪指令无法识别：自定义的ORG指令、length, \$, ?, 等
- 移位指令定义与课本有偏差
- 故：语法以课本为准，虚拟机仅作为辅助工具



## 程序模板

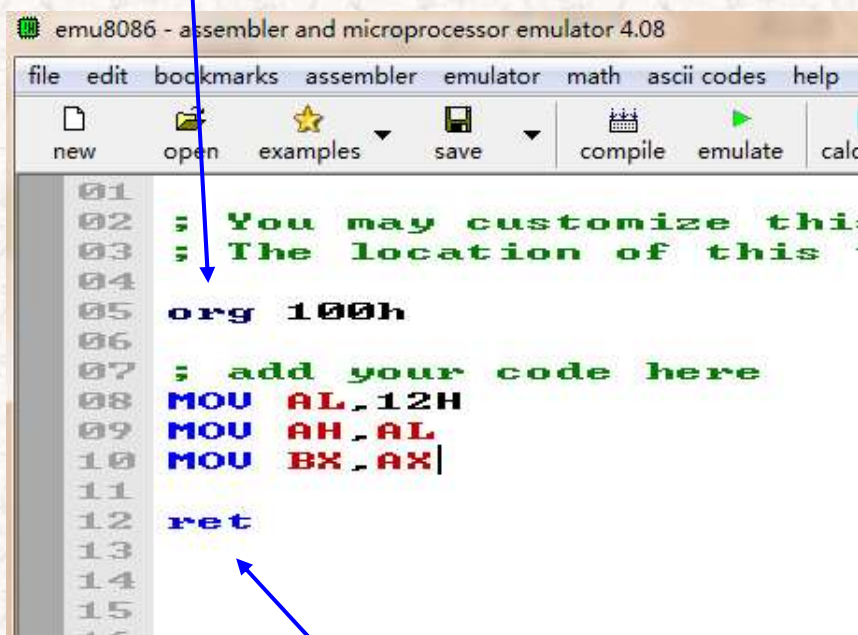
- 虚拟机提供**com**和**exe**两种模板，前者适用于小程序，不需要段定义伪指令，所有段默认在同一个**64KB**空间；后者适用于大程序，需要定义段；
- 打包文件中有一个编写好的**exe**程序模板，**mould.asm**，供大家参考，可以在其基础上修改

# .com类型小程序举例—编辑执行

此代码的**逻辑地址**（0700H虚拟机特定，0100H伪指令指定）

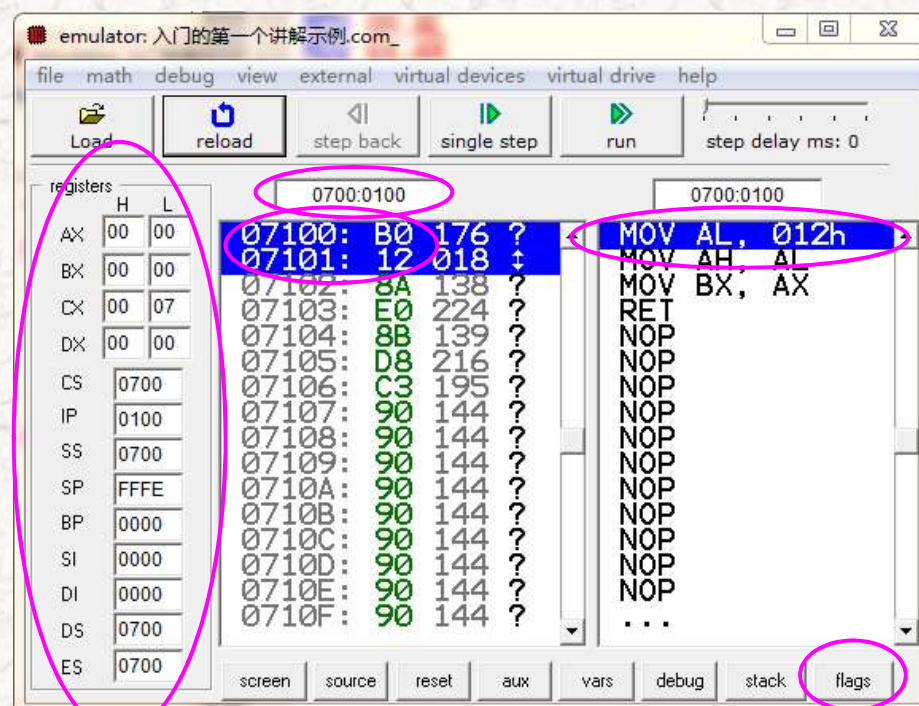
伪指令**ORG**  
指定段内偏移地址

即将执行的第一条汇编指令、**机器码**及存放地址  
**8086**内部寄存器，注意**CS**、**SS**、**DS**、**ES**、**SP**和**IP**的值



```
01  
02 ; You may customize this  
03 ; The location of this  
04  
05 org 100h  
06  
07 ; add your code here  
08 MOV AL, 12H  
09 MOV AH, AL  
10 MOV BX, AX  
11  
12 ret  
13  
14  
15  
16
```

返回**DOS**操作系统



emulator: 入门的第一个讲解示例.com

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers	H	L
AX	00	00
BX	00	00
CX	00	07
DX	00	00
CS	0700	
IP	0100	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0700:0100

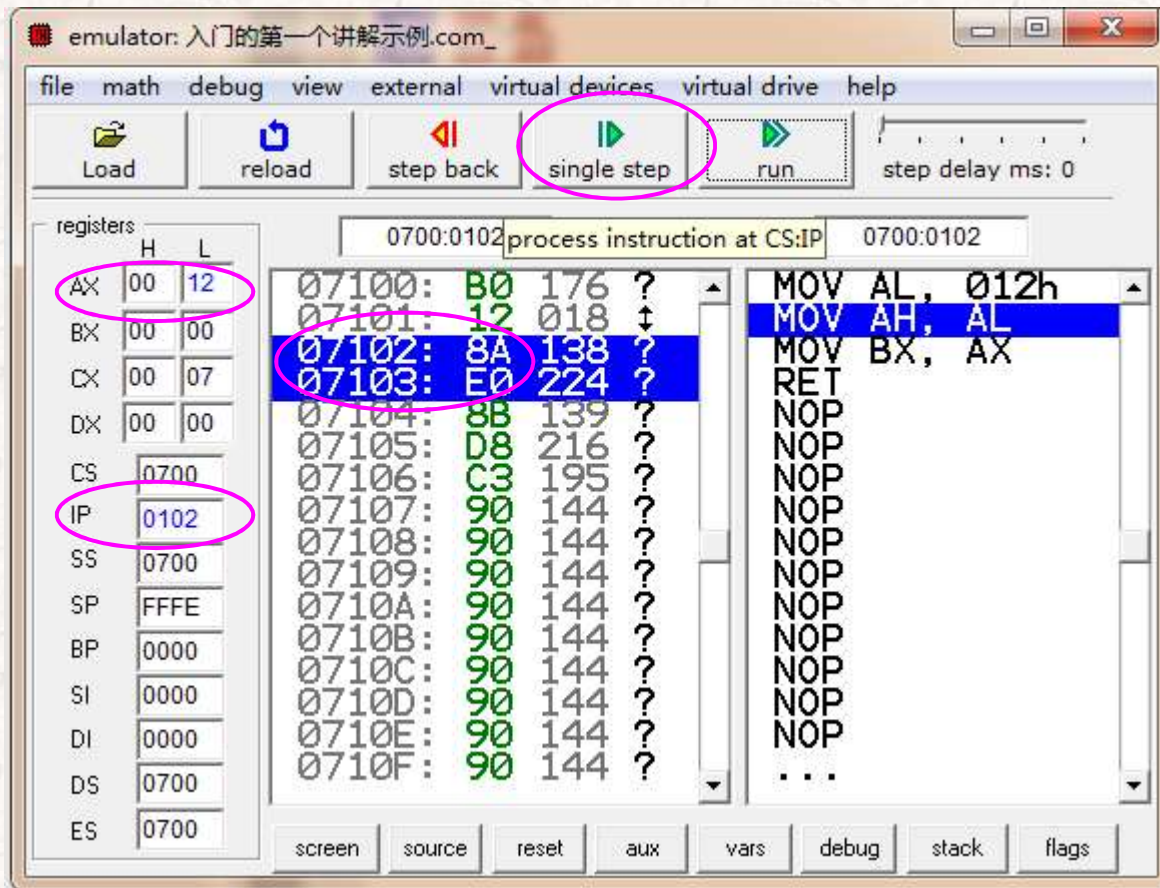
Address	Hex	Dec	Op
07100	B0	176	?
07101	12	18	?
07102	8A	138	?
07103	E0	224	?
07104	8B	139	?
07105	D8	216	?
07106	C3	195	?
07107	90	144	?
07108	90	144	?
07109	90	144	?
0710A	90	144	?
0710B	90	144	?
0710C	90	144	?
0710D	90	144	?
0710E	90	144	?
0710F	90	144	?

MOV AL, 012h  
MOV AH, AL  
MOV BX, AX  
RET  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
...

screen source reset aux vars debug stack flags



# .com类型小程序举例—单步调试



单步执行

寄存器AL的值被（指令）修改

指令指针IP被（微处理器自动）修改，指向第二条指令：  
**MOV AH,AL**

高亮即将执行的第二条汇编指令、机器码及其存放地址



# .com类型小程序举例—数据定义

- 伪指令**DB**：定义字节型变量，存放在存储器中

```
01  
02 ; You may customize this and  
03 ; The location of this templ.  
04  
05 org 100h  
06  
07 JMP START  
08  
09 X DB 11H  
10 Y DB 12H  
11  
12 START: MOV BL, X  
13         MOV BH, Y  
14  
15 ret  
16  
17
```

指令**JMP**  
用于跳过下面的数据区

伪指令**DB**  
定义了两个字节变量

定义的变量，不是指令，不可执行

COM类型程序没有段的定义，所以从汇编后的第一个字节开始执行

可能是：

- (1) 指令的机器码 (✓)
- (2) 伪指令定义的数据区 (✗)

必须跳过数据区，用**JMP** 指令

# 以mould.asm为例学习汇编程序架构1

- 汇编架构：若干程序模块，每个模块含四个段定义（不一定全部包括，用到什么定义什么）
  - 代码段----- 一定要有
  - 数据段----- 需要使用变量时应定义数据段
  - 堆栈段----- 有涉及堆栈的操作时应定义堆栈段
  - 扩展段----- 有涉及扩展段操作时应定义扩展段（例如串操作指令）
  - 段定义：由段定义伪指令和段属性声明伪指令进行说明
  - 变量定义：由变量定义伪指令进行说明
  - 另外还有：汇编结束伪指令
- 
- 汇编程序由指令（8086可执行）和伪指令（8086不可执行）组成，注意区分
  - 指令-----全部在代码段中

## 以mould.asm为例学习汇编程序架构2

- **DATA1 SEGMENT** ;段定义伪指令，可以改变段名
- ;在此加入你的变量定义
- **DATA1 ENDS** ;段定义结束伪指令，名字和前面的段定义相同
- **STACK1 SEGMENT** ;段定义伪指令，可以改变段名
- **STT DB 100 DUP(0FFH)** ;变量定义伪指令，可以改变堆栈名和深度
- **;TOP EQU LENGTH STT** ;堆栈深度获取方式1，准备赋值给SP，供参考
- **;TOP EQU \$-STT** ;堆栈深度获取方式2，准备赋值给SP，供参考
- **STACK1 ENDS** ;段定义结束伪指令，名字和前面的段定义相同
- **CODE1 SEGMENT** ;段定义伪指令，可以改变段名
- **ASSUME CS:CODE1,DS:DATA1,SS:STACK1** ;代码段、数据段、堆栈段的声明，相应改变段名，在这里指明所有定义的段的属性
- **START:** ;程序开始执行的第一条指令的标号，标号名可变
- .....



