

汇 编 语 言

与

接 口 技 术

第4章 80X86汇编语言程序设计

主编：王让定 朱莹

宁波大学信息学院



本章主要内容



中断系统的功能

-  MASM宏汇编语句结构以及开发过程
-  MASM汇编语言表达式、运算符
-  程序段的定义和属性
-  复杂数据结构
-  宏汇编
-  常用的系统功能调用
-  汇编语言顺序程序设计

本章主要内容



中断系统的功能

-  汇编语言分支程序设计
-  汇编语言循环程序设计
-  串处理程序设计
-  子程序设计
-  高级汇编语言程序设计
-  汇编语言与C语言混合编程



常用的系统功能调用

PART 06

- DOS系统是单用户的操作系统DOS系统提供给用户的编程界面大体有如下几种：
 - 裸机层软件开发，利用芯片或板卡（适配器）支持的寄存器或内存数据区编程。需要用户非常清楚设备的硬件细节，编程较复杂，但软件实现速度最快。
 - BIOS级软件开发，利用BIOS基本输入输出系统所提供的一些服务功能编程。
 - 系统功能级软件开发，利用DOS系统提供的系统功能编程。



BIOS功能级软件接口及实现

1. BIOS级软件接口概述

- BIOS基本输入输出系统由一批子程序组成，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。
- 大多数的驱动程序以软件中断方式调用，少数由硬件中断调用。

常用BIOS服务功能

BIOS服务	功能号	功 能
打印屏幕服务	05H	将当前视频页内容送到默认打印机
视频服务	10H	为显示适配器提供I/O支持
软盘服务	13H	提供软盘的读、写、格式化、初始化、诊断
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
系统服务	15H	系统级子服务
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
日期时间服务	1AH	设置和读取时间、日期、声源等

(补充内容) X86体系架构中的中断系统 (实模式中断)

- **中断的概念：**
- 中断是指由于某种事件的发生（硬件的或软件），计算机暂停执行当前的程序，转而执行另一程序，去处理发生的事件，处理完毕后又返回原程序继续执行的过程。
- **中断源、中断请求、中断响应、中断处理、中断返回。**

(补充内容) X86体系架构中的中断系统 (实模式中断)

- **一、中断指令：**
- 在实模式下，在内存地址最低处(00000H ~ 003FFH) 设置1K字节长度的中断矢量表，表中存放系统中所有中断服务程序地址（中断矢量），中断矢量以4个字节存放在中断矢量表中，每个中断矢量包含一个中断服务程序的16位段值和16位偏移地址，因此，中断矢量表允许存放256个中断矢量。
- 中断矢量表的索引又称中断矢量地址指针由中断类型码 n 乘以4得到。

(补充内容) X86体系架构中的中断系统 (实模式中断)

- 微处理器获取中断类型码 n 的方法有3种
- 第一种是在中断指令中直接给出;
- 第二种是由外部事件引发, 向微处理器发出中断请求, MPU在中断响应周期, 通过外部数据总线获取 (外部可屏蔽中断)。
- 第三种是由内部或者外部事件引发, 向微处理器发出中断请求, 但其中断类型码是MPU已知的, 不需判别。

(补充内容) X86体系架构中的中断系统 (实模式中断)

- **中断指令**INT——软件中断触发：
- 本节介绍的中断指令用于引发内部软件中断，中断类型码在中断指令中给出。
- **中断指令格式**：INT n
- **功能**：产生中断类型码为n的软中断，该指令包含中断操作码和中断类型码两部分。
- 中断类型码n为8位，取值范围为0 ~ 255 (00H ~ FFH)。

(补充内容) X86体系架构中的中断系统 (实模式中断)

- **中断指令引发的中断执行过程为：**
- 将标志寄存器FLAGS (或EFLAGS) 压入堆栈;
- 清除TF和IF标志位;
- CS, IP/EIP压入堆栈;
- 实模式下, $n \times 4$ 获取中断矢量表地址指针; 保护模式下, $n \times 8$ 获取中断描述符表地址指针;
- 根据地址指针, 从中断矢量表或中断描述符表中取出中断服务程序地址送IP/EIP和CS中, 控制程序转移去执行中断服务程序。

(补充内容) X86体系架构中的中断系统 (实模式中断)

- **中断返回指令格式：** IRET/IRETD:
- **功能：** 该指令实现在中断服务程序结束后，返回到主程序中中断断点处，继续执行主程序。
- **中断返回执行过程：**
 - IRET指令弹出堆栈中数据送IP, CS, FLAGS;
 - IRETD指令弹出堆栈中数据送EIP, CS, EFLAGS。

(补充内容) X86体系架构中的中断系统 (实模式中断)

- 其它中断类指令如下:

格 式

说 明

CLI

清除中断允许标志, $0 \rightarrow IF$

STI

置位中断允许标志, $1 \rightarrow IF$

INTO

溢出中断

LIDT SRC

根据SRC所指存储单元内容修改中断
描述符表寄存器IDTR的基限和限长

SIDT DEST

将中断描述符表寄存器IDTR内容保
存到DEST所指的内存单元



DOS功能级软件接口及调用实例

1. DOS功能级软件接口调用概述

- **系统功能调用**是MS-DOS为程序员编写汇编语言源程序提供的一组子程序，包括设备管理、文件管理和目录管理等。
- DOS规定使用软中断指令INT21H作为进入各功能子程序的总入口，再为每个功能调用规定一个功能号，引用功能号即可进入相应的子程序入口
- **主要包括：磁盘管理、内存管理、基本输入输出管理等。**
- 子功能都有一个00H~57H的功能编号
- INT 21H

- 00H~0CH : 字符I/O管理。包括键盘、显示器、打印机、异步通信口的管理
- 0DH~24H: 文件管理。包括复位、选择磁盘, 打开、关闭、删除文件, 顺序读、写文件, 建立文件, 重命名文件, 查找驱动器分配表信息, 随机读、写文件, 查看文件长度
- 25H~26H: 非设备系统调用。包括设置中断向量, 建立新程序段
- 27H~29H: 文件管理。包括随机块读写、分析文件名
- 2AH~2EH: 非设备系统调用。包括读取、设置日期、时间
- 2FH~38H: 扩充的系统调用。包括读取DOS版本号, 终止进程, 读取中断矢量, 读取磁盘空闲空间
- 39H~3BH: 目录组。包括建立子目录, 修改当前目录, 删除目录项

- 3CH~46H: 扩充的文件管理。包括建立、打开、关闭文件, 从文件或设备读写数据, 在指定路径删除、移动文件, 修改文件属性, 设备I/O控制, 复制文件标志47H取当前目录组。
- 48H~4BH: 扩充的内存管理。包括分配内存、释放已分配的内存、分配内存块、装入或执行程序等
- 4CH~4FH: 扩充的系统调用。包括终止进程、查询子程序的返回代码、查找第一个相匹配的文件、查找下一个相匹配的文件
- 50H~53H: 扩充的系统调用。供DOS内部使用
- 54H~57H: 扩充的系统调用。包括读取校验状态、重新命名文件、设置读取日期及时间

2. 基本I/O功能调用

- 键盘输入（1号调用）
- 无回显的键盘输入（8号调用）
- 控制台输入输出（6号调用）
- 无回显的控制台输入（7号调用）
- 打印输出（5号调用）
- 输出字符串（9号调用）
- 输入字符串（0AH调用）
- 异步通信输入输出（3、4号调用）
- 日期与时间的设置与获取（2AH、2BH、2CH、2DH调用）



DOS功能级软件接口及调用实例

DOS系统功能调用的使用方法

- ① 传送入口参数到指定的寄存器中;
- ② 把要调用功能的功能号送入AH寄存器中;
- ③ 用INT21H指令转入子程序入口;
- ④ 相应的子程序运行结束后, 可以按照规定取得出口参数。

3. DOS功能调用实例（1） 演示用例 CharIO.asm

- 单字符输入（1号调用）
- 其调用格式为
- `MOV AH, 1H`
- `INT 21H`
- 该功能调用无入口参数。其功能为系统等待键盘输入，如是Ctrl+Break键则退出；否则将键入字符的ASCII码送入AL寄存器中，并且通过显示器显示该字符

- 单字符输入例程（1号调用）

`MOV AH, 1H`

`INT 21H`

`MOV achar, AL`

注意：系统会等待键盘输出

3. DOS功能调用实例（2）

- 单字符输出（2号调用）
- 其调用格式为
- `MOV DL, 'A'`
- `MOV AH, 2`
- `INT 21H`
- 执行2号系统功能调用，将置入DL寄存器中的字符（以ASCII码形式表示）通过显示器显示出来（或从打印机输出）

- 单字符输出例程（2号调用）

```
MOV AL, achar
XOR AL, 20H      ;invert
upper - lower case
MOV achar, AL
MOV DL, achar
MOV AH, 2H
INT 21H
```

3. DOS功能调用实例（3）

- 标准I/O设备RW（6号调用）
 - input without wait
- 其调用格式为
- `MOV DL, 'A'`
- `MOV AH, 2`
- `INT 21H`
- 若DL= 0FFH，从键盘输入字符，
 - ZF=0，AL中为键入的字符值，
 - ZF=1，尚无键按下；
- 若DL ≠ 0FFH，将DL中的ASCII码值对应的字符输出到屏幕上)

- 单字符输出例程（6号调用）

do_something: **XXXX**

`MOV AH, 06 ;尝试获取输出`

`MOV DL, 0FFH`

`INT 21H`

`CMP AL, 'q'`

`JE QUITDOS`

`JMP do_something`

注意：系统测试输出，没有取得结果后不等待，继续执行程序

3. DOS功能调用实例（4）

- 显示字符串（9号调用）
- 其调用格式为
 - `MOV DX, OFFSET the-string`
 - `MOV AH, 09H`
 - `INT 21H`
- 功能是将指定的内存缓冲区中的字符串从显示器显示输出
- 要求DX指向数据段内的缓冲区首地址；缓冲区中的字符串以字符‘\$’作为结束标志

