第4章 x86汇编语言程序设计

1. MASM语句结构及开发过程
2. MASM语言表达式和运算符
3. 程序段的定义和属性
4. 常用的系统功能调用
5. 汇编程序设计

汇编语言程序特点

x86汇编程序例：

存储器中首地址为ARRAY的M个字数组，编写程序，求出该数组的内容之和（不考虑溢出），并把结果存入TOTAL中，程序段如下：

```
DATA SEGMENT
          i   DB  0
          M   DB  10
   ARRAY   DW   1,2,3,4,5,6,7,8,9,10
   TOTAL   DW   ?
DATA  ENDS
CODE  SEGMENT
        ASSUME   DS:DATA, CS: CODE
START：MOV   AX，DATA
          MOV   DS，AX
          MOV   CL，M
          MOV   AX，0
          MOV   SI，AX
   XUN：ADD   AX，ARRAY[SI]
          ADD   SI，2
          LOOP  XUN
          MOV   TOTAL，AX
           …
CODE  ENDS
        END  START
```

```
#include <iostream>
using namespace std;
const int  M=10;
int main( )
{
      int  i=0,TOTAL=0;
      int  array[M]={1,2,3,4,5,6,7,8,9,10};
      while (i<M)
        {   TOTAL=TOTAL+array[i];
            i ++;          }
       cout <<"sum="<<TOTAL<<"\n";
       return 0;
}
```

1. MASM语句结构及开发过程

```
stack   SEGMENT
        DB   100  DUP('stack')
stack   ENDS
data    SEGMENT
        <数据、变量在此定义>
data    ENDS

code    SEGMENT
        ASSUME  CS:code, DS:data, ES:data , SS:stack
start:  MOV    AX, data
        MOV    DS, AX
        MOV    ES, AX
        <此处加入你自己的程序段>
        MOV    AH, 4CH
        INT    21H
code    ENDS
        END    start
```

堆栈段

数据段

代码段

1.1 MASM源程序的一般结构

- 程序按段编写，与8086内存按段管理对应；
- 一个源程序由若干段组成，如数据段、代码段、堆栈段等；至少有一个代码段，为确保程序的执行和正确返回，需要堆栈段；
- 每段由数条语句构成，每条语句占一行，指令性语句和指示性语句(伪指令语句）；
- 程序中设有返回DOS的功能，使程序执行完后返回DOS操作系统；
- 程序最后为 END 结束语句，后跟程序启动地址，启动地址是指示程序开始执行的第一条语句；
- 对相应的段寄存器赋值，程序中用到内存操作数时，应按操作数的寻址方式对段寄存器赋值。

```
;汇编语言程序结构例一 movs.asm
aa  SEGMENT         ;数据段1
      str1  DB  'Hello!'
aa  ENDS
bb  SEGMENT         ;数据段2
      str2  DB  6 dup (?)
bb  ENDS
cc   SEGMENT        ;代码段
    ASSUME   CS:cc, DS:aa, ES:bb
    start: CLD
           MOV   AX , aa
           MOV   DS , AX
            LEA    SI ,  str1
           MOV   AX , SEG  str2
           MOV   ES , AX
           MOV   DI ,OFFSET str2
           MOV   CX , 6
           REP   MOVSB
           MOV   AH , 4CH
            INT    21H     ;返回DOS
cc      ENDS
        END  start    ;指示程序结束
```

汇编语言的语句类型

- 硬指令语句

产生目标代码，CPU可执行的语句，主要由CPU指令组成。

- 伪指令语句

不产生目标代码，仅仅在汇编过程中告诉汇编程序如何汇编。
如：源程序有几个段、变量定义等。

- 宏指令语句

指令序列，汇编时用相应的指令序列的目标代码替代。

汇编语句行的构成

- **执行性语句**：即指令性语句(硬指令)包括四部分：

  [标号：] 硬指令助记符 [操作数] [；注释]

  例： AGAN：ADC SUM，AX ；累加

- **说明性语句**：即指示性语句(伪指令),由四部分组成：

  [名字] 伪指令助记符 [操作数] [；注释]

  例： DA_BYTE DB 50H , 50 , 0caH ；变量定义

■ 名字

根据语句功能的不同，名字可用来表示段名、变量名、标号、过程名以及常量名等。**由编程人员自行定义的有意义的字符序列。**

名字用一个标识符表示， 标识符的规定:

① 由字符A~Z ， a~z ， 0~9及符号@、$、下划线_ 等组成，最长31个字符，超出部分忽略；

② 不能用数字开头，以免与十六进制数相混；

③ 不使用汇编程序中的保留字，如指令的助记符等；

④ 对定义的符号不区分大小写。

■ 助记符

助记符可以是指令中的操作码、伪操作中的助记符。

– **指令**，汇编程序将其翻译成机器语言指令。

MOV AX, 100H → B8 00 01

– **伪操作**，汇编程序据其要求的功能进行处理。

data SEGMENT → data与一段值对应

string DB ‘China’ → string与一内存地址对应

■ 操作数

操作数或操作数地址。

- 操作数多于一个时，用逗号分开；
- 操作数可以是常数、寄存器、存储器操作数、标号名、过程名或表达式等。

例：

```
data1   DB   12, 34, 56          ；十进制
MOV     AL, ‘G’                 ；字符
```

注释项

分号引出，用来说明语句或程序的功能，**增强可读性。本身不参与汇编**，汇编程序对分号后的内容不做处理。

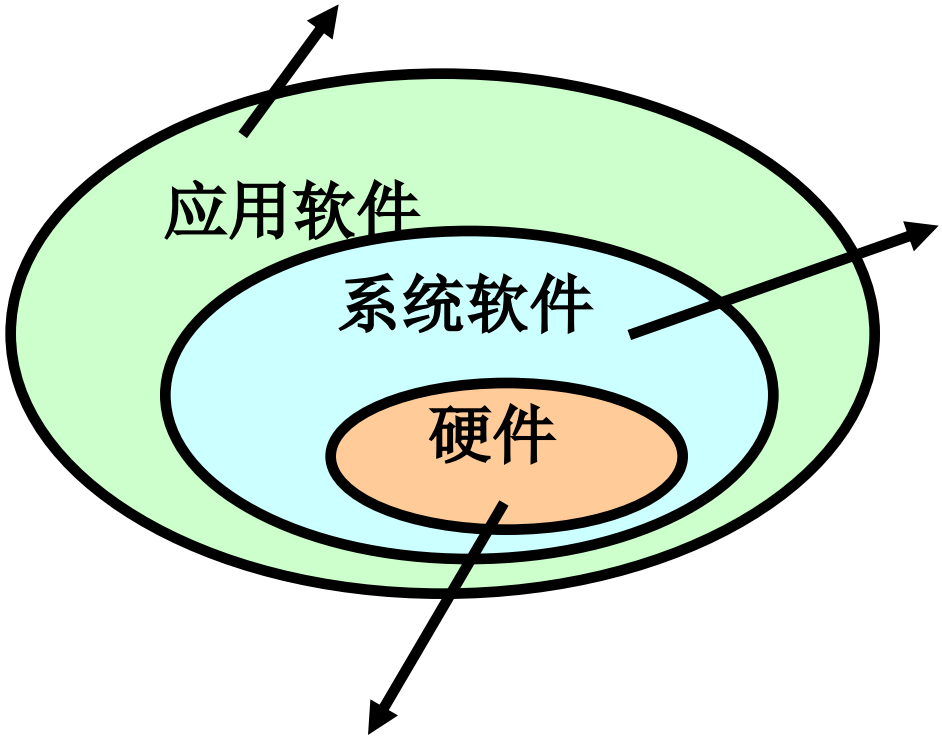
可放在语句最前，暂时注释某语句，调试程序用。

例：

```
……
;   MOV   AH,   2        ;显示提示信息
;   MOV   DL,  'A'
;   INT    21H
……
```

1.2 汇编语言程序的开发过程

用户开发的程序：ABC.exe等



操作系统：DOS系统
编辑器：EDIT.exe
汇编程序：MASM.exe
连接程序：LINK.exe
调试程序：DEBUG.exe

CPU、存储器(ROM、RAM)、I/O接口、输入、输出设备

DOS环境汇编语言开发流程

D:>EDIT ABC.asm

D:>MASM ABC;

有语法错，回EDIT下改该程序

D:>LINK ABC;