## 以mould.asm为例学习汇编程序架构3

- **CODE1 SEGMENT** ;段定义伪指令，可以改变段名
- **ASSUME CS:CODE1,DS:DATA1,SS:STACK1** ;以下开始指令，即8086可执行代码
- **START:** ;程序开始执行的第一条指令的标号，标号名可变
- **MOV AX,STACK1** ;堆栈段寄存器初始化
- **MOV SS,AX**
- **; MOV SP,TOP** ;与前面TOP定义相配合使用，供参考
- **MOV SP,100** ;根据自己伪指令中设置的深度填写
- **MOV AX,DATA1** ;数据段寄存器初始化
- **MOV DS,AX** ;如果需要定义扩展段，请仿照上述语法添加
- **;在此加入你的代码**
- **HLT** ;暂停
- **CODE1 ENDS** ;段结束伪指令
- **END START** ;汇编结束伪指令，指明程序入口为start，可以改变标号名

请用**EXE**格式完成练习

# 编程练习1-加法和标志位

- 给AL、BL、CL寄存器分别赋立即数，计算 $CL=AL+BL$ ，观察标志位变化
- 提示：
- 加法指令 **ADD**
- 思考：
- **MOV**和**ADD**指令源和目的操作数位宽是否可以不一样？
- 立即数有位宽信息吗？
- 执行哪条指令标志位发生了变化？哪些指令不影响标志位？
- 运算数据分别为有符号数和无符号数时结果的真值是多少

AL	BL	CL	运算结果	标志位CF、OF	无符号数结果	有符号数结果
11H	0FFH	22H	10H			
80H	7FH	22H	0FFH			
02H	7FH	22H	81H			
80H	80H	22H	0H			

# 编程练习1—代码示例

本练习没有用到数据段和堆栈段，可以不进行相关定义和初始化，仅保留代码段

```
CODE SEGMENT
ASSUME CS:CODE
START:
```

```
    MOV AL, 11H
```

```
    MOV BL, 0FFH
```

```
    MOV CL, 22H
```

```
    ADD AL,BL
```

```
    MOV CL,AL
```

```
    HLT
```

```
CODE ENDS
```

```
END START
```

;段定义

;程序开始执行的第一条指令的标号

;寻址方式？

;注意十六进制立即数最高位为字母时前面加0

;立即数可以写为 (11H\*2)等表达式

;

;程序停止

;段结束

;汇编结束，从start开始执行

AL	BL	CL	运算结果	标志位CF、OF	无符号数结果	有符号数结果
11H	0FFH	22H	10H	CF=1, OF=0	272 (溢出)	16
80H	7FH	22H	0FFH	CF=0, OF=0	255	-1
02H	7FH	22H	81H	CF=0, OF=1	129	129 (溢出)
80H	80H	22H	0H	CF=1, OF=1	256 (溢出)	-256 (溢出)



- 直线结构编程练习

# 本单元练习

- 练习2. 在存储器中定义2个字节型有符号数变量ADD1=80H、ADD2=0E2H、字型有符号数变量SUM=99H，读出ADD1、ADD2，求和，结果写入SUM中
- 答案：SUM=0FF62H
- 练习3. 在存储器中定义5个无符号字型变量，分别命名为D1、D2、D3、D4、D5，并分别初始化为0FFFFH、0、5544H、33H、0，请执行：D1加1，D2减1，D4-D3后写入D5，D4-D3-CF后写入D4
- 答案：D1=0，D2=0FFFFH，D3=5544H，D4=0AAEEH，D5=0AAEFH
- 练习4. 在存储器中定义3个字节型变量ADD1=80H、ADD2=092H、ADD3=0A3H，字型变量SUM1=99H，SUM2=88H，将ADD1~ADD3作为无符号数求和写入SUM1，作为有符号数求和写入SUM2
- SUM1=01B5H, SUM2=0FEB5H

## 编程练习2-变量定义

- 在存储器中定义2个字节型有符号数变量ADD1=80H、ADD2=0E2H、字型有符号数变量SUM=99H，读出ADD1、ADD2，求和，结果写入SUM中

- 提示：

- 字节型变量定义伪指令 DB
- 字型变量定义伪指令 DW
- 有符号数位宽扩展指令 CBW (字节扩展到字)

- 思考：

- 直接引用变量的名称是什么寻址方式？
- 有符号无符号变量的定义是否相同？
- 变量存放在哪个段？字节和字型变量如何存放的？
- 有符号、无符号数如何扩展位宽？

SUM=0FF62H



## 编程练习3-字变量增减操作

- 在存储器中定义5个无符号**字型**变量，分别命名为D1、D2、D3、D4、D5，并分别初始化为0FFFFH、0、5544H、33H、0，请执行：D1加1，D2减1，D4-D3后写入D5，D4-D3-CF后写入D4

- 提示：

- 加1指令 **INC**
- 减1指令 **DEC**
- 减法指令 **SUB, SBB**

- 思考：画出存储器中各字型变量在如何存放
- 哪些寄存器可以进行字型运算？
- SUB与SBB的区别？

**D1=0, D2=0FFFFH, D3=5544H,  
D4=0AAEEH, D5=0AAEFH**

## 练习4-位宽扩展

- 在存储器中定义3个字节型变量ADD1=80H、ADD2=092H、ADD3=0A3H，字型变量SUM1=99H，SUM2=88H，将ADD1~ADD3作为无符号数求和写入SUM1，作为有符号数求和写入SUM2
- 提示：
- 可采用带进位加法指令

ADC

SUM1=01B5H

SUM2=0FEB5H

## 编程练习2-代码示例

**DATA1 SEGMENT**

**ADD1 DB 080H**

**ADD2 DB 0E2H**

**SUM DW 99H**

**DATA1 ENDS**

;段定义，可以改变段名

;伪指令

;预留一个字的空間，注意本虚拟机不支持“？”

**STACK1 SEGMENT**

**STT DB 100 DUP(0FFH)**

**;TOP EQU LENGTH STT**

**;TOP EQU \$-STT**

**STACK1 ENDS**

;段定义，可以改变段名

;可以改变堆栈名和深度

;堆栈深度获取方式1，虚拟机不支持

;堆栈深度获取方式2，虚拟机不支持

**CODE1 SEGMENT**

**ASSUME CS:CODE1,DS:DATA1,SS:STACK1**

**START:**

;段定义，可以改变段名

;程序开始执行的第一条指令的标号，标号名可变



MOV AX,STACK1	;堆栈段初始化
MOV SS,AX	
;MOV SP,TOP	;虚拟机不支持，实际本指令可运行
MOV SP,100	;根据自己设置的深度填写
MOV AX,DATA1	;数据段初始化
MOV DS,AX	;如果需要定义扩展段，请仿照上述语法添加
;在此加入你的代码	
CODE:MOV AL, ADD1	;寻址方式？
CBW	;将字节有符号数扩展到字再进行相加，寻址方式？
MOV BX,AX	
MOV AL,ADD2	
CBW	;加数扩展到相同位宽
ADD AX, BX	
MOV SUM, AX	
HLT	;暂停
CODE1 ENDS	;代码段结束
END START	;汇编结束，从start开始执行，可以改变标号名

## 以下代码运行SUM值不变，为什么？

.....（数据段和变量定义略）

**CODE1 SEGMENT** ;段定义，可以改变段名

**ASSUME CS:CODE1,DS:DATA1,SS:STACK1**

**START:** ;程序开始执行的第一条指令的标号，标号名可变

**CODE: MOV AL, ADD1**

**CBW**

**MOV BX,AX**

**MOV AL,ADD2**

**CBW**

**ADD AX, BX**

**MOV SUM, AX**

**HLT**

**CODE1 ENDS**

**END START**

;暂停

;代码段结束

;汇编结束，从**start**开始执行，可以改变标号名

没有初始化数据段寄存器

学会调试，查看中间结果

实际上，第一条指令执行就不正确

## 编程练习3-代码示例

参考代码（省略了段定义、声明、段寄存器初始化）

D1 DW 0FFFFH ; 注意字型变量数据如何存放

D2 DW 0

D3 DW 5544H

D4 DW 33H

D5 DW 0

CODE: .....

MOV AX, D1 ; 方式1, 通过寄存器计算, 注意位宽一致

INC AX ; 影响哪些标志位? CF呢?

MOV D1, AX ; 结果写回

DEC WORD PTR D2 ; 方式2, 直接对变量计算

; 为什么需要显式说明位宽? DEC影响CF吗? ZF呢?

MOV AX, D3 ; 不能两个存储单元相减, 故先放入寄存器

MOV BX, D4 ; D4经过BX写入D5

MOV D5, BX

SUB D5, AX

SBB D4, AX ; 带进位相减, 结果写入目的操作数

HLT

注意: 变量存储 指令格式  
寻址方式 标志位影响



## 练习4-代码示例

(省略段定义、声明、段寄存器初始化)

```
add1 db 080h
add2 db 092h
add3 db 0a3h
sum1 dw 99h
sum2 dw 88h
```

CODE: .....

```
    mov ah,0 ;无符号数位扩展
    mov al,add1
    add al,add2
    adc ah,0
```

```
add al,add3
adc ah,0
mov sum1,ax
```

```
mov al,add1
cbw ;有符号数位扩展
mov bx, ax
mov al, add2
cbw
add bx,ax
mov al,add3
cbw
add bx,ax
mov sum2,bx
HLT
```

## 本单元练习

- 练习5. 在存储器中定义2个双字型无符号数变量，分别命名为ADD1、SUM，各包含两个变量，其中ADD1初值分别为89ABCDEFH，0A1234567H，读出ADD1中两个加数，相加，写入SUM第一个变量，ADD1中第2个变量减去第1个变量的结果写入SUM第二个变量中。
- 答案：SUM=2ACF1356H, 17777778H
- 练习6. 定义字型变量X、Y、Z，result，均为无符号数，计算 $result = X + Y * 10 + Z / 5$ ，初始化X=555，Y=666，Z=7777
- 答案：result=8770 or 2242H
- 拓展练习：见下一页

## 本单元拓展练习

- 无符号字节型变量**score**，存放**12**位同学的成绩，初始值分别为：**75, 85, 95, 63, 45, 89, 99, 80, 75, 60, 83, 82**定义字节型变量**AVE**，其中包含两个变量。
- **(1)** 计算平均成绩，舍弃小数部分，写入**AVE**第**1**个变量中
- **(2)** 计算平均成绩，小数部分四舍五入，写入**AVE**第**2**个变量中
- **思考：**
- 不使用判断跳转如何实现四舍五入？
- **答案：** **AVE = 77, 78**



## 编程练习5-双字操作及带进位加减法

- 在存储器中定义2个双字型变量，分别命名为ADD1、SUM，各包含两个变量，其中ADD1初值分别为89ABCDEFH，0A1234567H，读出ADD1中两个加数，相加，写入SUM第一个变量，ADD1中第2个变量减去第1个变量的结果写入SUM第二个变量中
- 提示：
- 双字型变量定义伪指令 DD
- 相关指令 CLC, STC
- 在vars中改变变量的size和element，可以观察不同位宽和个数的变量
- 思考：双字变量在存储器中存放的顺序是怎样的？
- 如何进行长位宽数据的加减运算（注意CF）？
- 如何分别访问双字的高字和低字？

SUM=2ACF1356H, 17777778H

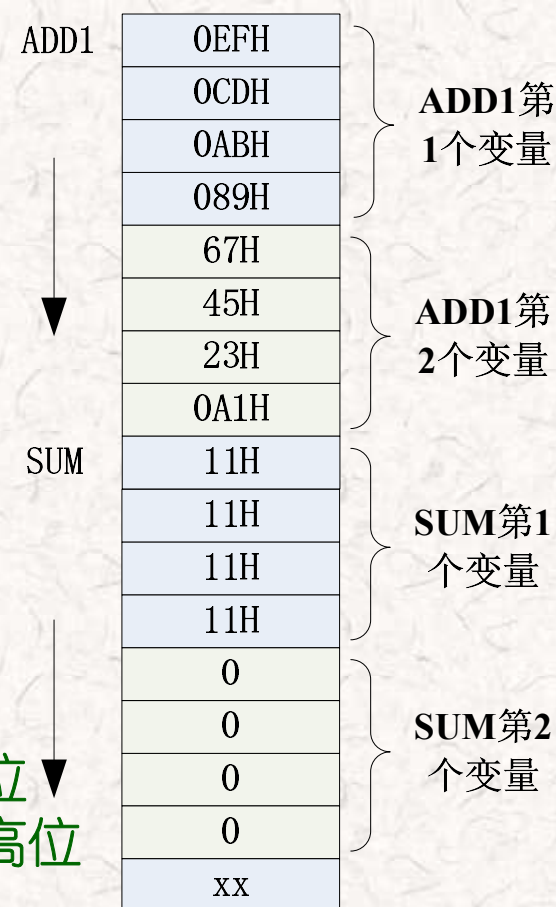
## 编程练习6-无符号乘法

- 定义字型变量X、Y、Z，result，均为无符号数，计算 $\text{result} = X + Y * 10 + Z / 5$ ，初始化X=555，Y=666，Z=7777，观察CF和OF的变化情况
- 提示：
- 无符号除法指令 **DIV**
- 无符号乘法指令 **MUL**
- 思考：可以直接乘以立即数吗？
- 字相乘，结果的位宽？字除法，被除数的位宽？

