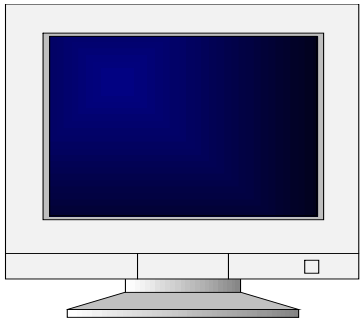
第五章

汇编语言程序设计



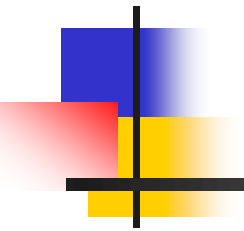
主要内容

- 汇编语言源程序的结构
- 汇编语言语句格式
- 伪指令
- 功能调用
- 汇编语言程序设计

§4.1 汇编语言源程序

了解：

- 汇编语言源程序的结构
- 汇编语言语句类型及格式

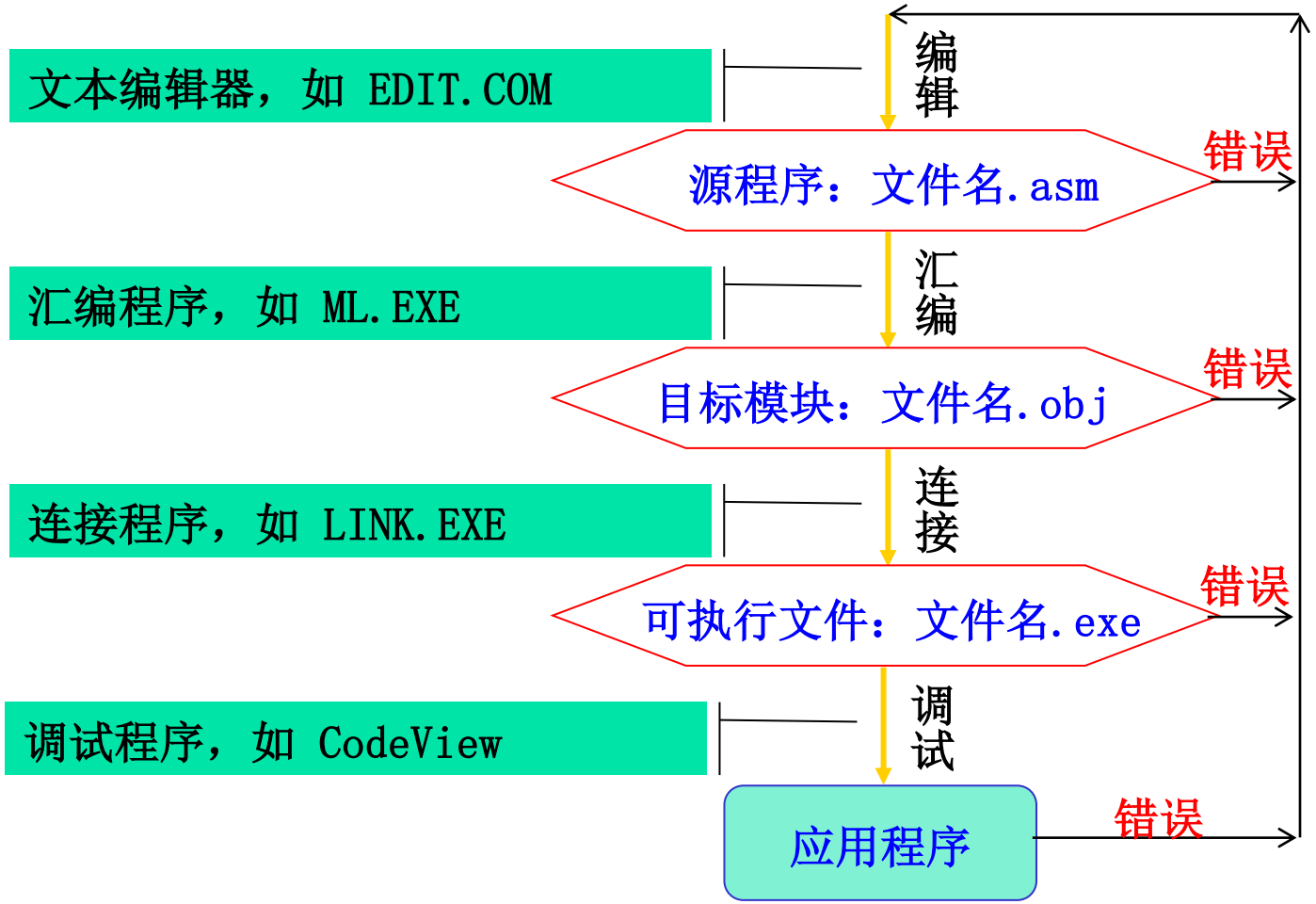


一、汇编语言源程序结构

1. 汇编语言源程序与汇编程序

- **汇编语言源程序** → 用助记符编写
- **汇编程序** → 源程序的编译程序

2. 汇编语言程序设计与执行过程



3. 汇编语言源程序结构

```
数据段名    SEGMENT
            …
数据段名    ENDS

附加段名    SEGMENT
            …
附加段名    ENDS
```

```
堆栈段名    SEGMENT
            …
堆栈段名    ENDS

代码段名    SEGMENT
            …
代码段名    ENDS
            END
```

一个完整源程序结构

```
DSEG    SEGMENT
        DATA1   DB    1, 2,  3  DUP (? )
        DATA2   DW   1234H
DSEG    ENDS
ESEG    SEGMENT
        DB   20   DUP (? )
ESEG    ENDS
SSEG    SEGMENT   STACK    'STACK'
        DB   200   DUP (? )
SSEG    ENDS
```

可以分成多行写，例如：
ASSUME CS: CSEG
ASSUME DS: DSEG

```
CSEG  SEGMENT
        ASSUME   CS：CSEG，DS：DSEG，
  ES：ESEG，SS：SSEG
START：MOV   AX，DSEG
        MOV   DS，AX
        MOV   AX，ESEG
        MOV   ES，AX
        MOV   AX，SSEG
        MOV   SS，AX
          ⋮
CSEG    ENDS
        END   START
```

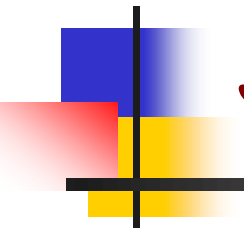
段寄存器初始化
——将段地址送相应的段寄存器

源程序代码

【例】编写一个两个字相加的程序

```
DSEG    SEGMENT                          ；定义数据段
        DATA1   DW   1234H               ；定义被加数
        DATA2   DW   5678H               ；定义加数
DSEG    ENDS                             ；数据段结束
ESEG    SEGMENT                          ；定义附加段
        SUM     DW   2 DUP（？）          ；定义存放结果区
ESEG    ENDS                             ；附加段结束
CSEG    SEGMENT                          ；定义代码段
```

```
; 下面的语句说明程序中定义的各段分别用哪个段寄存器寻址
ASSUME  CS：CSEG，DS：DSEG，ES：ESEG
START：  MOV   AX，DSEG ；START为程序开始执行的启动标号
         MOV   DS，AX              ；初始化DS
         MOV   AX，ESEG
         MOV   ES，AX              ；初始化ES
         LEA   SI，SUM             ；存放结果的偏移地址送SI
         MOV   AX，DATA1           ；取被加数
         ADD   AX，DATA2           ；两数相加
         MOV   ES：[SI]，AX        ；和送附加段的SUM单元中
         HLT
CSEG     ENDS                      ；代码段结束
         END   START               ；源程序结束
```



二、汇编语言语句类型及格式

1. 汇编语言语句类型

指令性语句

CPU执行的语句，能够生成目标代码

指示性语句

CPU不执行，而由汇编程序执行的语句，不生成目标代码

2. 汇编语言语句格式

指令性语句：

[标号：] [前缀] 助记符 [操作数], [操作数] [； 注释]

指令的符号地址

标号后要有冒号

操作码

注释前加分号

```
LOOPER：  MOV   AL，[SI] ；将[SI]所指向的内容赋给AL
```

标号有三种属性：段、偏移量和类型。

① 标号的段属性是定义标号在程序段的段地址，当程序中引用一个标号时，该标号的段值应在CS寄存器中。

② 标号的偏移量属性表示标号所在段的起始地址到定义该标号的地址之间的字节数。偏移量是一个16位无符号数。

③ 标号的类型属性有两种：NEAR和FAR。前一种标号可以在段内被引用；后一种标号可以在其它段被引用。

指示性语句格式

[名字] 伪指令助记符 操作数 [，操作数，...] [；注释]

变量的符号地址

其后不加冒号

指示性语句中至
少有一个操作数

```
DATA1 DB 0F8H，60H，0ACH ；定义含有3个元素的字节变量
```

3. 操作数

- 寄存器
- 存储器单元
- 常量
- 变量或标号
- 表达式

常 量

- 数字常量:以字母A～F开头的十六进制数，前面要用0表达，以避免与其他符号混淆。
- 字符串常量:用单引号引起的字符或字符串
- 例：‘A’
  - **MOV AL,’ A’**
- **例：’ABCD’**

汇编时被译成对应的ASCII码41H，42H，43H，44H

常量并不被分配内存空间，因为存储在内存空间中的内容是可变的

变 量

- 代表内存中的数据区，程序中视为存储器操作数
- 变量的属性：

段 值 —— 变量所在段的段地址

偏移量 —— 变量单元地址与段首地址之间的位移量。

类 型 —— 字节型、字型和双字型

变量定义格式：<变量名>DB/DW/DD/DQ/DT<表达式>

变量空间分配：变量与常量不同，汇编程序会给变量被分配存储空间，EQU定义的常量则不占用存储空间。

注意**变量的偏移地址**和**变量的值**两者的区别，且变量一旦定义，则**有类型**

变量名	偏移地址	变量的值
S1	0000H	00H
S2	0004H	31H
NW	0009H	0078H

DS:0000H	00H	S 1
0001H	00H	
0002H	00H	
0003H	16H	
0004H	31H	S 2
0005H	32H	
0006H	33H	
0007H	00H	N B
0008H	00H	
0009H	78H	N W
000AH	00H	
000BH	9CH	
000CH	FFH	
000DH	04H	A S
000EH	00H	
000FH	02H	B 3
0010H	02H	
0011H	0BH	
0012H	41H	
0013H	16H	