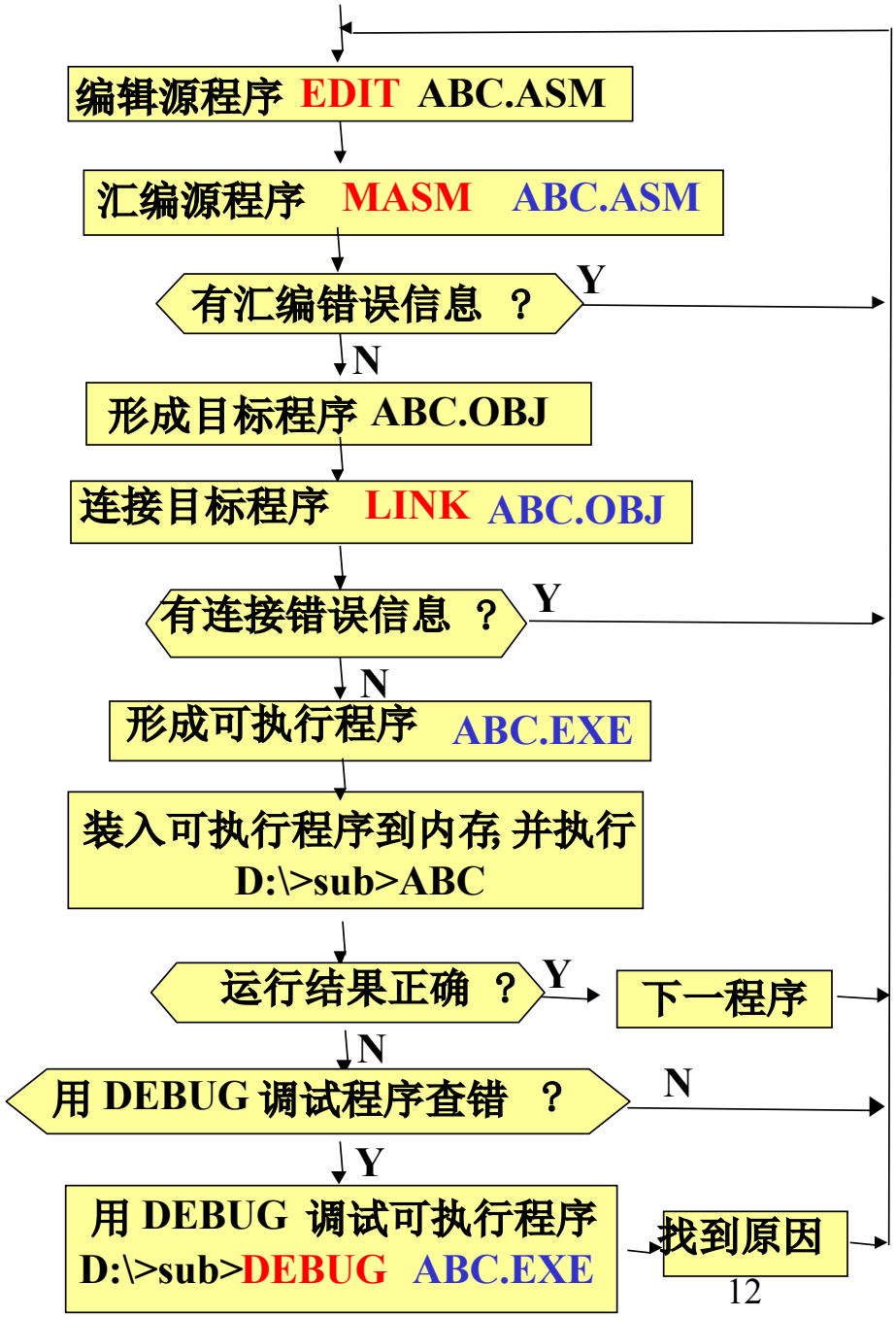
有错，回在EDIT下改程序

D:>ABC

运行结果错，回EDIT下改程序

或在DEBUG下调试，找原因。

D:>DEBUG ABC.exe



2. MASM语言表达式和运算符

2.1 常量

通常在源程序中指令操作数的位置，具有固定值的数值。

- **整数**：必须以数字开头！

  二进制数——以“B” 结尾

  八进制数——以 “Q” 结尾

  十进制数——不带结尾字母，或以“D” 结尾

  十六进制数——以“H” 结尾

  （以A、B、C、D、E、F开头的十六进制数前面加0，与以H结尾的标识符区别。如 MOV AL, 0AH 与 MOV AL, AH ）

- **字符串**

连续的字节单元，每个单元均存放ASCII码。用包括在单引号中的字母表示（单引号本身不算作字符）。如‘abcdef’，对应的数值依次是61H~66H。

■ 符号定义伪指令 EQU 、=

格式：　　符号　EQU　表达式

　　符号　=　表达式

功能：给表达式指定一个等价的符号名。

说明：

① “=”后的表达式只能是常数。

② EQU后的表达式可以是数值、字符串、寄存器名、指令助记符等。

③ EQU不能重复定义，必须解除赋值后，才能重新定义；“=”伪指令可以重复定义，其作用域从定义点到重新定义之前。

■ 解除符号定义伪指令的格式：

格式：　　PURGE　符号1，符号2，……

```
例：    NEW_PORT    EQU     PORT_VAL+1
        PURGE       NEW_PORT
        NEW_PORT    EQU     POTR_VAL+8
```

2.2 变量和标号

C语言：

```
int  i=0,TOTAL=0;
int  array[M]={1,2,3,4,5,6,7,8,9,10};
```

汇编语言：

```
        i    DB   0
    TOTAL    DB   0
    array    DB   1,2,3,4,5,6,7,8,9,10
```

- **变量**——编程人员为内存单元起的名字。有3种属性：
  - **段基址**：该变量所在段的首址（16位）
  - **偏移量**：该变量的段内偏移量或有效地址（16位）
  - **类型**：该变量存放的数据长度，数据类型是字节、字或双字，对应的值为1、2、4
- **标号**——编程人员为目标地址单元起的名字，对应于该指令在代码段中的存放地址。也有3种属性，仅类型与变量不同：
  - **类型**：标号的类型共有远(Far)、近(Near)两种，对应的值是-2、-1

数据定义伪指令格式

（1）基本形式一

[变量名] 伪指令名 表达式1，表达式2，……，表达式n

功能：为程序中的数据分配内存空间，并设置相应内存单元的初始值。

说明：

① 变量名是一个符号地址，凡是可以使用内存操作数的地方，都可以使用变量。变量名表示其后数据的首地址，多个数据构成一个数组。

② 伪指令名主要包括下列几种：

DB：定义字节，后面的每个数据占1个字节

DW：定义字，后面的每个数据占1个字

DD：定义双字，后面的每个数据占2个字

DF：定义双字，后面的每个数据占3个字

DQ：定义双字，后面的每个数据占4个字

DT：定义双字，后面的每个数据占5个字

③ 表达式

- **数值表达式**

```
DA_BYTE   DB   50H , 50 , 0caH
DA_WORD   DW   0a3f1H , 498dH
A1            DD   12345H
```

- **不带引号的？，表示只保留内存空间，初始值未定义**

```
DA_B    DB   ? , ?          ；要求分配两个字节单元
DA_W   DW   ? , ?          ；要求分配两个字单元
```

- **字符串表达式**

数据项可以写成字符串形式，只能用DB、DW、DD定义，而且DW、DD语句定义的串只允许是两个字符。定义多于两个以上字符的字符串时，只能使用DB伪指令，不能使用DW和DD等伪指令。

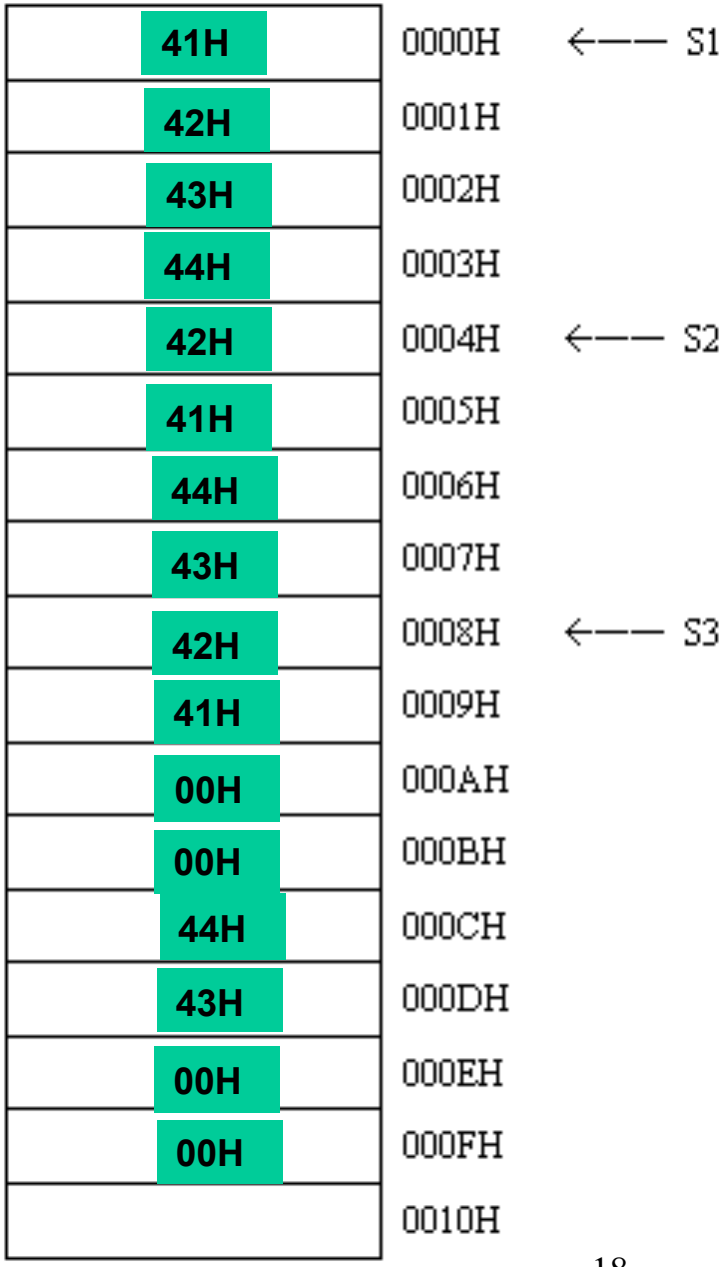
例：数据段定义如下

S1　　DB　‘ABCD’

S2　　DW　‘AB’, ‘ CD’

S3　　DD　‘AB’, ‘ CD’

存储器中存放情况如图所示。



（2）基本形式二：

[变量名]　伪指令名　表达式 1　DUP(表达式 2)

功能：用于定义重复的数据或分配一数据块空间，并设置相应内存单元的初始值。

```
例：D_B1  DB   20H  DUP(?)          ；保留20H个字节
    D_B2  DB   10H  DUP('ABCD')    ；字符串 'ABCD' 重复10H次
    D_W1  DW   10H  DUP(4)         ；字4重复10H次
```

注意：

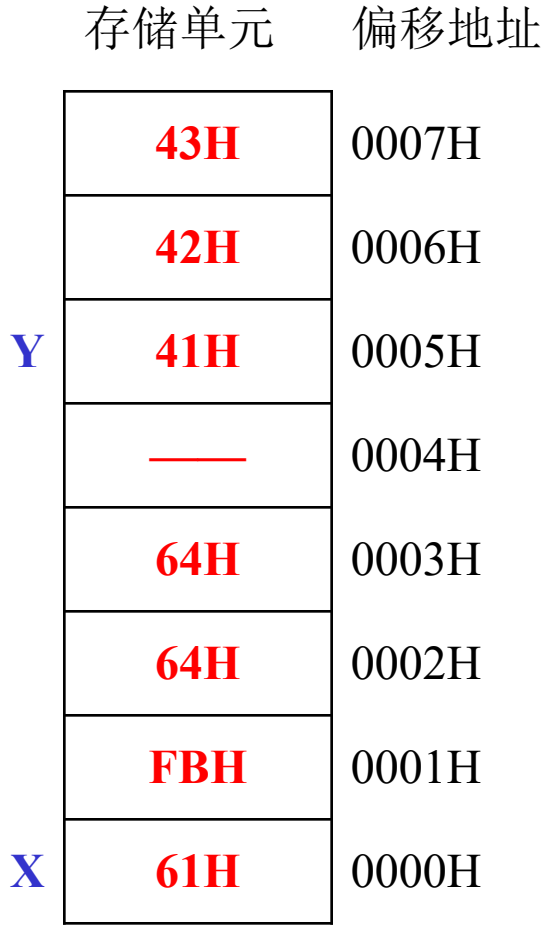
变量可以定义在程序中的任何段（包括代码段），但一般定义在数据段。

例：数据段定义如下

```
DATA  SEGMENT
    X     DB    'a', -5
          DB   2  DUP(100), ?
    Y     DB    'ABC'
DATA  ENDS
```

执行下列指令序列后，该存储区中各单元的值有何变化？

```
MOV     AL, X
DEC     X+1
MOV     Y, AL
```



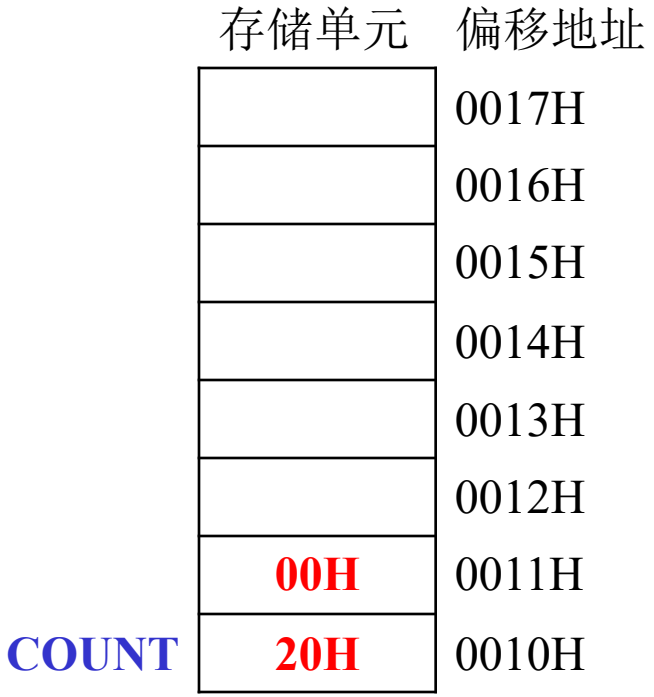
例：

```
DATA  SEGMENT
      COUNT      DW      8000H,  ?, 'AB'
      MAXINT     EQU     64H
      NUMBER     DW      MAXINT
      ARRAY      DW      MAXINT DUP(0)
DATA  ENDS
```

	存储单元	偏移地址
	00H	000BH
	00H	000AH
	00H	0009H
ARRAY	00H	0008H
	00H	0007H
NUMBER	64H	0006H
	41H	0005H
	42H	0004H
	——	0003H
	——	0002H
	80H	0001H
COUNT	00H	0000H