**result=8770**  
**or 2242H**

## 练习5参考代码 注意：变量存储 指令格式 寻址方式 标志位影响

- ADD1 DD 89ABCDEFH, 0A1234567H;
- SUM DD 11111111H, 0 ; 以上数据如何存放 ?
- CODE:段寄存器初始化部分略
- MOV AX, ADD1 ; 取第一个加数低字
- ADD AX, 4[ADD1] ; 加上第二个加数低字
- MOV SUM, AX
- MOV AX, 2[ADD1] ; 取第一个加数的高字
- ADC AX, ADD1+6 ; 带进位加第二个加数的高字
- MOV SUM+2, AX
- MOV AX, ADD1+4 ; 取第二个变量的低位
- SUB AX, ADD1 ; 减去第一个变量的低位
- MOV SUM+4, AX ; 写入SUM第二个变量的低位
- MOV AX, 6[ADD1] ; 取第二个变量的高位
- SBB AX, ADD1+2 ; 带进位减去第一个变量的高位
- MOV SUM+6, AX ; 结果写入SUM第二个变量的高位
- HLT





无符号数计算:  $\text{result} = X + Y * 10 + Z / 5$

## 练习6参考代码

思路: 注意**MUL**和**DIV**指令的隐含寻址和位宽变化, 位宽扩展方式

- X      DW 555
- Y      DW 666
- Z      DW 7777
- RESULT DW 0

- code: .....
- MOV AX, Y
- MOV BX, 10
- MUL BX
- 字相乘结果在**DX**, **AX**中
- ADD AX, X

MOV BX, AX ;结果暂存  
;要得到商为字, 需双字除以字

MOV AX, Z

**XOR DX, DX**

; 被除数为双字**DX**, **AX**

MOV CX, 5

DIV CX ; 商在**AX**中

**ADD BX, AX**

MOV RESULT, BX

HLT

X DB 75, 85, 95, 63, 45, 89, 99, 80, 75, 60, 83, 82

省略数据定义、段定义、  
段寄存器初始化等部分

MOV AX,0	ADD AL,X+8
MOV AL,X	ADC AH,0
ADD AL,X+1	ADD AL,X+9
ADC AH,0	ADC AH,0
ADD AL,X+2	ADD AL,X+10
ADC AH,0	ADC AH,0
ADD AL,X+3	ADD AL,X+11
ADC AH,0	ADC AH,0
ADD AL,X+4	MOV BL,12
ADC AH,0	DIV BL
ADD AL,X+5	MOV AVE,AL
ADC AH,0	MOV BL,5
ADD AL,X+6	SUB BL,AH
ADC AH,0	ADC AL,0
ADD AL,X+7	MOV AVE+1,AL
ADC AH,0	HLT

## 拓展练习参考代码

重点：如何实现四舍五入？

;余数大于等于6则CF=1  
;四舍五入

## 本单元练习

- 练习7. 定义字节型变量X、Y、Z，字型变量result,均为有符号数，计算 $result = X + Y * 10 + Z / 5$ ，初始化 $X = 55$ ， $Y = -66$ ， $Z = 0B3H$  (-77)。答案：result=-620 或 0FD94H
- 练习8. 设 $AX = 1122H$ ， $BX = 3344H$ ，将AX取反，BX取负，AX与BX相与后压栈，再弹出到CX。答案：CX=0CC9CH
- 练习9. 请不使用乘除法，对无符号数8FFFH执行除以8,再乘以4的操作；对有符号数-3200执行除以8再乘以-4的操作，两个结果依次写入字型变量result（定义为两个字深度）中
- 答案：result=47FCH 0640H



## 编程练习7-有符号数乘除法

- 定义字节型变量X、Y、Z，字型变量result,均为有符号数，计算 $result = X + Y * 10 + Z / 5$ ，初始化X=55，Y=-66，Z=0B3H (-77)

- 提示：

- 有符号除法指令
- 有符号乘法指令
- 位宽扩展指令

IDIV

IMUL

CBW, CWD

result=-620  
或 0FD94H

- 思考：

- 执行哪一步CF和OF置位？说明什么？
- 计算OFFFH（字）除以1（字节），分别作为有符号数和无符号数，观察结果，并说明为什么

## 编程练习8-堆栈操作

- 设AX=1122H，BX=3344H，将AX取反，BX取负，之后AX与BX相与后压栈，再弹出到CX

- 参考：

- 进栈指令                    **PUSH**
- 出栈指令                    **POP**
- 取反指令                   **NOT**
- 与指令                      **AND**
- 取负指令                   **NEG**

- 思考：数据进出堆栈的位宽是多少？（操作数可以是立即数吗？）

**CX=0CC9CH**

- 压栈后数据在堆栈中如何存放？

## 编程练习9-移位操作

- 请不要使用乘除法，对无符号数8FFFH执行除以8,再乘以4的操作；对有符号数-3200执行除以8再乘以-4的操作，两个结果依次写入字型变量result（定义为两个字深度）中
- 参考：
  - 字型变量重复定义伪指令 **DW 个数 DUP（初值）**
  - 算术右移指令 **SAR**
  - 逻辑右移指令 **SHR**
  - 算术逻辑左移指令 **SHL/SAL**
- 思考：乘或除以2的整数幂可以用什么实现？
- 绝对值较小的无符号数高位是什么？有符号数呢？

**result=47FCH**  
**0640H**



**result=X+Y\*10+Z/5**

思路：注意**IMUL**和**IDIV**用法，有符号数的位宽扩展方式

- X DB 55
- Y DB -66
- Z DB 0B3H
- RESULT DW 0
- code: .....
- MOV AL, Y
- **MOV BL, 10**
- IMUL BL ;字节相乘,积为字
- MOV BX,AX ;结果暂存
- MOV AL, X
- **CBW** ;有符号数扩展
- ADD BX, AX

## 练习7参考代码

```
MOV AL,Z ;Z/5结果需为字
CBW ;字节扩展到字
CWD
;商为字，被除数应为双字
MOV CX,5
IDIV CX ;商在AX中
ADD BX, AX
MOV RESULT, BX ;字结果写回

HLT
```

## 乘法指令MUL 和 IMUL溢出说明

- 定义：字节相乘，结果为字（**AL\*源操作数->AX**）；字相乘，结果为双字（**AX\*源操作数->DX, AX**）
- 溢出问题：字节相乘，乘积用字存放不会溢出；字相乘，乘积用双字存放不会溢出；
- 标志位情况：**CF、OF**受影响，其它标志位不定
- 字节相乘分析：结果超过一个字节的范围，**CF、OF**有效；
- **MUL**：高字节非零；**IMUL**：高字节非全部为符号位；
- 字相乘分析：结果超过一个字的范围，**CF、OF**有效；
- **MUL**：高字非零；**IMUL**：高字非全部为符号位；

## 除法指令DIV 和 IDIV溢出说明

- 定义：字除以字节，商为字节（AX/源操作数->AL）；双字除以字，结果为字（DX, AX/源操作数->AX）
- 溢出问题：字除以字节，商超过一个字节则溢出，产生除法错中断；双字除以字，商超过一个字则溢出，产生除法错中断；（除法错中断见第八章）
- 判断方法：被除数高半部分的绝对值大于除数则判为溢出
- 标志位情况：全部无定义
- 除法错怎么办？提前检测，避免出错



## 练习8参考代码

- ; 省略伪指令和段初始化部分.....
- **MOV AX,1122H**
- **MOV BX,3344H**
- **NOT AX**
- **NEG BX**
- **AND AX,BX**
- **PUSH AX**
- **POP CX**
- **HLT**

补充问题：令堆栈段寄存器为0，栈底为0200H，编程进行SS和SP初始化，所定义的堆栈最多可放多少个字？

```
MOV AX,0  
MOV SS,AX  
MOV SP,0200H
```

**100H**个字

## 练习9参考代码

RESULT DW 2 DUP(0)

CODE: MOV AX,8FFFH

MOV CL,3

SHR AX,CL ;逻辑右移

MOV CL,2

SHL AX,CL ;逻辑左移

MOV RESULT,AX

MOV AX,-3200

MOV CL,3

SAR AX,CL; 算术右移

MOV CL,2

SAL AX,CL; 算术左移

NEG AX

MOV RESULT+2,AX

HLT

# 总结1：传送类指令

- MOV 目的操作数, 源操作数
- PUSH 源操作数 (16位)
- POP 目的操作数 (16位)
- XCHG 目的操作数, 源操作数
- XLAT (隐含寻址 AL, BX)
- IN, OUT
- LEA, LDS, LES
- PUSHF, POPF, LAHF, SAHF



## 总结2：算术逻辑运算类指令

- ADD, ADC, INC
- SUB, SBB, DEC, CMP, NEG
- MUL, IMUL --- --参与运算的操作数位宽不同
- DIV, IDIV --- --参与运算的操作数位宽不同
- CBW, CWD
- NOT, AND, OR, XOR, TEST
- SHL/SAL
- SHR, SAR
- ROL, ROR, RCL, RCR

## 总结3：指令中操作数的限制

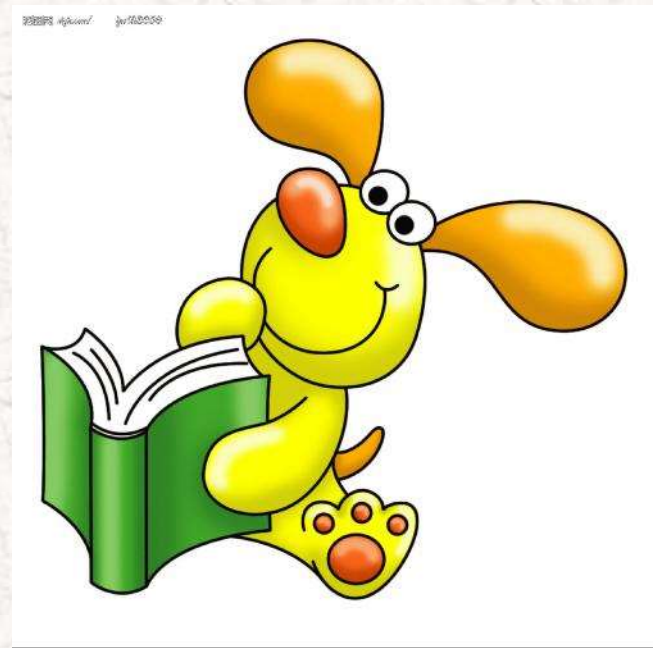
- **MOV PUSH POP XCHG指令：**
  - 源操作数和目的操作数必须位宽一致
  - **IP、CS**不能作为操作数
  - 立即数不能作为目的操作数
  - 两个操作数不能同时为存储器（串操作指令除外）
  - 立即数不能直接向段寄存器赋值
  - 段寄存器之间不能赋值
- **对于所有指令：**
  - 如果指令中没有位宽信息，则必须显式说明，仅**寄存器寻址**操作数具有明确的位宽信息
  - 说明方法：**WORD PTR**，**BYTE PTR**

## 练习：判断以下指令是否正确

- 1 MOV DS, AX ✓
- 2 MOV CS, AX
- 3 MOV SS, 2000H
- 4 PUSH 0100H
- 5 MOV IP, CS
- 6 MOV X, 80H (X是变量)
- 7 SHL AX, 3
- 8 ADD BX, AL
- 9 SUB AX, 55+9 ✓
- 10 CBW BL
- 11 MOV 99, SI
- 12 XCHG 99, AX
- 13 DIV AX, CL
- 14 IMUL -3
- 15 MOV DI, AX ✓
- 16 MOV DS, ES
- 17 MOV X, Y (均为变量)
- 18 MUL [BX]