例1：某个数据段为：

```
DATA   SEGMENT
S1     DB    0,?,?,22
S2     DB    '123'
COUNT  EQU   $-S1
NB     DB    2 DUP(?)
NW     DW    120,-100
AS     DW    S2      ;用地址表达式定义变量
B3     DB    2 DUP(2),1 DUP(11,'A'),16H
DATA   ENDS
```

$—汇编地址计数器，记录当前汇编地址，Count的值为7

表达式

- 算术运算
- 逻辑运算
- *关系运算
- 取值运算和属性运算
- 其它运算

算术运算和逻辑运算符

- 算术运算符
  - +，-，*，/，MOD
- 逻辑运算符
  - AND，OR，NOT，XOR
- 例：
  - MOV AL，8 AND 4
  - MOV AL，8+4-1

取值运算符

- 用于分析存储器操作数的属性
  - 获取变量的属性值

OFFSET → 取得其后变量或标号的偏移地址

SEG → 取得其后变量或标号的段地址

TYPE	取变量的类型
LENGTH	取所定义存储区的长度
SIZE	取所定义存储区的字节数

- Type 返回值：

（1）对于变量有3种。

1：字节型； 2：字型；4：双字型；

（2）对于标号有2种：

—1：NEAR（段内）

—2：FAR（段间）

取值运算符例

```
MOV   AX，SEG   DATA
 MOV   DS，AX
 MOV   BX，OFFSET   DATA
```

等价于

```
LEA   BX，DATA
```

取值运算符例

- 若BUFFER存储区用如下伪指令定义：

  BUFFER DW 200 DUP(0)

则：

TYPE	BUFFER	等于2
LENGTH	BUFFER	等于200
SIZE	BUFFER	等于400

注意：SIZE＝LENGTH＊TYPE

属性运算符

- 用于指定其后存储器操作数的类型
- 运算符：**PTR**
- 例：

MOV BYTR PTR[BX]，12H

其它运算符

- **方括号：**

  [　] → 方括号中内容为操作数的偏移地址

- **段重设符**

  段寄存器名：[　] → 用于修改默认的段基地址

上述运算符和操作符构成表达式时的优先级：

高

（）、[ ]、< >、LENGTH、SIZE、WIDTH、MASK；

PTR、OFFSTE、SEG、TYPE、THIS、段操作符；

HIGH、LOW；

*、/、MOD、SHL、SHR；

+、-；

EQ、NE、LT、LE、GT、GE；

NOT；

AND、OR、XOR；

低

SHORT

从左到右顺序运算

§4.2 伪指令

掌握：

- 伪指令的格式及实现的操作
- 伪指令的应用

伪指令

- 由汇编程序执行的“指令系统”
- 作用：
  - 定义变量；
  - 分配存储区
  - 定义逻辑段；
  - 指示程序开始和结束；
  - 定义过程等。

常用伪指令

- 数据定义伪指令
- 符号定义伪指令
- 段定义伪指令
- 结束伪指令
- 过程定义伪指令
- 宏命令伪指令

一、数据定义伪指令

- 用于定义数据区中变量的类型及大小
- 格式：

[变量名] 伪指令助记符 操作数，… ；[注释]

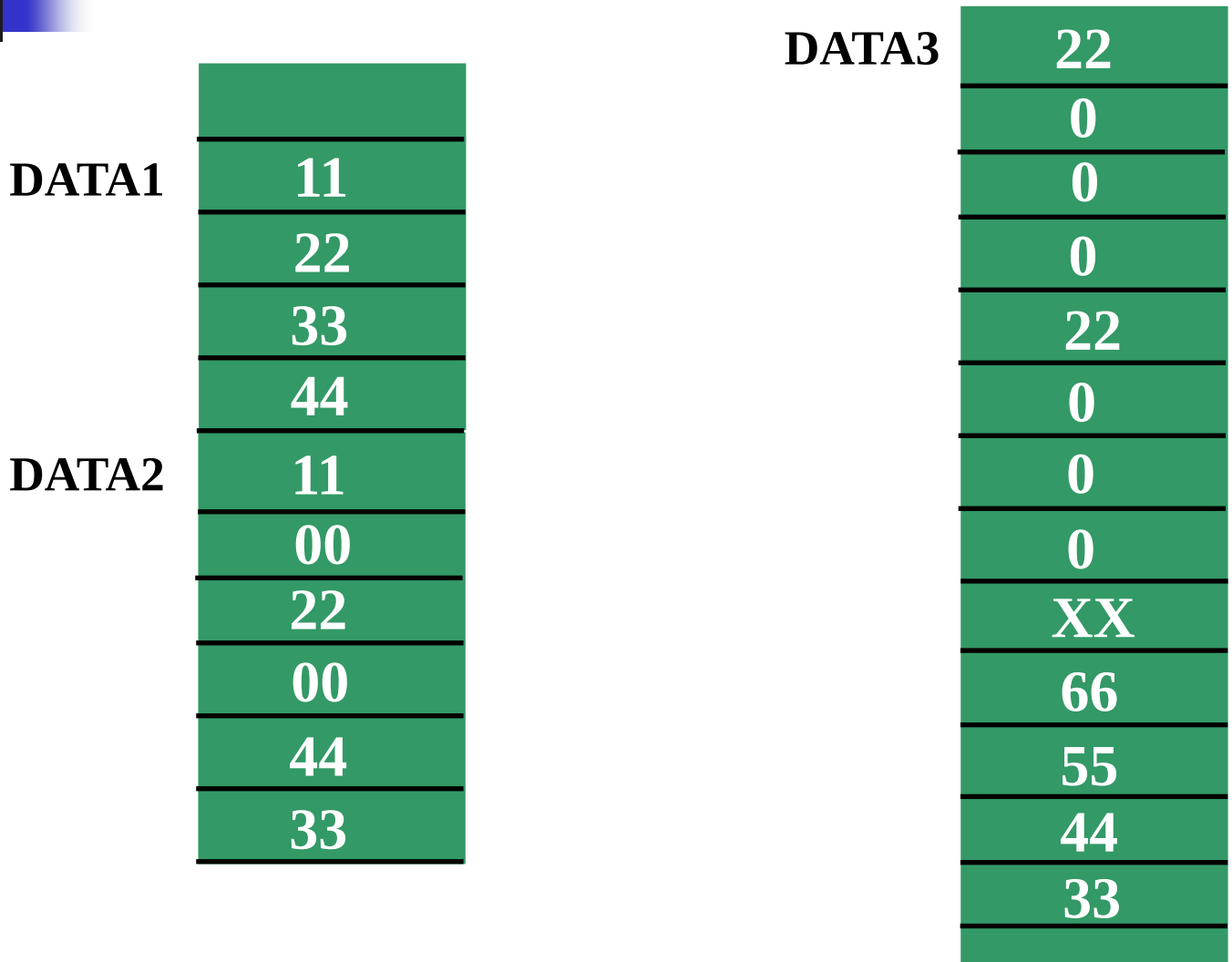
1. 数据定义伪指令助记符

- DB　　定义的变量为字节型
- DW　　定义的变量为字类型（双字节）
- DD　　定义的变量为双字型（4字节）
- DQ　　定义的变量为4字型（8字节）
- DT　　定义的变量为10字节型

数据定义伪指令例

- DATA1　DB　11H，22H，33H，44H
- DATA2　DW　11H，22H，3344H
- DATA3　DD　11H*2，22H，33445566H

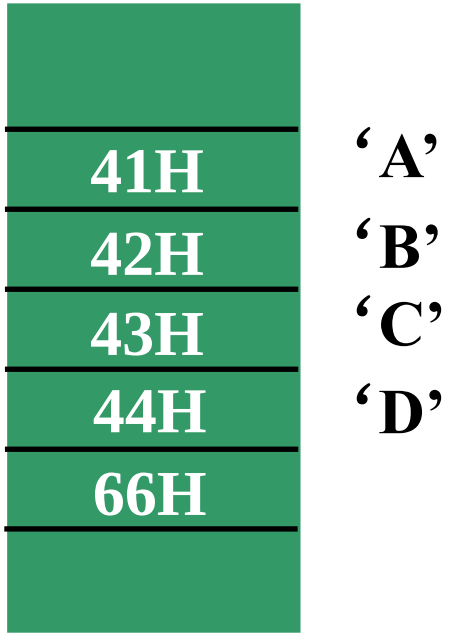
数据定义伪指令例_变量在内存中的分布



数据定义伪指令的几点说明

- 伪指令的性质决定所定义变量的类型；
- 定义字符串必须用DB伪指令
- 例：

DATA1　DB　‘ABCD’，66H



2. 重复操作符

- **作用：**
  - **为一个数据区的各单元设置相同的初值**
- **目的：**
  - **常用于声明一个数据区**
- **格式：**

  [变量名] 伪指令助记符 n DUP（初值，...）
- **例：**

  M1 DB 10 DUP (0)