例：

```
DATA  SEGMENT
    ORG          0010H
     WNUM     EQU   5678H
     COUNT    DW     20H
DATA  ENDS
```



程序：

```
MOV    AX, [BX+SI+WNUM]
MOV    AX, COUNT
MOV    AX, [SI+COUNT]
LEA     BX, COUNT
MOV    BX, OFFSET COUNT
```

等价于：

```
MOV     AX, [BX+SI+5678H]
MOV     AX, [0010H]
MOV     AX, [SI+10H] / COUNT[SI]
LEA     BX, [0010H]
MOV     BX, 0010H
```

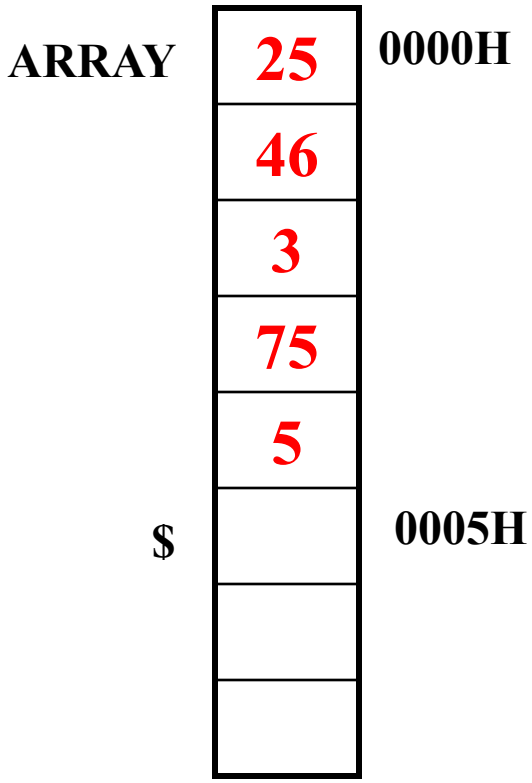
■ 地址计数器 $

表示位置计数器的当前值，“$”可以在数值表达式中使用，它在表达式里的值是程序下一个所能分配的存储单元的偏移地址。

例：

```
SORTD  SEGMENT
    ARRAY    DB    25，46，3，75，5
    COUNT    EQU   $ - ARRAY
SORTD  ENDS
```

- ✓ ARRAY的偏移地址值为0000H
- ✓ $ 的偏移地址值为0005H
- ✓ COUNT = 0005H – 0000H = 5



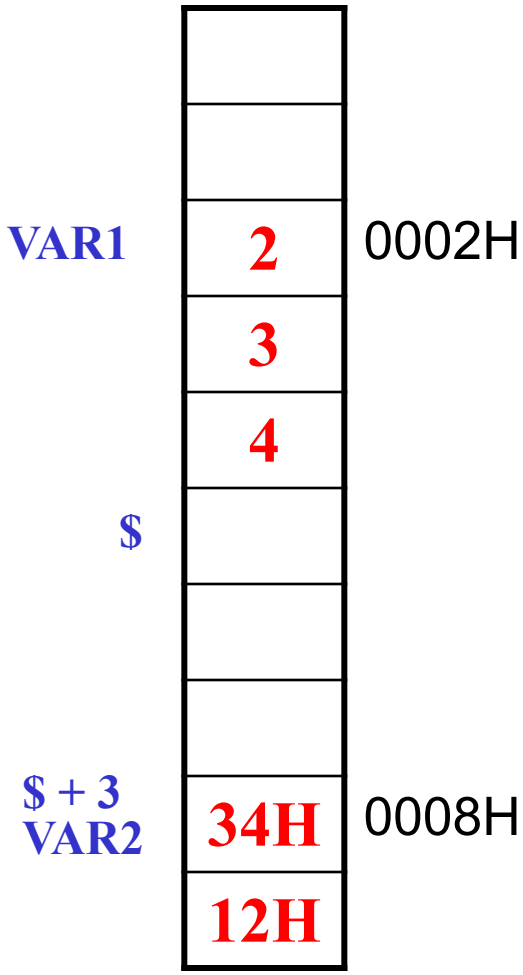
■ 定位伪指令

ORG 起始位置设定伪指令

指出源程序或数据块的起点。利用ORG伪指令可以改变位置计数器的值。

下一个地址变量的偏移地址值由ORG确定

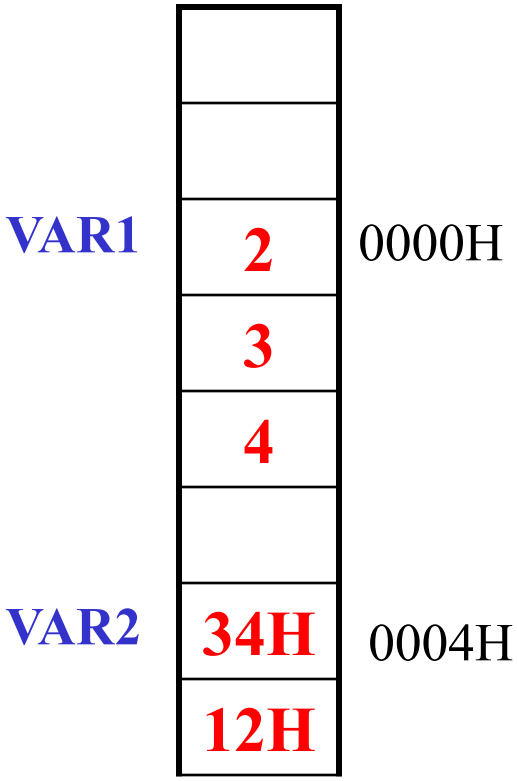
```
DATA   SEGMENT
       ORG    0002H
       VAR1   DB   2，3，4
       ORG    $ + 3
       VAR2   DW   1234H
DATA  ENDS
```



EVEN 偶数地址指向设定伪指令

设置当前偏移地址指针指向偶数地址，若已指向偶数地址，则不变。

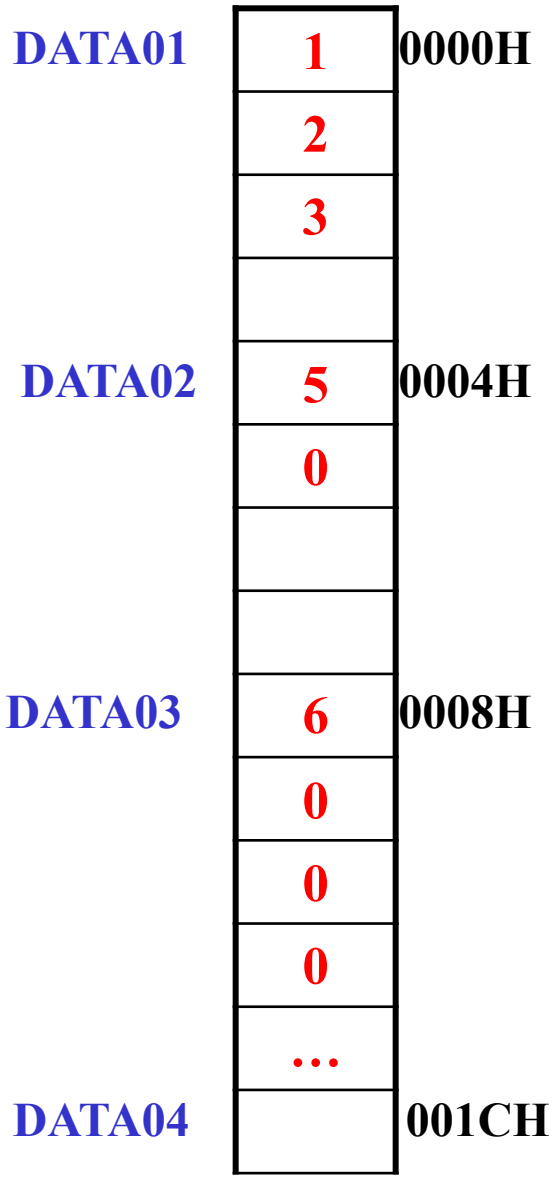
```
DATA  SEGMENT
   VAR1   DB   2，3，4
   EVEN
   VAR2   DW   1234H
DATA  ENDS
```



ALIGN n 地址指向n的整数倍地址伪指令

若当前偏移地址指针已指向n的整数倍地址，则不作调整；否则将地址指针指向下一个n的整数倍地址。

```
DATA  SEGMENT
   DATA01   DB   1，2，3
   EVEN
   DATA02   DW   5
   ALIGN    4
   DATA03   DD   6
   ORG      $+10H
   DATA04   DB   ‘ABC’
DATA  ENDS
```



2.3 表达式

■ 常用运算符

– **算术运算符**　+、-、*、/、MOD

例：　32　MOD　5　　；结果为2

– **逻辑运算符**

AND	24H AND 0FH = 04H
OR	24H OR 0FH = 2FH
XOR	24H XOR 0FH = 2BH
NOT	NOT 24H = 0DBH

– **关系运算符**

EQ （等于）	若PP=25，则25 EQ PP = 0FFFFH
NE （不等于）	25 NE PP = 0
LT （小于）	25 LT 26 = 0FFFFH
LE （小于等于）	25 LE PP = 0FFFFH
GT （大于）	26 GT PP = 0FFFFH
GE （大于等于）	24 GE PP = 0

– **移位运算符**　　SHL，SHR

例:

```
MOV   AL, 21H   SHL   2        ；结果为84H
MOV   AL, 0101B  SHL(2*2)    ;等价于MOV  AL,  01010000B
```

– **高低运算符**　　HIGH，LOW，HIGHWORD，LOWWORD

例:

```
MOV   AH, HIGH  8765H      ;等价于MOV  AH, 87H

DD_VALUE   EQU   0FFFF1234H
MOV  AX,  LOWWORD  DD_VALUE
                           ;等价于MOV  AX, 1234H
```

■ 地址运算符

– SEG （求段地址）， 格式： SEG ＜符号名＞

– OFFSET（求偏移地址），格式： OFFSET ＜符号名＞

– TYPE （求符号名类型值），格式：TYPE ＜符号名＞

– LENGTH（求符号名分配的项数）， 格式： LENGTH ＜符号名＞

数据项必须是用重复格式DUP（ ）定义，其他情况则回送1

类型	byte	word	dword	NEAR	FAR
类型值	1	2	4	-1 (0FFFH)	-2 (0FFEH)

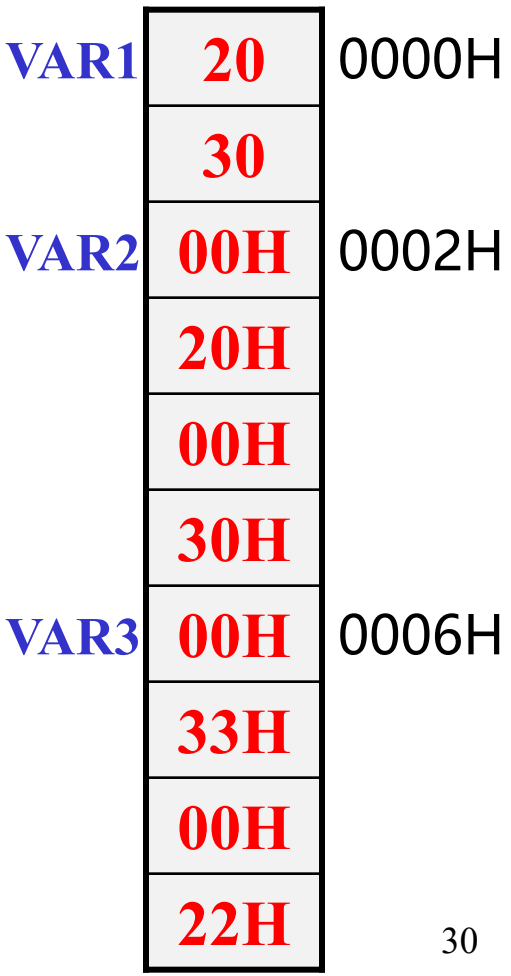
– SIZE （求为符号名分配的字节数） 格式： SIZE ＜符号名＞

回送分配给该符号名的字节数

例：物理地址 02000H 开始存数据，即段地址为 0200H，偏移地址从 0000H 开始，定义数据段 DATA 如下：

```
DATA  SEGMENT
 VAR1  DB  20，30
 VAR2  DW  2000H，3000H
 VAR3  DD  22003300H
DATA  ENDS

MOV  BX，SEG  VAR1     ;   (BX) = 0200H
MOV  BX，SEG  VAR2     ;   (BX) = 0200H
MOV  BX，SEG  VAR3     ;   (BX) = 0200H
MOV  BX，OFFSET  VAR1  ;   (BX) = 0000H
MOV  BX，OFFSET  VAR2  ;   (BX) = 0002H
MOV  BX，OFFSET  VAR3  ;   (BX) = 0006H
MOV  BL，TYPE  VAR1    ;   (BL) = 1
MOV  BL，TYPE  VAR2    ;   (BL) = 2
MOV  BL，TYPE  VAR3    ;   (BL) = 4
```