- 第三章作业
- 1~3;
- 5~7;
- 8: 改为不使用**XLAT**指令 (使用亦可)
- 9, 10
- 13
- 15, 16
- 17题只考虑**CF**和**IF**的置位和清除



- 循环结构
- 复杂程序流程，请先画出流程图
- 对照每个环节，填写相应指令
- 重点：地址指针寄存器的用法
- 相关寻址方式：**MOV AX,[BX]** **MOV AX,2[BX]**
- **MOV AL, [BX+SI]** **MOV BH, 1[BP][DI]**
- 其中：寄存器间接寻址是基础

## 寄存器间接寻址的定义

- **BX, BP, SI, DI** 四个寄存器可以作为地址指针寄存器
- 形式: **[BX]**, 操作数是以寄存器的值为偏移地址的**存储单元**
- 举例: 若**BX=2000H, SI=1000H, DI=10H, BP=100H**
- **MOV AL, [BX]**                      **DS:2000H** 存放的字节 -> **AL**
- **MOV [DI], AX**                      **AX -> DS:10H** 开始的两个单元
- **MOV AL, 2[SI]**      寄存器相对寻址, **DS:1002H** 存放的字节 -> **AL**
- **MOV WORD PTR [SI][BX], 20H** 基址变址寻址, 表示:  
    **20H**写入 **DS:3000H**, **0**写入**DS:3001H**
- **MOV AX, 2[DI][BP]** 相对基址变址寻址, 表示:  
    **SS:0112H**开始存放的字->**AX** (**BX or BP**与 **SI or DI**组合)



# 循环结构

- 初始化
  - 设置循环次数
  - 设置变量的初始状态
- 循环体
  - 需要循环执行的内容
  - 修改循环变量
- 循环控制
  - 循环次数减1
  - 不为0则跳转到循环体开始处，为0则循环结束（继续向下执行）

## 举例说明寄存器间接寻址的用法

- 例如：有4K个字存放在以下数据区
- 2000H:0000H, 2000H:0002H, ....., 2000H:1FFFFH
- 如何依次读取上述数据区的每个字？
- 用寄存器存放数据区首单元偏移地址，上例中BX=0；
- 令DS=2000H，用寄存器间接寻址方式 **MOV AX,[BX]** 得到 2000H:0000H~0001H中存放的字
- **BX+2**后，则**MOV AX, [BX]**会得到2000H:0002H~0003H中存放的字
- 反复执行上一步骤直到数据读完（通过循环计数实现）

# 利用循环实现连续读取4K个字

- 1.段寄存器初始化
  - 2.指针指向数据区的首地址
  - 3.设置循环次数
  - 4.读取指针指向的那个字
  - 5.指针+2
  - 6.计数值减1
  - 7.如未减到0则返回执行步骤3
  - 8.程序结束
- **MOV AX,2000H**
  - **MOV DS,AX**
  - **MOV BX, 0**
  - **MOV CX, 4096**
  - **NEXT: MOV AX,[BX]**
  - **ADD BX,2**
  - **DEC CX**
  - **JNE NEXT**
  - **HLT**



## 本单元练习

- **练习10.** 将**AX**中的数字高低位颠倒后写入**DX**，保持**AX**的值不变，同时将**AX**中“1”的个数记录在**BL**中；
- **练习11.** 定义**字节型**变量**X**，**80**字节深度，全部初始化为**OFFH**，从地址为**110**的输入端口读取**80**个字节，依次写入**X**
- **练习12.** 设输出端口**112**接**8**盏**LED**灯，设该端口输出数据的某位为**1**，则灯亮，为**0**，则灯灭，编程令一盏灯从最低位到最高位不断循环点亮，每次移位操作前循环延时**10**次
- **拓展练习：**在输出端口**112**所连接的**8**盏灯中，同时点亮**2**盏灯，并使每次点亮的**2**盏灯有不同的组合，共计产生**20**种不同的亮灯方式循环演示，每种亮灯方式之间循环延时**10**次

## 编程练习10-循环移位

- 将**AX**中的数字高低位颠倒后写入**DX**，保持**AX**的值不变，同时将**AX**中“1”的个数记录在**BL**中；
- 参考：
- 循环移位指令： **ROL, ROR**
- 带进位循环移位： **RCL, RCR**
- 判断进位位是/否为0跳转 **JC, JNC**
- 思考：
- 左移、右移，循环移位，移出位到了哪里？
- 参考答案：
- 输入**AX=0055H**, 输出： **AX=0055H, DX=AA00H, BL=4**
- 输入**AX=8000H**, 输出： **AX=8000H, DX=0001H, BL=1**

## 编程练习11-IO输入

- 定义**字节型**变量**X**，**80**字节深度，全部初始化为**OFFH**，从地址为**110**的输入端口读取**80**个字节，依次写入**X**
- 提示：
- 使用**simple-io**模板（模板为**COM**类型，各段重叠，从头开始执行，故需跳过数据区）
- 字节型变量重复定义 **DB 个数 DUP(初值)**
- 取变量偏移地址（形成地址指针）**1 MOV REG, OFFSET 变量名**
- 取变量偏移地址（形成地址指针）**2 LEA REG, 变量名**
- 判断是/否为**0**跳转 **JZ, JNZ**
- 读**IO**操作 **IN AL, 端口地址（小于256时）**
- 使用**aux**中的**memory**观察存储器中数据，注意**段地址**和**偏移地址**



# simple\_io.asm

- ; this sample shows how to access virtual ports (0 to 0FFFFh).
- ; these ports are emulated in this file: c:\emu8086.io
- ; this new technology allows to make external add-on devices
- ; for emu8086, such as led displays, thermostat, stepper-motor, etc...
- ; "devices" folder contains sample device that works with this sample.
- ; (with visual basic source code).
- #start=simple.exe#
- #make\_bin#
- name "simple"
- ; write byte value 0A7h into the port 110:
  - mov al, 0A7h
  - out 110, al
- ; write word value 1234h into the port 112:
  - mov ax, 1234h
  - out 112, ax
- mov ax, 0 ; reset register.
- ; read byte from port 110 into AL:
  - in al, 110
- ; read word from port 112 into AX:
  - in ax, 112
- hlt

JMP START  
X DB .....  
START: 代码  
.....

win7以上需在管理员账号下运行

## 编程练习12-IO输出，多重循环

- 设输出端口**112**接**8**盏**LED**灯，设该端口输出数据的某位为**1**，则灯亮，为**0**，则灯灭，编程令一盏灯从最低位到最高位不断循环点亮，每次移位操作前延时循环**10**次
- 提示：
- 写IO操作                      **OUT 地址, AL (地址小于256)**
- 循环左移指令                **ROL**
- 循环延时举例
- **MOV CH,10 ; 循环次数**
- **DELAY: DEC CH**
- **JNZ DELAY**

## 拓展练习--点灯进阶篇

- 在输出端口**112**所连接的**8**盏灯中，同时点亮**2**盏灯，并使每次点亮的**2**盏灯有不同的组合，共计产生**20**种不同的亮灯方式循环演示，每种亮灯方式之间循环延时**10**次
- 思考：
- 寻找能够以统一的方式生成各种点灯模式的方法

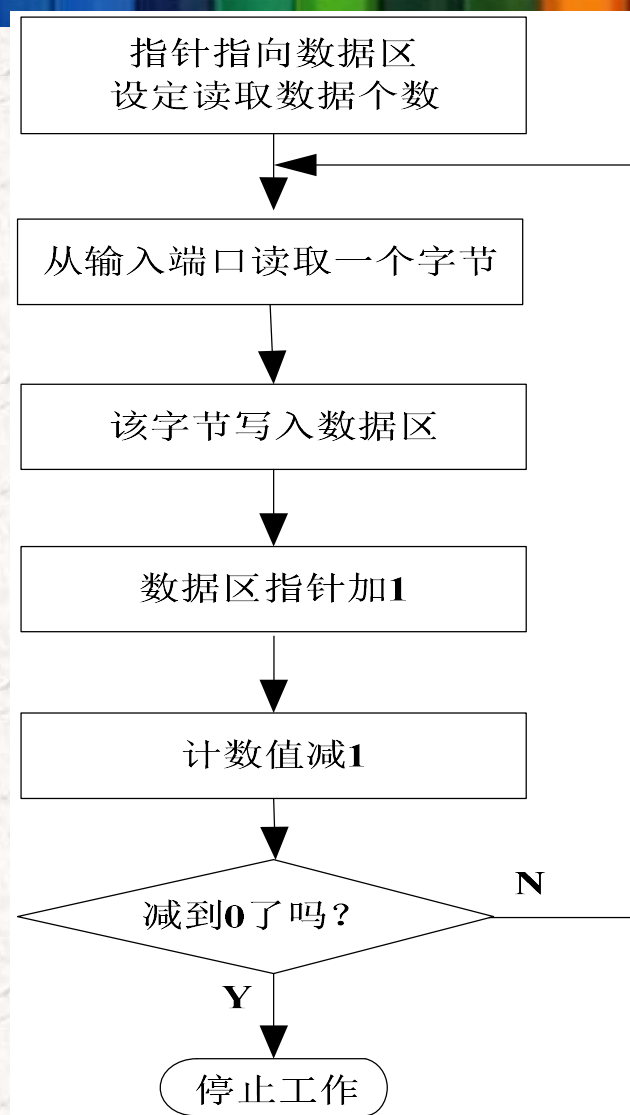


## 练习10参考代码

MOV AX, 55H	; AX初值
MOV CL, 16	; 设循环次数
MOV BL, 0	; 记录“1”的个数
CONT:ROL AX, 1	; 循环左移, 16次后AX恢复原值
JNC ISZERO	; 判断移出位是否为1
INC BL	; 为1则BL加1, INC不影响CF
ISZERO:RCR DX, 1	; 将CF移入到DX的最高位
DEC CL	; 循环次数减1
JNZ CONT	; 没完成则继续循环

# 练习11

## 流程图示例



## 练习11参考代码

```
#start=simple.exe#
```

```
#make_bin#
```

```
name "simple"
```

```
JMP CODE
```

```
X DB 80 DUP(0FFH)
```

```
CODE:LEA DI,X
```

; 取变量偏移地址, MOV DI, OFFSET X 亦可

```
MOV CL,80
```

```
; read byte from port 110 into AL:
```

```
RDNEXT: IN AL,110
```

; IO寻址, 注意与立即寻址区别

```
MOV [DI], AL
```

; 目的操作数寄存器间接寻址

```
INC DI
```

; 修改地址指针

```
DEC CL
```

; 修改计数值

```
JNZ RDNEXT
```

; 循环未完成则跳转

```
HLT
```



## 练习12参考代码

```
#start=simple.exe#  
#make_bin#  
name "simple"  
code:MOV AL,01H  
      MOV CL,8  
PATTERN:OUT 112, AL  
        ROL AL,1  
        MOV CH,10  
DELAY: DEC CH  
        JNZ DELAY  
        DEC CL  
        JNZ PATTERN  
        JMP CODE
```

;如循环次数大于1必须写入CL

;延时

## 参考代码

```
#start=simple.exe#
```

```
#make_bin#
```

```
name "simple"
```

```
jmp code
```

```
X DB 03H, 06H, 0BH, 18H, 30H, 60H,  
0B0H, 05H, 0AH, 14H, .....
```

```
; 定义输出模式
```

```
code:MOV SI, OFFSET X
```

```
MOV CL,20
```

## 拓展练习参考代码

```
PATTERN:MOV AL,[SI]
```

```
INC SI
```

```
OUT 112,AL
```

```
MOV CH,10
```

```
DELAY: DEC CH
```

```
JNZ DELAY
```

```
DEC CL
```

```
JNZ PATTERN
```

```
JMP CODE
```

## 两种IO寻址方式

- 地址小于256: **OUT 地址, AL** ;地址小于256
- **IN AL, 地址** ;地址小于256
- 地址大于一字节: **MOV DX, 地址** ;地址超过一字节
- **OUT DX, AL**
- **IN AL, DX**

Tip: 比较一下这三条语句

**IN AL, 80H**

**MOV AL, 80H**

**MOV AL, [80H]**

Tip: 想想这两条指令是否正确?