3. “？”的作用

- 表示随机值，用于预留存储空间
- MEM1　DB　34H，’A’，？

　　DW　20　DUP（？）

随机数
占1个字节单元

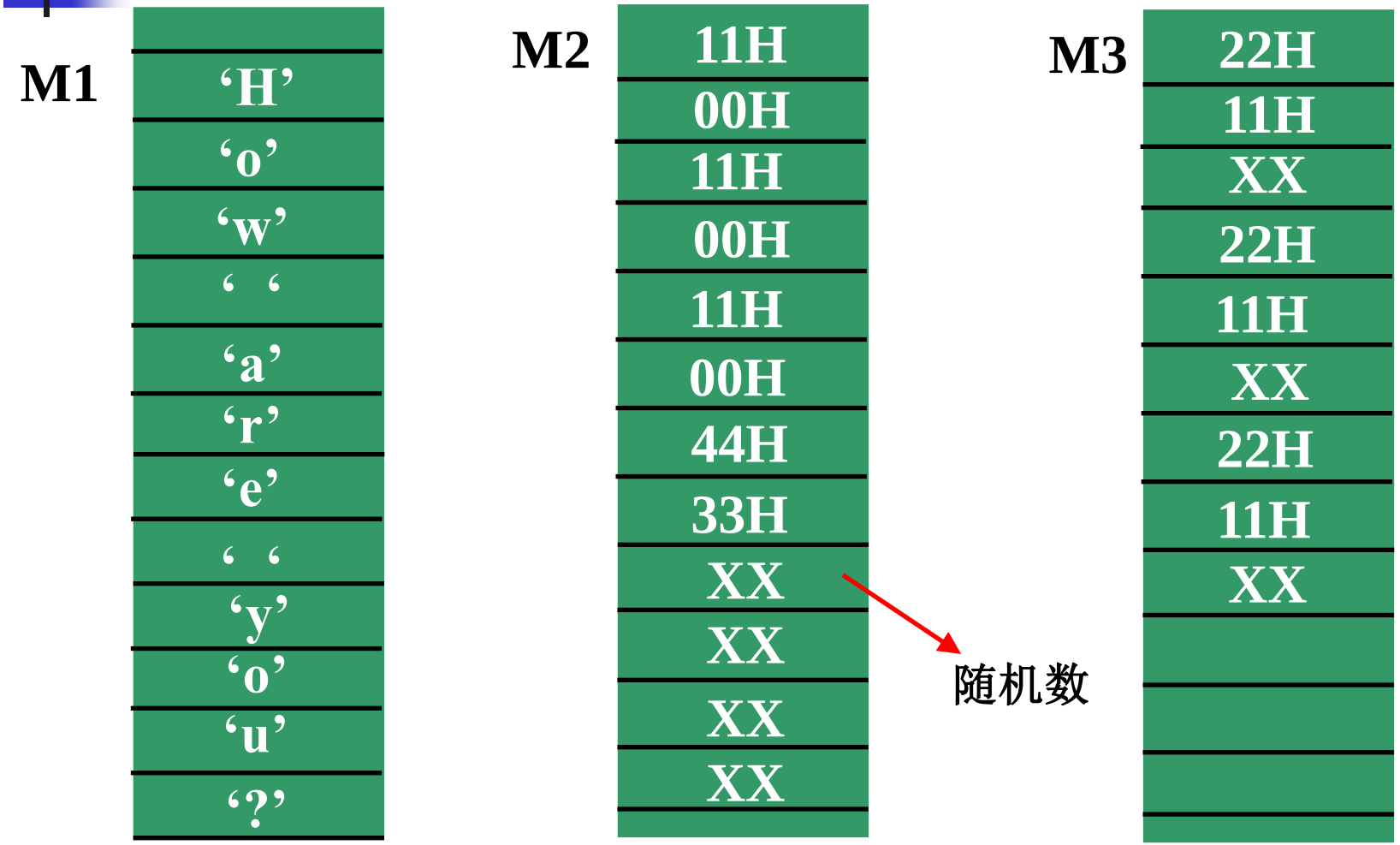
预留40个字节单元，每单元为随机值

数据定义伪指令例

- M1　DB　‘How are you?’
- M2　DW　3　DUP(11H)，3344H
  　　DB　4　DUP（？）
- M3　DB　3　DUP（22H，11H，？）

变量在内存中的分区

数据定义伪指令例


M1
'H'
'o'
'w'
' '
'a'
'r'
'e'
' '
'y'
'o'
'u'
'?'
M2
11H
00H
11H
00H
11H
00H
44H
33H
XX
XX
XX
XX
随机数
M3
22H
11H
XX
22H
11H
XX
22H
11H
XX

二、符号定义伪指令

- 格式：

  符号名 EQU 表达式 等价于 符号名 = 表达式

- 操作：

  用符号名取代后边的表达式，不可重新定义

- 例：

```
CONSTANT      EQU    100
VAR   EQU   30H+99H
```

EQU说明的表达式不占用内存空间

三、段定义伪指令

- 说明逻辑段的起始和结束；
- 说明不同程序模块中同类逻辑段之间的联系形态

段定义伪指令格式

段名　SEGMENT［定位类型］［组合类型］［'类别'］

⋮

段名　ENDS

说明逻辑段的起点

说明不同模块中同名段的组和连接方式

```
段名　SEGMENT
　　⋮
段名　ENDS
```

只需要掌握这种形式

定位类型

- PARA：段的起点从节边界开始（16个字节为1节）

  段的起始地址应为：XXXX0H（被16整除）

- BYTE：段的起点从存储器任何地址开始（字节边界开始）

- WORD：段的起点从偶地址开始（字边界开始）

- PAGE：段的起点从页边界开始（256个字节为1页）

  段的起始地址应为：XXX00H（被256整除）

组合类型

- 与其它模块中的同名段在满足定位类型的前提下具有的组合方式：
  - NONE：　不组合
  - PUBLIC：　依次连接（顺序由LINK程序确定）
  - COMMON：　覆盖连接
  - STACK：　堆栈段的依次连接
  - AT　表达式：段定义在表达式值为段基的节边界
  - MEMORY：　相应段在同名段的最高地址处。

类别

- 不同模块连接时将相同类别的段放在连续的内存区域中

段定义伪指令例

```
DATA   SEGMENT
       MEM1   DB   11H，22H
       MEM2   DB   ‘Hello!’
       MEM3   DW   2 DUP（？）
DATA   ENDS
```

变量在逻辑段中的位置就代表了它的偏移地址

表示变量所在逻辑段的段地址

表示变量的类型

四、设定段寄存器伪指令

- 说明所定义逻辑段的性质
- 格式：

ASSUME 段寄存器名:段名[，段寄存器名:段名，…]

五、结束伪指令

- 表示源程序结束
- 格式：

  END［标号］

六、过程定义伪指令

- 用于定义一个过程体
- 格式：

过程入口的符号地址

```
过程名   PROC [ NEAR / FAR ]
          ⋮
         RET
过程名   ENDP
```

过程定义及调用例

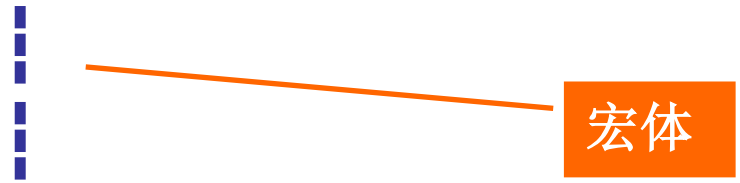
- 定义延时子程序
  - DELAY PROC
  - PUSH BX
  - PUSH CX
  - MOV BL, 2
  - NEXT: MOV CX, 4167
  - W10M: LOOP W10M
  - DEC BL
  - JNZ NEXT
  - POP CX
  - POP BX
  - RET
  - DELAY ENDP

- 调用延时子程序:
  - CALL DELAY

七、宏命令伪指令

- 宏 → 源程序中由汇编程序识别的具有独立功能的一段程序代码
- 格式：

宏命令名　MACRO　<形式参数>



ENDM

八、其它伪指令

ORG → **段内程序代码或变量的起始偏移地址**

- **格式：**

  ORG 表达式

  计算值为非负常数

- **例：**

  ORG 2000H

ORG可置于代码段、数据段的任何地方
无ORG时则从偏移地址为0000H开始存放代码和数据

【例】给汇编地址计数器（指针）$赋值

```
DATA    SEGMENT
        ORG  10；置$值为10,表示下面的代码将从偏移地址10位置开始。
        VAR1   DW   100H，200H
        ORG  $+5 ；置$的值为14+5，即为19
        VAR2   DB   1，2，   $+1，$+2
        N EQU  $-VAR2       ；=4
DATA    ENDS
```

注：$表示当前汇编地址指针的值。

```
dseg    segment
        org   10
        var1  dw 100h,200h
        org   $+5
        var2  dw 1,2  ,$+1，  $+2
        n     equ $-var2
dseg    ends
code    segment'code'
        assume cs:code,ds:dseg
start:  mov ax,dseg
        mov ds,ax
        mov dl,n
        mov ah,02h
        int 21h
        mov ax,4c00h
        int 21h
code    ends
        end start
```

请注意DW，若采用地址指针$表达式初始化变量时，在masm5编译和debug调试发现，这里不能采用DB，只能采用DW

```
-r
AX=FFFF  BX=0000  CX=0030  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0000   NV UP EI PL NZ NA PO NC
076C:0000 B86A07        MOV     AX,076A
-t

AX=076A  BX=0000  CX=0030  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0003   NV UP EI PL NZ NA PO NC
076C:0003 8ED8          MOV     DS,AX
-t

AX=076A  BX=0000  CX=0030  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076C  IP=0005   NV UP EI PL NZ NA PO NC
076C:0005 B208          MOV     DL,08
-t

AX=076A  BX=0000  CX=0030  DX=0008  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076C  IP=0007   NV UP EI PL NZ NA PO NC
076C:0007 B402          MOV     AH,02
-t

AX=026A  BX=0000  CX=0030  DX=0008  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076C  IP=0009   NV UP EI PL NZ NA PO NC
076C:0009 CD21          INT     21
-
```

```
dseg    segment
        org   10
        var1  dw 100h,200h
        org   $+5
        var2  db 1,2  ,$+1，  $+2
        n     equ $-var2
dseg    ends
code    segment'code'
        assume cs:code,ds:dseg
start:  mov ax,dseg
        mov ds,ax
        mov dl,n
        mov ah,02h
        int 21h
        mov ax,4c00h
        int 21h
code    ends
        end start
```

若改为db，则显示第5行错误

```
C:\>cd masm5

C:\MASM5>masm liuyao.asm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [liuyao.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:
liuyao.asm(5): error A2042: Constant expected

  51708 + 464836 Bytes symbol space free

      0 Warning Errors
      1 Severe  Errors

C:\MASM5>
```

