



第四章 汇编语言程序开发

张华平 副教授 博士

Email: kevinzhang@bit.edu.cn

Website: <http://www.nlpir.org/>

@ICTCLAS张华平博士

大数据搜索挖掘实验室 (wSMS@BIT)





重点范围

- (1) 【重点讲解】汇编语言编程基本知识、Windows汇编语言程序设计
- (2) 【重点讲解】分支与循环程序设计、浮点运算
- (3) 【一般性讲解】程序优化





4.1 汇编语言基础知识

机器语言：直接用二进制代码的机器指令表示的语言。

汇编语言：用指令助记符、符号地址、标号等符号书写程序的语言。

高级语言：高级语言是一种类似于人类语言的语言。





4.1.1 汇编语言概述

汇编语言是一种符号化了的机器语言，即用指令助记符、符号地址、标号等符号书写程序的语言。





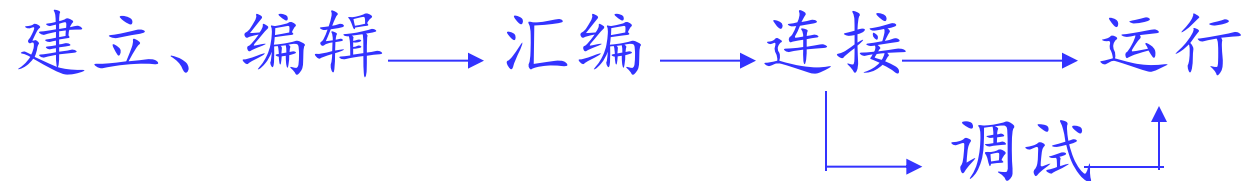
三种语言的比较

	机器语言	汇编语言	高级语言
基本形式	二进制	助记符	语句
编译程序	不需要	需要	需要
执行效率	高	高	低
占用空间	少	少	多
CPU依赖性	依赖	依赖	不依赖
编程难度	复杂	中等	容易



汇编语言实践环境基于DOS平台。

当汇编语言源程序编好后，要使其实现功能，需经过以下过程：





4.1.2 汇编语言编程环境

平 台：Intel 80X86/Pentium

DOS/虚拟8086模式 (V86)

Windows/保护模式

MASM5.1 MASM6.11 MASM32





4.1.2 汇编语言编程环境

上机过程（实模式）

上机过程：masm→link→.exe / .com

编辑：temp.asm

汇编：masm temp.asm→temp.obj

连接：link temp.obj→temp.exe

调试方式：Debug





4.1.2 汇编语言编程环境

上机过程（保护模式）；集成开发环境VS

上机过程：ml→link

编辑：temp.asm

汇编：ml /c /coff hello.asm→hello.obj

连接：link /subsystem:console[windows]

hello.obj→hello.exe

调试方式：WinDebug





4.1.3 汇编语言语句格式

汇编语句：**指令、伪指令、宏指令**。

每条**指令**语句都生成机器代码，各对应一种CPU操作，在程序运行时执行。

伪指令语句由汇编程序在汇编过程中执行，数据定义语句分配存储空间，其它伪指令不生成目标码。

宏指令是用户按照宏定义格式编写的一段程序，可以包含指令、伪指令、甚至其他宏指令。





汇编语言语句格式:

[名字] 助记符 <操作数> [;注释]

