



第3章 汇编语言程序设计



3.3 顺序程序设计

- ◇ 没有分支、循环等转移指令的程序，会按指令书写的前后顺利依次执行，这就是顺序程序。
- ◇ 顺序结构是最基本的程序结构。
- ◇ 完全采用顺序结构编写的程序并不多见。

顺序结构程序设计实例

例题3.4 采用查表法，将一位16进制数转换为其对应字符的ASCII码并显示。

```
.data          ;数据段
ASCII db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h
        ; 对应0 ~ 9的ASCII码
        db 41h,42h,43h,44h,45h,46h
        ; 对应A ~ F的ASCII码
hex db 04h,0bh
        ; 假设两个数据
```

Wj0304.asm


例3.4 代码段




```
.code                                ;代码段
start:  mov ax,@data
        mov ds,ax
        mov bx,offset ASCII          ;BX指向ASCII码表
        mov al,hex                    ;AL取得一位16进制数
                                           ;恰好就是ASCII码表中的位移
        and al,0fh                    ;只有低4位是有效的，高4位清0
        xlat                          ;换码：AL←DS:[BX + AL]
        mov dl,al                     ; 入□参数：DL←AL
        mov ah,2                      ; 02号DOS功能调用
        int 21h                       ; 显示一个ASCII码字符
```



例3.4 代码段（续）

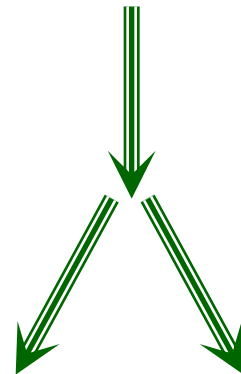


```
mov al,hex+1    ; 转换并显示下一个数据  
and al,0fh  
xlat  
mov dl,al  
mov ah,2  
int 21h
```



3.4 分支程序设计

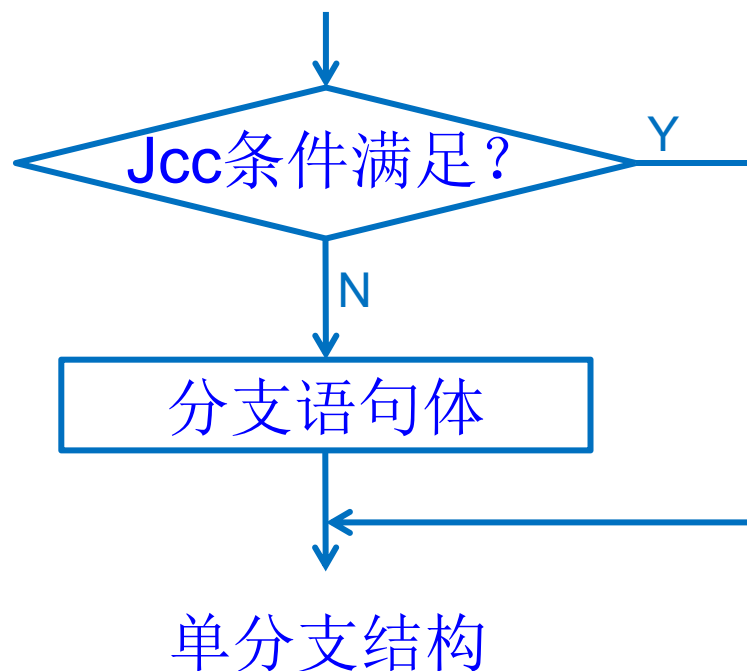
- ◇ 分支程序根据条件成立与否决定某些语句是否执行。
- ◇ 分支控制通过转移指令**Jcc**和**JMP**实现。
- ◇ 判断的条件是**CMP**、**TEST**等指令执行后建立的状态标志。
- ◇ 分支结构有
 - ◆ 单分支结构
 - ◆ 双分支结构
 - ◆ 多分支结构