- 单字符输出例程（9号调用）

```
. DATA
```

```
HINTMSG DB 0DH, 0AH, 'q touched,  
quit to dos. $'
```

```
. CODE
```

```
MOV DX, OFFSET HINTMSG
```

```
MOV AH, 09H
```

```
INT 21H
```


3. DOS功能调用实例（5） 演示用例book4_16/4-16plus.asm

- 显示字符串（0A号调用）
- 其调用格式为（见右侧）
- 功能为将键盘输入的字符串写入内存单元中
 - 缓冲区第一个字节存放规定字符串的最大字节数，
 - 第二个字节由系统送入实际键入的字符数，
 - 从第三个字节开始用于存放键入的字符串，
 - 最后通过键入回车键来表示字符串的结束
- 如果实际键入的字符数未达到最大规定数，其缓冲区的空余区间填0；如果实际键入数超过缓冲区的容量，则超出的字符自动丢失，而且响铃警告。注意，回车键值也存于缓冲区中

- 单字符输出例程（0A号调用）

. DATA

- `BUF DB 20`
- `DB ?`
- `DB 20 DUP (?)`

. CODE

- `MOV DX, OFFSET BUF`
- `MOV AH, 0AH`
- `INT 21H`

2. 视频服务

- INT 10H
 - AH寄存器选择视频服务功能
 - 待写的字符或像素值一般在AL寄存器中传递。
 - 功能调用保存BX、CX、DX及段寄存器的值。其他寄存器的内容（特别是SI、DI）不保存
 - X坐标（列号）在CX（图形功能）中或DL（正文功能）中传递
 - 显示页在BH中传递，显示页从零开始计数

例如：利用BIOS视频服务的AH=0CH子功能实现写像素点。

- 调用子程序之前，需要用DX保存行号，CX保存列号，AL保存颜色值，这是0CH子功能要求的。

```

WRITINGPIXEL      PROC      NEAR
    PUSH          AX
    MOV           AH, 0CH
    INT           10H
    POP           AX
WRITINGPIXEL      ENDP

```

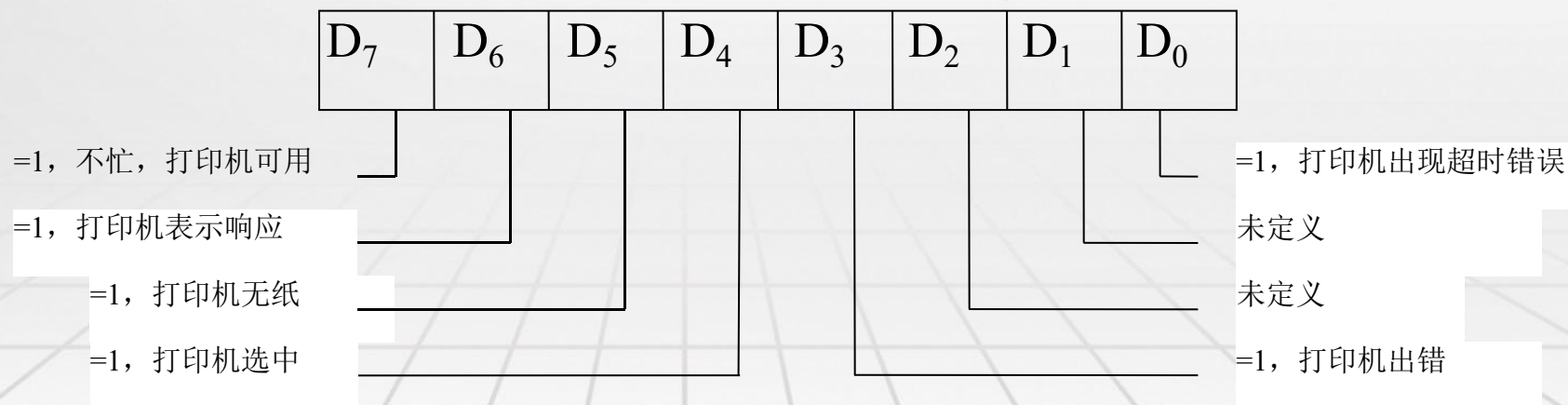
3. 键盘服务

- 键盘ISR（键盘中断服务程序） INT 09H硬件中断
- 键盘DSR（键盘设备服务程序）， INT 16H软件中断
 - BIOS键盘服务包括3个子功能：0号功能、1号功能、2号功能

子功能号	含 义	出 口 参 数
AH=0	从键盘输入一个字符	AL=ASCII码 (或0) AH=扫描码 (或扩展扫描码)
AH=1	判断键盘有无字符输	ZF=0 有键按下, 键代码保存在AX中 ZF=1 无键按下
AH=2	当前键盘特殊键状态	AL=KB-FLAG的变量

4. 并行打印机服务

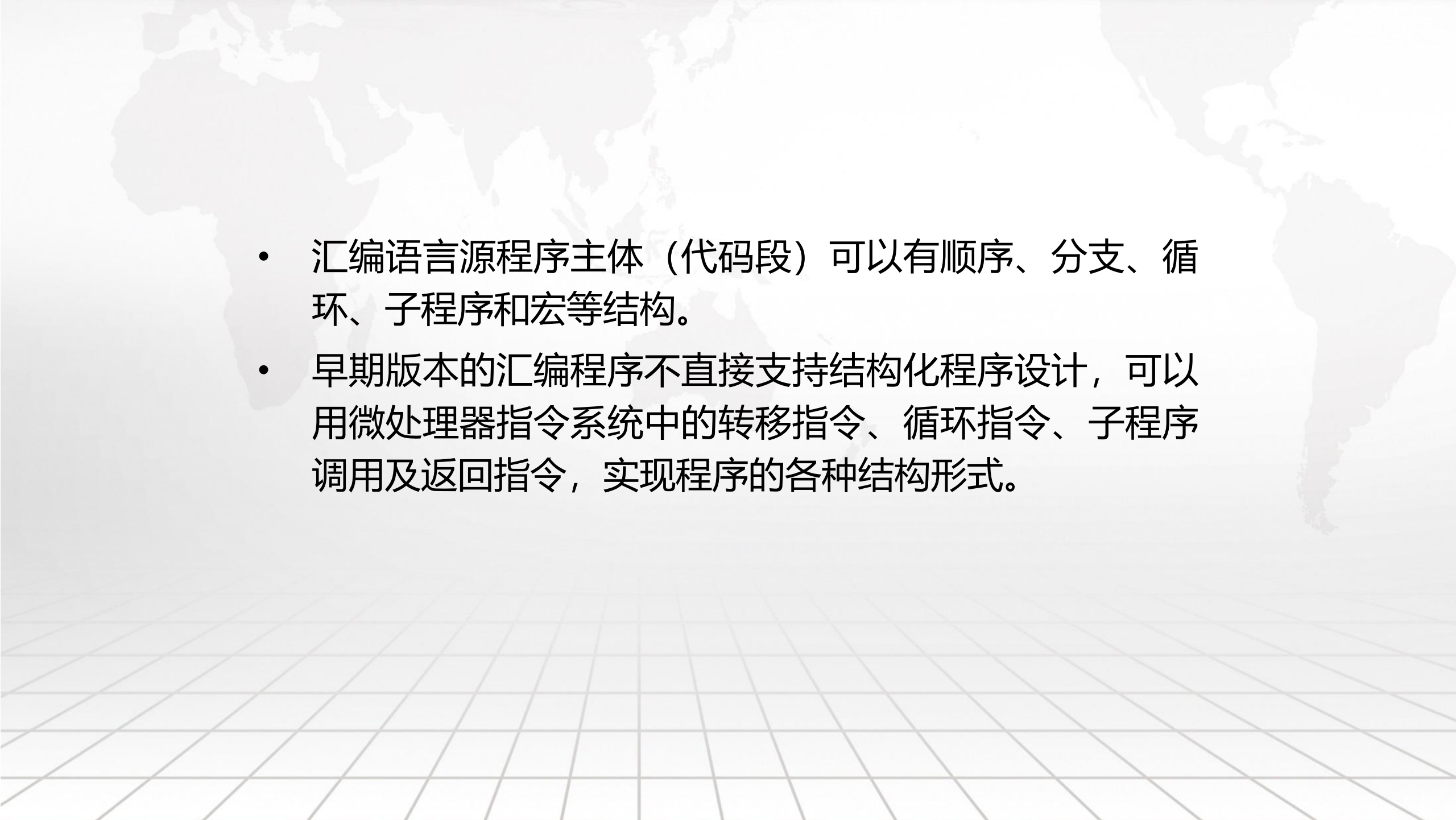
- INT 17H
 - 0号功能（给打印机传送一个字符）
 - 1号功能（初始化打印机）
 - 2号功能（读打印机状态）
- 均可通过AH返回打印机状态字节





顺序结构程序设计

PART 07

- 
- 汇编语言源程序主体（代码段）可以有顺序、分支、循环、子程序和宏等结构。
 - 早期版本的汇编程序不直接支持结构化程序设计，可以用微处理器指令系统中的转移指令、循环指令、子程序调用及返回指令，实现程序的各种结构形式。

- **例** 求两个数的平均值。这两个数分别放在X单元和Y单元中，而平均值放在Z单元中。

根据题意，所设计的程序如下：

```
.MODEL    SMALL
.STACK
.DATA
X          DB      8CH
Y          DB      64H
Z          DB      ?
```

. CODE

. STARTUP

```
MOV    AL, X      ; AL ← 8CH
ADD    AL, Y      ; AL ← 8CH+64H
MOV    AH, 00H    ; AH ← 00H
ADC    AH, 00H    ; 进位送AH
MOV    BL, 2      ; 除数2 → BL
DIV    BL         ; AX除以BL的内容,
                  ; 商 → AL, 余数 → AH
MOV    Z, AL      ; 结果送入Z单元
        .EXIT 0
```