**IN AL, DX**

**MOV AL, DX**



- 1.分支结构
- 尤其需要流程图
- 2.软件中断指令之一：**DOS**功能调用

# DOS功能调用入门举例

- 在屏幕上显示字符“A”，按一个键后显示字符“B”

MOV AH,2

MOV DL,'A'

INT 21H

;待显示字符的ASCII写入DL

;DOS功能调用2，显示字符

MOV AH,7

INT 21H

;等待键入字符

;DOS功能调用7，输入按键不回显，结果在AL中

MOV AH,2

MOV DL,'B'

INT 21H

HLT

DOS功能调用1，输入按键并回显。P133有更多DOS功能调用介绍

## 本单元练习

- **练习13**设有冷库温控系统，输入端口**110**输入补码表示的环境温度，输出端口**112**的**D0**位为**1**，开启加热；**D1**位为**1**，开始制冷；**112**的**D7:2**位用于其它控制任务，需保持不变（假设**112**端口状态可以读入）。要求：温度高于**5°**，开启制冷，低于**-5°**，开始加热，其它情况不加热也不制冷。
- **练习14**.屏幕上显示字符串“**How are you ?**”等待键盘输入，如果是“**Y**”或“**y**”，则显示“**Fine, and you?**”，如果是“**N**”或“**n**”，则显示“**Realy bad**”，按其它键显示“**Pardon ?**”。也可自行设计语句，要求每句话带回车换行，不断循环



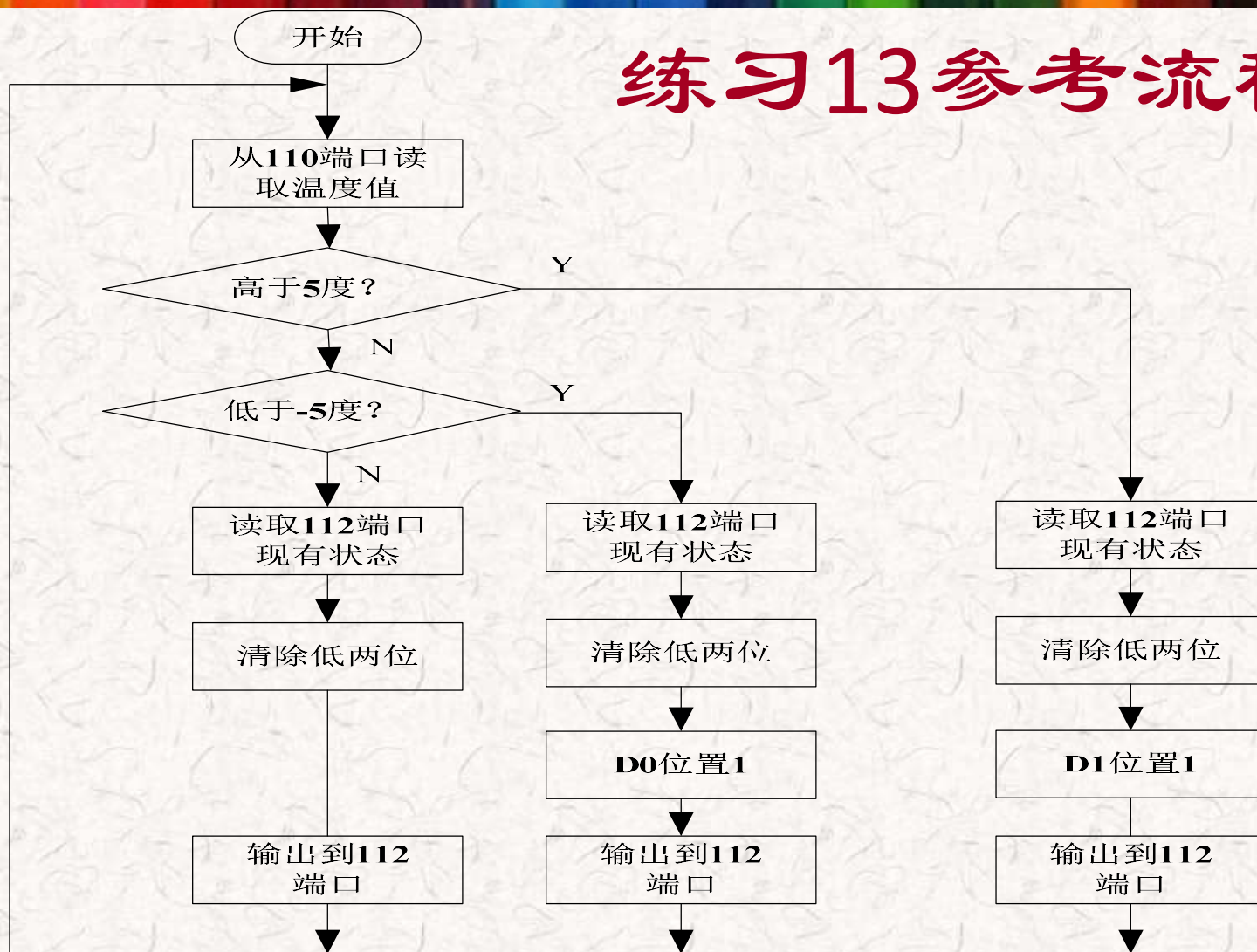
## 编程练习13-有符号数比较

- 设有冷库温控系统，输入端口**110**输入补码表示的环境温度，输出端口**112**的**D0**位为**1**，开启加热；**D1**位为**1**，开始制冷；**112**的**D7:2**位用于其它控制任务，需保持不变（假设**112**端口状态可以读入）。要求：温度高于**5°**，开启制冷，低于**-5°**，开始加热，其它情况不加热也不制冷。
- 思考：如何只对**指定位**进行置位或清除？
- 提示：
- 比较指令：**CMP**
- 有符号数比大小**JG, JGE, JL, JLE**
- 逻辑或运算**OR**
- 逻辑与运算**AND**

## 编程练习14-DOS功能调用练习

- 屏幕上显示字符串“**How are you ?**”等待键盘输入，如果是“**y**”或“**y**”，则显示“**Fine, and you?**”，如果是“**N**”或“**n**”，则显示“**Realy bad**”，按其它键显示“**Pardon ?**”。也可自行设计语句，要求每句话带回车换行，不断循环。
- 提示：
- 显示字符串：DOS功能调用9
- 输入按键并回显DOS功能调用1
- 输入按键不回显DOS功能调用7
- 字符串定义DB ‘How are you ?’
- 回车的ASCII码0DH
- 换行的ASCII码0AH
- 注意大小写字母ASCII不同

## 练习13参考流程图





## 练习13参考代码

#start=simple.exe#

#make\_bin#

name "simple"

CONT:IN AL,110

CMP AL,5

JG COOL ;转去制冷

CMP AL,-5

JL HEAT ; 转去加热

IN AL,112

AND AL,0FCH; 清除低两位

OUT 112,AL

JMP CONT

HEAT:IN AL,112

AND AL,0FCH

OR AL, 01H ;D0置1,加热

OUT 112,AL

JMP CONT

COOL:IN AL,112

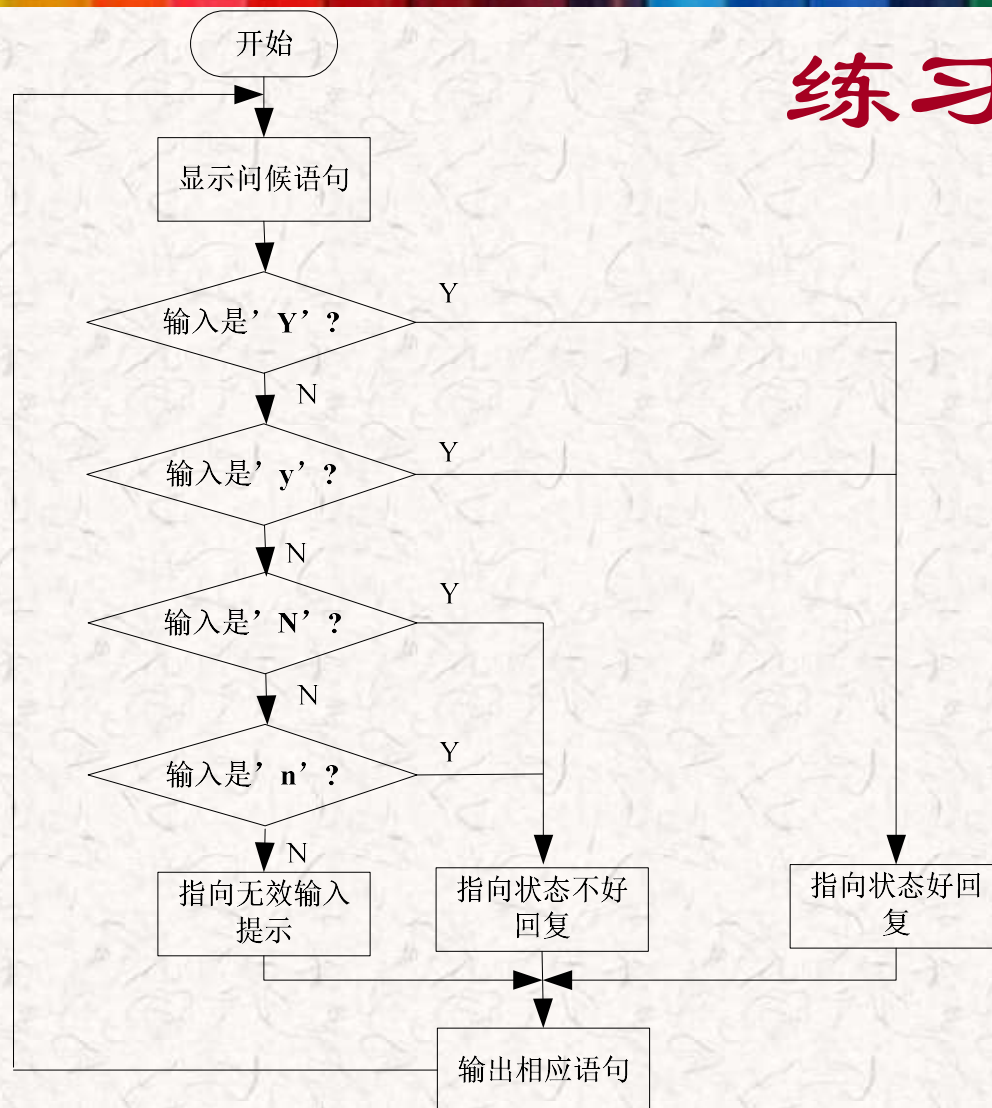
AND AL,0FCH

OR AL,02H ;D1置1, 制冷

OUT 112,AL

JMP CONT

## 练习14参考流程图



ASK DB 'How are you',3FH,0DH,0AH,'\$' ;'\$'结束符  
YS DB 'Fine,and you',0DH,0AH,'\$'  
NS DB 'Realy bad',0DH,0AH,'\$'  
ELSE DB 'Pardon',3FH,0DH,0AH,'\$'

## 练习14参考代码

CODE:MOV AH,9  
LEA DX, ASK  
INT 21H  
MOV AH,7  
INT 21H ; AL为键值  
CMP AL,'Y'  
JZ DIS\_Y  
CMP AL,'y'  
JZ DIS\_Y  
CMP AL,'N'  
JZ DIS\_N

CMP AL,'n'  
JZ DIS\_N  
LEA DX,ELSE  
ANS: MOV AH,9  
INT 21H  
JMP CODE  
DIS\_Y: LEA DX,YS  
JMP ANS  
DIS\_N: LEA DX,NS  
JMP ANS



- 过程调用

# CALL和RET指令的执行过程

- 执行 **CALL XXXX** 指令时，**CS**和**IP**指向下一条指令
- （硬件自动）将**IP**（或**CS**和**IP**）入栈
- （硬件自动）将被调用过程入口的段地址和偏移地址赋值给**IP**（或**CS**和**IP**）
- 执行 **XXXX** 过程
- 最后一条指令：**RET**
- （硬件自动）将栈顶的数据弹出，赋值给**IP**（或**IP**和**CS**）
- 回到**CALL**的下一条指令处执行