其它伪指令

- NAME--------为目标程序设定一个名字
- 格式：

  NAME　模块名
- TITLE--------为程序清单指定打印标题
- 格式：

  TITLE　标题名
- .8086--------汇编程序将在8086／8088方式下操作
- .386--------汇编程序将在80386方式下操作

§4.3 功能调用

- DOS 功能调用
- BIOS功能调用

DOS 功能调用

- 包含多个子功能的功能包；
- 用软中断指令调用，中断类型码固定为21H；
- 各子功能采用功能号来区分。

DOS 功能调用

- 设备管理
- 目录管理
- 文件管理
- 其它

DOS 功能调用

- **调用格式:**

```
MOV   AH，功能号
〈置相应参数〉
INT   21H
```

一、单字符输入

● 键盘输入单字符（1号调用）

格式：MOV　AH，1
　　INT　　21H

功能：等待键盘输入，输入字符的ASCII码送入AL寄存器，并在显示器上显示该字符。

- 无入口参数；
- 执行时系统等待键盘输入，若键入Ctrl-Break，则退出；
- 功能号7和8也可接收键盘输入的单字符，但无回显。

例：程序中有时需要用户对提示做出应答。

```
GET_KEY: MOV   AH,1    ;等待键入字符
         INT   21H     ;结果在AL中
         CMP   AL,’Y’  ;是’Y’?
         JZ    YES     ;是,转YES
         CMP   AL,’N’  ;是’N’?
         JZ    NO      ;是,转NO
         JMP   GET_KEY ;否则继续等待输入
    YES:      …
              …
    NO:       …
```

单字符输入例

```
GET_KEY:   MOV   AH,1
           INT   21H
           CMP   AL,'Y'
           JZ    YES
           CMP   AL,'N'
           JZ    NO
           JMP   GET_KEY
YES:       ┆
 NO:       ┆
```

交互式应答程序

二、字符串输入

- 注意问题：
  - 调用格式
  - 字符输入缓冲区的定义

1. 调用格式

- AH ← 功能号0AH
- DS：DX ← 字符串在内存中的存放地址

⑩ INT 21H

2.键盘输入字符串（0A号调用）

格式：MOV　AH，0AH
　　LEA　DX，<字符串缓冲区首地址>
　　INT　　21H

功能：等待键盘输入，输入字符串的ASCII码送内存单元。

注意：先定义一个缓冲区；最后输入回车键以示字符串输入结束：未填满，剩余填零；超过缓冲区容量，自动丢失并响铃。

缓冲区格式：

N1	N2	(预留的N1个字节的存储单元)	0DH

N1：缓冲区长度(最大键入字符数)，　**N2**：实际键入的字符数(不包括回车符)

例：设在数据段定义键盘缓冲区如下：

```
    STR1  DB  10,?,10 DUP(?)
；程序段：
    LEA  DX，STR1
    MOV  AH，0AH
    INT  21H    ；最多从键盘接收10个按键
```

输入字符串程序段

- DAT1　DB　20，？，20　DUP（？）

⋮

- LEA　DX，DAT1

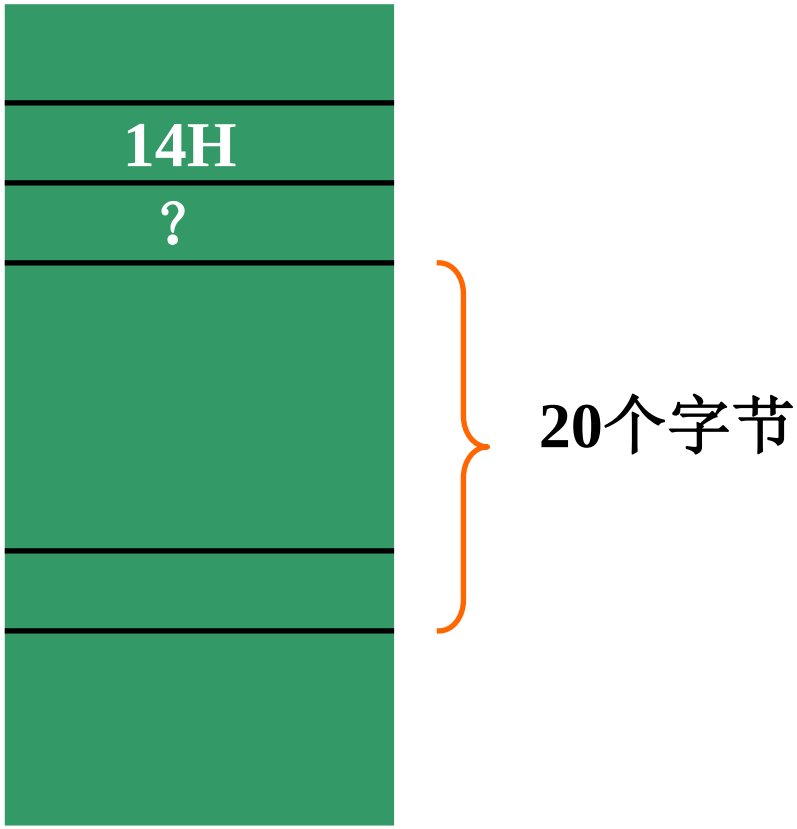
  MOV AH，0AH

  INT　21H

注意该位置表示实际输入的字符数，在数据段定义时可以随便输入一个数，相当于初值。

输入缓冲区

定义后的输入缓冲区初始状态：



三、单字符显示输出

- **AH** ← 功能号02H
- **DL** ← 待输出字符
- INT 21H

将DL寄存器中的字符在显示器上显示，如果要退出按〈CTRL〉+〈BREAK〉

单字符显示输出例

- MOV AH，02
- MOV DL，41H
- INT 21H

执行结果：
屏幕显示A

四、字符串输出显示

- AH ← 功能号O9H
- DS：DX ← 待输出字符串的偏移地址
- INT 21H

被显示的字符串必须以‘$’结束，且所显示的内容不应出现非可见的ASCII码

字符串输出显示例

```
DATA   SEGMENT
       MESS1   DB   ‘Input String:’, ODH, OAH,’ $’
       DATA   ENDS
CODE   SEGMENT
         ┇
MOV    AH, 09
MOV    DX, OFFSET   MESS1
INT    21H
         ┇
```

五、返回操作系统（4CH号调用）

格式：MOV　AH，4CH

　　INT　21H

功能：终止当前程序并返回调用程序(操作系统)。

【例】编写程序完成在屏幕上显示字符串“Example of string display!”

```
DATA    SEGMENT
STR     DB  0DH，0AH，‘Example of string display！ $’
DATA    ENDS
；定义了数据段，0DH回车符ASCII码、0AH换行符ASCII码
```

```
STACK  SEGMENT  PARA  STACK ’STACK’
       DB   100 DUP（0）
STACK  ENDS
；定义了堆栈段，本程序没使用定义的堆栈段
```

```
CODE   SEGMENT
   ASSUME     DS：DATA，CS：CODE，SS：STACK
BEGIN：       MOV    AX，DATA
              MOV    DS，AX
              LEA    DX，STR
              MOV    AH，9
              INT    21H
              MOV    AH，4CH
              INT    21H
CODE   ENDS
              END      BEGIN
```

【例】编写程序完成从键盘上输入一字符串到输入缓冲区，然后将输入的字符串在显示器上以相反的顺序显示。

解题思路：

（1）先按0AH号、9号功能要求定义输入输出缓冲区；

（2）调用0AH号功能输入字符串到BUFA缓冲区；

（3）通过一段循环程序将输入缓冲区的字符按相反顺序传送到输出缓冲区BUFB中；

（4）通过调用9号系统功能显示输出。

```
DATA   SEGMENT
   INFO1  DB  0DH，0AH，  ‘INPUT  STRING：  $’
   INFO2  DB  0DH，0AH，  ‘OUTPUT  STRING：  $’
   BUFA        DB      81
               DB      ?
               DB      80 DUP（0）
     ORG  $+10
   BUFB        DB      81 DUP（0）
DATA   ENDS
；定义数据段，安排提示信息、输入输出数据区
```

```
     STACK   SEGMENT  PARA  STACK ’STACK’
               DB  200 DUP（0）
     STACK   ENDS
     ；定义数据段
```

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

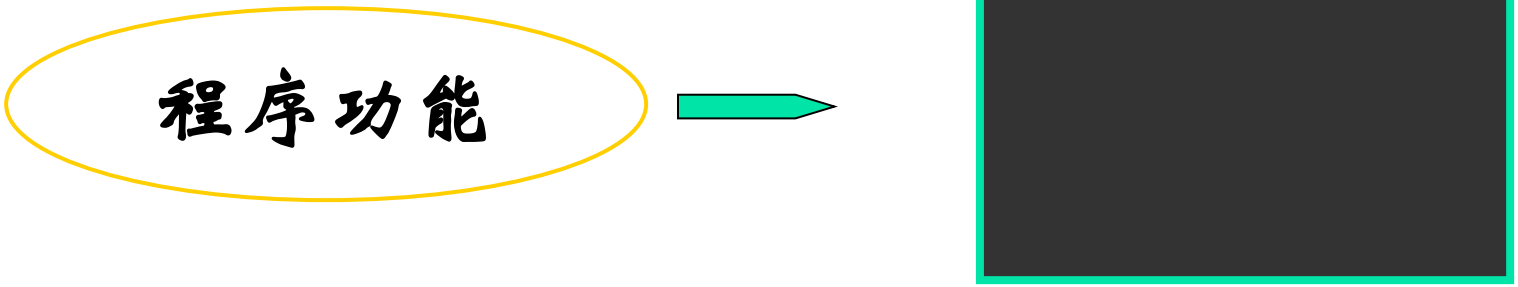
```
    LEA  DX，INFO1
    MOV  AH，9
    INT  21H          ；9号功能调用，显示输入提示信息
    LEA  DX，BUFA     ；输入字符串缓冲区首地址送DX
    MOV  AH，10       ；0AH号功能调用，功能号送AH寄存器
    INT  21H
```

```
    LEA  SI，BUFA+1
    MOV  CH，0
    MOV  CL，[SI]     ；取字符长度→CX
    ADD  SI，CX       ；SI为源数据指针指向字符串尾部
    LEA  DI，BUFB     ；DI为目的数据指针，首地址BUFB
NEXT：MOV  AL，[SI]
    MOV  [DI]，AL
    DEC  SI
    INC  DI
    LOOP  NEXT
    MOV  BYTE PTR [DI]，‘$’
```

```
    LEA  DX，INFO2
    MOV  AH，9
    INT  21H    ；显示提示信息
    LEA  DX，BUFB
    MOV  AH，9
    INT  21H    ；反显输入字符串
    MOV  AH，4CH ；返回DOS
    INT  21H
CODE  ENDS
    END  START
```