END



分支程序设计

PART 08



无条件转移指令和条件转移指令

1. 无条件转移指令

在8086/8088系统中，程序的寻址由CS和IP两部分组成。改变CS和IP或只改变IP都能使程序转移到一个新的地址去执行。

指令格式：JMP dest

JMP reg/m16

指令功能：JMP指令无条件地转移到指令所指定的目标地址，
dest标号提供转移目的地址，或者由寄存器、存储器提供转移目的地址。

根据目标地址相对于转移指令的位置，转移可分为短转移、段内转移和段间转移。

- 短（SHORT）转移属于相对转移，指在段内短距离（-128~127）转移。
- 段内（NEAR）转移指CS值不变，只给出地址偏移值的转移；这种转移的目标地址与转移指令都在同一段内。
- 段间（FAR）转移指CS段值和IP值都发生改变的转移。在这种情况下，程序从一个段转移到另一段中的某一地址去执行，因此，JMP指令中要同时给出段值和偏移值。

例

JMP NEAR PTR TABLE[BX] ; 为段内转移,
IP ← [BX+TABLE][BX+TABLE +1]

JMP FAR PTR TABLE[BX] ; 为段间转移,
IP ← [BX+TABLE][BX+TABLE +1]

; CS ← [BX+TABLE+2][BX+TABLE +3]

在16位寻址方式下, 可以用WORD和DWORD来区分不带标号的转移是段内转移还是段间转移。

例 3. 60

JMP WORD PTR [BX] ; 为段内转移,
IP ← [BX][BX+1]

JMP DWORD PTR [BX] ; 为段间转移,
IP ← [BX][BX+1], CS ← [BX+2][BX+3]

例

DRV TBL LABEL WORD

DW DRV\$INIT

DW MED1\$CHK

⋮

DW ENTRY

⋮

ENTRY:

⋮

MOV AX, SEG DRV TBL

MOV DS, AX

MOV SI, NUMBER ; NUMBER为表中ENTRY存储地址相对

; DRV TBL的偏移量

JMP NEAR PTR DRV TBL[SI]

2. 条件转移指令

1) 无符号数条件转移指令格式及其功能

- JA/JNBE label ; 高于/不低于等于, 条件满足则转移到label指定的地址。
- JAE/JNB label ; 高于等于/不低于, 条件满足则转移到label指定的地址。
- JB/JNAE label ; 低于/不高于等于, 条件满足则转移到label指定的地址。
- JBE/JNA label ; 低于等于/不高于, 条件满足则转移到label指定的地址。
- JC label ; 有进位, 条件满足则转移到label指定的地址。
- JE/JZ label ; 等于/为0, 条件满足则转移到label指定的地址
- JNC label ; 无进位, 条件满足则转移到label指定的地址。
- JNE/JNZ label ; 不等于/不为0, 条件满足则转移到label指定的地址
- JNP/JPO label ; 非偶/奇, 条件满足则转移到label指定的地址
- JP/JPE label ; 偶/偶, 条件满足则转移到label指定的地址。

2) 带符号数条件转移指令格式及其功能

- JG/JNLE label ; 大于/不小于等于, 条件满足则转移到label指定的地址。
- JGE/JNL label ; 大于等于/不小于, 条件满足则转移到label指定的地址。
- JL/JNGE label ; 小于/不大于等于, 条件满足则转移到label指定的地址。
- JLE/JNG label ; 小于等于/不大于, 条件满足则转移到label指定的地址。
- JNO label ; 无溢出, 条件满足则转移到label指定的地址。
- JNS label ; 正数或0, 条件满足则转移到label指定的地址。
- JO label; 溢出, 条件满足则转移到label指定的地址。
- JS label; 负数, 条件满足则转移到label指定的地址。

3) 据CX中的值来决定转移的指令格式及其功能

- JCXZ label ; CX中的值为0, 则转移到label指定的地址。



分支结构程序设计举例

- 单分支IF - THEN
- 双分支IF - THEN—ELSE
- 条件转移Jcc和无条件转移JMP指令用于实现程序的分支结构。
 - JMP指令仅实现了转移到指定位置,
 - Jcc指令则可根据条件转移到指定位置或不转移而顺序执行后续指令序列。

1. 单分支结构

例 计算AX中符号数绝对值的程序段。

CMP AX, 0

JGE NONNEG ; 分支条件: $AX \geq 0$

NEG AX ; 条件不满足, 负数,
 ; 执行分支体进行求补

NONNEG: MOV RESULT, AX ; 条件满足,
 ; 为正数, 保存结果

CMP AX, 0

JL YESNEG ; 分支条件: $AX < 0$

JMP NONENG ; 条件不满足, 正数,
; 转向保存结果

YESNEG: NEG AX ; 条件满足, 为负数,
; 需要求补

NONNEG: MOV RESULT, AX ; 保存结果

选择分支条件不当, 不仅多了一个JMP指令, 而且也
容易出错。

2. 双分支结构

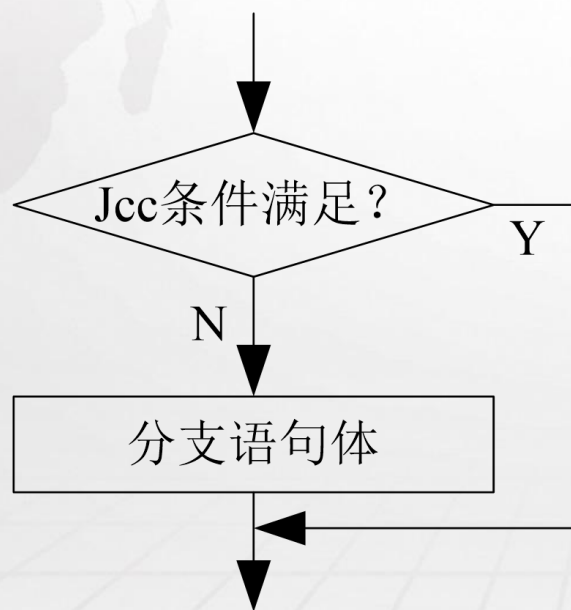
- 对于双分支程序，顺序执行的分支语句体1不会自动跳过分支语句体2，所以分支语句体1最后一定要有一条JMP指令跳过分支体2，即分支汇点处；否则将进入顺序分支语句体2而出现错误。

例 显示BX最高位的程序段。

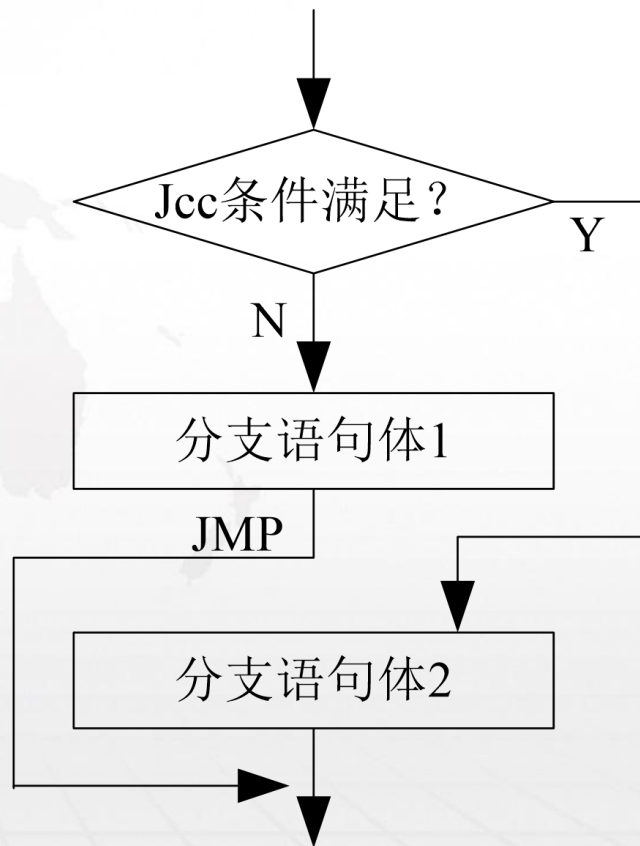
```
SHL    BX, 1    ; BX最高位移入CF标志
JC     ONE      ; CF = 1, 即最高位为1, 转移
MOV    DL, ' 0'  ; CF = 0, 即最高位为0
                ; DL ← ' 0'

        JMP     TWO      ; 一定要跳过另一个分支体
ONE:    MOV     DL, ' 1'  ; DL ← '1'
TWO:    MOV     AH, 2
        INT     21H      ; 显示
```

- 单分支与双分支



A) 单分支结构



B) 双分支结构

3. 分支程序设计

例 判断方程 $AX^2+BX + C = 0$ 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。

分析：二元一次方程有根的条件是： $B^2 - 4AC \geq 0$ 。
依据题意，首先计算出 B^2 和 $4AC$ ，然后比较两者大小，根据比较结果分别给TAG赋不同的值。

例 设计根据键盘输入的1~8数字转向8个不同的处理程序段的程序。

分析：在数据段定义一个存储区，顺序存放8个处理程序段的起始地址。由于所有程序都在一个代码段，所以，用字定义伪指令DW存入偏移地址。另外，为了具有良好的交互性，程序首先提示输入数字，然后判断是否为1~8。不是有效数字，则重新提示；若是有效数字，则形成表中的正确偏移，并按地址表跳转。



循环程序设计

PART 09



循环指令

循环指令实际上也令LOOP是条件转移指令，它用存放在CX中的数作为循环重复计数值递减计数，直到CX中的数为0时终止循环。

1) LOOP循环指令

指令格式： LOOP label

指令功能：在执行LOOP指令时，处理器先将CX中计数值减1，再判其值：如果计数值不为0，则转到LOOP指令中指出的目标语句（执行循环体内的语句）；如果计数值为0，则执行LOOP指令后的下条指令（退出循环）。如果CX的初值为0，则循环将进行 2^{16} 次。