LENGTH 和 SIZE：只对DUP指令有效，否则为1

- 数据段定义

```
DATA  SEGMENT
      VAR1    DB    100 DUP（?）
      VAR2    DW    100 DUP（?）
      VAR3    DB     ? , ? , ? , ?
DATA  ENDS
```

- 指令功能

```
MOV  BX，LENGTH  VAR1          ;   (BX) = 100
MOV  BX，LENGTH  VAR2          ;   (BX) = 100
MOV  CX，LENGTH  VAR3          ;   (CX) = 1
MOV  BX，SIZE  VAR1            ;   (BX) = 100
MOV  BX，SIZE  VAR2            ;   (BX) = 200
MOV  DX，SIZE  VAR3            ;   (DX) = 1
```

■ 地址运算符

– 类型属性操作符 PTR

```
MOV  BYTE PTR[1000H], 0
MOV  WORD PTR[1000H], 0
```

– 创建当前地址操作符 THIS

```
B_VAR      EQU  THIS  BYTE
W_VAR      DW  10 DUP(0)
F_JUMP     EQU  THIS  FAR
N_JUMP: MOV  AX, W_VAR
```

– LABEL 功能等同 THIS

– 距离属性操作符 SHORT

```
JMP  SHORT  N_JUMP
```

■ 汇编表达式

- 数字表达式　　MOV　　DX，(6*A-B)/2
- 地址表达式　　MOV　　AX，ES:[BX+SI+1000H]

■ 运算符的优先级

优先级	运算符
1	( ),< >,[ ], . ,LENGTH,SIZE,WIDTH,MASK
2	PTR, OFFSET, SEG, TYPE, THIS
3	HIGH, LOW
4	*, /, MOD, SHL, SHR
5	+, -
6	EQ, NE, GT, LT, GE, LE
7	NOT
8	AND
9	OR, XOR
10	SHORT

书例4.3：屏幕上显示字符串1357？？？？？？？？？？

■ 计算机中字符、数码转换的处理

– 计算机处理字符时，常用的字符编码是ASCII码，数字和字母的ASCII码是一个有序序列

数字0~9 ： 30H ~ 39H

大写字母A~Z ： 41H ~ 5AH

小写字母a~z ： 61H ~ 7AH

– 计算机处理信息时，其对象都是二进制数，外设(显示器、打印机、键盘等) 用ASCII码与CPU进行信息传送。

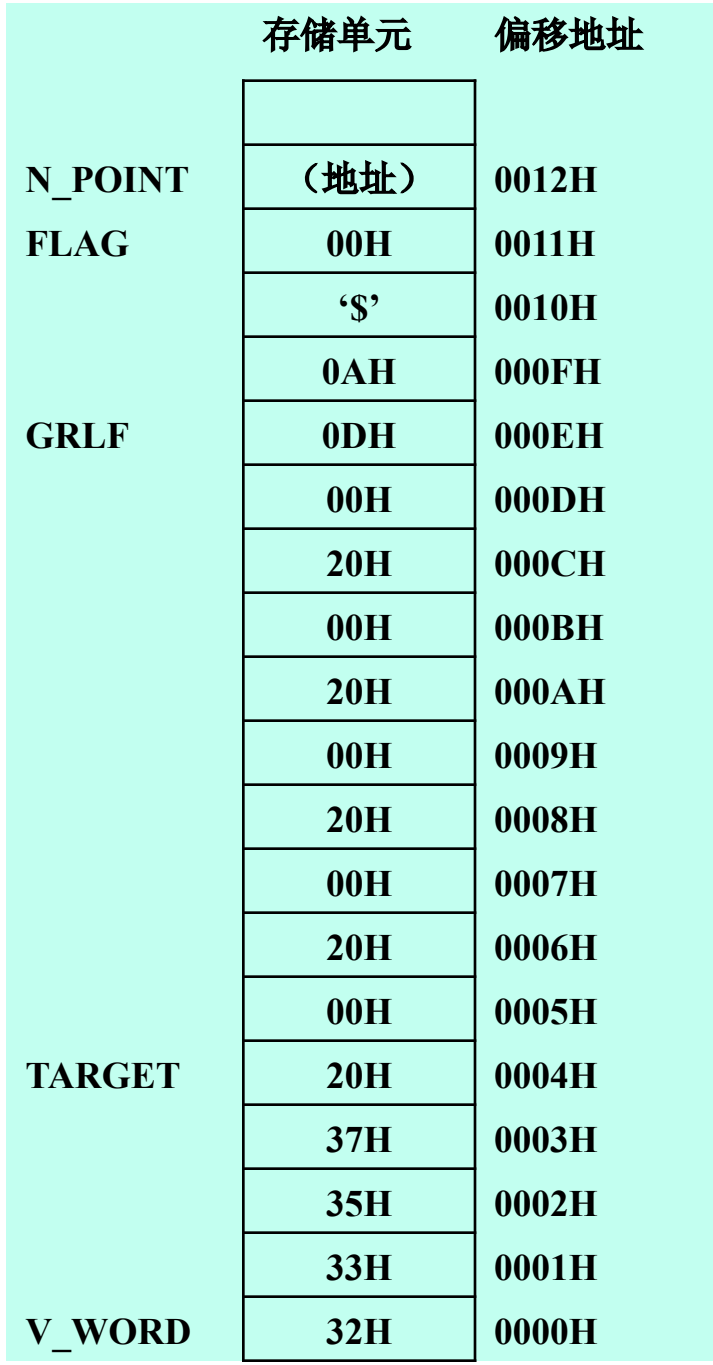
• 键盘上按下某一字符键 (如’9’)，键盘接口向键盘缓冲区送去的是该字符的ASCII码(如39H)，不是送数字09H。

• 文本方式下，要在显示器上显示某一字符(如’A’)， 须将该字符的ASCII码(如41H)送显示缓冲区，不是送数字0AH。

书例4.3要求在显示器上显示字符串‘1357??????????’，则需将该字符串每个字符的ASCII码值存入存储单元，然后调用显示字符串的中断，在显示器上显示字符串，其代码如下所示：

```
.MODEL  SMALL
.STACK
.DATA
    V_BYTE    EQU    THIS BYTE
    V_WORD    DW     3332H, 3735H
    TARGET    DW      5 DUP (20H)
    GRLF      DB     0DH, 0AH, '$'
    FLAG      DB     0
    N_POINT   DW     OFFSET  S_LABEL
```

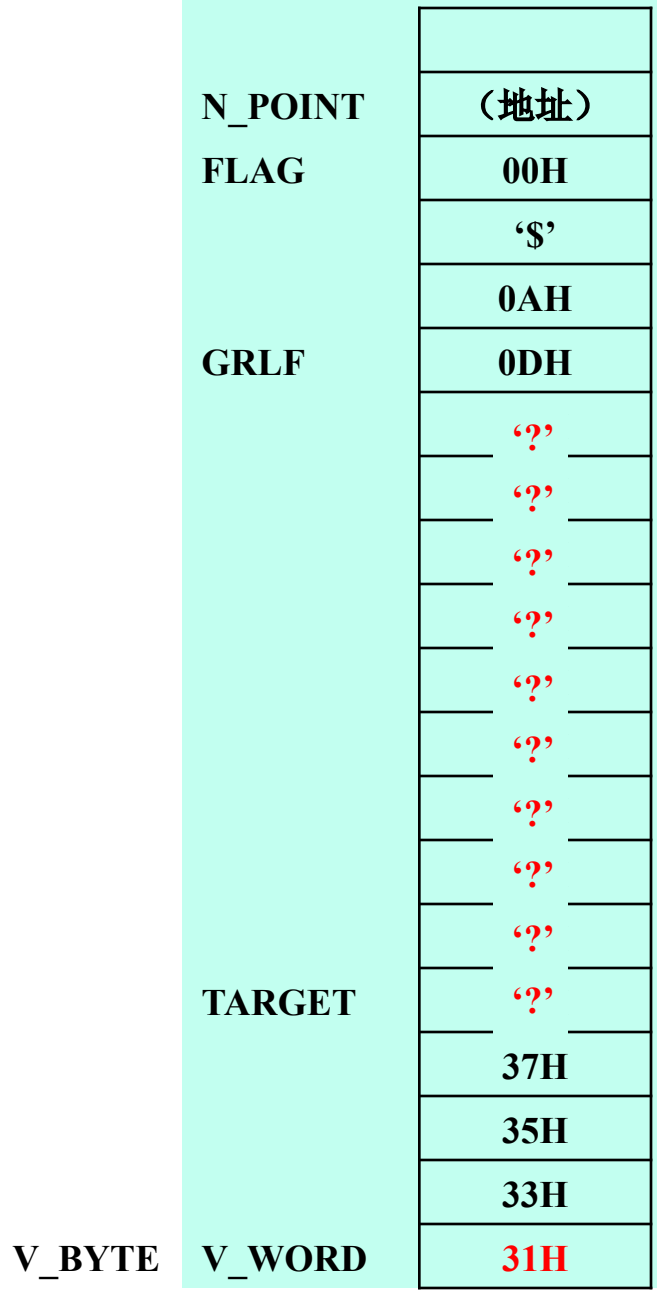
V_BYTE



```
.CODE
.STARTUP
        MOV     AL,  BYTE PTR V_WORD
        DEC     AL
        MOV     V_BYTE, AL
N_LABEL: CMP    FLAG, 1
        JZ      S_LABEL
        INC     FLAG
        JMP     SHORT  N_LABEL
S_LABEL: CMP    FLAG, 2
        JZ      NEXT
        INC     FLAG
        JMP     N_POINT
  NEXT:MOV      AX, TYPE  V_WORD
        MOV     CX, LENGTH  TARGET
        MOV     SI, OFFSET  TARTGET
W_AGAIN:MOV     [SI], AX
        INC     SI
        INC     SI
        LOOP    W_AGAIN
```

	存储单元	偏移地址
N_POINT	（地址）	0012H
FLAG	00H	0011H
	‘$’	0010H
	0AH	000FH
GRLF	0DH	000EH
	00H	000DH
	02H	000CH
	00H	000BH
	02H	000AH
	00H	0009H
	02H	0008H
	00H	0007H
	02H	0006H
	00H	0005H
TARGET	02H	0004H
	37H	0003H
	35H	0002H
	33H	0001H
V_BYTE V_WORD	31H	0000H