§4.4源程序两种格式书写

- 第一种格式从MASM 5.0开始支持
  - 简化段定义格式
- 第二种格式MASM 5.0以前就具有
  - 完整段定义格式



```
;lt301a.asm（文件名）
        .model small        ;定义程序的存储模式
        .stack              ;定义堆栈段
        .data               ;定义数据段
string  db ’Hello, Everybody !’,0dh,0ah,’$’
        ;在数据段定义要显示的字符串
        .code               ;定义代码段
        .startup            ;程序起始点，建立DS、SS
        mov dx,offset string  ;指定字符串
        mov ah,9
        int 21h             ;利用功能调用显示信息
        .exit 0             ;程序结束点，返回DOS
        end                 ;汇编结束
```

简化段定义格式

简化段定义格式
MASM 6.x支持

```
;SampleA.ASM
        .model small
        .stack
        .data
        ...             ;在数据段定义数据
        .code
        .startup
        ...             ;在代码段填入指令序列
        .exit 0
        ...             ;子程序代码
        end
```

简化段定义格式
MASM 5.x支持

```
;SampleC.ASM
        .model small
        .stack
        .data
        ...             ;在数据段定义数据
        .code
start: mov ax,@data
        mov ds,ax
        ...             ;在代码段填入指令序列
        mov ax,4c00h
        int 21h
        ...             ;子程序代码
        end start
```

```
;lt301b.asm (文件名)
stack   segment stack   ;定义堆栈段
        dw 512 dup(?)
        ;堆栈段有512字 (1024字节) 空间
stack   ends            ;堆栈段结束
data    segment         ;定义数据段
string  db 'Hello, Everybody !',0dh,0ah,'$'
data    ends
code    segment 'code' ;定义代码段
        assume cs:code,ds:data,ss:stack
start:  mov ax,data     ;建立DS段地址
        mov ds,ax
```

完整段定义格式

```
        mov dx,offset string
        mov ah,9
        int 21h
        mov ax,4c00h
        int 21h          ;利用功能调用返回DOS
code    ends             ;代码段结束
        end start
        ;汇编结束，同时指明程序起始点
```

完整段定义格式

完整段定义格式
MASM 5.x支持

```
;SampleB.ASM
stack   segment stack
        dw 512 dup(?)
stack   ends
data    segment
        ...             ;在数据段定义数据
data    ends
code    segment 'code'
        assume cs:code,ds:data,ss:stack
start:  mov ax,data
        mov ds,ax
        ...             ;在代码段填入指令序列
        mov ax,4c00h
        int 21h
        ...             ;子程序代码
code    ends
        end start
```

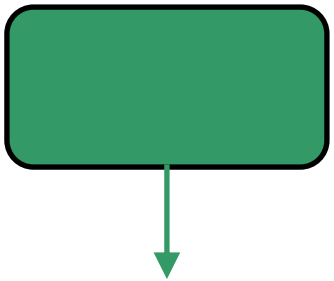
§4.5 汇编语言程序设计

设计步骤：

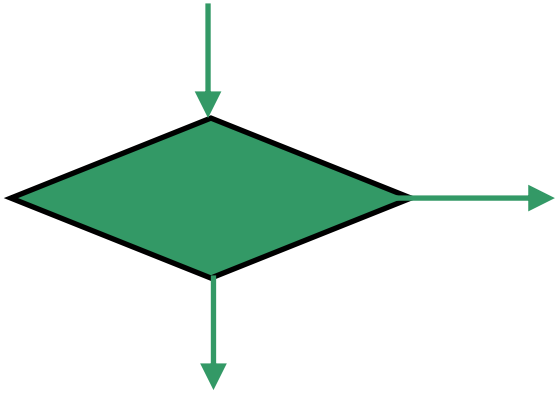
- 根据实际问题抽象出数学模型
- 确定算法
- 画程序流程图
- 分配内存工作单元和寄存器
- 程序编码
- 调试

程序流程图符号

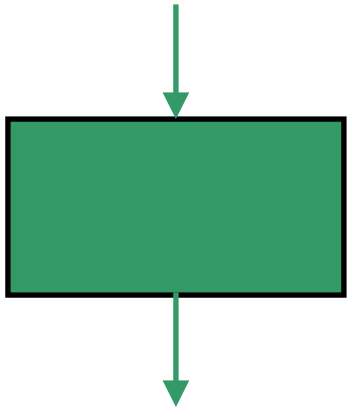
起始：



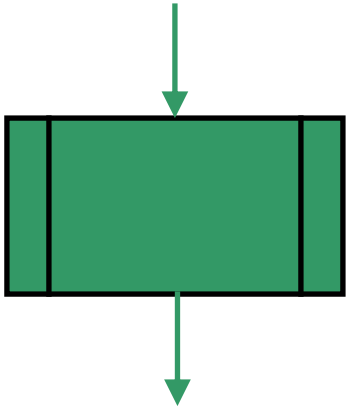
判断：



功能：



子过程：



汇编语言程序结构

- 顺序结构
- 循环结构
  - 先判定型
  - 后判定型
- 分支结构
  - 单分支
  - 多分支

顺序结构程序

什么是顺序结构程序？

【例】已知某班学生的英语成绩按学号（从1开始）从小到大的顺序排列在TAB表中，要查的学生的学号放在变量NO中，查表结果放在变量ENGLISH中。试编写程序实现成绩查询。

分析：

1，建立数据表，BX ← TAB偏移地址

2，学号NO 值送AL

3、查表：AL ←[BX+AL]

5、返回操作系统，使用DOS功能调用

```
MOV AH,4CH
INT 21H
```

```
DATA   SEGMENT
   TAB  DB  80，85，86，90，88，98，78，86，65
   NO   DB  6
   ENGLISH   DB    ?
DATA   ENDS
；定义数据段，成绩表、学号、成绩
```

```
      STACK   SEGMENT  PARA  STACK ’STACK’
                  DB  200 DUP（0）
      STACK   ENDS
      ；定义数据段
```

```
      CODE   SEGMENT
         ASSUME  DS: DATA, SS: STACK, CS: CODE
      BEGIN：MOV  AX，DATA
          MOV  DS，AX
          LEA  BX，TAB
          MOV  AL，NO
          DEC   AL
          XLAT
          MOV  ENGLISH，AL
          MOV  AH，4CH
          INT   21H
      CODE  ENDS
          END  BEGIN
```

学号是两位数
如何处理？
（十位、个位）

顺序结构程序

思考：

程序运行时改为有键盘输入学号，会改程序吗？

分析：1、建立数据表，BX ← TAB偏移地址

2、键盘输入学号送AL，使用DOS功能调用

```
MOV AH,01H
INT 21H    ;键盘输入的数据送AL
```

3、将键盘输入的字符转换为数字

```
AND AL, 0FH ；个位
```

4、查表：AL ←[BX+AL]

5、返回操作系统，使用DOS功能调用

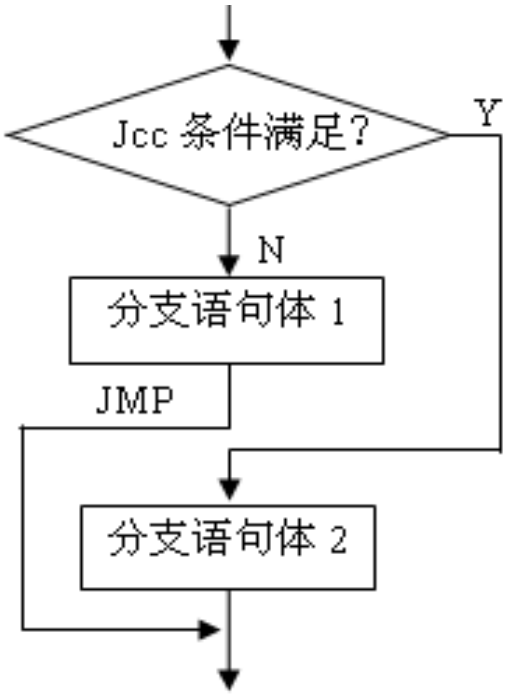
```
MOV AH,4CH
INT 21H
```

分支结构程序

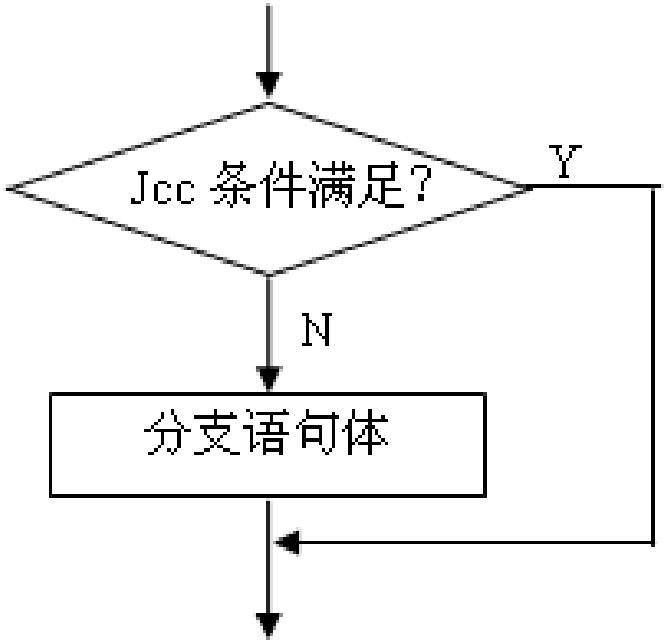
分支结构程序也称条件结构，根据不同条件转移到不同程序段执行相关分支程序。有双分支和多分支。