2) L00PE/L00PZ循环指令

指令格式: L00PE label

L00PZ label

L00PE和L00PZ是同一条指令的两种不同的助记符，其指令功能是：

执行本指令时，将CX寄存器的内容减1并回送CX寄存器，ZF不受CX减1的影响。

这两条指令执行后是否循环，依下述条件判断：

如果 $CX \neq 0$ 且 $ZF=1$ ，则转移到目的标号执行， $IP \leftarrow IP（现行）+ 偏移量$ 。

如果 $CX \neq 0$ 且 $ZF=0$ ，则停止循环，按指令顺序执行。

如果 $CX=0$ （无论ZF如何）停止循环，按指令顺序执行。

3) LOOPNE/LOOPNZ循环指令

指令格式：LOOPNE/ LOOPNZ label

LOOPNE和LOOPNZ是同一条指令的两种不同的助记符，其指令功能：

将CX寄存器的内容减1并回送CX寄存器，ZF不受CX减1的影响。

- 判断是否循环的条件是：

如果 $CX \neq 0$ 且 $ZF=0$ ，继续循环，转到目的标号执行，即 $IP \leftarrow IP$ （现行）+偏移量。

如果 $CX \neq 0$ 且 $ZF=1$ ，则停止循环，按指令顺序执行。

如果 $CX=0$ 停止循环，按指令顺序执行。

注意：LOOPE/LOOPZ与LOOPNE/LOOPNZ指令对标志位没有影响



循环程序设计举例

1. 循环结构

- **循环初始部分** 开始循环准备必要的条件，如循环次数、循环体需要的初始值等。
- **循环体部分** 是循环工作的主要部分，是为完成某种特定功能而设计的重复执行的程序段。
- **修改部分** 对循环条件、相关信息（如计数器的值、操作数地址等）进行修改的程序段。
- **循环控制部分** 判断循环条件是否成立，决定是否继续循环，

2. 循环程序设计

例 计算1~100数字之和，并将结果存入字变量SUM中。

分析：程序要求 $SUM = 1 + 2 + 3 + \dots + 99 + 100$ ，这是一个典型的记数循环，完成100次简单加法。

编写一个100次的计数循环结构；循环开始前将被加数清0，加数置1，循环体内完成一次累加，每次的加数递增1。LOOP指令要求循环次数预置给CX，每次循环CX递减1。

循环体内加数就可以直接用循环控制变量CX，简化循环体。

```
.MODEL      SMALL
. STACK
. DATA
    SUM     DW    ?
. CODE
. STARTUP
        XOR    AX, AX    ; 被加数AX清0
        MOV    CX, 100
AGAIN:  ADD    AX, CX    ; 从100, 99, ..., 2, 1倒
序累加
        LOOP   AGAIN
        MOV    SUM, AX  ; 将累加和送入指定单元
. EXIT 0
END
```

例 把一个字符串中的所有大写字母改为小写字母，该字符串以0结尾。

分析：这是一个循环次数不定的循环程序结构，宜用转移指令决定是否结束循环，并应该先判断后循环。循环体判断每个字符，如果是大写字母则转换为小写，否则不予处理；循环体中具有分支结构。大小写字母的ASCII码不同之处是：大写字母D5 = 0，而小写字母D5 = 1。

```
.MODEL SMALL
.STACK
.DATA
STRING DB 'HELLO, EVERYBODY!' , 0
.STARTUP

AGAIN: MOV BX, OFFSET STRING
MOV AL, [BX] ; 取一个字符
OR AL, AL ; 是否为结尾符0
JZ DONE ; 是, 退出循环
CMP AL, 'A' ; 是否为大写A~Z
JB NEXT
CMP AL, 'Z'
JA NEXT
OR AL, 20H ; 是, 转换为小写字母 (使D5 = 1)
MOV [BX], AL ; 仍保存在原位置
NEXT: INC BX
JMP AGAIN ; 继续循环
DONE: .EXIT 0
END
```



串处理程序设计

PART 10

8086串操作指令

- 一般情况下，串操作处理的数据指在内存中地址连续的字节串（或字串），也就是处理一组数据或字符，数据串的长度最长为64KB。
- 串操作类指令要求SI为源串操作数指针，DI为目标操作数指针，每条指令一般只处理一个字节（或字）数据，而且每处理完一个数据，其地址指针会自动 ± 1 （或 ± 2 ），指向下一个要处理的数据。指针是增还是减取决于状态寄存器（FLAGS）中DF的状态（DF=0，则增址；DF=1，则减址）。指针是增（减）1还是增（减）2取决于字节操作还是字操作。
- 单独的串操作指令只能对单个串元素进行操作，只有在这些指令前加上重复前缀REP，才能实现对整个串的操作；在重复前缀的作用下，串指令的基本操作不断重复。

(1) 串传送指令

指令格式：MOVSB、MOVSW

指令功能：串传送指令MOVSB、MOVSW分别将DS：SI所指向的串元素移送到由ES：DI指向的位置。MOVSB作用于字节元素，MOVSW作用于字元素，根据DF的值确定增值或减值SI与DI。

当串传送指令与重复前缀REP一起使用时，才能完成将整个串从源存储区域传送到目的存储区域的操作。为此，程序必须在执行串传送指令前设置好CX、SI和DI的值。

(2) 串比较指令

指令格式：CMPSB、CMPSW

指令功能：串字节比较指令CMPSB把寄存器SI所指向的一个字节数据与由寄存器DI所指向一个字节数据采用相减方式比较，相减结果影响有关标志位(AF，CF，OF，PF，SF和ZF)，但不会影响两个操作数，然后根据方向标志位DF的使SI和DI之值分别增1或减1。串字比较指令CMPSW把寄存器SI所指向的一个字数据与由寄存器DI所指向的一个字数据比较，结果影响有关标志位，但不会影响两个操作数，然后根据方向标志位DF的使SI和DI之值分别增2或减2。

(3) 串扫描指令

指令格式：SCASB、SCASW

指令功能：串字节扫描指令SCASB把累加器AL的内容与由寄存器DI所指向一个字节数据采用相减方式比较，相减结果影响有关标志位(AF, CF, OF, PF, SF和ZF)，但不影响两个操作数，然后根据方向标志DF使之值增1或减1。串字扫描指令SCASW把累加器AX的内容与由寄存器DI所指向的一个字数据比较，结果影响标志，然后根据方向标志DF使DI之值增2或减2。

(4) 串装入指令

指令格式：LODSB

LODSW

指令功能：串装入指令是将寄存器SI所指的字节或字数据装入AL或AX，如果操作数的类型为字节，则采用LODSB指令；如果操作数的类型为字，则采用LODSW指令。使用上述格式的串装入指令时，仍必须先给SI赋合适的值。

(5) 串存储指令

指令格式：STOSB

STOSW

指令功能：字符串存储指令只是把累加器的值存到字符串中，即替换字符串中的一个字符。串存储指令STOSB/STOSW把累加器AL/AX的内容送到寄存器DI所指向的存储单元中，然后根据方向标志DF使DI值增1/2或减1/2。

(6) 重复前缀

指令格式：REP

REPE/REPZ

REPNE/REPNZ

这三种重复前缀指令不能单独使用，只能加在串操作指令之前，用来控制跟在其后的字符串操作指令，使之重复执行，重复前缀不影响标志位。

1) REP

无条件重复前缀。

指令功能：该指令无条件执行其后的指令，由CX寄存器指定重复次数，每执行一次，则 $CX \leftarrow CX - 1$ ，若 $CX \neq 0$ ，继续执行其后的指令，一直到 $CX = 0$ 为止。REP前缀常与MOVS和STOS串操作指令配合使用。

2) REPE/REPZ 有条件重复前缀。

- 指令功能：该指令条件为相等/结果为0时重复前缀，该前缀与CMPS和SCAS串操作指令配合使用。
- 只有当CX \neq 0且ZF=1（表示两个操作数比较相等）时，继续执行其后的比较或扫描指令；否则当CX=0或者ZF=0（表示两个操作数比较不相等）时，则停止执行其后的字符串指令，结束该操作。

3) REPNE/REPNZ

有条件重复前缀。

指令功能：该指令条件为不相等/结果不为0时重复前缀。

- 有关串操作指令的重复前缀、操作数以及地址指针所用的寄存器等情况归纳如下，见表3.3所示。

指令	重复前缀	操作数	地址指针寄存器
MOVS	REP	目标	ES:DI
		源	DS:SI
CMPS	REPE/REPZ	目标	DS:SI
		源	ES:DI
SCAS	REPE/REPZ	目标	ES:DI
LODS	无	源	DS:DI
STOS	REP	目标	ES:DI



80386串操作指令

位串操作指令

- 串操作指令包括 MOVSB、CMPSB、SCASB、LODSB、STOSB、INSB、OUTSB。
- 前5个指令是8086中原有的，在80386指令系统增加了对32位寄存器的支持，如MOVSB可以写成MOVSBQ、MOVSBW、MOVSD。它们的源变址寄存器、目的变址寄存器及计数器是用ESI、EDI、ECX还是用SI、DI、CX由它所在的段决定。若是32位段，则使用前者，否则使用后者。



串操作程序设计举例

例

S_POINT DD S_ADDR

D_POINT DD D_ADDR

S_ADDR DB...

⋮

D_ADDR DB...