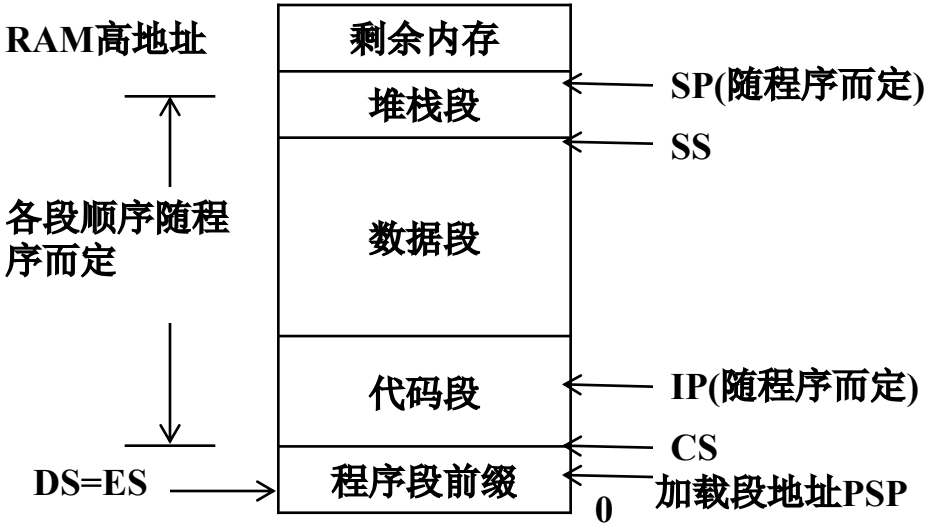
```
        MOV    CX, SIZE  TARGET
        MOV     AL, '?'
        MOV     DI, OFFSET  TARGET
B_AGAIN:MOV     [DI], AL
        INC      DI
        LOOP    B_AGAIN
        MOV     DX, OFFSET  V_WORD
        MOV     AH, 9
        INT      21H
```



3. 程序段的定义和属性

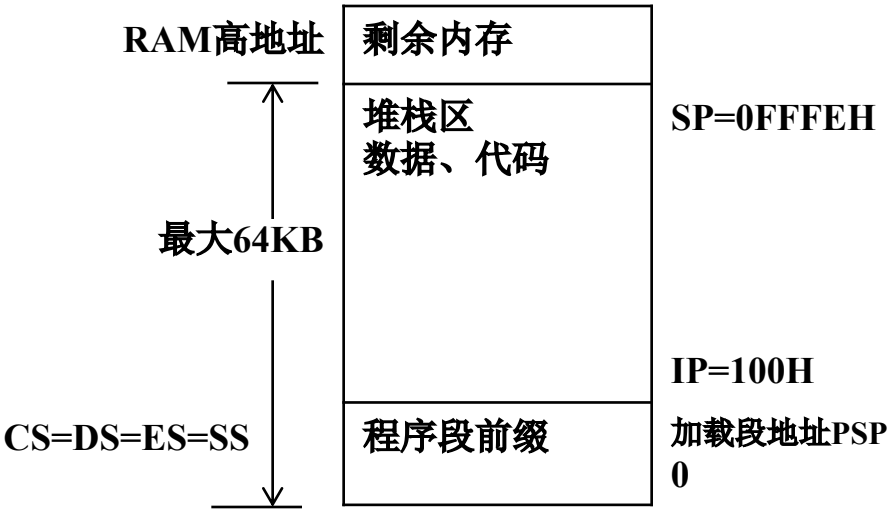
■ DOS的程序结构

– EXE程序



程序段前缀：256字节

– COM程序



3.1 简约段定义伪指令

- 程序开始时使用 .MODEL SMALL伪指令
- 段定义可简化为：

```
.MODEL  SMALL          ；定义程序的存储模式
.STACK                 ；定义堆栈段
.DATA                  ；定义数据段
  …                    ；数据定义
.CODE                  ；定义代码段
.STARTUP               ；程序起始点，建立DS、SS内容
  …                    ；程序代码
.EXIT 0                ；程序结束点，返回DOS
  …                    ；子程序代码
END                    ；汇编结束
```

存储模式

MODEL　存储模式 [,语言类型][,操作系统类型][,堆栈选项]

存储模式决定一个程序的规模，也决定子程序调用、指令转移和数据访问的默认属性

- **TINY**
- **SMALL** 一般的程序都采用的模式。一个程序至多只有一个代码段和一个数据段（数据段、堆栈段和附加段的总和，共用一个基地址），每段大小不大于64KB。所以该模式下的程序最多128KB。
- **COMPACT**
- **MEDIUM**
- **LARGE**
- **HUGE**
- **FLAT** 创建32位程序。

程序开始伪指令

.STARTUP

功能：按照给定的CPU类型，根据MODEL语句选择的存储模式等，产生程序开始执行的代码，同时指定程序开始执行的起始地址。DOS环境下，初始化DS值，调整SS和SP。

例如：SMALL模式下，对应8086CPU，STARTUP语句将被汇编成如下启动地址(不调整SS：SP)

```
START: MOV     AX, @DATA
       MOV     DS, AX
```

START是一个标号，标记汇编程序启动地址。

■ 程序终止伪指令

.EXIT [返回数码]

功能：终止程序执行，返回操作系统。可选参数是一个返回的数码，通常用0宝石没有错误。

例如：

.EXIT 0 对应的代码是

```
MOV     AX, 4C00H
INT     21H
```

■ 汇编结束伪指令

END [标号]

功能：汇编程序MASM到此结束汇编过程。

程序中所有有效语句应放在END语句之前，汇编程序对END之后的语句不进行处理。

3.2 完整段定义伪指令

■ 段定义伪指令

段名　SEGMENT　[定位] [组合] [段字][‘类别’]

…

段名　ENDS

功能：定义一个逻辑段的开始和结束。

（1）定位：用来规定段起始边界的要求，默认为 PARA

- PARA：段起始地址的最低4位必须为0

（2）组合：表示本段与其他段的关系，为连接程序使用，默认为PRIVATE

- PRIVATE：与其他段不发生关系，每段都有自己的基地址
- AT表达式：把本段装在表达式值所指定的段地址上（值为16位；不能指定代码段）

（3）段字：支持32位段设置

（4）类别名：为连接程序使用，把类别名相同的段放在连续的存储区间（一般有‘STACK’、‘CODE’、‘DATA’）。

■ 指定段寄存器伪指令

汇编程序必须知道程序的段结构，并知道在指令执行时哪个段由哪个段寄存器指定，这个信息在代码段的开始由ASSEMU语句提供。

ASSUME 段寄存器：段名[，段寄存器：段名，……]

当程序运行时，由于DOS的装入程序负责把CS初始化成正确的代码段地址，SS初始化为正确的堆栈段地址，因此用户在程序中就不必设置。但是，在装入程序中DS/ES寄存器由于被用作其它用途，因此，在用户程序中必须用两条指令对DS/ES进行初始化，以装入用户的数据段地址。

例如：CODE SEGMENT
```
        ASSUME  CS：CODE，DS：DATA，SS：STACK
        MOV   AX，DATA
        MOV   DS，AX
CODE ENDS
```

书例4.2：屏幕上显示字符串HELLO, EVERYBODY!

```
STACK  SEGMENT  STACK
          DW       512 DUP(?)
STACK  ENDS
DATA   SENGMENT
     STRING      DB ‘Hello，Everybody!’,0DH, 0AH, ‘$’
DATA   ENDS
CODE   SEGMENT  ‘CODE’
          ASSUME
          CS:CODE,DS:DATA,SS:STACK
START: MOV      AX, DATA
          MOV      DS, AX
          MOV      DX, OFFSET  STRING
          MOV      AH, 9
          INT      21H
          MOV      AX, 4C00H
          INT      21H
CODE   ENDS
END       START
```

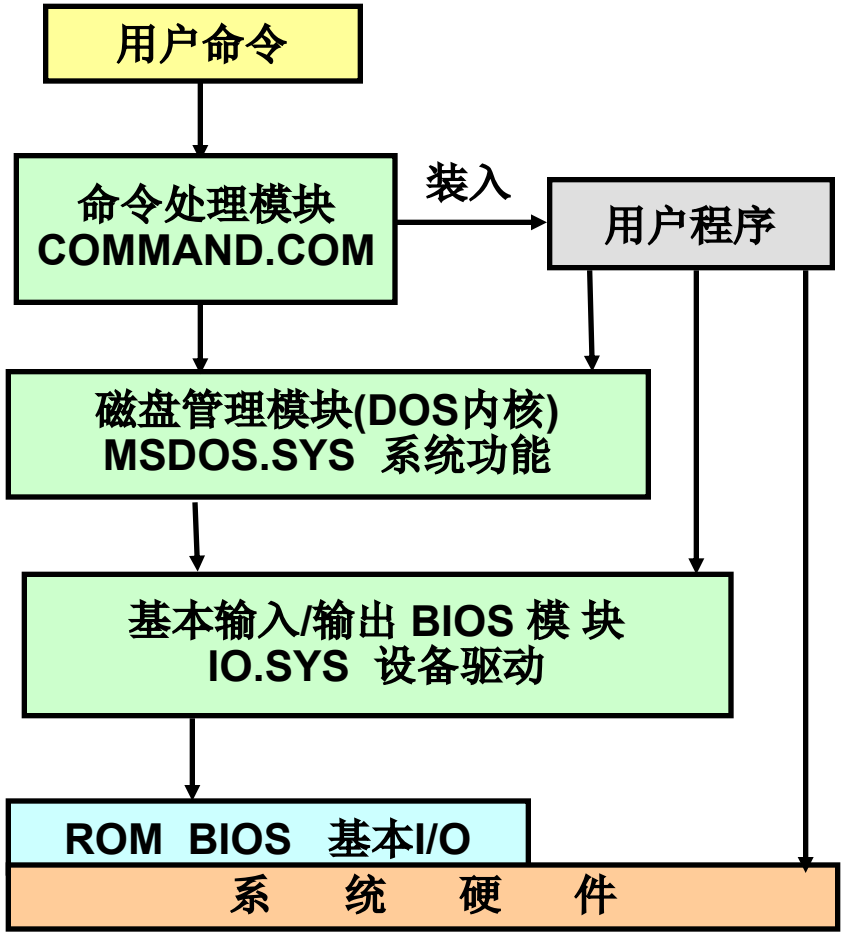
4. 常用的系统功能调用

■ DOS系统调用概述

DOS系统调用提供许多子程序供用户调用，包括：

- 磁盘管理
- 内存管理
- 基本输入输出管理
- …

使用中断指令 INT 21H完成相应子程序的调用，书表4.5系统功能调用说明。



4.1 DOS功能调用

- 汇编程序的运行结果，或是保留在寄存器中，或是保留在存储器中，不能方便直观的看到
- 使用INT 21H完成子程序调用需要入口参数及出口参数
- 基本I/O功能调用见书表4.6

功能号	功能描述	使用说明
01H	键盘输入，屏幕显示	入口参数：无 出口参数：AL存放输入字符
03H	异步通信输入	入口参数：无 出口参数：AL存放异步通信口接收的数据
04H	异步通信输出	入口参数：DL存放要输出的数据 出口参数：无
05H	打印输出	入口参数：DL存放待打印的字符 出口参数：无
09H	输出字符串	入口参数：DS:DX指向内存中一个以$结束的字符串 出口参数：无
0AH	键盘接收字符串，存内存缓冲区	入口参数：DS:DX指向输出缓冲区 出口参数：DS:DX指向输出缓冲区
2AH	读取日期	入口参数：无 出口参数：CX：年；DH：月；DL：日
2BH	设置日期	入口参数：CX：年；DH：月；DL：日 出口参数：AH=00，设置成功；AH=0FFH，无效