其中带[]的内容是可选的。





名字域是语句的符号地址，可以由26个大小写英文字母、0~9数字、_、\$、@、? 等字符组成，数字不能出现在名字的第一个字符位置。

指令的名字叫做标号，必须以冒号（:）结束。它提供给循环或转移指令的转向地址。

伪指令的名字可以是变量名、过程名、段名等。通常，名字具有三属性：段基址、偏移量和类型。

标号的类型有NEAR型和FAR型，变量的类型有字节、字、双字、四字等。





助记符域给出操作的符号表示，可以是指令助记符、伪指令助记符等。例如加法指令的助记符是ADD。

操作数域为操作提供必要的信息。每条指令语句的操作数个数已由系统确定，例如加法指令有两个操作数。

注释域用以说明本条语句在程序中的功能，要简单明了。注释以分号（;）开始。





4.2 常用伪指令

4.2.1、数据定义伪指令

4.2.2、符号定义伪指令

4.2.3、操作符伪指令

4.2.4、框架定义伪指令





4.2.1、数据定义伪指令

数据定义伪指令用来定义程序中使用的数据。这是一组使用频率很高的语句，因为大部分程序都会涉及到数据问题。

格式：**[变量名]** **助记符** **操作数**

功能：为变量分配单元，并为其初始化或者只预留空间。





说明：

①变量名是可选的，需要时由用户自己起。它是该数据区的符号地址，也是其中第一个数据项的偏移量。程序通过变量名引用其中的数据。





②

助记符是数据类型的符号表示。

助记符	数据类型	一个数据项字节数
DB (BYTE)	字节型	1
DW (WORD)	字型	2
DD (DWORD)	双字型	4
DQ (QWORD)	四字型	8
DF (FWORD)	六字节型	6
DT (TBYTE)	10字节型	10

注：()中是在MASM6.11版本中可以使用的助记符。必须掌握DB、DW、DD。





③ 操作数

操作数可以是数字常量、数值表达式、字符串常量、地址表达式、?、<n> DUP (操作数,) 形式。





a. 数字常量及数值表达式

操作数可以是数值表达式，数字中若出现字母形式，不区分大小写。如下所示：

十进制数：以D结尾，汇编语言中缺省值是十进制数，所以D可以省略不写。

二进制数：以B结尾。例如，10100011B，10100011b。

十六进制数：以H结尾。例如，12H，12h，0AB56H，0ab56h。

八进制数：以Q或O（字母）结尾。例如，352Q。





b. 字符串常量

在汇编语言中字符需要用单引号括起来，其值为字符的ASCII值。因为每个字符占用一个字节，所以最好用DB助记符定义字符串。例如，'A'的值为41H。'abc'的值为616263H。





c. 地址表达式

操作数可以是地址符号。若只定义符号的偏移量部分，则使用DW助记符。若要定义它的双字长地址指针（既含16位偏移量又含段基址），则使用DD助记符，其中低字中存放偏移量，高字中存放段基址。

例如，“VAR DW LAB”语句在汇编后VAR中含有LAB的偏移量。





d. ?

在程序中使用“?”为变量预留空间而不赋初值。

e. <n> DUP (操作数,)

若要对某些数据重复多次, 可以使用这种格式。其功能是把 () 中的内容复制n次。DUP可以嵌套。



例:

M1	DB	15, 67H, 11110000B, ?
M2	DB	'15' , ' AB\$'
M3	DW	4*5
M4	DD	1234H
M5	DB	2 DUP (5, ' A')
M6	DW	M2 ;M2的偏移量
M7	DD	M2 ;M2的偏移量、段基址

设以上数据自1470:0000开始存放, 则为:

0F 67 F0 00 31 35 41 42 24 14 00 34 12 00 00 05 41 05 41 04 00
04 00 70 14。





```
LEA DX, M4+2  
MOV CX, M4+2
```





可直接通过变量名引用变量，但要注意类型匹配。例如以下程序片段：

```
MOV AL, M1           ; (AL) = 15
MOV BX, M3           ; (BX) = 20
ADD M3, 6            ; (M3) = 26
MOV AL, M2           ; (AL) = '1' = 31H
MOV BL, M2+2         ; (BL) = 'A' = 41H
MOV M1+3, BL         ; (M1+3) = 41H
```

M2+2的这种表示形式？





4.2.2、符号定义伪指令

有时程序中会多次出现同一个表达式，为方便起见可以用符号定义伪指令给该表达式定义一个符号，以便于引用及减少程序修改量，并提高程序的可读性。**汇编后该符号代表一个确定的值。**





1. 等值EQU伪指令

格式：符号名 EQU 表达式

功能：用符号名代表表达式或表达式的值。

说明：表达式可以是任何有效的操作数格式。

例如常数、数值表达式、另一符号名或助记符。

注意：用EQU定义的符号在同一个程序中不能再定义。





例.