- 双分支结构

典型的双分支结构程序的流程图：



条件成立跳转执行第2个分支语句体，否则顺序执行第1个分支语句体。注意第1个分支体后一定要有一个JMP指令跳到第2个分支体后

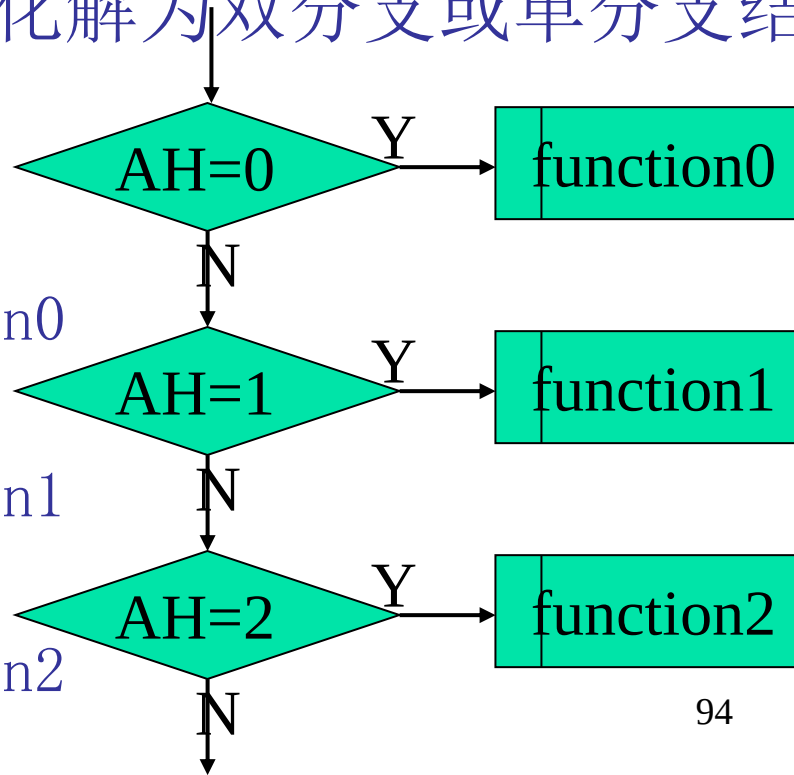


条件成立跳转，否则顺序执行分支语句体；注意选择正确的条件转移指令和转移目标地址

- 多分支程序设计

多个条件对应各自的分支语句体，哪个条件成立就转入相应分支体执行。多分支可以化解为双分支或单分支结构的组合，例如：

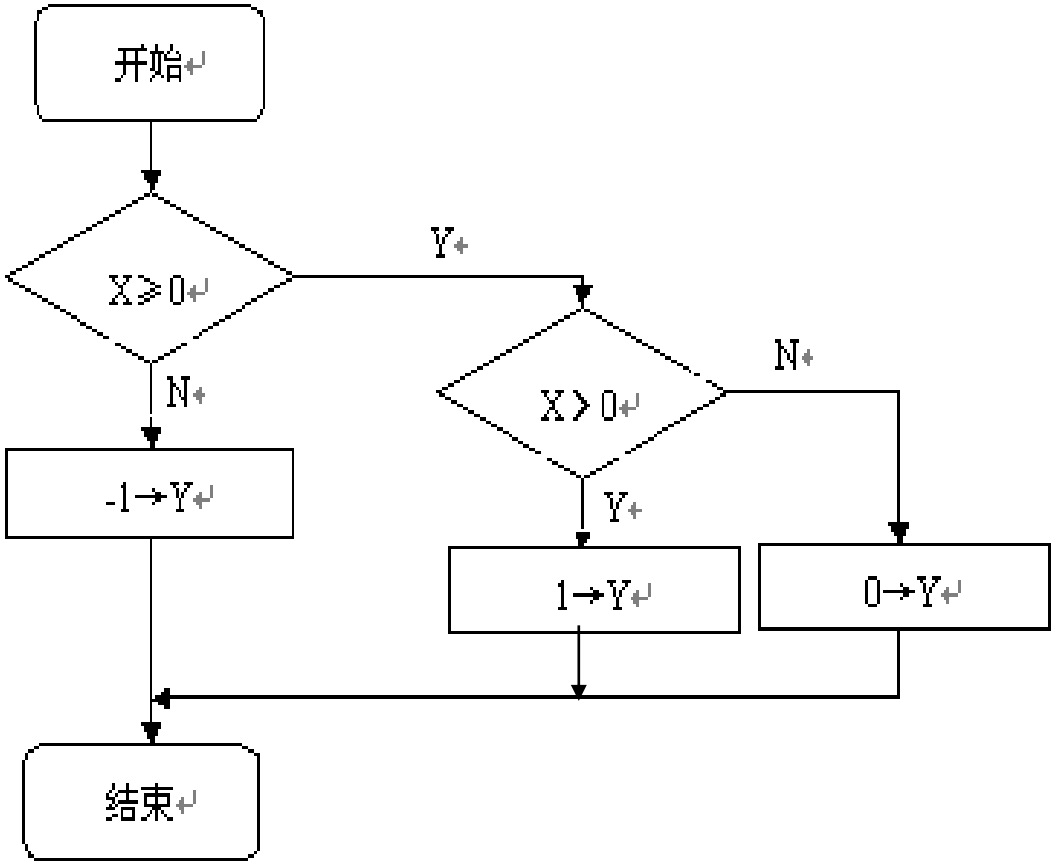
```
Xor ah,ah ;等效于cmp ah,0
jz function0;ah=0，转向function0
dec ah    ;等效于cmp ah,1
jz function1;ah=1，转向function1
dec ah    ;等效于cmp ah,2
jz function2;ah=2，转向function2
```



【例】：编写计算下面函数值的程序：

$$Y=\begin{cases}1 & ;X>0\\ 0 & ;X=0\\ -1 & ;X<0\end{cases}$$

设输入数据为X、输出数据Y，皆为字节变量。程序流程图如右图所示。



```
DATA    SEGMENT
        X  DB  -10          ；定义数据段，假设x值为-10
        Y  DB  ？           ；给函数值y预留一个单元
DATA    ENDS
SSEG  SEGMENT  PARA  STACK ’STACK’
        DB  200 DUP（0）
SSEG  ENDS
```

```
CODE    SEGMENT
        ASSUME  DS：DATA，SS：SSEG，CS：CODE
START：         MOV  AX，DATA
                MOV  DS，AX
                MOV  AX，SSEG
                MOV  SS，AX
```

```
        CMP  X，0       ；与0进行比较
        JGE  A1         ；X≥0转A1
        MOV  Y，-1      ；X ＜0时，-1→Y
        JMP  EXIT
A1：    JG  A2          ；X＞0转A2
        MOV  Y，0       ；X=0时，0→Y
        JMP  EXIT
```

```
A2：    MOV  Y，1
EXIT：  MOV  AH，4CH
        INT  21H
CODE    ENDS
        END  START
```

地址表形成多分支

利用地址表法实现多分支程序设计的一般方法为：

把各分支程序段的入口地址（一般是偏移地址，也可以使段地址与偏移地址）依次存放在数据段的一个表中，形成地址表。取各分支程序段的编号作为给分支入口地址的表地址的位移量。某个分支程序入口地址的表地址为：