## 过程（子程序）调用举例1

将存储在数据区中**100**个非组合的**BCD**码转换为**ASCII**码，将转换程序编制为一个子程序

```
DATA      SEGMENT
BCD1      DB 01, 02, 03, ...      ; 共100个待转换数
CUNT      EQU $-BCD1
DATA      ENDS
```

```
STACK1    SEGMENT PARA STACK
STAPN     DW  20H DUP(?)
TOP       EQU  LENGTH STAPN
STACK1    ENDS
```

```
CODE      SEGMENT
          ASSUME CS: CODE, DS: DATA, SS: STACK1
```



```

START:  MOV    AX, DATA
        MOV    DS, AX ; 数据段初始化
        MOV    AX, STACK1
        MOV    SS, AX
        MOV    SP, TOP ;堆栈段初始化
        MOV    CX, CUNT
        MOV    SI, OFFSET BCD1
        CALL   BSC ;注意过程的入口参数
        MOV    AH, 4CH
        INT     21H      ;主程序结束返回DOS

```

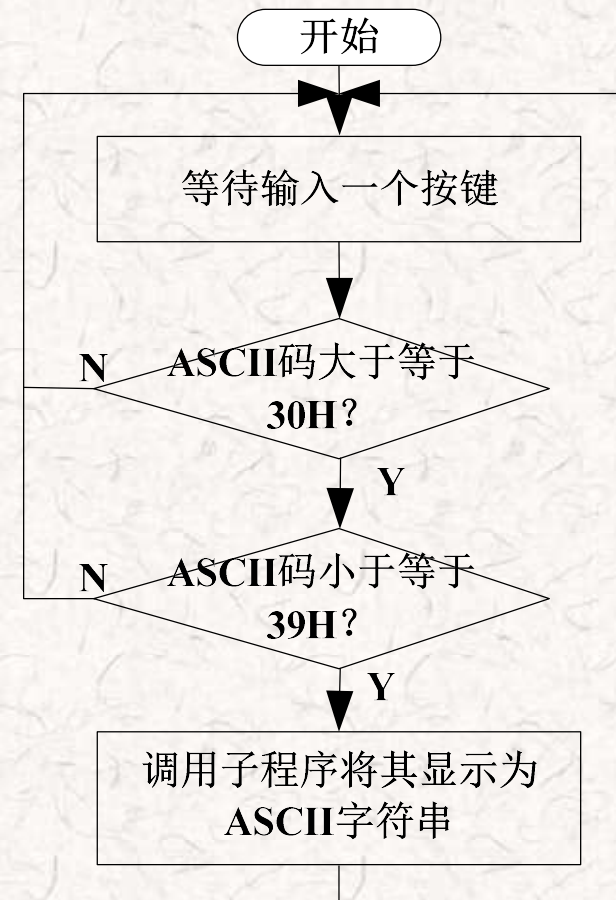
```

BSC      PROC      ; 子程序的定义
CYCLE:   OR BYTE PTR [SI], 30H
        INC      SI
        DEC      CX
        JNZ      CYCLE
        RET
BSC      ENDP
CODE     ENDS
        END      START

```

## 过程（子程序）调用举例2

- 接收屏幕按键，如果是数字，显示其**ASCII**码，例如1，显示“31H”，如果按下其它键，则继续等待。不断循环。要求将数字转为**ASCII**码显示的部分采用过程调用的方式实现
- 可以采用查找表的方式实现



## 参考代码

省略段定义伪指令和段寄存器初始化部分

```
V1 DB '30H$', '31H$', '32H$', '33H$', '34H$', '35H$', '36H$', '37H$',  
      '38H$', '39H$'
```

STAR: 初始化 .....

```
WAIT1: MOV DX, OFFSET V1  
        MOV AH, 7  
        INT 21H  
        CMP AL, 30H  
        JC WAIT1 ;无符号数比大小  
        CMP AL, 40H  
        JNC WAIT1  
        CALL SHOW ; DX、AL为参数  
        JMP WAIT1  
        HLT
```

SHOW: PUSH DX ;寄存器保护

```
        MOV AH, 0  
        SUB AL, 30H  
        SHL AX, 1  
        SHL AX, 1 ; 乘以4  
        ADD DX, AX  
        MOV AH, 9  
        INT 21H ;显示字符串  
        POP DX ;寄存器恢复  
        RET ; 过程返回
```



# 子程序调用常见错误--堆栈

- 必须定义堆栈，并对SS和SP初始化
- 跳转的目的地址必须在本子程序内
- 最后一条指令必须是ret
- 进出栈实际操作必须对应，call与ret，push与pop

从端口82H读入1个字节，如果小于10，则转为ASCII码输出到80H

NEXT: IN AL, 82H

CALL FAR PTR ASC

CONT: MOV [DI], AL

INC DI

JMP NEXT

ASC PROC

PUSH AX

CMP AL, 10

JNC CONT (错误)

ADD AL, 30H

OUT 80H, AL

POP AX

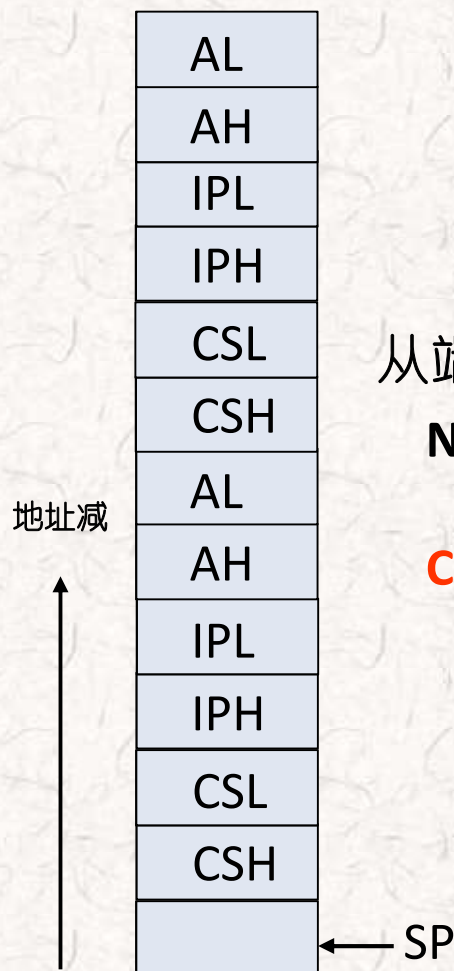
RET

ACS ENDP

1. 设读入为2

2. 设读入为12

3. 设读入为9



## 本单元练习

- **练习15**：定义深度为7的字节型无符号变量X和Y，其中X的初值分别为12H, 34H, 56H, 78H, 0ABH, 97H, 0EFH, Y的初值为0H。编程使X中数据按照从大到小的顺序写入变量Y。将适当的功能以子程序调用方式实现。
- **练习16**：通过键盘输入2位十六进制数，并以'H'结尾，将其转变为8bit二进制数输出，并以'B'结尾，不断循环，如果输入错误，给出'ERROR'提示，并重新输入，将转换过程编写为子程序实现
- **练习17**：**练习17**综合练习，设计PPT讲解具体要求见后

## 编程练习15-子程序调用

- 定义深度为7的字节型无符号变量X和Y，其中X的初值为初值分别为12H, 34H, 56H, 78H, 0ABH, 97H, 0EFH, Y的初值为0H。编程使X中数据按照从大到小的顺序写入变量Y。要求将适当的功能以子程序调用方式实现
- 提示：
- 过程调用 `CALL`
- 过程返回 `RET`
- 当前位置符 `$` (用于伪指令中表示静态数据)
- 思考：
- X和Y为有符号数或无符号数，编程有什么不同？



## 编程练习16-子程序调用

- 通过键盘输入2位十六进制数，并以'H'结尾，将其转变为8bit二进制数输出，并以'B'结尾，不断循环，如果输入错误，给出'ERROR'提示，并重新输入，将十六进制到二进制的转换过程编写为子程序实现（输入支持大小写）
- 提示：
- 输入一个字符串                      DOS功能调用0AH
- 输出字符串                            DOS功能调用09H
- 注意:DOS功能调用0AH需要首先初始化接收数据区（建议全部初始化为'\$'，解析数据时注意数据区格式）
- 子程序调用注意保护寄存器

## 编程练习17-综合练习

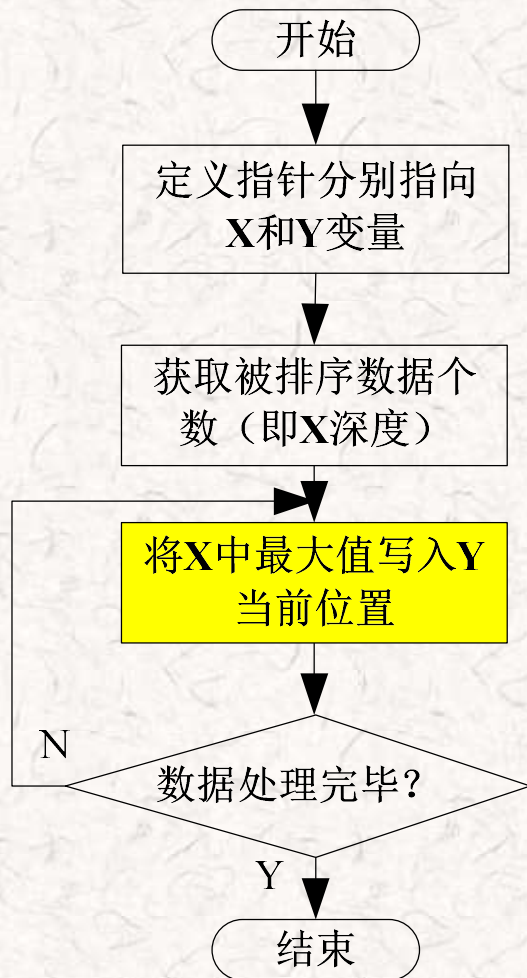
- 设计一个计算器，完成2个小于等于4位的十进制有符号整数(-9999~9999)的“+、-、\*、/”运算。
- 屏幕显示“**please enter number1**”，等待接收数字并回显，回车表明数字输入结束；换行显示“**please enter operator +、-、\*、/**”，如果输入正确的运算符则输出“**please enter number2**”，在输入数字并回车后以十进制方式显示运算结果。循环执行上述过程。
- 如果数字或运算符输入错误，给出提示（自行设计提示语）并再次提请输入。如果运算结果超出-9999~9999，提示发生溢出，并返回头重新输入。
- 重点：如何合理划分子程序？
- 提示：参考例4.47, 4.48，使用过程调用
- 独立编写，两周内完成，最终提交asm代码和PPT

# 综合练习要求

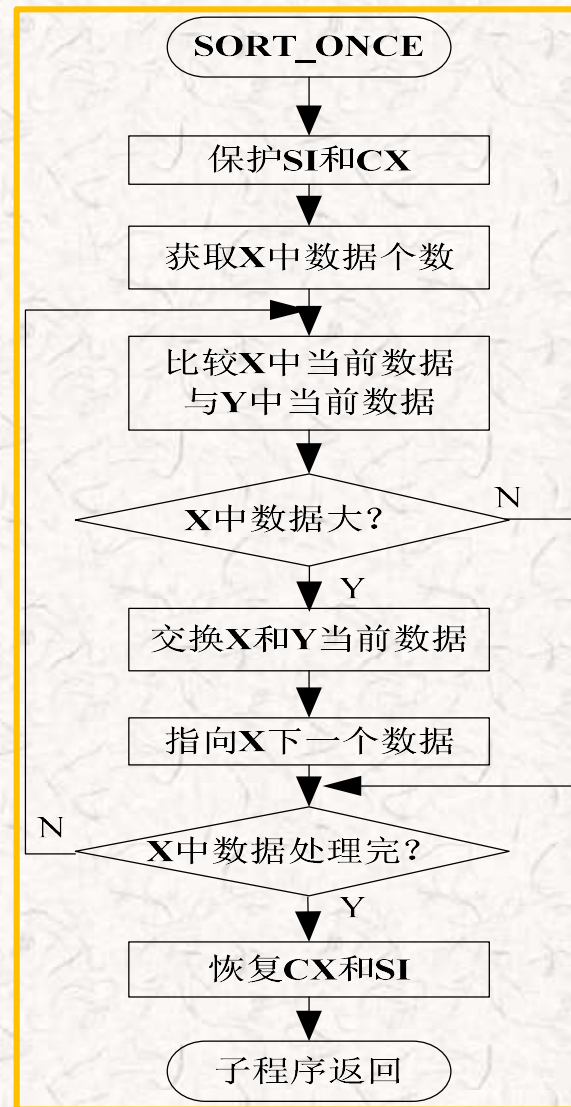
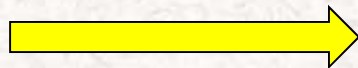
- **PPT**说明以下内容:
- 程序中使用的变量
- 主程序流程图
- 子程序流程图, 子程序入口和出口参数说明
- 十种寻址方式各举**2**个例子
- 写出**10**个错误的指令的例子