■ DOS功能调用方法

INT 21H

通常按照如下4个步骤进行：

(1) 在AH寄存器中设置系统功能调用号

(2) 在指定寄存器中设置入口参数

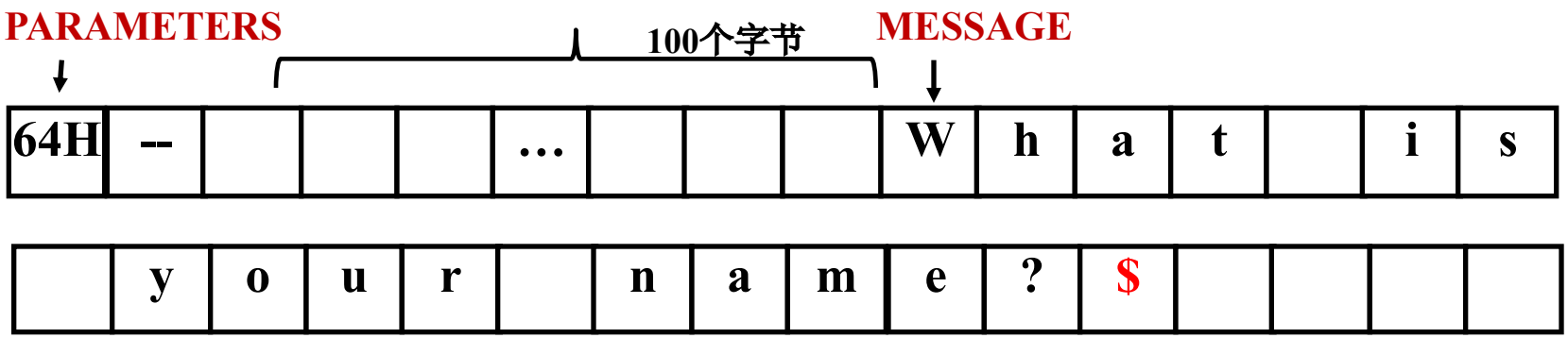
(3) 执行指令INT 21H实现中断服务程序的功能调用

(4) 根据出口参数分析功能调用执行情况

例：利用DOS调用在屏幕上显示提示信息，然后接收键盘输入信息，并存入缓冲区。

```
PARAMETERS    DB      100                          ；数据定义
              DB      ?
              DB      100 DUP(?)
MESSAGE       DB      'What is your name?'
              DB      '$'

DISP:    MOV  DX, OFFSET MESSAGE                   ；屏幕显示
         MOV  AH, 09H
         INT  21H
KEY:     MOV  DX, OFFSET PARAMETERS                ；键盘输入
         MOV  AH, 0AH
         INT  21H
```



例　利用键盘输入的字符产生分支：

```
        MOV   AH, 1          ;等待从键盘输入
        INT    21H
        CMP   AL, 'Y'        ;是’Y’?
        JZ     yes
        CMP   AL, 'y'        ;是’y’?
        JZ     yes
no:     . . .
        . . .
        JMP    exit
yes:    . . .
          . . .
exit:   . . .
```

4.2 BIOS中断调用

BIOS是固化在ROM中的一组I/O设备驱动程序，为系统各主要部件提供设备级的控制，负责管理系统内的输入输出设备，直接为DOS操作系统和应用程序提供底层设备驱动服务。

大多数以软件中断方式调用，少数以硬件中断调用。

调用格式：

INT n ；n=05H~1FH

常用BIOS服务功能见书P144，表4.7

BIOS服务	功能号	功　能
打印屏幕服务	05H	当前视频内容送默认打印机
视频服务	10H	为显示适配器提供I/O支持
硬盘服务	13H	提供硬盘的读、写、格式化、初始化、诊断
串行通信服务	14H	为串行适配器提供字符输入输出
键盘服务	16H	为键盘提供I/O支持
并行打印机服务	17H	为并行打印机提供I/O支持

■ BIOS中断调用及实现

■ 视频服务

视频服务由INT 10H 启动，一般的步骤是：

① AH选择视频服务功能，AL或BL选择子功能

② AL存放待显示字符或像素值

③ 功能调用时，保存BX, CX, DX及段寄存器值

④ X坐标在CX存放（图形显示）；在DL存放 （正文显示）

⑤ BH存放显示页（0开始计数）

例：光标移到3行14列

```
MOV     AH, 02H
MOV     DH, 3
MOV     DL, 14
INT     10H
```

■ 键盘服务

例：16H键盘中断处理

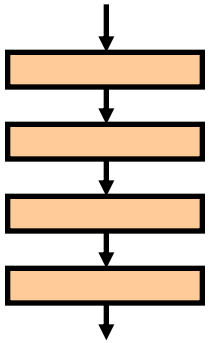
操作系统和应用程序调用INT 16H指令，对键盘进行检测和设置，并读取键盘缓冲区中的键码，有4种功能。

功能号	功 能	返回参数
AH=0	键盘缓冲区读取1个字符的键码送到AX	AH=系统扫描码 AL=字符的ASCII或0
AH=1	检测键盘缓冲区中是否有键码	ZF=0则有键码并读入AX ZF=1则无键码
AH=2	读取特殊键的状态标志	AL中为读取的状态标志
AH=3	设置键盘速率和延迟时间 BL=速率，BH=延迟时间	无

5. 汇编程序结构

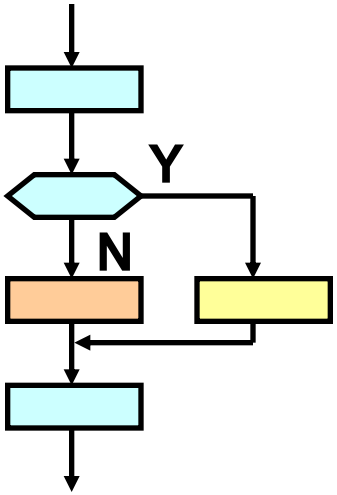
■ 汇编程序结构

顺序



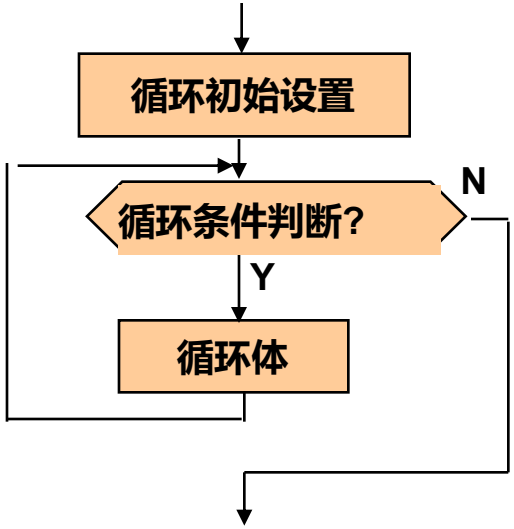
```
MOV  AX，DATA
MOV  DS，AX
MOV  CL，M
MOV  AX，0
MOV  SI，AX
```

分支




```
…
CMP  AL, BL
JG   great
     AL≤BL处理
JMP  exit
great:  AL>BL处理
exit:   …
```

循环




```
     MOV  CL，M
     MOV  AX，0
     MOV  SI，AX
XUN：ADD  AX，ARRAY[SI]
     ADD  SI，2
     LOOP XUN
```

■ 顺序结构

例：查表的方法将一位十六进制数（0~9，A~F）转换成相应的ASCII码。

```
DATA     SEGMENT
   TABLE  DB   30H,31H,32H,33H,34H,35H,36H,37H,38H,39H,41H,42H
          DB   43H,44H,45H,46H
    HEX    DB   4
    ASC    DB   ?
DATA     ENDS
STACK1   SEGMENT
         DW   20H DUP(0)
STACK1   ENDS
COSEG    SEGMENT
         ASSUME  CS:COSEG,DS:DATA,SS:STACK1
START:   MOV   AX, DATA
         MOV   DS, AX
         LEA   BX, TABLE
         XOR   AH, AH
         MOV   AL, HEX
         ADD   BX, AX
         MOV   AL, [BX]
         MOV   ASC, AL
COSEG    ENDS
         END SATRT
```

■ 分支结构

(1）无条件转移指令

JMP reg/m16

– 转移距离

SHORT	相对于当前指令地址 -128~127
NEAR	段内转移，CS不变，指令给出IP
FAR	段间转移，CS、IP均需给出

– 寻址方式

- 直接寻址
- 间接寻址

例：
```
JMP  CX
JMP  2000H
JMP  1000H：2000H
JMP  DWORD PTR[SI]
```

例：
```
     MOV  AX, 12
     MOV  CX, 5
QQ:  ADD  AX, CX
     JMP  QQ
```

（2）条件转移指令

Ⅰ）单个条件标志

A) **JZ(或JE)** 如果ZF=1,则转移

B) **JNZ(或JNE)** 如果ZF=0,则转移

C) **JS** 如果SF=1,则转移

D) **JNS** 如果SF=0,则转移

E) **JO** 如果OF=1,则转移

F) **JNO** 如果OF=0,则转移

G) **JP** 如果PF=1,则转移

H) **JNP** 如果PF=0,则转移

II）用于二个无符号数的比较

A）JB（JNAE、JC） A<B

B）JNB（JAE、JNC） A ≥ B

C）JBE（JNA） A ≤ B

D）JNBE（JA） A >B

III）用于二个有符号数的比较

A）JL（JNGE） A<B

B）JNL（JGE） A ≥ B

C）JLE（JNG） A ≤ B

D）JNLE（JG） A >B

IV) JCXZ

JCXZ LABEL 如果CX=0,则转移

分支程序例：

注意：分支的开始点和结束点

例4.22：计算AX中符号数绝对值

```
        CMP     AX, 0
        JGE     NONEG
        NEG     AX
NONEG:  MOV     RESULT, AX
```

```
        CMP     AX, 0
        JL      YESNEG
        JMP     NONEG
YESNEG: NEG     AX
NONEG:  MOV     RESULT, AX
```

例4.23：显示BX最高位

```
        SHL     BX, 1
        JC      ONE
        MOV     DL, '0'
        JMP     TWO
ONE:    MOV     DL, '1'
TWO:    MOV     AH, 2
        INT     21H
```

```
        MOV     DL, '0'
        SHL     BX, 1
        JNC     TWO
        MOV     DL, '1'
TWO:    MOV     AH, 2
        INT     21H
```

分支程序例:

```
GMAX:  MOV   BX, 2000H
       MOV   AL, BYTE PTR [BX]
       MOV   CX, 14H
P1:    CMP   AL, BYTE PTR [BX]
       JAE   P2
       MOV   AL, BYTE PTR [BX]
P2:    INC   BX
       DEC   CX
       JNZ   P1
       MOV   BX, 3000H
       MOV   BYTE PTR [BX], AL
```

该程序段功能？

例4.24：判断AX^2+BX+C=-是否有实根，若有实根，则将字节变量TAG置1，否则置0。假设 A、B、C均为字节变量，数据范围为-128~+127。

```
.MODEL  SMALL
.STACK
.DATA
    _A          DB      ?
    _B          DB      ?
    _C          DB      ?
    TAG         DB      ?
.CODE
.STARTUP
        MOV     AL, _B
        IMUL    AL
        MOV     BX, AX
        MOV     AL, _A
        IMUL    _C
        MOV     CX, 4
        IMUL    CX
        CMP     BX, AX
        JGE     YES
        MOV     TAG, 0
        JMP     DONE
YES:    MOV     TAG, 1
DONE:   .EXIT 0
        END
```