表地址=编号*2+入口地址首地址

根据条件首先在地址表中找到转移的目标地址，然后转到相应位置，从而实现多分支。

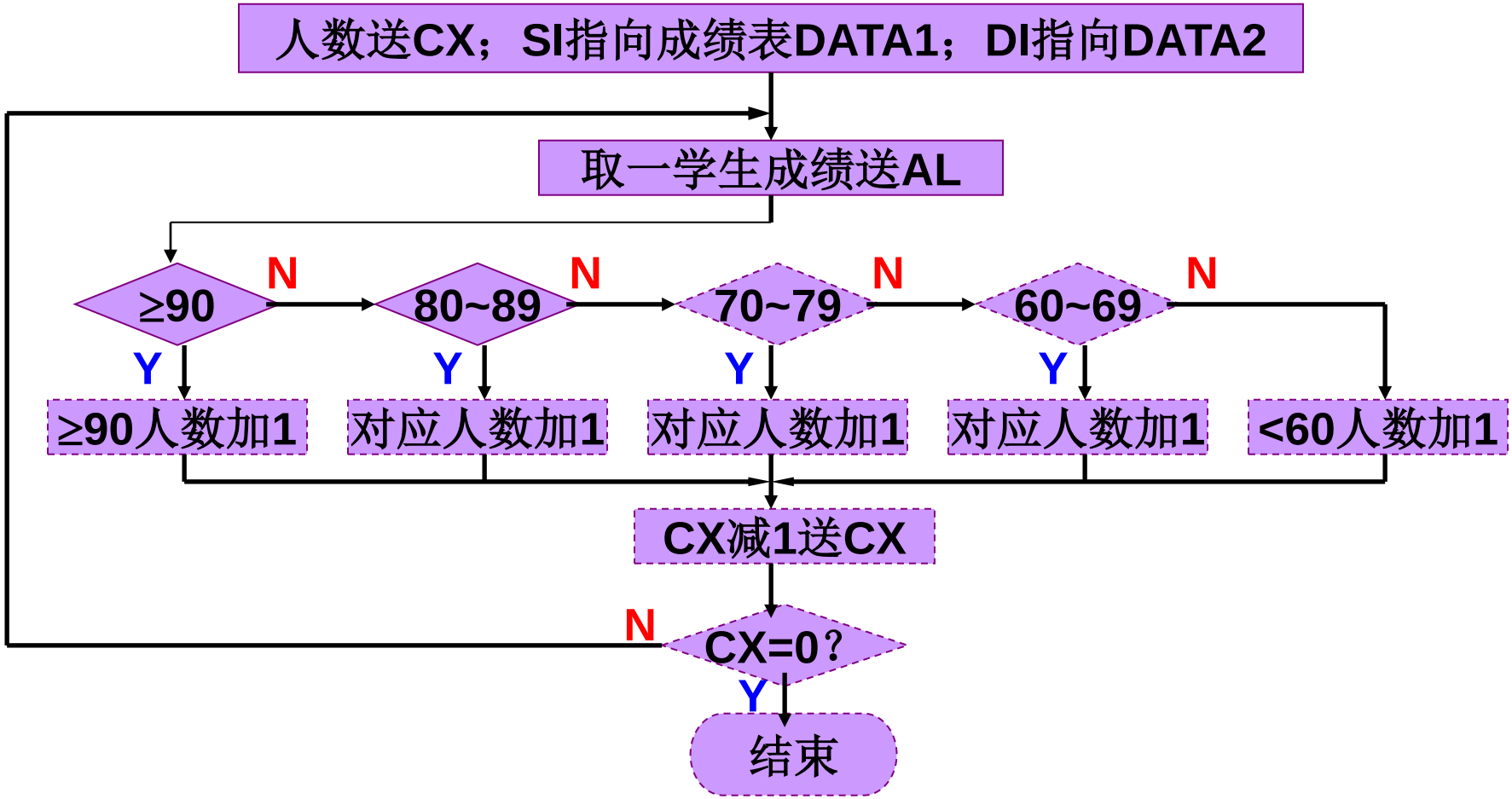
【例】用地址表法编写程序实现从低到高逐位检测一个字节数据，找到第一个非0的位数。检测时，为0，则继续检测；为1，则转移到对应的处理程序段显示相应的位数。

```
DATA SEGMENT
      NUM DB 78H
     ADTAB DW AD0，AD1，AD2，AD3，AD4，AD5，AD6，AD7；地址表
DATA ENDS
CODE SEGMENT
     ASSUME CS:CODE，DS:DATA
START:MOV AX,DATA
```

```
      MOV AL,NUM
      MOV DL,' ?'
      CMP AL,0
      JZ DISP
      MOV BX,0;BX用来记录位1的位数
AGAIN:SHR AL,1
      JC NEXT
      INC BX
      JMP AGAIN
NEXT:SHL BX,1
      JMP ADTAB[BX]
AD0: MOV DL,' 0'
      JMP DISP
AD1: MOV DL,' 1'
    JMP DISP
AD2:MOV DL,' 2' ; 第2位为1,
   JMP DISP
AD3:MOV DL,' 3'
   JMP DISP
AD4:MOV DL,' 4'
   JMP DISP
AD5:MOV DL,' 5'
   JMP DISP
AD6:MOV DL,' 6'
   JMP DISP
AD7:MOV DL,' 7'
DISP:MOV AH,2  ; 显示
     INT 21H
    MOV AH,4CH ; 返回DOS
     INT 21H
CODE ENDS
    END START
```

分支结构程序

【例】：在DATA1开始的连续80个单元中存放80位同学某门课的考试成绩（0~100）请编程序统计大于等于90，80~89分，70~79分，60~69分，小于60分的人数，分别放在同一数据段DATA2开始的5个单元中。