# 练习15流程图



入口条件：SI指向Y中当前即将被写入的数据



## 练习15参考代码

JMP START

X DB 12H, 34H, 56H, 78H, 0ABH,  
97H, 0EFH

CONT DB \$-X ;注意与EQU的区别

Y DB 7 DUP(0)

START:MOV BX,OFFSET X

MOV CL,CONT

MOV SI,OFFSET Y ;入口参数

NEXT: CALL SORT\_ONCE

INC SI

DEC CL

JNE NEXT

RET

SORT\_ONCE: PUSH BX

PUSH CX

MOV CL,CONT

CMPNEXT: MOV AL,[BX]

CMP AL, [SI]

JC NOTCHG

XCHG [SI],AL ;大的放入Y

MOV [BX],AL ;小的放入X

NOTCHG: INC BX

DEC CL

JNE CMPNEXT

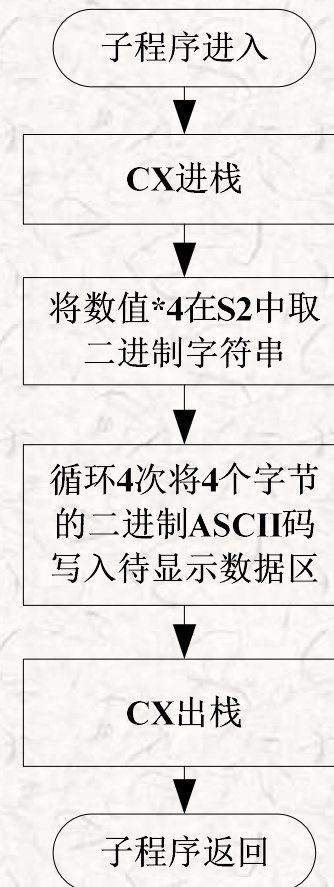
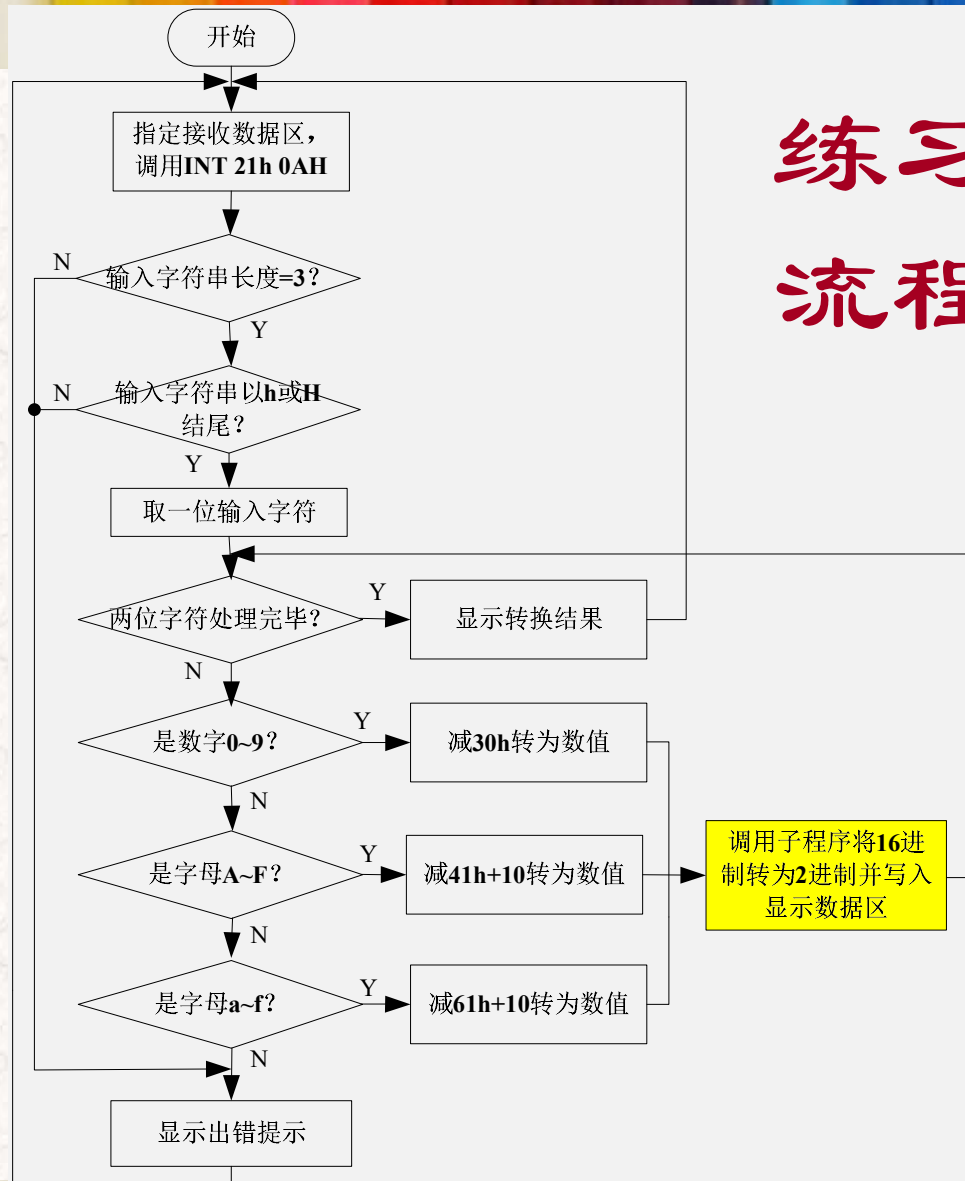
POP CX

POP BX

RET

# 练习16 流程图

入口条件：  
AL存放一位16进制数的数值  
DI指向转换结果数据区





## 练习16参考代码

```
org 100h  
JMP START
```

```
S1 DB 0DH,0AH,'ERROE! INPUT AGAIN:',0DH,0AH,'$';
```

```
S2 DB '0000', '0001', '0010', '0011', '0100', '0101', '0110', '0111', '1000',  
'1001', '1010', '1011', '1100', '1101', '1110', '1111'; 一位16进制转2进制
```

```
S3 DB 0DH,0AH,'00000000B',0DH,0AH,'$' ;结果显示
```

```
S4 DB 6 DUP('$') ;输入数据区初始化
```

```
START: MOV AH, 0AH
```

```
MOV DX, OFFSET S4
```

```
INT 21H
```

```
MOV BX,DX
```

```
CMP BYTE PTR 1[BX],3 ;个数对吗 ?
```

```
JNZ ERROR
```

0	24H
1	3
2	31H
3	32H
4	42H
5	0DH

```
CMP BYTE PTR 4[BX],'H'
```

```
JE CONT
```

```
CMP BYTE PTR 4[BX],'h' ;结尾对吗 ?
```

```
JE CONT
```

```
JMP ERROR
```

CONT: INC BX ;注意POINT TO NUMBER  
MOV DI,OFFSET S3  
INC DI  
INC DI ;结果指针, 跳过回车换行  
  
MOV CL,3 ;注意次数+1  
NEXT: DEC CL  
JZ INPUT\_OVER  
INC BX  
  
MOV AL,[BX]  
CMP AL,30H ;30H-39H,41H-46H,61H-66H  
JC ERROR  
CMP AL,3AH  
JC RIGHTD ;是数字  
CMP AL,41H  
JC ERROR

CMP AL,47H ;是A~F  
JC RIGHTA1  
CMP AL,61H  
JC ERROR  
CMP AL,67H  
JC RIGHTA2 ;是a~f  
ERROR: MOV AH,9  
MOV DX,OFFSET S1  
INT 21H  
JMP START  
  
INPUT\_OVER: MOV AH,9 ;结果显示  
MOV DX,OFFSET S3  
INT 21H  
JMP START

**RIGHTD:** SUB AL,30H ;转数值后调用子程序  
CALL CHANGE  
JMP NEXT

**RIGHTA1:** SUB AL,41H ;转数值后调用子程序  
ADD AL,0AH  
CALL CHANGE  
JMP NEXT

**RIGHTA2:** SUB AL,61H ;转数值后调用子程序  
ADD AL,0AH  
CALL CHANGE  
JMP NEXT

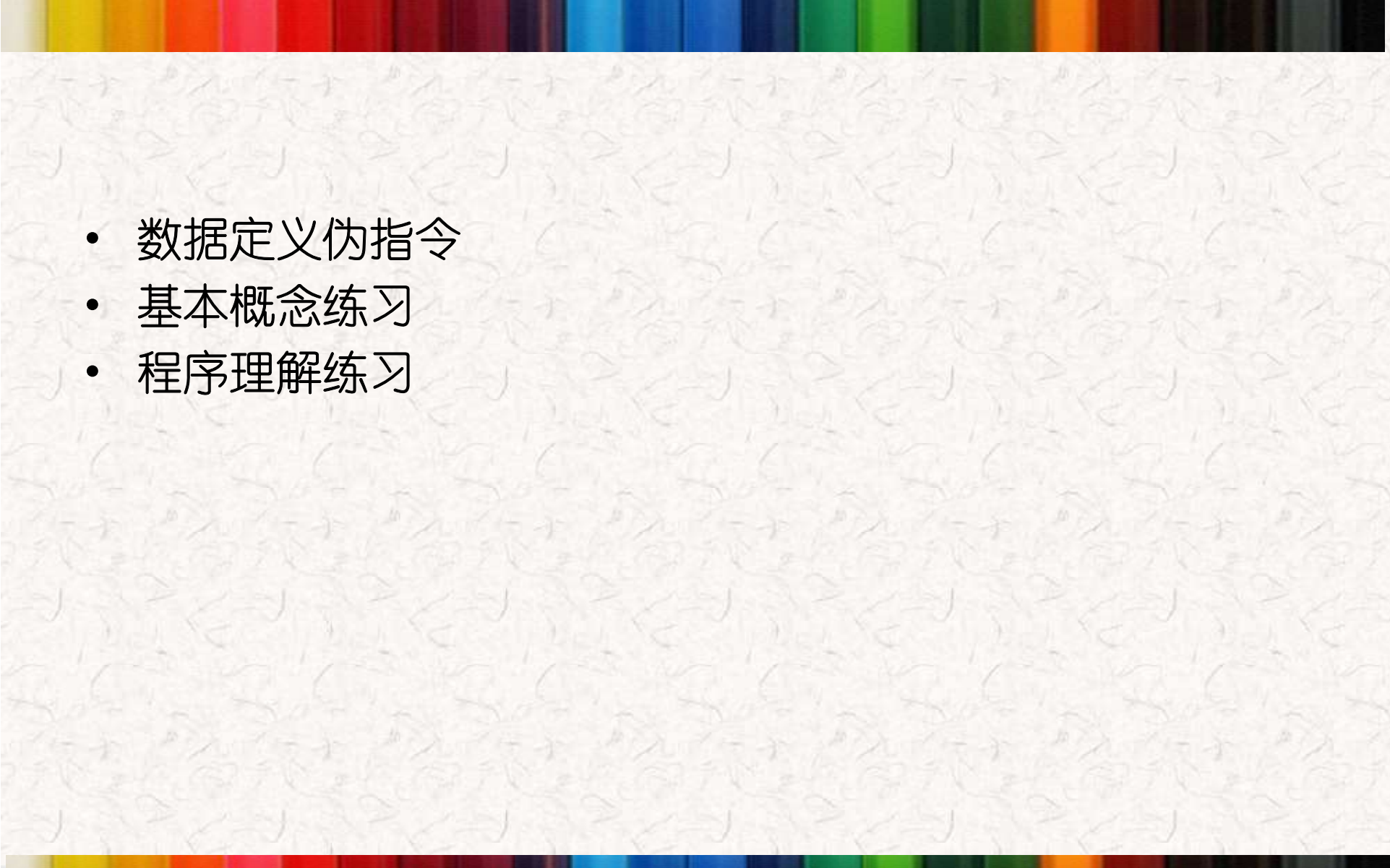
AL存放一位16进制数的数值  
DI指向转换结果数据区

**CHANGE:** PUSH CX ;保护CX  
MOV SI,OFFSET S2  
MOV AH,0; ;扩展到16位  
SHL AX,1 ;乘以4  
SHL AX,1  
ADD SI,AX ;指向2进制字符串  
MOV CL,4  
**MOVE:** MOV AL,[SI]  
MOV [DI],AL ;写入输出数据区  
INC SI  
INC DI  
DEC CL  
JNE MOVE  
POP CX ;恢复CX  
RET