例4.25：根据键盘输入的1~8数字转向8个不同的处理程序段。

```
.MODEL  SMALL
.STACK
.DATA
    MSG     DB   'Input number(1~8):', 0DH, 0AH, $
    MSG1    DB   'Chapter1: …', 0DH, 0AH, $
    MSG2    DB   'Chapter2: …', 0DH, 0AH, $
    MSG3    DB   'Chapter3: …', 0DH, 0AH, $
    MSG4    DB   'Chapter4: …', 0DH, 0AH, $
    MSG5    DB   'Chapter5: …', 0DH, 0AH, $
    MSG6    DB   'Chapter6: …', 0DH, 0AH, $
    MSG7    DB   'Chapter7: …', 0DH, 0AH, $
    MSG8    DB   'Chapter8: …', 0DH, 0AH, $
  TABLE   DW   DISP1,DISP2,DISP3,DISP4,DISP5,
               DISP6,DISP7,DISP8
```

```
.CODE
.STARTUP
START1:MOV      DX,  OFFSET MSG
        MOV     AH, 9
        INT     21H
        MOV     AH,  1
        INT     21H
        CMP     AL,  '1'
        JB      START1
        CMP     AL,  '8'
        JA      START1
        AND     AX, 000FH
        DEC     AX
        SHL     AX, 1
        MOV     BX, AX
        JMP     TABLE[BX]
START2:MOV      AH, 9
        INT     21H
.EXIT 0
DISP1:  MOV     DX,  OFFSET  MSG1
        JMP     START2
DISP2:  MOV     DX,  OFFSET  MSG2
        JMP     START2
DISP3:  MOV     DX,  OFFSET  MSG3
        JMP     START2
.......
END
```

（3）循环控制指令

格式：LOOP　　目的地址

相当于：DEC　CX

JNZ　目的标号

例：START: ADD　AX，ARRAY[SI]

ADD　SI，2

LOOP　START

LOOPZ/LOOPE：CX≠0，且ZF=1，循环。

LOOPNZ/LOOPNE：CX≠0，且ZF=0，循环。

注意：所有循环指令所发生的转移都只能是短转移

■ 循环程序

注意：循环程序的循环条件和结束点

例4.27：计算1~100数字之和，结果存入字变量SUM。

```
.MODEL  SMALL
.STACK
.DATA
      SUM          DW       ?
.CODE
.STARTUP
            XOR      AX, AX
            MOV      CX, 100
 AGAIN: ADD      AX, CX
            LOOP     AGAIN
            MOV      SUM, AX
.EXIT 0
END
```

循环程序例： 5个连续字节单元的累加

```
DATA    SEGMENT                          ;定义数据段
    D1    DB  5 DUP（？）
    CC    EQU  $－D1
  SUM     DW   0
DATA    ENDS                             ;数据段定义结束
STACK SEGMENT                            ;定义堆栈段
    DB  100  DUP （？）
STACK   ENDS                             ;堆栈段定义结束
CODE    SEGMENT                          ;定义代码段
ASSUME  DS:DATA，SS:STACK，CS:CODE
BEGIN：  MOV     AX，DATA
         MOV     DS，AX
         LEA     SI，D1
         MOV     CX，CC
         CLD                             ；（DF）=0，串指针自动增量
         LODS    BYTE PTR [SI]
         MOV     AH，0                   ；清AX的高字节
         CLC                             ；清CF位
AGAN：   ADC     SUM，AX
         LODS    BYTE PTR [SI]
         LOOP    AGAN
         MOV     AH，4CH                 ；完成，返回DOS
         INT     21H
CODE    ENDS                             ；代码段结束
        END     BEGIN                    ；整个程序结束
```

定义数据

定义堆栈

例4.30：采用“冒泡法”把一个长度已知的数组元素按照从小到大排序。假设数组元素为无符号字节。

```
.MODEL  SMALL
.STACK
.DATA
      ARRAY       DB      56H, 23H, 37H, 78H,0FFH, 0, 12H, 99H, 64H, 0B0H
                  DB      78H, 80H, 23H, 1, 4, 0FH, 2AH, 46H, 32H, 42H
      COUNT       EQU     ($-ARRAY)/TYPE ARRAY
.CODE
.STARTUP
            MOV     CX, COUNT
            DEC     CX
 OUTLP:     MOV     DX, CX
            MOV     BX, OFFSET  ARRAY
 INLP:      MOV     AL, [BX]
            CMP     AL, [BX+1]
             JNA    NEXT
            XCHG    AL, [BX+1]
 NEXT:      INC     BX
            DEC     DX
            JNZ     INLP
            LOOP    OUTLP
.EXIT 0
END
```

■ 子程序设计

（1）子程序定义伪指令 PROC 和 ENDP

子程序名　PROC　[NEAR/FAR]

　　子程序体

子程序名　ENDP

（2）子程序调用伪指令　CALL

CALL　子程序名/地址

例：　CALL　1000H

　　CALL　EAX

　　CALL　2500H:3600H

　　CALL　DWORD　PTR　[DI]

（3）子程序返回伪 指令　RET 和 RET n

RET

RET　n　　返回后使SP/ESP的值加上n

子程序定义例：（实现回车换行功能的子程序）

```
DPCRLF      PROC
        PUSH    AX
        PUSH    DX
        MOV     DL, 0DH
        MOV     AH, 2
        INT     21H
        MOV     DL, 0AH
        MOV     AH, 2
        INT     21H
        POP     DX
        POP     AX
        RET
DPCRLF      ENDP
```

■ 子程序的参数传递

● 寄存器传递参数

例4.42：ARRAY是10个元素的数组，每个元素是8位数据，试用子程序计算数组元素的校验和，并将结果存入变量RESULT。(寄存器参数传递)

```
.MODEL  SMALL
.STACK
.DATA
        COUNT       EQU     10
        ARRAY       DB      12H, 25H, 0F0H, 0A3H,3, 68H, 71H, 0CAH, 0FFH, 90H
        RESULT      DB      ?
.CODE
.STARTUP
            MOV     BX, OFFSET ARRAY
            MOV     CX, COUNT
            CALL    CHECKSUMA
            MOV     RESULT, AL
.EXIT 0
CHECKSUMA   PROC
            XOR     AL, AL
SUMA:       ADD     AL, [BX]
            INC     BX
            LOOP    SUMA
            RET
CHECKSUMA   ENDP
END
```

入口参数：
BX, CX

出口参数：
AL

■ 子程序的参数传递

- 变量传递参数

例4.43：ARRAY是10个元素的数组，每个元素是8位数据，试用子程序计算数组元素的校验和，并将结果存入变量RESULT。(COUNT/ARRAY/RESULT参数传递)

```
.MODEL  SMALL
.STACK
.DATA
        COUNT       EQU     10
        ARRAY       DB      12H, 25H, 0F0H, 0A3H,3, 68H, 71H, 0CAH, 0FFH, 90H
        RESULT      DB      ?
.CODE
.STARTUP
            CALL    CHECKSUMB
.EXIT 0
CHECKSUMB   PROC
            PUSH    AX
            PUSH    BX
            PUSH    CX
            XOR     AL, AL
            MOV     BX, OFFSET ARRAY
            MOV     CX, COUNT
SUMB:       ADD     AL, [BX]
            INC     BX
            LOOP    SUMB
            MOV     RESULT, AL
            POP     CX
            POP     BX
            POP     AX
            RET
CHECKSUMB   ENDP
END
```

入口参数：
ARRAY, COUNT

出口参数：
RESULT

■ 子程序的参数传递

● 堆栈传递参数

例4.44：ARRAY是10个元素的数组，每个元素是8位数据，试用子程序计算数组元素的校验和，并将结果存入变量RESULT。 (堆栈参数传递)

入口参数：
ARRAY偏移地址，ARRAY个数压入堆栈

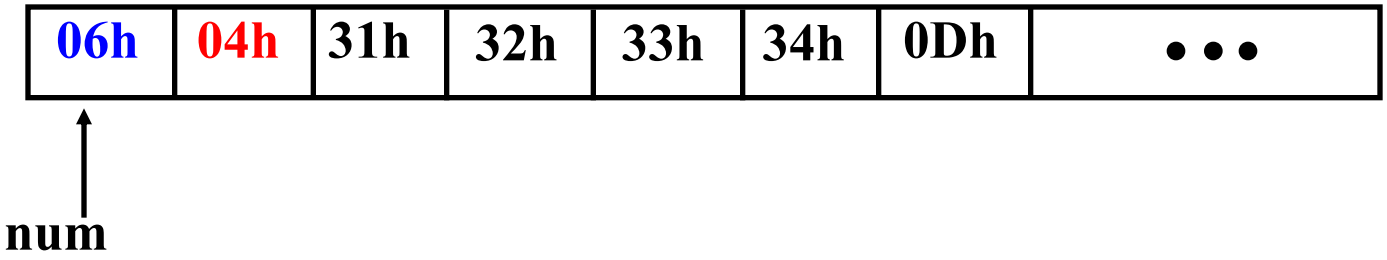
出口参数：
AL

```
.MODEL  SMALL
.STACK
.DATA
      COUNT       EQU     10
      ARRAY       DB      12H, 25H, 0F0H, 0A3H,3, 68H, 71
      RESULT      DB      ?
.CODE
.STARTUP
          MOV     AX, OFFSET ARRAY
          PUSH    AX
          MOV     AX, COUNT
          PUSH    AX
          CALL    CHECKSUMC
          ADD     SP, 4
          MOV     RESULT, AL
.EXIT 0
CHECKSUMC     PROC
          PUSH    BP
          MOV     BP, SP
          PUSH    BX
          PUSH    CX
          MOV     BX, [BP+6]
          MOV     CX, [BP+4]
          XOR     AL, AL
SUMC:     ADD     AL, [BX]
          INC     BX
          LOOP    SUMC
          POP     CX
          POP     BX
          POP     BP
          RET
CHECKSUMC     ENDP
END
```

子程序编程例：

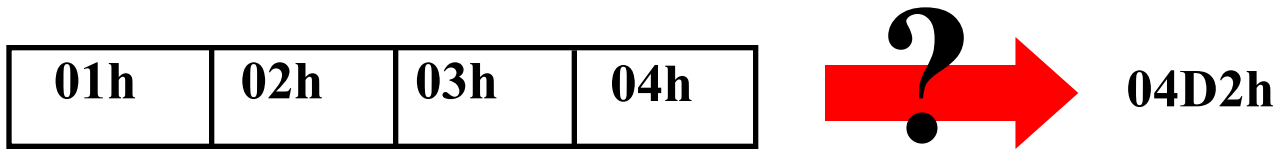
键盘输入的十进制数据转换成相应大小的十六进制数值存放在BX寄存器中。

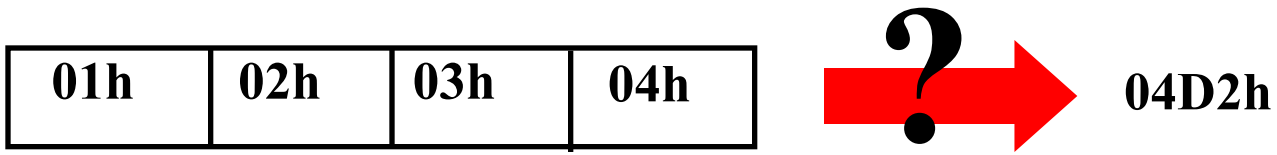
分析： 从键盘输入’ 1234’（表示1234）
用0AH功能输入，则缓冲区存放的内容为：