3.4.1 单分支结构程序设计

- ◇ 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。

实例：求绝对值



单分支结构程序实例

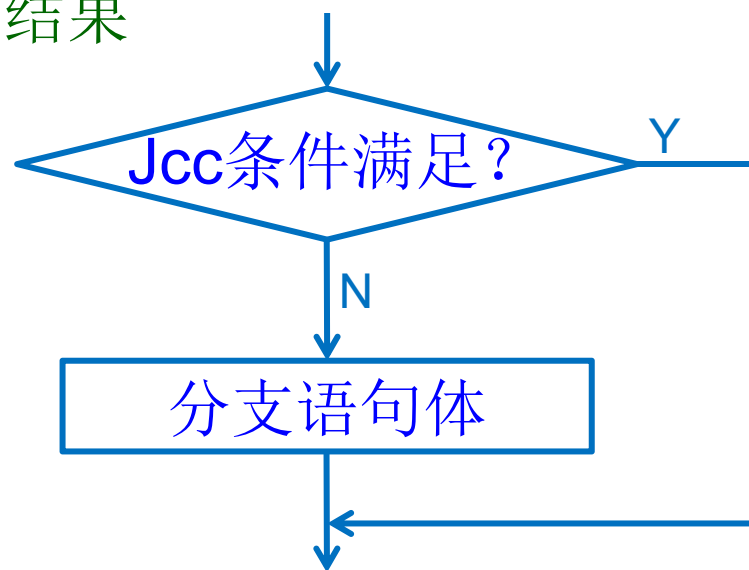
例：计算AX中有符号数的绝对值，结果用变量result保存。

```
cmp ax,0
```

```
jge nonneg      ; 条件满足 ( $AX \geq 0$ )，转移
```

```
neg ax          ; 条件不满足，求补
```

```
nonneg: mov result,ax ; 保存结果
```