⋮

LDS SI, S_POINT

LES DI, D_POINT

CLD

MOV CX, LENGTH

REP MOVSB

； 源数据区

； 目的数据区

； 置全地址指针DS: SI值

； 置全地址指针ES: DI值

； 将方向标志DF清0

； 置串长度值

例

把DATA1段从0020H开始的30H个字节串和DATA2段从0100H开始的30H个字节串进行比较。

解：依题意，程序段为：

MOV AX, DATA1

MOV DS, AX

MOV AX, DATA2

MOV ES, AX

MOV SI, 0020H

MOV DI, 0100H

MOV CX, 0030H

CLD

REPE CMPS

例 编写程序查找STR1串中是否有字母‘J’，如果有输出‘Y’，否则输出‘N’。

DATAS SEGMENT

STR1 DB ‘ASDFGHJK’

DATAS ENDS

CODES SEGMENT

ASSUME CS:CODES, DS:DATAS, ES:

DATAS

START:

MOV AX, DATAS

MOV DS, AX

MOV ES, AX

MOV DI, OFFSET STR1

MOV AL, ‘J’

MOV CX, 8

REPNZ SCASB; 字符搜索

JNZ JM1

MOV DL, ‘Y’

MOV AH, 2

INT 21H

JMP EXT

JM1:MOV DL, ‘N’

MOV AH, 2

INT 21H

EXT:MOV AH, 4CH

INT 21H

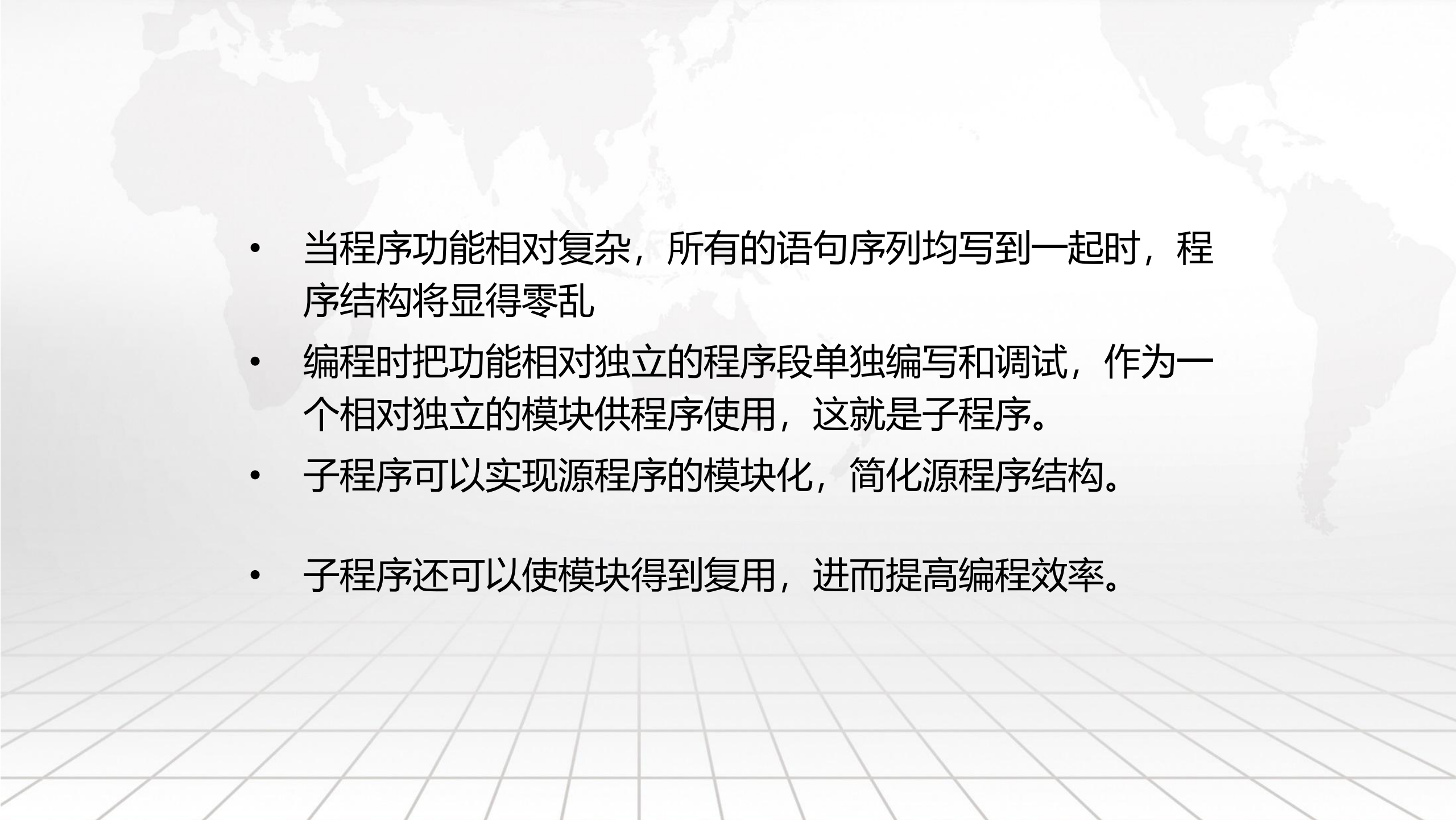
CODES ENDS

END START



子程序设计

PART 03

- 
- 当程序功能相对复杂，所有的语句序列均写到一起时，程序结构将显得零乱
 - 编程时把功能相对独立的程序段单独编写和调试，作为一个相对独立的模块供程序使用，这就是子程序。
 - 子程序可以实现源程序的模块化，简化源程序结构。
 - 子程序还可以使模块得到复用，进而提高编程效率。



子程序的定义与调用

1. 子程序的定义

过程名 PROC [NEAR/FAR]

过程体

过程名 ENDP

2. 过程的调用与返回

- 由指令CALL和RET来完成
- 要保证RET指令执行前堆栈栈顶的内容刚好是过程返回的地址
- 注意寄存器的保护与恢复

子程序调用与返回指令

(1) 调用指令CALL

用于调用子程序。

1) 段内直接调用

指令格式：CALL DST

要求：DST为子程序名。

指令功能：调用一个近子程序，该子程序在本段内。指令汇编后，得到CALL的下一条指令与被调用的子程序入口地址之间的16位相对偏移量DISP。段内调用指令功能是将当前指令指针IP内容自动压入堆栈，堆栈指针SP自动减2，然后将相对偏移量DISP送到IP中，使程序控制转移到调用的子程序。

2) 段内间接调用

指令格式: CALL SRC

要求: SRC为16位寄存器或者各种寻址方式的存储器操作数。

指令功能: 将指令指针寄存器IP的内容自动压入堆栈, 然后将SRC的内容传送到IP, 转到子程序的入口地址去执行。

例 CALL CX ; 子程序入口地址的偏移量为CX的内容

CALL WORD PTR [BX+SI+20H] ; 子程序入口地址:

$IP \leftarrow [BX+SI+20H]$ 单元中的字数据

3) 段间直接调用

指令格式: CALL DST

要求: DST给出子程序入口地址的完整信息, 即段基址CS和偏移量IP。

指令功能: 把现行CS和IP值压入堆栈, 再把子程序对应段基址和偏移地址分别送入CS和IP, 程序转向子程序执行。

例 CALL 2500H:1000H ; 子程序入口地址为
2500H:1000H

CALL FAR PTR PROCNAME ; 子程序入口地址在另一段内

例 假设在存储器中存有两个目标子程序地址 SIP0和SIP1，序号分别为0，1，下面一段程序根据SI寄存器的赋值，能调用不同目标子程序。

```
PROCTABL DD SIP0
```

```
DD SIP1
```

```
⋮
```

```
MOV AX, SEG PROCTABL
```

```
MOV DS, AX
```

```
MOV SI, ENTRY
```

； ENTRY是目标子程序的序号的4倍，

； 即相对于PROCTABL的字节地址

```
CALL FAR PTR PROCTABL [SI]
```

(2) 返回指令RET

1) 段内返回

指令格式：RET

指令功能：把保护堆栈区的断点偏移地址送入指令指针寄存器IP，返回调用程序的地方继续执行。

2) 段间返回

指令格式：RET

指令功能：把保护堆栈区的断点偏移地址送入指令指针寄存器IP，段地址送到CS寄存器，返回调用程序的地方继续执行。使用该指令时，应注意保证CALL指令的类型与子程序中RET指令的类型匹配，以免发生返回地址错误。如果在定义子程序中，该子程序定义为远子程序FAR，对应的返回指令RET属于段间返回，若子程序定义为近子程序NEAR，对应的返回指令RET属于段内返回。

3) 带参数返回

指令格式：RET n

注意：n为常数或表达式，n只能是16位偶数，不能为奇数。

该指令也分为段内（近子程序）返回和段间（远子程序）返回，其操作与返回指令RET基本相同，不同之处是最后将修改堆栈指针SP，即 $SP \leftarrow SP + n$ 。

3. 子程序设计举例

例 编制一个过程把AL寄存器内的二进制数用十六进制形式在屏幕上显示出来。

分析：AL中8位二进制数对应2位十六进制数，先转换高4位成ASCII码并显示，然后转换低4位并显示。屏幕显示采用02号DOS功能调用。



子程序的参数传递

- 主程序在调用子程序时，通常需要向其提供一些数据，对于子程序来说就是入口参数（输入参数）
- 子程序执行结束也要返回给主程序必要的的数据，这就是子程序的出口参数（输出参数）。
 1. 用寄存器传递参数
 2. 用变量传递参数
 3. 用堆栈传递参数

1. 用寄存器传递参数

例 设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中。

分析：子程序完成元素求和，主程序需要向它提供入口参数，使得子程序能够访问数组元素。子程序需要回送求和结果这个出口参数。本例我们采用寄存器传递参数。

主程序

```
.MODEL      SMALL
.STACK
.DATA
    COUNT    EQU    10          ; 数组元素个数
    ARRAY    DB  12H, 25H, 0F0H, 0A3H, 3, 68H, 71H
                DB  0CAH, 0FFH, 90H      ; 数组
    RESULT   DB      ?           ; 校验和
.CODE
.STARTUP    ; 设置入口参数（含有DS←数组的段地址）
    MOV     BX, OFFSET ARRAY    ; BX←数组的偏移地址
    MOV     CX, COUNT          ; CX←数组的元素个数
    CALL    CHECKSUMA          ; 调用求和过程
    MOV     RESULT, AL         ; 处理出口参数
.EXIT 0
```

- 子程序

```
CHECKSUMA PROC
```

```
        XOR     AL, AL      ; 累加器清0
```

```
SUMA:   ADD     AL, [BX]; 求和
```

```
        INC     BX        ; 指向下一个字节
```

```
        LOOP    SUMA
```

```
        RET
```

```
CHECKSUMA ENDP
```

```
END
```