分支结构程序

【例】 统计BUF+1开始的N个字符组成的字符串中数字、大写字母、和其它字符的个数，并存入字符串后面的三个单元中。

分析：

（1）数字ASCII码在30H~39H之间
　　大写字母ASCII码在41H~5AH之间

（2）CX计数，初值 为BUF 单元的内容

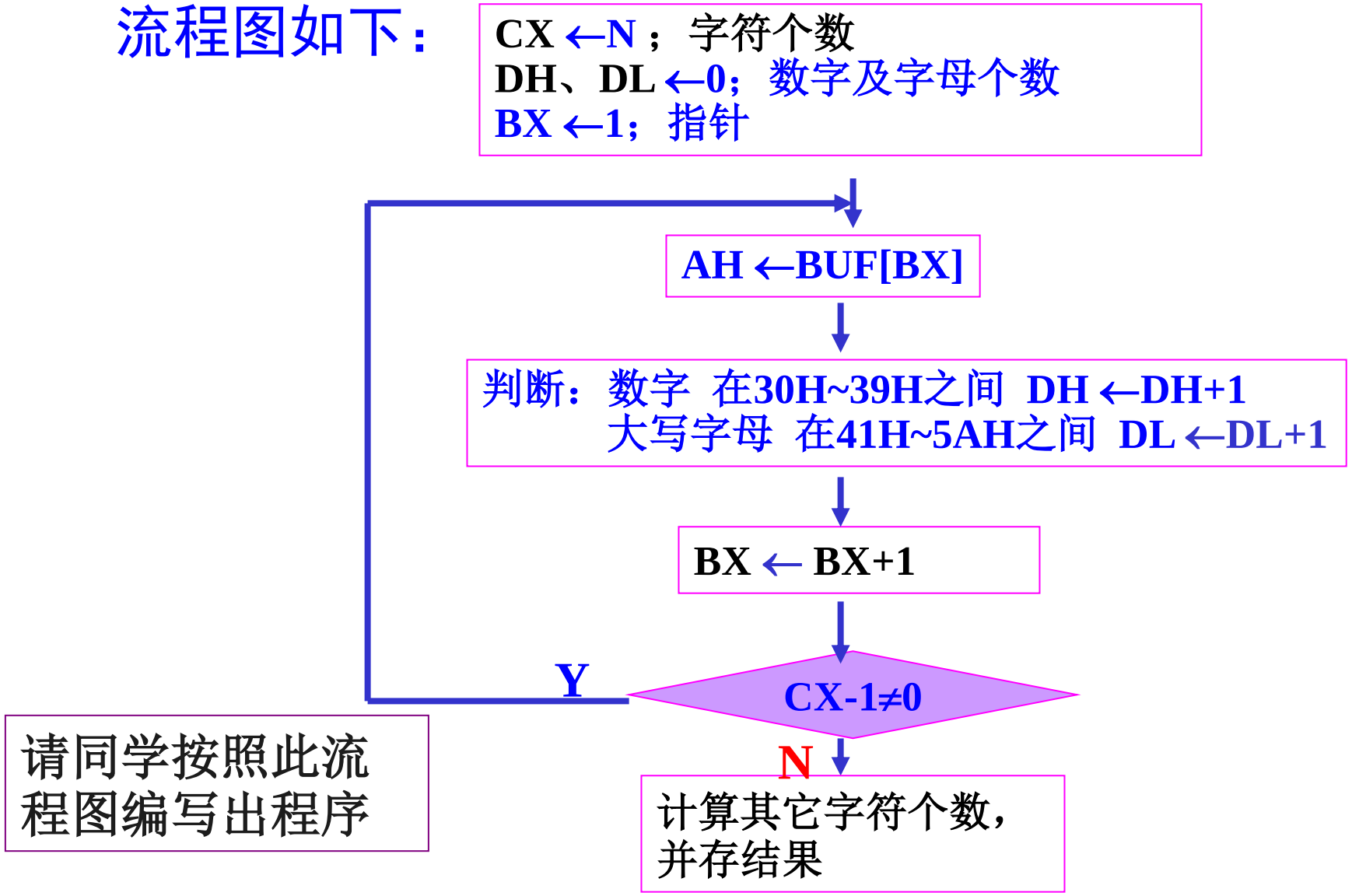
（3）BX作指针初值为1

（4）DH，DL存放数字和字母的个数

（5）其它字符的个数=N-DH-DL

分支结构程序

流程图如下：



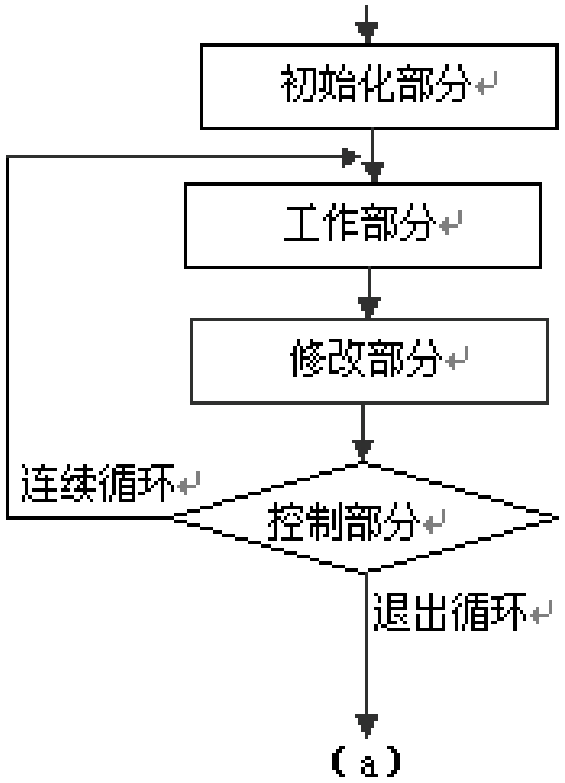
循环结构程序

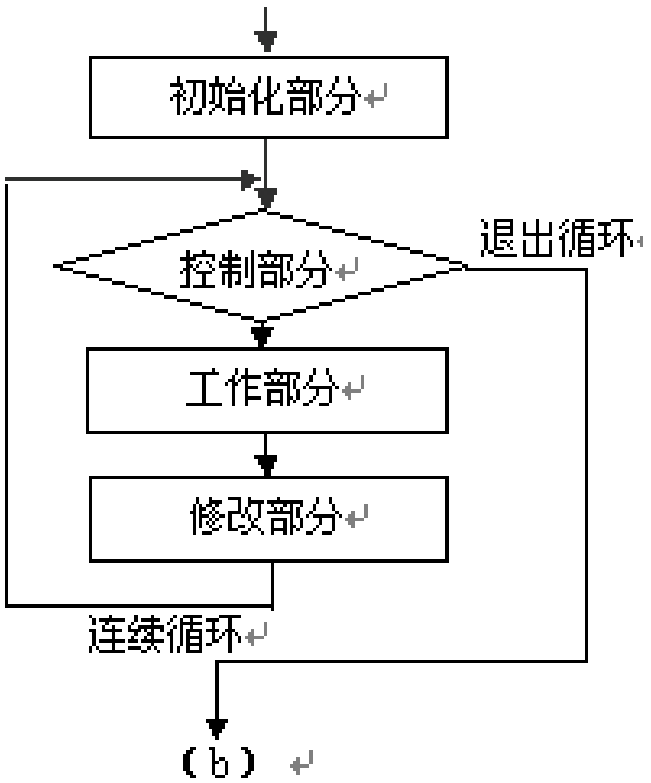
一、 循环程序的结构

循环结构是指重复执行某公共程序段若干次，直到满足条件，才结束循环操作。有单重循环和多重循环。构成：

1．初始化部分；2．循环体部分；3．循环控制部分

直到型循环：先执行循环体，再判条件是否满足。若不满足则执行循环体，否则退出循环





当型循环：当条件满足时，执行循环体，否则退出循环

循环结构程序

二、单重循环程序设计

常用循环控制方法有：

（1）计数器控制：循环次数已知的情况；

（2）条件控制：循环次数未知，循环过程中某个特定条件是否满足；

（3）开关变量控制：根据开关变量的值确定进入下一个循环体，有循环条件来控制整个循环的执行。

（4）逻辑尺控制：以某一存储单元中某位是1或0去执行不同的循环体。

循环结构程序

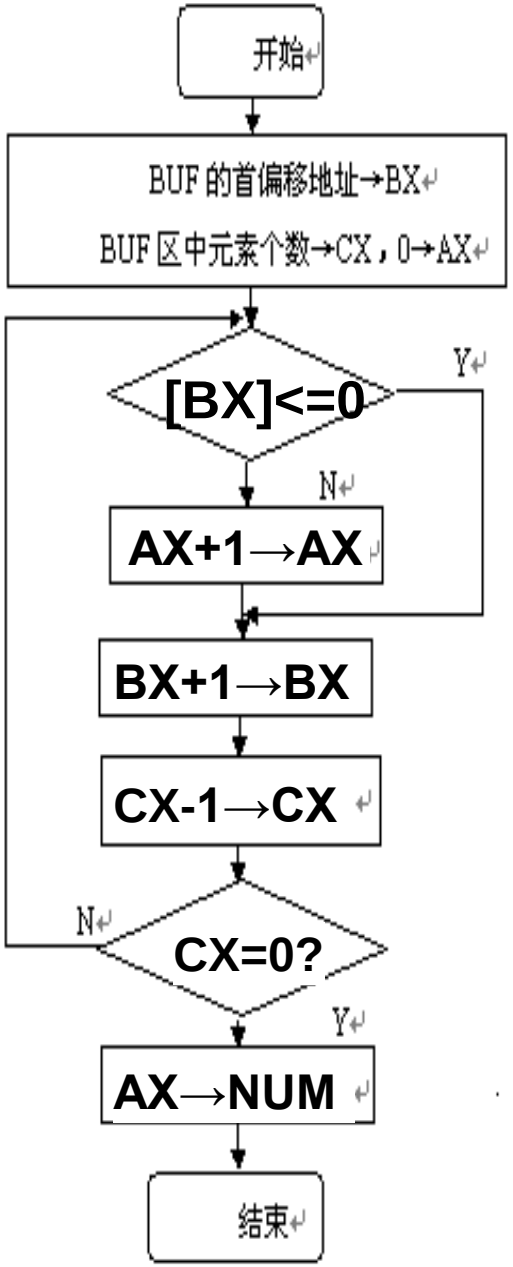
【例】 在以BUF为首址的存贮区存放一批字节数，试统计其中正数的个数。

BX指向首址BUF；

CX为计数器；

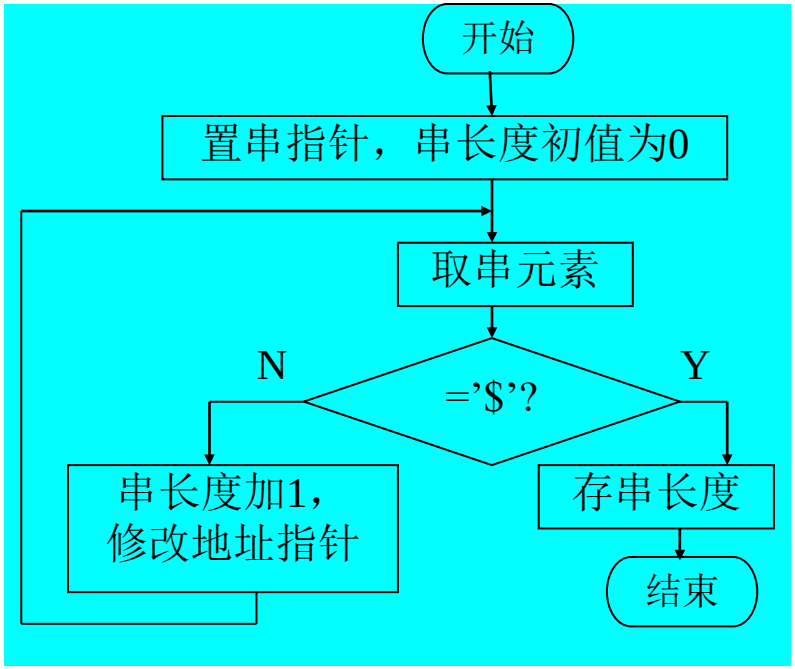
正数个数送AX，初值为0。

程序流程图



```
DATA    SEGMENT
    BUF  DB  -10,16,78,9,-36,87,44,79,36,54,97,75,3
    N    EQU   $ - BUF
    NUM  DB  ？ ；留一个单元，存放正数个数
DATA    ENDS
CODE    SEGMENT
        ASSUME  DS：DATA，CS：CODE
START：  MOV  AX，DATA
         MOV  DS，AX
         MOV  AX，0000H
         LEA   BX，BUF
         MOV  CX，N
NEXT:   CMP  [BX]，0    ；与0进行比较
        JLE  A1         ；X≤0转A1
        INC  AX         ；X＞0，AX加1
A1：     INC  BX
        LOOP  NEXT
        MOV  NUM，AX ；正数个数送NUM单元
    MOV  AH，4CH
    INT  21H
CODE    ENDS
    END  START
```

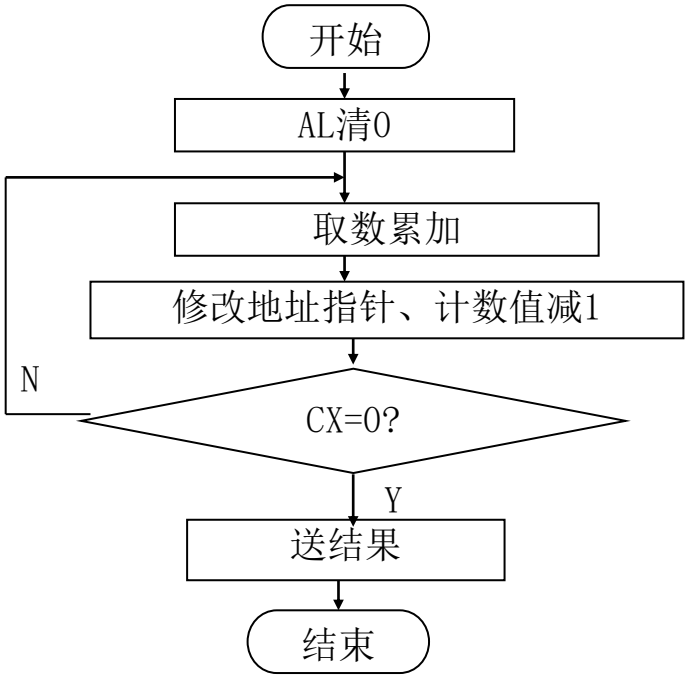
【例】在STR开始的缓冲区中存放有一个字符串，结束符‘$’，计算该字符串的长度并存入LEN单元。



```
DATA   SEGMENT
       STR DB  'JHHJKHKFHKJ$'
       LEN   DB ?
DATA   ENDS
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
       MOV DS,AX
       LEA SI,  STR
       XOR BL,BL
LOP:MOV AL,[SI]
       CMP AL,24H
       JZ STOP
       INC BL
       INC SI
       JMP LOP
STOP:MOV LEN,BL
       MOV AH,4CH
       INT 21H
CODE ENDS
       END START
```

【例】求以BUF为首地址的10个内存单元的无符号数据和。

已知其和小于等于255，将结果存入第11个内存单元



```
DATA   SEGMENT
       BUF   DB
       12H, 38H, 46H, 0BH, 09H,
       41H, 32H, 56, 02H, 26H
       RES    DB ?
DATA   ENDS
CODE   SEGMENT
       ASSUME CS:CODE, DS:DATA
START:MOV  AX, DATA
      MOV  DS, AX
      MOV  AL, 0
      MOV  CX, 0AH
      LEA  BX, BUF
   LP:ADD  AL, [BX]
      INC  BX
      LOOP LP
      MOV  RES, AL
      MOV  AH, 4CH
      INT  21H
CODE  ENDS
      END  START
```

【例】 在以BUF为首址的字存储区中存放有N个有符号数，现需将它们按大到小的顺序排列在BUF存储区中，试编写其程序。

一组数	10	8	16	90	32
第一遍	10	16	90	32	8
第二遍	16	90	32	10	8
第三遍	90	32	16	10	8

本章注意点

- 完整的汇编语言源程序结构
  - 定义逻辑段，说明段的含义，初始化段寄存器
- 伪指令
  - 数据定义方式
- 字符及字符串的输入和显示输出
  - 字符输入缓冲区的定义，输出字符串的定义
- 源程序的编写
  - 几种结构（顺序、循环、分枝等）