单分支结构程序实例 解法2

例：计算AX中有符号数的绝对值，结果用变量result保存。

；解法2，不恰当的分

cmp ax,0

jl yesneg ; 条件满足 ($AX < 0$)，转移

jmp nonneg

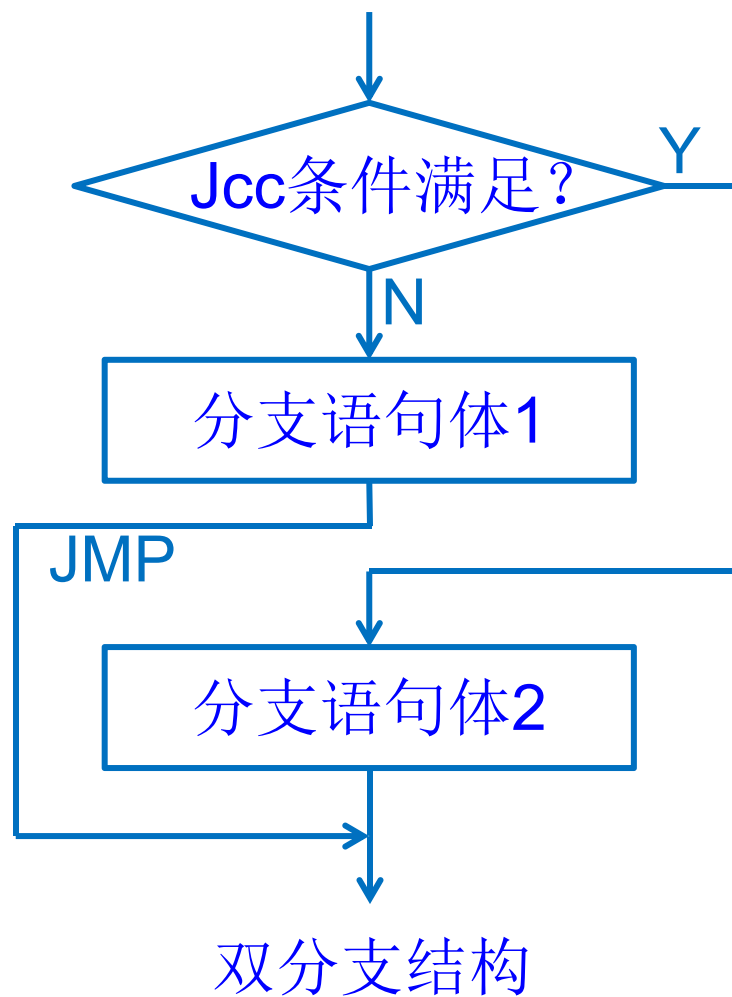
yesneg: neg ax ; 条件满足，求补

nonneg: mov result,ax ; 保存结果

3.4.2 双分支结构程序设计

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。



双分支结构程序实例

例:编程显示BX的最高位 (解1)

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,30h

; CF = 0, 即最高位为0: DL ← 30H = '0'

jmp two ; 一定要跳过另一个分支体

one: mov dl,31h ; DL ← 31H = '1'

two: mov ah,2

int 21h ; 显示

可以用JNC替换JC

双分支结构程序实例

例:编程显示BX的最高位 (解2: 用JNC替换JC)

shl bx,1 ; BX最高位移入CF标志

jnc one ; CF = 0, 即最高位为0, 转移

mov dl,31h

; CF = 1, 即最高位为1: DL ← 31H = '1'

jmp two ; 一定要跳过另一个分支体

one: mov dl,30h ; DL ← 30H = '0'

two: mov ah,2

int 21h ; 显示

转换为单分支结构

双分支结构程序优化

例：显示BX的最高位（解3：单分支实现）

mov dl,'0' ; DL←30H = '0'

shl bx,1 ; BX最高位移入CF标志

jnc two ; CF = 0, 即最高位为0, 转移

mov dl,'1'

; CF = 1, 即最高位为1: DL←31H = '1'

two: mov ah,2

int 21h ; 显示

双分支结构程序优化

例：显示BX的最高位（解4：无分支）

```
mov dl,0
```

```
shl bx,1      ; BX最高位移入CF标志
```

```
adc dl,30h
```

; $CF = 0$, $DL \leftarrow 0 + 30h + 0 = 30H = '0'$

; $CF = 1$, $DL \leftarrow 0 + 30h + 1 = 31H = '1'$

```
mov ah,2
```

```
int 21h      ; 显示
```