2. 用变量传递参数（同上例）

用COUNT，ARRAY和RESULT变量。主程序只要设置数据段DS，就可以调用子程序；子程序直接采用变量名存取数组元素。

```
.MODEL    SMALL
.STACK
.DATA
    COUNT    EQU    10          ; 数组元素个数
    ARRAY    DB  12H, 25H, 0F0H, 0A3H, 3, 68H, 71H
              DB  0CAH, 0FFH, 90H          ; 数组
    RESULT   DB      ?          ; 校验和
.CODE
.STARTUP                                ; 含有DS ← 数组的段地址
    CALL    CHECKSUMB          ; 调用求和过程
.EXIT 0
```




```
CHECKSUMB      PROC
PUSH          AX
PUSH          BX
PUSH          CX
XOR           AL, AL                ; 累加器清0
MOV           BX, OFFSET ARRAY      ; BX ← 数组的偏移地
MOV           CX, COUNT             ; CX ← 数组的元素个数
SUMB: ADD      AL, [BX]              ; 求和
      INC      BX
      LOOP     SUMB
MOV           RESULT, AL            ; 保存校验和
POP           CX
POP           BX
POP           AX
RET
CHECKSUMB      ENDP
END
```

3. 用堆栈传递参数（同上例）

- 通过堆栈传递参数，主程序将数组的偏移地址和元素个数压入堆栈，然后调用子程序；
- 子程序通过BP寄存器从堆栈相应位置取出参数（非栈顶数据），求和后用AL返回结果。
- 由于共用数据段，所以没有传递数据段基地址。本例利用堆栈传递入口参数，但出口参数仍利用寄存器传递。

```
.MODEL SMALL
.STACK
.DATA
COUNT EQU 10 ; 数组元素个数
ARRAY DB 12H, 25H, 0F0H, 0A3H, 3, 68H, 71H, 0CAH, 0FFH,
90H ; 数组
RESULT DB ? ; 校验和
.CODE
.STARTUP
MOV AX, OFFSET ARRAY ; 设置入口参数
PUSH AX ; 压入数组的偏移地
址
MOV AX, COUNT
PUSH AX ; 压入数组的元素个数
CALL CHECKSUMC ; 调用求和过程
ADD SP, 4 ; 主程序平衡堆栈
MOV RESULT, AL ; 保存校验和
.EXIT 0
```

```
CHECKSUMC      PROC
    PUSH BP
    MOV BP, SP    ; BP指向当前栈顶, 用于取出入口参数
    PUSH BX      ; 保护使用的BX和CX寄存器
    PUSH CX
    MOV BX, [BP+6] ; BX ← SS: [BP+6] (数组的偏移地址)
    MOV CX, [BP+4] ; CX ← SS: [BP+4] (数组的元素个数)
    XOR AL, AL    ; 累加器清0
    SUMC: ADD AL, [BX] ; 求和: AL ← (AL) + DS: [BX]
    INC BX
    LOOP SUMC
    POP CX        ; 恢复寄存器
    POP BX
    POP BP
    RET
CHECKSUMC      ENDP
END
```



子程序的嵌套、递归与重入

- 1. 子程序的嵌套

- 子程序内包含有子程序的调用就是子程序嵌套。
- 嵌套深度（即嵌套的层次数）逻辑上没有限制，
- 由于子程序的调用需要在堆栈中保存返回地址以及寄存器等数据，因此实际上受限于开设的堆栈空间。

我们可以写成过程，形成过程（子程序）嵌套：

```
ALDISP PROC                                ; 显示AL中的2位十六进制数
    PUSH    AX                             ; 保护入口参数
    PUSH    CX
    PUSH    AX                             ; 暂存数据
    MOV     CL, 4
    SHR     AL, CL; 转换AL的高4位
    CALL    HTOASC                          ; 子程序调用（嵌套）
    POP     AX                             ; 转换AL的低4位
    CALL    HTOASC                          ; 子程序调用（嵌套）
```

；显示AL中的2位十六进制数

; 保护入口参数

PUSH AX

；暂存数据

SHR AL, CL; 转换AL的高4位

子程序调用（嵌套）

；转换AL的低4位

子程序调用（嵌套）

POP AX

; 子程序返回

ENDP


```
HTOASC PROC ; 将AL低4位表达的一位十六进制数转换为ASCII码
        PUSH     AX      ; 保护入口参数
        PUSH     BX
        PUSH     DX
        MOV BX, OFFSET ASCII ; BX指向ASCII码表
        AND AL, 0FH      ; 取得一位十六进制数
        XLAT      ES: ASCII ; 换码: AL ← CS: [BX+AL]
        MOV      DL, AL ; 显示
        MOV AH, 2
        INT 21H
        POP DX
        POP BX
        POP AX
        RET
ASCII DB 30H, 31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H
      DB 41H, 42H, 43H, 44H, 45H, 46H; 子程序的数据区
HTOASC ENDP
```

2. 子程序的递归

- 子程序直接或间接地嵌套调用自身时称为递归调用
- 含有递归调用的子程序称为递归子程序。
- 递归子程序必须采用寄存器或堆栈传递参数，递归深度受堆栈空间的限制。

例 编制计算 $N! = N \times (N-1) \times (N-2) \times \cdots \times 2 \times 1$
($N \geq 0$) 的程序

分析：已知递归定义

$$N! = \begin{cases} N \times (N-1)! & N > 0 \\ 1 & N = 0 \end{cases}$$

- 求 $N!$ 可以设计成输入参数为 N 的递归子程序
- 每次递归调用的输入参数递减1。
- 如果 $N > 0$ ，则由当前参数 N 乘以递归子程序返回值得到本层返回值；
- 如果递归参数 $N = 0$ ，得到返回值为1

堆栈段

	栈底
3	
IP (主程序)	
AX (原始)	
BP (原始)	←BP1
2	
IP	
AX	
BP1	←BP2
1	
IP	
AX	
BP2	←BP3
0	
IP	
AX	
BP3	←BP4

3. 子程序的重入

- 子程序的重入是指子程序被中断后又被中断服务程序所调用
- 能够重入的子程序称为可重入子程序。
- 在子程序中，注意利用寄存器和堆栈传递参数和存放临时数据，而不要使用固定的存储单元（变量），就能够实现重入。

子程序的应用

例 从键盘输入有符号的10进制数的子程序。

分析：子程序从键盘输入一个有符号的10进制数。负数用“-”引导，正数直接输入或用“+”引导。子程序还包含将ASCII码转换为二进制数的过程，其算法如下：

- ①首先判断输入正数还是负数，并用一个寄存器记录下来；
- ②接着输入0~9数字（ASCII码），并减30H转换为二进制数；
- ③然后将前面输入的数值乘10，并与刚输入的数字相加得到新的数值；
- ④重复②、③步，直到输入一个非数字字符结束；
- ⑤如果是负数进行求补，转换成补码；否则直接将数值保存。

例 向显示器输出有符号十进制数的子程序。

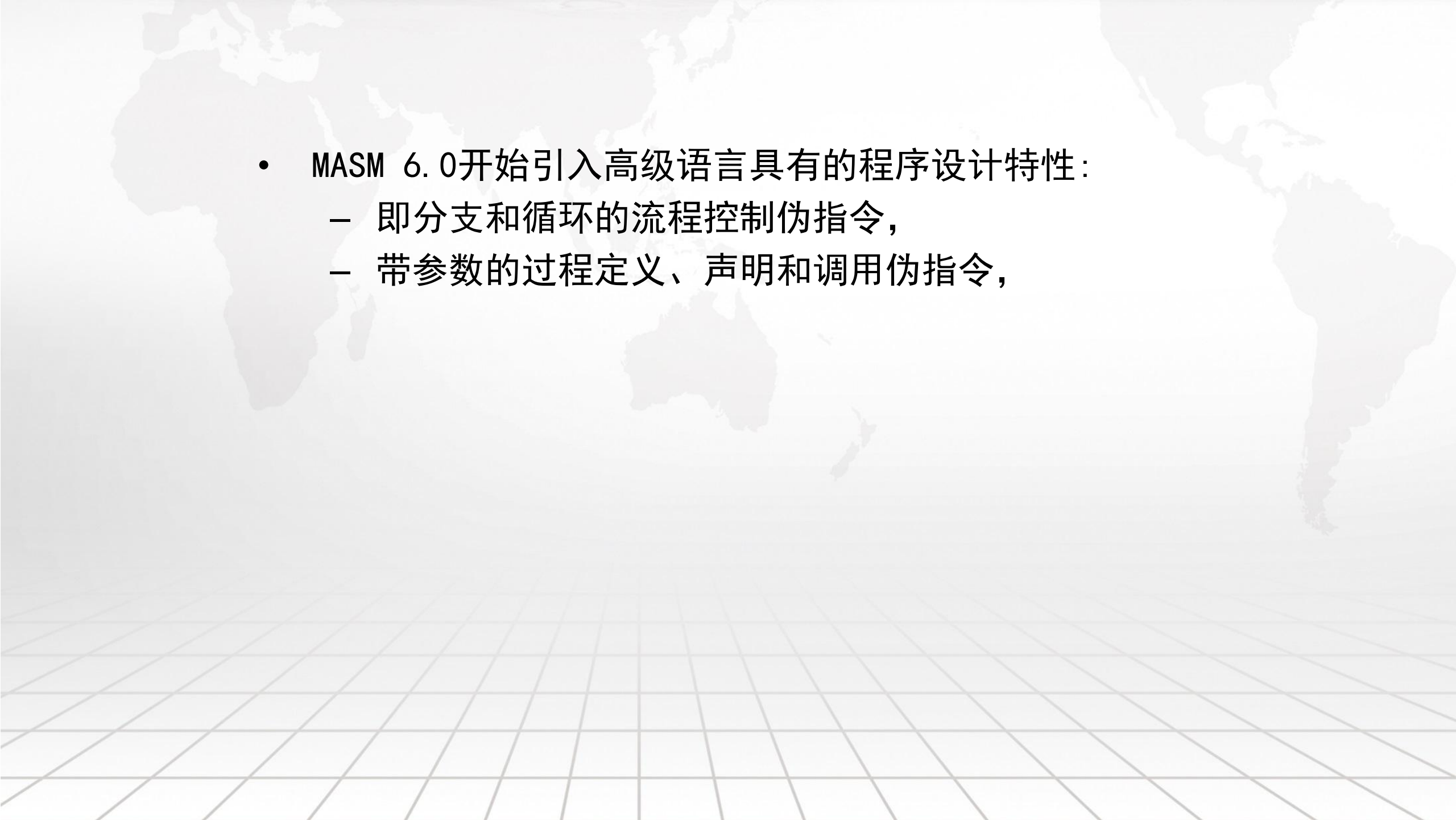
分析：子程序在屏幕上显示一个有符号十进制数，负数用“-”引导。子程序还包含将二进制数转换为ASCII码的过程，其算法如下：