## 总结：控制转移类指令

- 无条件转移：JMP
- 单一标志位条件转移指令：
- JE/JNE(JZ/JNZ) JC/JNC JS/JNS JO/JNO JP/JNP
- 无符号数比较标志位组合条件转移指令：JA JAE JB JBE
- 有符号数比较标志位组合条件转移指令：JG JGE JL JLE
- 子程序调用与返回指令：CALL RET
- 循环指令：LOOP
- 中断指令：INT n
- 溢出中断指令：INTO
- 中断返回指令：IRET

- 
- 数据定义伪指令
  - 基本概念练习
  - 程序理解练习

# 数据定义的练习1

FIRST

SECOND

THIRD

FOURTH

ORG 1000H

DB 30H;

DB 'GOOD';

DW 5566H;

EQU 78H;

FIRST

SECOND

THIRD

~~FOURTH~~

30H

47H

4FH

4FH

44H

66H

55H

DS:1000H

DS:1001H

DS:1002H

DS:1003H

DS:1004H

DS:1005H

DS:1006H

DS:1007H

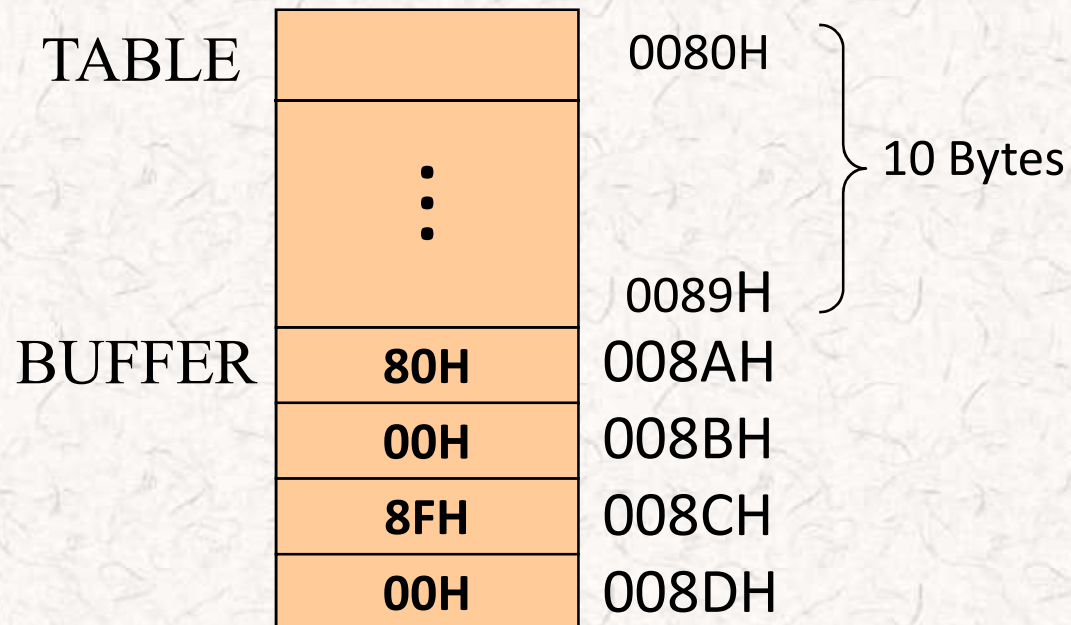
DS:1008H

DS:1009H



# DUP、地址标号、\$例子

```
ORG 80H  
TABLE DB 10 DUP(?)  
BUFFER DW TABLE, $+3
```



## 数据定义的练习2

- **ORG 0060H**
- **V1 DW '12'** V1: 0060H开始 31H, 32H ('1', '2') 伪指令中的字符串视为字节序列
- **V2 DB 20, '15','Cb'** V2: 0062H开始 14H,31H,35H,43H,62H
- **V3 DB 2 DUP(3, ?)** V3: 0067H开始 3H,?,3H,?
- **V4 EQU \$-V1**
- **V5 DW -1,-3** V5: 006BH开始 0FFH,0FFH,0FDH,0FFH
- **V6 DW V2,V5** V6: 006FH开始 62H,00H,6BH,00H
- 说明各变量所分配的存储空间及初始化的数据。（要求：标出各变量的起始偏移地址，初始化数据以16进制数表示，'A'的ASCII码为41H，'a'的ASCII码为61H，'0'的ASCII码为30H，不确定值用0表示）

**注意：** **MOV AX, V1** 的执行结果为 **AX=3231H**  
**MOV AX, '12'** 的执行结果为 **AX=3132H**

## 基本概念练习-指令理解与寻址方式

- **MOV AX, 2408H**
- **MOV BX, [SI]**
- **MOV [BP+100H], AX**
- **MOV DX, ES:[BX,SI]**
- **IN AL, 05H**
- **MOV CL, 0FFH**
- **MOV 5[BX], BL**
- **MOV BYTE PTR[BX+DI], '\$'**
- **MOV 9[BP+DI], DX**
- **MOV DS,AX**



## 基本概念练习-有效地址计算

- 已知：DS=1000H，BX=0200H，SI=02H，存储器中10200H~10205H分别存放10H,2AH,3CH,46H,59H,6BH,说明源操作数寻址方式、有效地址(如有)、AX的值

• MOV AX,0200H	立即寻址	/	0200H
• MOV AX,[200H]	直接寻址	0200H	2A10H
• MOV AX,BX	寄存器寻址	/	0200H
• MOV AX, [BX]	寄存器间接寻址	0200H	2A10H
• MOV AX,3[BX]	寄存器相对寻址	0203H	5946H
• MOV AX,[BX+SI]	基址变址寻址	0202H	463CH
• MOV AX,2[BX+SI]	相对基址变址寻址	0204H	6B59H

## 基本概念练习-指令理解

DATA SEGMENT

A DB '\$', 10H

B DB 'COMPUTER'

C DW 1234H, 0FFH

D DB 5 DUP(?)

E DD 1200459AH

- 以下指令执行的结果

- **MOV AL,A**

**AL=24H '\$'=24H**

- **MOV DX, C+2**

**DX=00FFH**

- **MOV BP, OFFSET B**

**BP=0002H**

- **MOV CX, DS:3[BP]**

**CX=5550H 'U'=55H,'P'=50H**

- **LEA DX,D**

**DX=000EH (或14)**

- **LDS SI, E**

**DS=DATA SI=0013H (或19)**

## 基本概念练习-指令对错

- 1 MOV DL,AX
- 2 MOV DS,0200H
- 3 MOV IP, 0FFH
- 4 MOV AX,[BX+BP]
- 5 MOV [SI+DI],DL
- 6 MOV CL, OFFSET X
- 7 IN BL, 05H
- 8 MOV 8650H,AX
- 9 MOV [BX],[1200H]
- 10 MOV [SI+BP+2],IP
- 11 MOV AL, ES:[BP]
- 12 MOV BX,OFFSET 0A20H
- 13 XCHA AL, 03
- 14 OUT AL, 0FFH
- 15 DIV X (X为变量名)
- 16 OUT [DX], AL





## 基本概念练习—转移目标地址

- CS=1200H, IP=0100H, DS=2000H, SI=3000H, BX=0300H, (20300H) = 4800H, (20302H) = 00FFH, TABLE = 0500H, PROG\_N标号地址为1200:0278H, PROG\_F标号地址为3400:0ABCH, 说明指令执行后转移到何处执行
- JMP PROG\_N                      CS=1200H IP=0278H
- JMP BX                            CS=1200H IP=0300H
- JMP [BX]                        CS=1200H IP=4800H
- JMP FAR PROG\_F                CS=3400H IP=0ABCH
- JMP DWORD PTR [BX]          CS=00FFH IP=4800H

## 基本概念练习—子程序调用

- CS=1200H, IP=0100H, SS=5000H, SP=0400H, DS=2000H, SI=3000H, BX=0300H, (20300H) = 4800H, (20302H) = 00FFH, TABLE = 0500H, PROG\_N标号地址为1200:0278H, PROG\_F标号地址为3400:0ABCH, 说明指令执行后转移到何处执行, 堆栈内容以及堆栈指针如何变化
- CALL PROG\_N CS=1200H IP=0278H 0100H进栈 SP=SP-2
- CALL BX CS=1200H IP=0300H 0100H进栈 SP=SP-2
- CALL [BX] CS=1200H IP=4800H 0100H进栈 SP=SP-2
- CALL FAR PROG\_F CS=3400H IP=0ABCH 1200H 0100H进栈 SP=SP-4
- CALL DWORD PTR [BX] CS=00FFH IP=4800H 1200H 0100H进栈 SP=SP-4

## 编程练习题 — 阅读代码

```
AT DB 0AH, 0BH, 0CH, 0DH, 0EH, 0FH, 00H
MOV BX, 0
LOP: MOV DL, AT[BX]
      CMP DL, 0
      JZ DONE
      ADD DL, 37H
      MOV AH, 2 ;2号功能调用将字符送到屏幕显示出来
      INT 21H
      INC BX
      JMP LOP
DONE: HLT
```

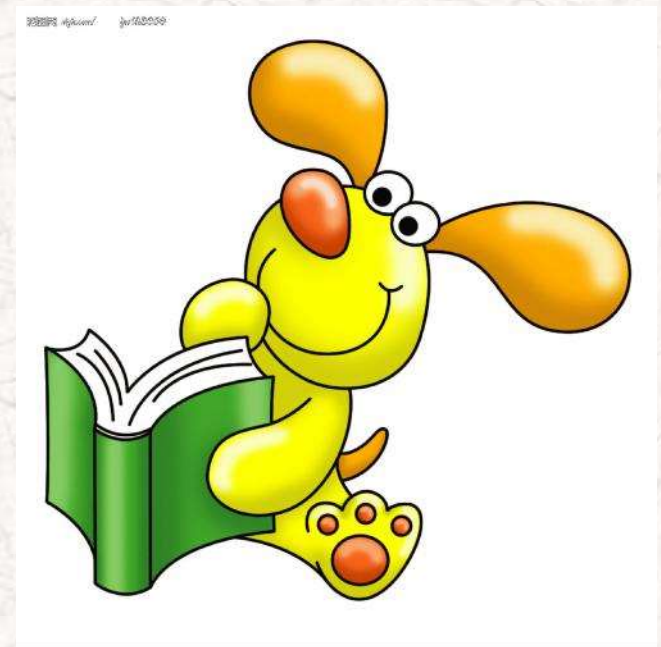
- 1) 上述程序段执行后在屏幕上显示的内容依次是 ABCDEF :
- 2) 执行后 **BX** = 6 。



# 一点扩展知识—CISC和RISC

- 复杂指令系统**CISC**
  - 特点：支持复杂的寻址方式，支持复杂功能的指令，指令长度不同（控制逻辑复杂）
  - 代表：**Intel X86**架构
- 精简指令系统**RISC**
  - 特点：只支持最常用的**20%**指令，简化寻址方式，以寄存器寻址为主，定长指令（简化译码和控制逻辑），专门的存储器访问指令
  - 代表：**ARM, RISC-V**，目前的主流架构
  - **Intel**的酷睿采用**RISC**内核微指令包装形成的**CISC**架构

- 第四章作业
- 1~4
- 6~7
- 12, 14, 18, 19



## 预告-第五章需要掌握的内容

- 5.1、5.3 了解，其中：5.2.1 掌握
- 5.4 重点学习