双分支结构程序实例

例3.5 显示压缩BCD码，无前导0。

;数据段

BCD db 04h

;代码段

mov dl,BCD


test dl,0ffh ; 如果BCD码为0，显示0

jz zero ; 双分支结构


test dl,0f0h ; 如果BCD码高位为0，不显示0

jz one ; 单分支结构

例3.5 显示压缩BCD码，无前导0（续）



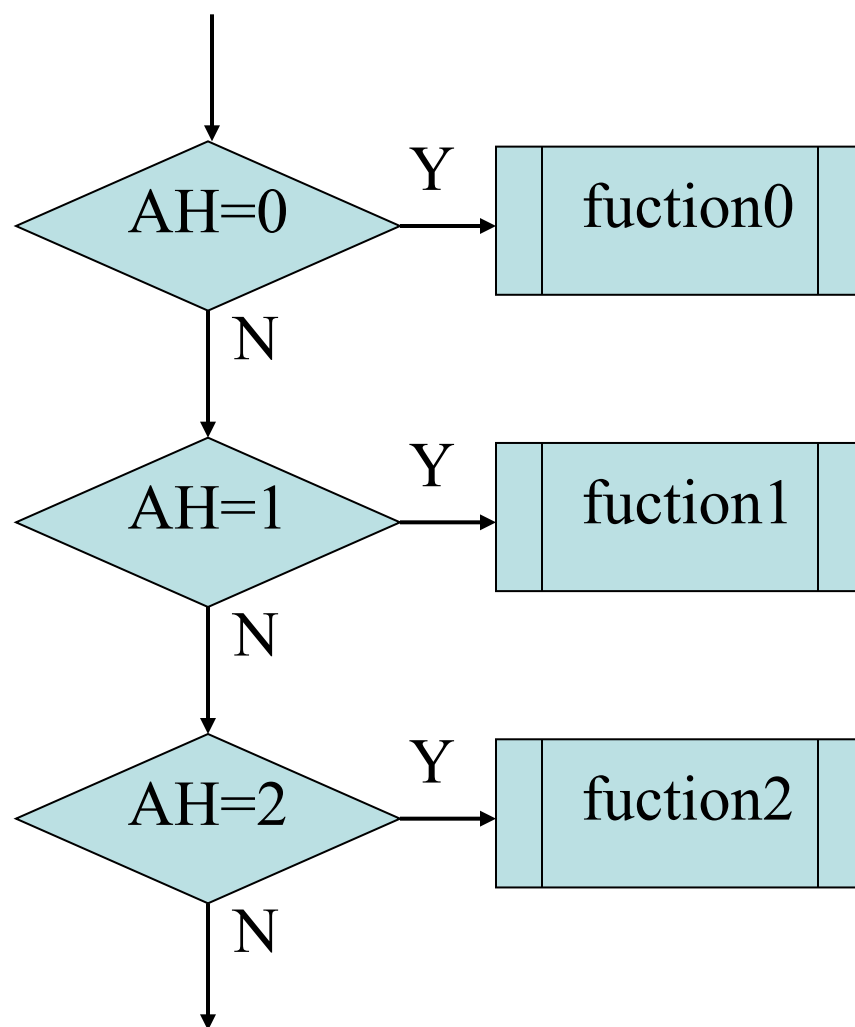
```
mov cl,4      ; 处理高位
shr  dl,cl
or   dl,30h   ; 转换为ASCII码
mov  ah,2     ; 显示高位
int  21h
mov  dl,BCD   ; 继续处理低位
and  dl,0fh
one: or  dl,30h ; 处理低位
     jmp two
zero: mov dl,'0'
two:  mov ah,2
     int 21h
```



3.4.3 多分支结构程序设计

多分支结构是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

```
or ah,ah    ; = cmp ah,0  
jz function0  
dec ah      ; = cmp ah,1  
jz function1  
dec ah      ; = cmp ah,2  
jz function2
```



利用地址表实现多分支结构

例3.7 从低位到高位检测一个字节数据，为0继续，为1则转移到对应的处理程序段，打印位值为1的位的序号，规定最低位序号为0。如果字节数据值为0，则打印输出“？”。

0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

输出“2”

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

输出“？”

wj0307.asm

例3.7 利用地址表实现多分支结构

;数据段

number db 78h ;给定一个数值，D3位为1

addr dw offset fun0, ... ,offset fun7

;取得各处理程序开始的偏移地址

;代码段

mov al,number


mov dl,'?' ;数值为0，显示一个问号“？”

cmp al,0 ;排除AL = 0的特殊情况


jz disp

wj0307.asm

例3.7 利用地址表实现多分支结构（续）



```
                mov bx,0                ;BX←记录为1的位数
again:          shr  al,1                ;最低位右移进入CF
                jc   next                ;为1, 转移
                inc  bx                  ;不为1, 继续
                jmp  again
next:           shl  bx,1                ;偏移地址是2个字节
                jmp  addrs[bx]           ;IP←[addrs + BX], 段内间址
fun0:           mov  dl,'0'
                jmp  disp
                ...
disp:           mov  ah,2                ;显示一个字符
                int  21h
```



大小写字母的比较和转换

'A' = 41H = 01000001B	'B' = 42H	... 'Z' = 5AH = 01011001B
'a' = 61H = 01100001B	'b' = 62H	... 'z' = 7AH = 01111001B

结论1: 大小写字母的**ASCII**码值相差**20H**

结论2: 大小写字母的**ASCII**码值仅**D5**位不同

方法1（加减指令）：“**ADD DL,20H**” “**SUB DL,20H**”

方法2（逻辑指令）：“**OR DL,20H**” “**AND DL,0DFH**”

大小写互换（异或指令）：“**XOR DL,20H**”