- ① 首先判断数据是零、正数或负数，是零显示“0”并退出；
- ② 是负数，显示“-”，求数据的绝对值；
- ③ 接着数据除以10，余数加30H转换为ASCII码压入堆栈；
- ④ 重复③步，直到余数为0结束；
- ⑤ 依次从堆栈弹出各位数字，进行显示。



高级汇编语言程序设计

PART 04

- 
- MASM 6.0开始引入高级语言具有的程序设计特性：
 - 即分支和循环的流程控制伪指令，
 - 带参数的过程定义、声明和调用伪指令，



条件控制伪指令

MASM 6.0引入. IF, . ELSEIF, . ELSE和. ENDIF伪指令

. IF条件表达式

分支体1

[. ELSEIF条件表达式

分支体2]

[. ELSE

分支体N]

. ENDIF

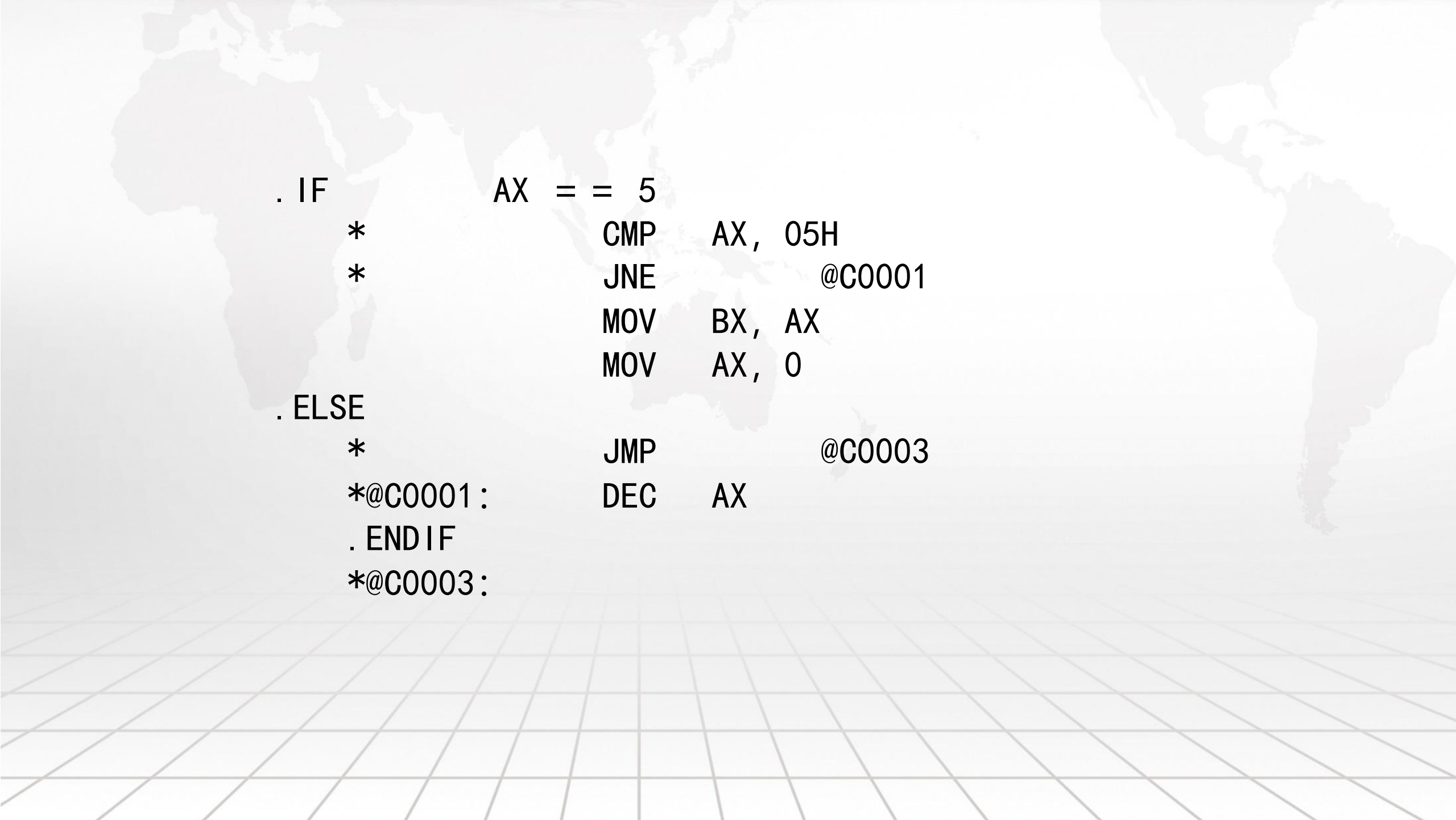
例如，求AX绝对值的单分支结构程序。

```
. IF      AX<0      ; 等价于. IF SIGN?  
    NEG    AX      ; 满足，求补  
. ENDIF  
  
    MOV    RESULT, AX
```



```
. IF      AX == 5
        MOV    BX, AX
        MOV    AX, 0
. ELSE
        DEC    AX
. ENDIF
```

LST文件中，汇编以后代码如下：



```
. IF      AX == 5
    *      CMP    AX, 05H
    *      JNE    @C0001
          MOV    BX, AX
          MOV    AX, 0
. ELSE
    *      JMP    @C0003
    *@C0001:  DEC    AX
. ENDIF
    *@C0003:
```



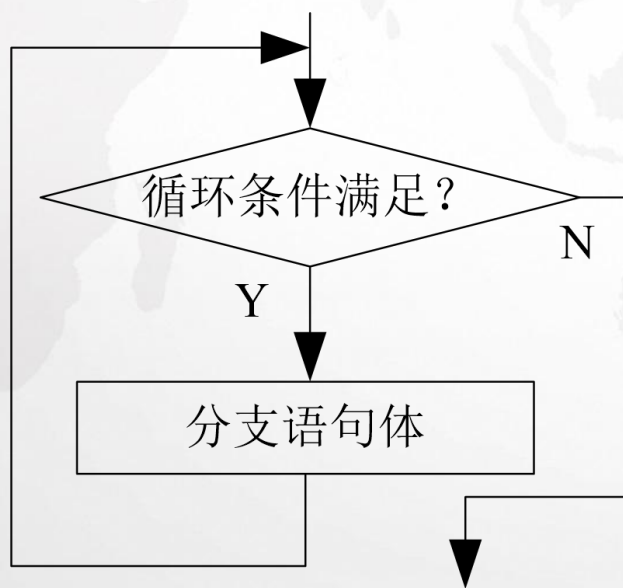

循环控制伪指令

1. WHILE结构的循环控制伪指令

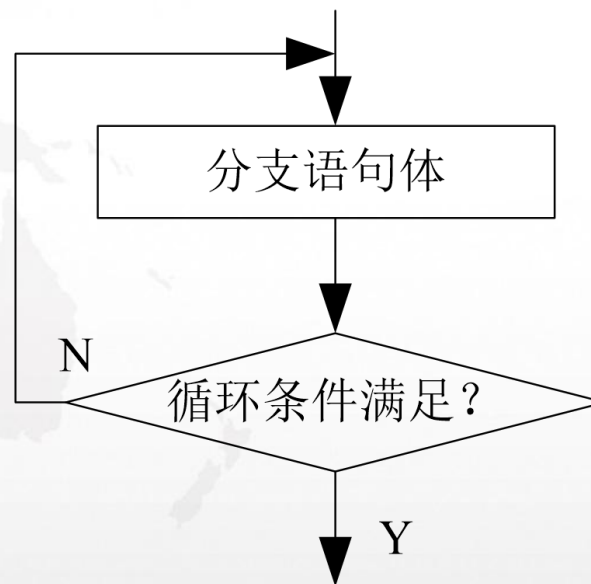
```
. WHILE 条件表达式  
    循环体  
. ENDW
```

2. UNTIL结构的循环控制伪指令

```
. REPEAT                ; 重复执行循环体  
    循环体  
. UNTIL 条件表达式    ; 直到条件为真
```



A) WHILE 循环结构



B) UNTIL 循环结构

过程声明和过程调用伪指令

1. 过程定义伪指令

在MASM 6.x中，带有参数的过程定义格式如下：

过程名 PROC [调用距离] [语言类型] [

[<起始参数>] [USES寄存器列表]

[, 参数: [类型]] . . .

LOCAL 参数表

. . . ; 汇编语言语句

过程名 ENDP

NEAR

/FAR

任何有效的类型

作用范围

- 默认是PUBLIC，表示其它模块可见
- PRIVATE表示对外不可见；
- EXPORT隐含有PUBLIC和FAR，表示该过程应该放置在导出表（Export entry table）中

2. 过程局部变量定义伪指令

- `LOCAL` 变量名[个数] [: 类型] [, . . .]

3. 过程声明伪指令

PROTO是一个过程声明伪指令，用于事先声明过程的结构。

过程名 PROTO [调用距离] [语言类型] [, 参数: [类型]]

4. INVOKE过程调用伪指令

- CALL是进行子程序调用的硬指令。
- 与PROTO配合使用的过程调用伪指令是INVOKE，它的格式如下：
 - INVOKE 过程名[, 参数, . . .]