问题： 要转换成1234 即 04D2h存放在BX中，怎么实现？

清ASCII码的高4位可得各数位大小值。





算法一： 数值大小 = 各位值×权值之和

= 千位×1000 + 百位×100 + 十位×10 + 个位

1234D = 1×1000 + 2×100 + 3×10 + 4

= 0000 0100 1101 0010B

= 04D2H

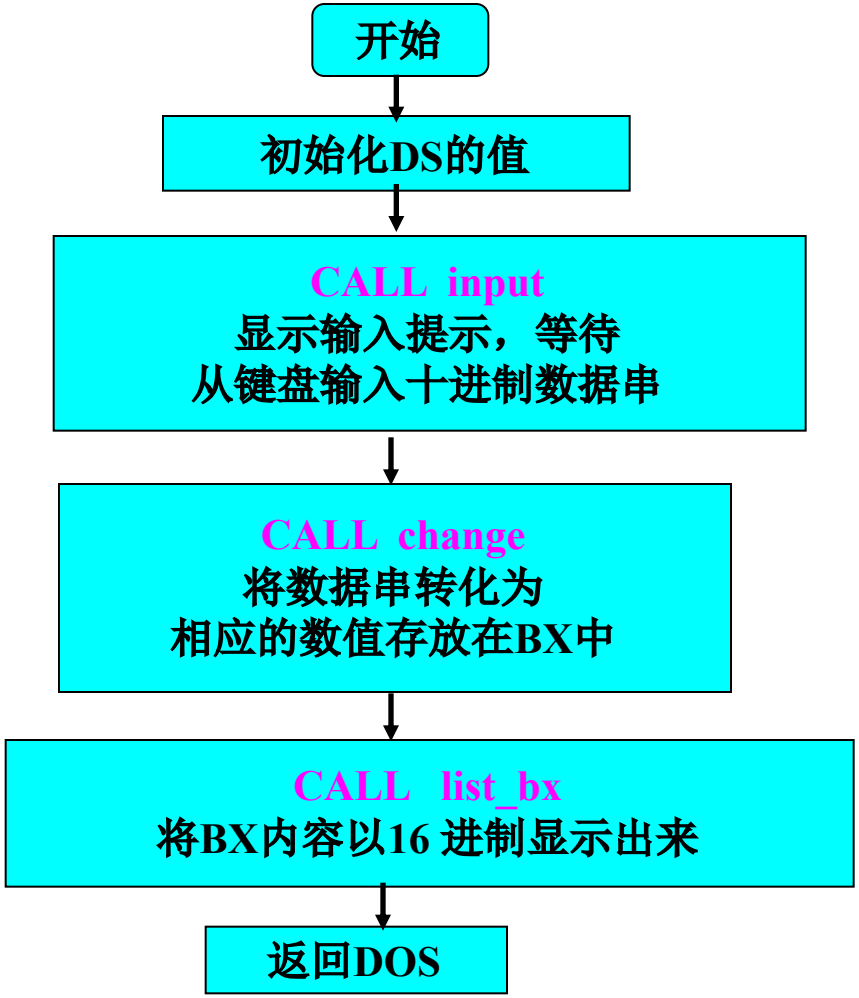
算法二： 数值大小 =部分和×10 +下一位数值

1234D = (((0×10 + 1) ×10 + 2) ×10 + 3) ×10 + 4

= 0000 0100 1101 0010B

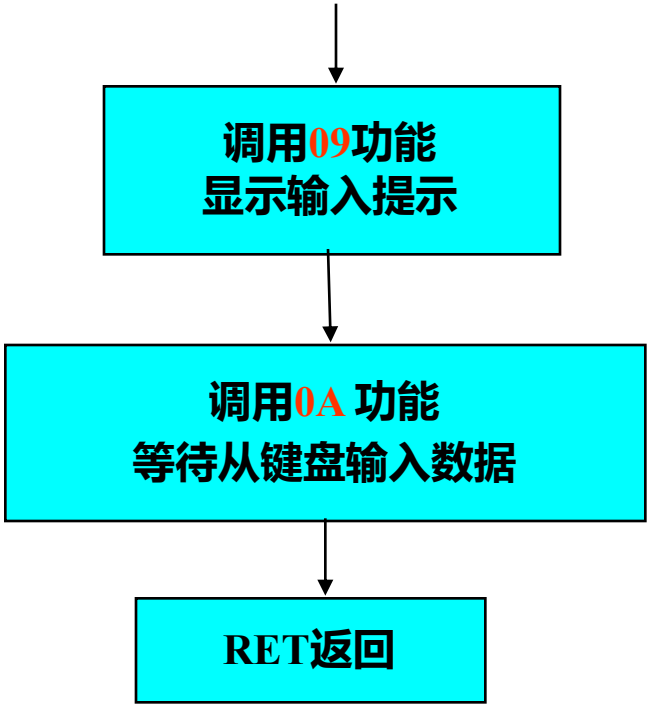
= 04D2H

部分和从0开始，循环次数等于输入的位数



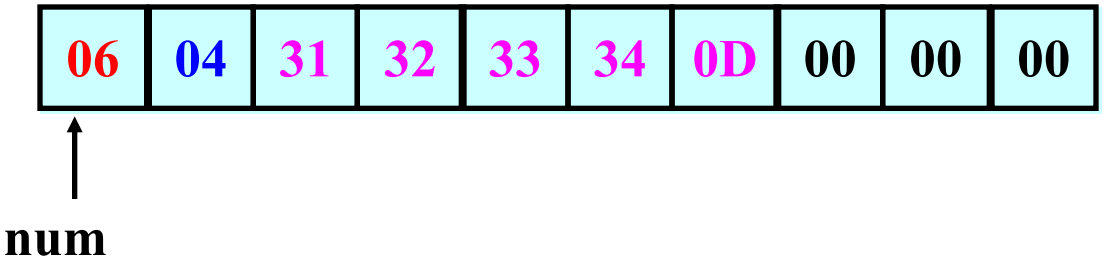
主程序流程图

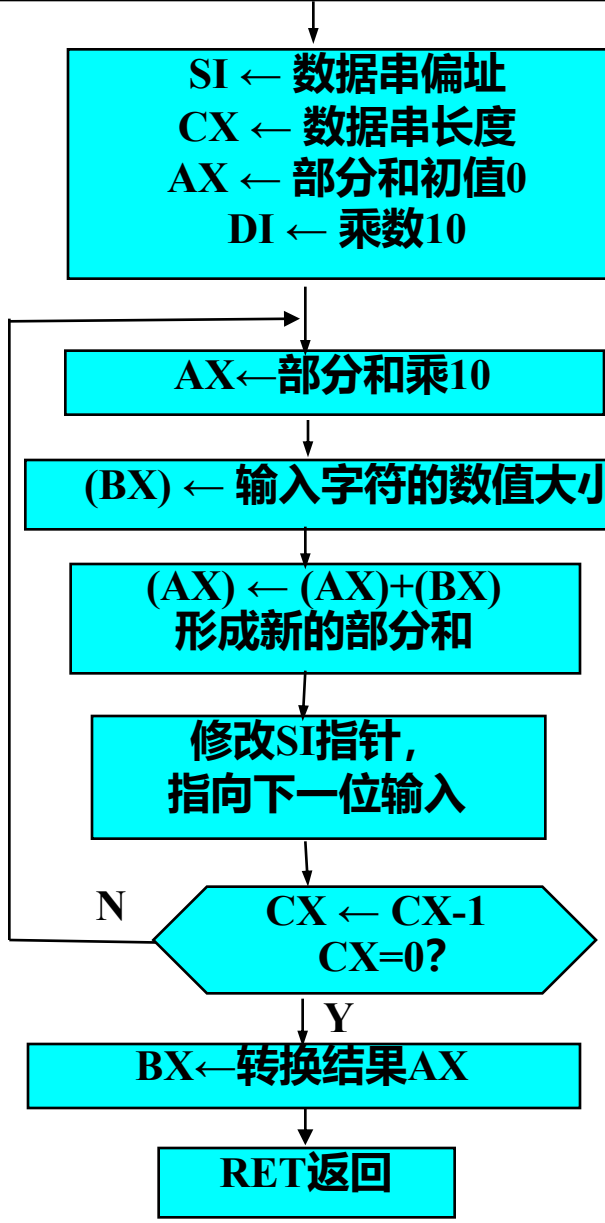
```
data    SEGMENT
        string  DB  'Input:','$'
        num     DB  6, ?, 6 DUP(?)
data    ENDS
CODE    SEGMENT
        ASSUME  CS:code, DS:data
start:  MOV   AX, data
        MOV   DS, AX
        CALL  input
        CALL  change
        CALL  list_bx
        MOV   AH, 4CH
        INT   21H
```

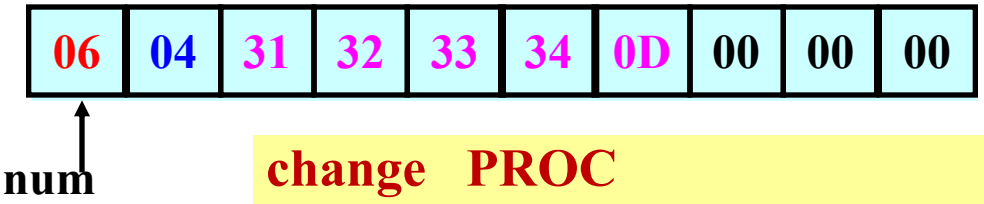


input
子程序流程图

```
input   PROC
        LEA   DX , string
        MOV   AH , 09H
        INT   21H
        LEA   DX , num
        MOV   AH , 0AH
        INT   21H
        RET
input   ENDP
```








```
change  PROC
        LEA   SI,  num+2
        MOV   CL,  num+1
        MOV   CH,  0
        MOV   AX,  0
        MOV   DI,  10
next:   MUL   DI
        MOV   BH,  0
        MOV   BL,  [SI]
        AND   BL,  0FH
        ADD   AX,  BX
        INC   SI
        LOOP  next
zero:   MOV   BX,  AX
        RET
change  ENDP
```

change子程序流程图

1234D = (((0×10 + 1) ×10 + 2) ×10 + 3) ×10 + 4

list_bx子程序流程图

```
list_bx   PROC
          MOV  CH, 4
          MOV  CL, 4
next:     ROL   BX, CL
          MOV  DL, BL
          AND   DL, 0FH
          ADD   DL, 30H
          CMP   DL, 39H
          JLE    print
          ADD   DL, 07H
print:    MOV   AH, 2H
          INT    21H
          DEC    CH
          JNZ    next
          RET
list_bx   ENDP
code        ENDS
          END   start
```

上面例程中存在的问题：

(1) 未处理输入非数字字符的情况

(2) 输入的十进制数范围为0~65535

(3) 当输入字符个数为0（直接回车）时，结果错误

(4) 未处理负数情况

(5) 结果的显示未换行

Input:1234　　最后看到的是：0D42t:1234

汇编和高级语言混合编程

■ 为什么混合编程?

(1) 提高速度和效率。

（2）实现某些C语言中不具备、但为不同机器所特有的功能。这是主要原因。

（3）利用通用的汇编语言例程。

■ 混合编程的方式

(1) 内嵌汇编语句

高级语言编程中嵌入汇编语句，如C/C++使用asm标识汇编语句

（2）模块连接

高级语言和汇编语言分别编译形成OBJ文件，然后连接在一起，生成可执行文件。

C语言内嵌汇编语句

C语言内嵌汇编指令

使用关键字asm语句:　　asm　<汇编语句>

例4.55

```
main( )
 {    int  i, a, b;
      printf(“Input two int num:\n”);
      scanf(“%d%d”, &a, &b);
      i = max(a, b);
      printf(“The maximum is %d”, i);
  }
int max (int v1, int v2)
  {  asm   mov    ax,  v1
     asm    cmp    ax,  v2
     asm    jge     exitfunc
     asm    mov    ax, v2
       exitfunc:
       return(_AX);
  }
```

C++语言内嵌汇编语句

VC++语言内嵌汇编指令

格式：_asm 汇编指令 [;]
　　_asm { 汇编指令 } [;]

例：
```
_asm mov   al, 2
_asm mov   dx, 0xD007
_asm out   dx, al
```

由于使用_asm识别汇编语句，所以可以写到一条代码行上：

```
_asm mov al, 2   _asm mov dx, 0xD007   _asm out dx, al
```

或写成语句块：

```
_asm { mov   al, 2
       mov   dx, 0xD007
       out   dx, al
       }
```

作业：

（1）练习

7,9,13,27

（2）纸质作业

10,11,35