汇编语言与C语言混合编程

PART 13



C程序嵌入汇编

C程序内嵌入汇编指令的方法简单明快，不必考虑两者间的接口，省去了独立汇编和连接的步骤，从而使用更加方便灵活，在内嵌汇编语句中可以通过名字引用C语言有效的标号、常量、变量名、函数名、宏等信息。

(1) 在嵌入的汇编指令前必须用关键字asm。

说明asm <操作码><操作数><;或换行>。

其中，操作码是有效的80X86指令及某些汇编伪指令，如db , dw, dd及extern。操作数可以是C语言中的常量、变量和标号，也可以是操作码可接受的数。内嵌的汇编指令用分号或换行作为结束。需要注意的是不能像MASM中那样用分号作为注释的开始，必须用C方式即/*... */来标出注释行。一条汇编指令不能超越两行，但同一行中可有多条汇编指令。

例4. 52

```
asm  mov ax, ds ; /* ds-> ax * /
```

```
asm  push ds ;
```

或者用换行结束也可以，如下：

```
asm mov ax, ds
```

```
asm push ds
```

(2) 内嵌汇编指令中的操作数

内嵌汇编指令的操作数可以是C语言程序中的符号，编译时它会自动转换为适当的操作数。当汇编指令中使用寄存器名时，只能是80X86的寄存器名，并且不分大小写。另外，在C语言中，DI和SI两个寄存器常被用来存放Register变量，只有在C程序中没有指定Register变量时，内嵌汇编程序才可以任意使用这两个寄存器。

(3) 汇编指令操作数可以是结构数据。

例4. 53

```
struct student  
{  
    int num ;  
    char name[10] ;  
} stu;
```

在汇编指令中可以用结构体的成员作为操作数

```
asm mov ax, stu.num
```


(4) 转移指令的执行。

内嵌汇编指令可以使用任何有条件和无条件转移汇编指令，他们只能在函数体内有效，不允许段间转移。由于在asm语句中无法给出标号，所以转移指令只能使用C语言中的goto语句用的标号。

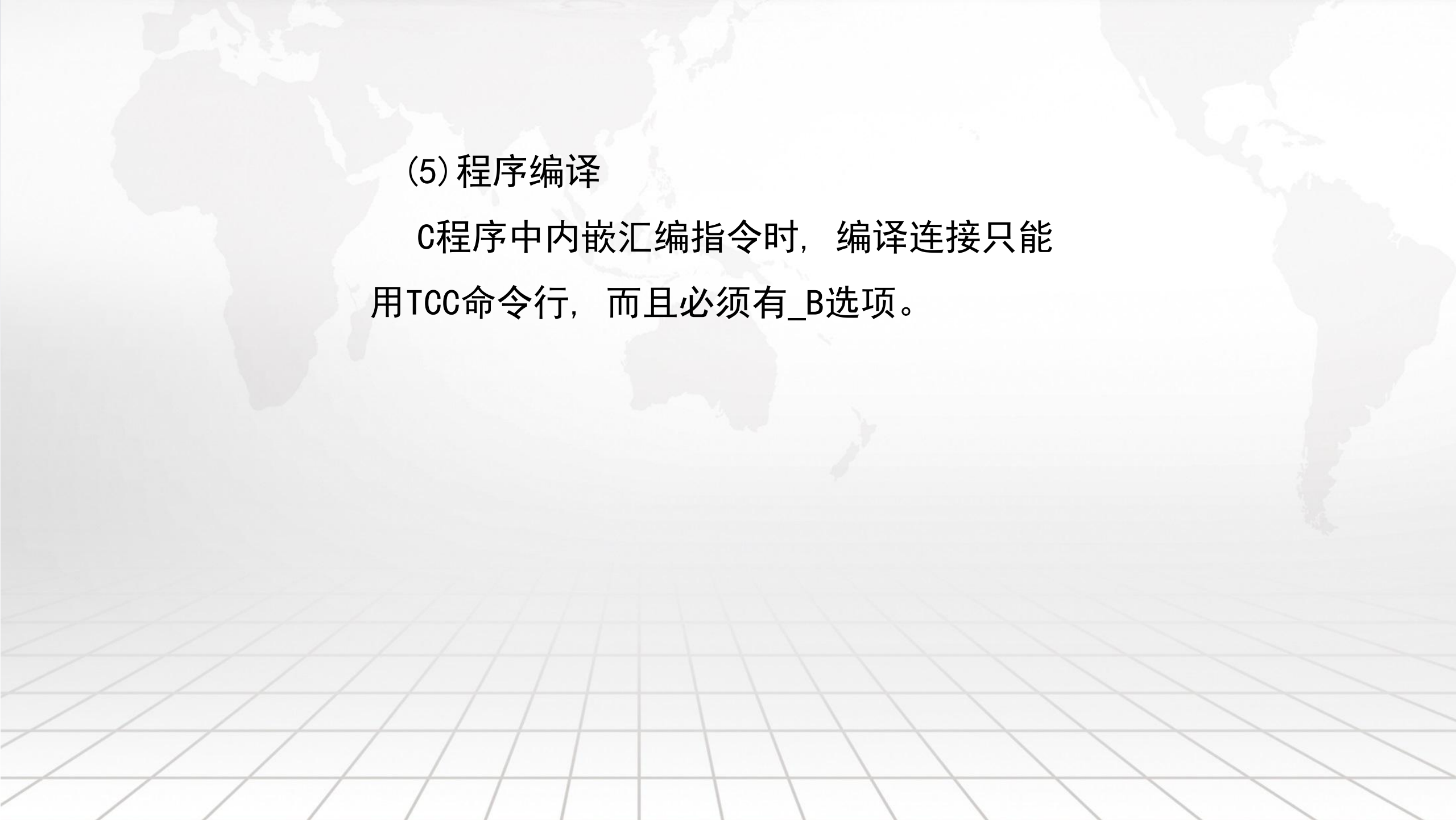
例4. 54

```
asm jmp exitfunc
```

```
.....
```

```
exitfunc;
```

```
return 1;
```



(5) 程序编译

C程序中内嵌汇编指令时，编译连接只能用TCC命令行，而且必须有_B选项。

注意：

- (1) C 程序调用汇编中的寄存器时，要用大写且前面加一个下划线，如上例中`return(_AX)`。
- (2) 在编译之前从TC下include文件夹中把`stdio.h`和`stdarg.h`复制到TC目录下，把lib文件夹中的`cs.obj`, `cs.lib`, `emu.lib`和`maths.lib`复制到TC目录下。
- (3) 若TC文件夹中没有`TASM.exe`可以把微软的`MASM.exe`复制到TC目录下并且改名为`TASM.exe`。
- (4) 编译时只能在纯DOS方式下(不是TC的SHELL os)，用TC中的TCC进行编译链接。编译时在命令行键入：`C:\TC\TCC _B asm.c`。可以生成文件`asm.bak`、`asm.obj`、`asm.map`、`asm.exe`。其中`asm.exe`是得到的可执行文件。

Microsoft C程序内嵌汇编指令方法。VC++6.0是微软公司为开发的32位应用程序而推出的基于C\C++的集成开发环境。它支持内嵌汇编指令的混合编程，并且VC++的编译器中已经集成了汇编指令编译器，使得编译连接非常方便。实现步骤如下：

(1) 在VC++中启用汇编语句的关键字是`_asm`。

(2) `_asm`不能单独出现，后面必须有汇编指令或者是由大括号括起来的多条指令。

(3) 在`_asm`所带的一组汇编指令中可以有标号，C\C++中的`goto`语句和汇编指令的跳转语句可跳到汇编指令组中的标号处，也可以跳转到汇编指令组外的标号处。

(4) 在`_asm`所带的汇编语块中只能调用没有重载的全局的C++函数，也可以调用声明为`extern C`类型的函数。因为C标准库的函数全部声明为这种形式，所以C标准库函数都可以调用。

(5) 对于类、结构体和共用体的成员变量，汇编指令可以直接使用。



汇编语言与C语言模块连接

1. 汇编语言与C语言模块结构

Microsoft C和Turbo C编译系统将不同类型的变量保存在不同段内，其具体情况如下：

BSS 段——存放未初始化的静态变量

DATA 段——存放所有的全局变量和已初始化的静态变量

CONST 段——存放只读常数

STACK 段——存放自动变量和函数参数

TEXT 段——存放程序的执行代码而且，经常将DATA，CONST，BSS 和STACK 段组合成一个DGROUP 段组。

C 编译系统对不同的段的段名、边界类型、结合类型及类型规定了统一的命名规则

段名	边界类型	结合类型	类型
TEXT	BYTE	PUBLIC	'CODE'
DATA	WORD	PUBLIC	'DATA'
BSS	WORD	PUBLIC	'BSS'
CONST	WORD	PUBLIC	'CONST'
STACK	PARA	STACK	'STACK'

2. 调用约定

- (1) 标识符的约定
- (2) 参数传递约定
- (3) 寄存器使用约定

3. 编程方法

(1) C程序调用汇编语言程序

例4. 58下面是在小模式下的混合编程例子。该文件有min. c 和min_num. asm 两个部分组成。功能为从5个数中找出最小的。min. c 是主程序，由C 语言编写。min_num. asm 是汇编子程序，完成寻找小数的工作。由min. c 来调用它。程序代码如下：

```
min. c
int extern min_num (int count , int x1, int x2, int x3,
int x4, int x5) ;
main ()
{int i;
i= min_num (5,7,0,6,1,12 ) ;
printf (“The mininum of five is % d\n”, i) ;
}
```

```
min_num. asm
TEXT SEGMENT BYTE PUBLIC ‘CODE ’
PUBLIC min_num
min_num proc far
    ASSUME CS: TEXT
    push bp
```

```
    mov bp , sp
    mov ax,0
    mov cx,[ bp+4]
    cmp cx , ax
    jle exit
    mov ax,[ bp+6]
    jmp ltest
comp: cmp ax ,[ bp +6]
        jle ltest
        mov ax,[ bp+6]
ltest: add bp,2
        loop comp
exit: pop bp
        ret
min_num endp
TEXT ENDS
end
```

(2) 汇编语言程序调用C函数