CR EQU 0DH ;回车符的ASCII值

LF EQU 0AH ;换行符的ASCII值

BEL EQU 07H ;响铃符的ASCII值

PORT_B EQU 61H ;定义PORT_B端口

B EQU [BP+6] ;[BP+6]用B表示

程序中可以通过符号引用这些值，例如：

MOV AL, CR ;等价于 MOV AL, 0DH

ADD BL, B ;等价于 ADD BL, [BP+6])

OUT PORT_B, AL ;输出到61H端口





EQU用途：增加程序可读性、缩短程序书写长度、避免因某些修改而带来的程序不一致性。

EQU伪指令除了以上用途外,经常使用它的一个场合是与\$配合,得到变量分配的字节数。如下所示:

```
MSG      DB    'This is first string.'  
Count    equ   $-msg  
Mov      cl, count ; (CL) = MSG的串长 = 21
```





这样做可以由汇编程序在汇编过程中自动计算字符串的长度，避免了编程者计数。特别是当因字符串改变而串长改变时，取串长的语句无需做任何修改。

由于用EQU定义的符号在同一个程序中不能再定义，所以以下语句是错误的：

```
CT EQU 1
```

```
CT EQU CT+1
```





2. 等号(=)伪指令

格式: 符号名 = 数值表达式

功能: 用符号名代替数值表达式的值

说明: 等号伪指令与EQU伪指令功能相似, 其区别是等号伪指令的表达式只能是常数或数值表达式。

用“=”定义的符号在同一个程序中可以再定义。通常在程序中用“=”定义常数。

例. DPL1 = 20H

 K = 1

 K = K+1





4.2.3 操作符伪指令

操作符可以出现在语句的操作数表达式中。该操作在汇编程序汇编时实现。包括算术、逻辑、关系、属性、返回值操作符。



1. \$ 操作符

ORG伪指令格式: **ORG** 数值表达式

功能: \$在程序中表示当前地址计数器的值。程序中的每一行都有一个地址,从程序的第一行到最后一行,地址计数器在不断地增加。

说明: 程序中一般不直接使用\$的值,而是使用它来计算变量占用的空间。

例. wVar WORD 0102h, 1000, 100*100

BYTESOFWVAR EQU \$-wVar; 值等于6。

MOV EAX, \$; 00401010 B8 10 10 40 00=MOV EAX, 0401010

3. OFFSET操作符

格式：OFFSET [变量|标号]

功能：OFFSET操作符用来取出变量或标号的地址（在段中的偏移量）。在32位编程环境中，地址是一个32位的数。

➤ 例4.9

➤ MOV EBX, DVAR2

➤ MOV EBX, OFFSET DVAR2

4、算术操作符

算术操作符包括 $+$ 、 $-$ 、 $*$ 、 $/$ 和MOD（取模）操作符。

算术操作符可以用在数值表达式或地址表达式中。





4.2.3 操作符伪指令

例.

```
X DW      12, 34, 56
CT EQU    ($-X)/2
MOV      CX , CT      ; (CX
) = 3
```

```
MOV      AX , X
ADD      AX , X+2 ; (AX) =
```

46



5、逻辑操作符

逻辑操作符包括AND、OR、XOR和NOT。逻辑操作符是按位操作的，它只能用在数值表达式中。

例.

```
PORT EQU 0FH
```

```
AND DL, PORT AND 0FEH
```

汇编后: AND DL, 0EH



6、关系操作符

关系操作符包括EQ、NE、LT、LE、GT、GE。其操作结果为一个逻辑值，若关系成立结果为真（全1），否则结果为假（0）。其中的操作数必须是数字或同段内的两个存储器地址。

例. 指令 MOV AL, CH LT 20的汇编结果:

MOV AL, 0FFH ;当CH<20时

或: MOV AL, 0 ;当CH≥20时



4.2.4 框架定义伪指令

汇编程序在缺省情况下只接受8086 / 8088的指令系统, 即使在386以上机器也是如此, 因此为8086 / 8088编写的程序在286以上的处理器都可以顺利执行。

为了能够使用其它微处理器或协处理器的指令系统编写高级的32位机软件, 需要在程序中添加选择微处理器的伪指令。



较常使用的选择微处理器伪指令有以下几种：

伪指令	功 能
. 286	选择80286微处理器指令系统
. 386	选择80386微处理器指令系统
. 486	选择80486微处理器指令系统
. 586	选择80586微处理器指令系统
. 8087	选择8087数字协处理器指令系统
. 287	选择80287数字协处理器指令系统
. 387	选择80387数字协处理器指令系统





4.2.4 框架定义伪指令

伪指令格式	功 能
.DATA	定义数据段
.DATA?	定义存放未初始化变量的数据段
.CONST	定义存放常量的数据段
.CODE	定义代码段
.STARTUP	指定加载后的程序入口点
.EXIT	返回DOS或父进程
.STACK size	建立一个堆栈段并定义其大小（size以字节为单位。若不指定size参数，则使用默认值1 KB）。
.MODEL 内存模式[,调用规则][,其他模式]	定义程序工作的模式





二、简化段定义常用结构中的伪指令

1. 定义存储模型伪指令

常用格式：`.model` 存储模型

功能：定义存储模型。





常用的存储模型有：

- ①TINY：所有代码和数据放置在一个段中。
- ②SMALL：所有代码在一个段内，所有数据在另一个段。
- ③MEDIUM：代码放置在多个段内。数据限制在一个段。
- ④COMPACT：代码在一个段内，数据可以在多个段内。
- ⑤LARGE：代码和数据被放置在多个段内。
- ⑥HUGE：单个数据项可以超过64K，其它同LARGE模型。
- ⑦FLAT：与TINY模型类似，所有代码和数据放置在一个段中。TINY模型的段是16位，FLAT32位。





2. 定义堆栈段尺寸伪指令

格式: `.stack size`

功能: 建立一个堆栈段并定义其大小

说明: 若不指定size参数, 则使用缺省值1KB。





4.3 汇编语言程序格式

➤ 4.3.1 用户界面 (User Interface)

- 字符用户界面 (Character User Interface, CUI)
- 图形用户界面 (Graphic User Interface, GUI)

➤ 4.3.2 控制台界面的汇编源程序

➤ 4.3.3 Windows界面的汇编源程序



4.3.2 控制台界面的汇编源程序

.386

.model flat, stdcall

option casemap:none

includelib msvcrt.lib

printf PROTO C :ptr sbyte, :VARARG

.data

szMsg byte "Hello World! %c", 0ah, 0

a byte 'Y' b byte "hello"

.code

start:

 invoke printf, offset szMsg, a

 invoke printf, offset szMsg, offset b

ret

end start

《汇编语言与接口技术》讲义/张华平



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY



4.3.2 控制台界面的汇编源程序

```
.386  
.model flat,stdcall  
option casemap:none
```

模式定义

```
includelib msvcrt.lib  
printf PROTO C :ptr sbyte, :VARARG
```

库文件及函数声明

```
.data  
szMsg byte 'Hello World!', 0ah, 0
```

数据部分

```
.code  
start:  
    invoke printf, OFFSET szMsg  
    ret  
end start
```

代码部分





4.3.2 控制台界面的汇编源程序

1. 模式定义

- `.386`

- `.model flat, stdcall`

- `option casemap:none`

- (1) `.386`语句定义了程序使用80386指令集。

- (2) `.model flat, stdcall`语句。

- (3) `option`语句。

`option`语句有许多选项，例如`option language`、`option segment`等，在Win32中需要定义`option casemap:none`，用以说明程序中的变量和子程序名是否对大小写敏感





4.3.2 控制台界面的汇编源程序

2. `include lib` 语句

■ `include lib` 库文件名



4.3.2 控制台界面的汇编源程序

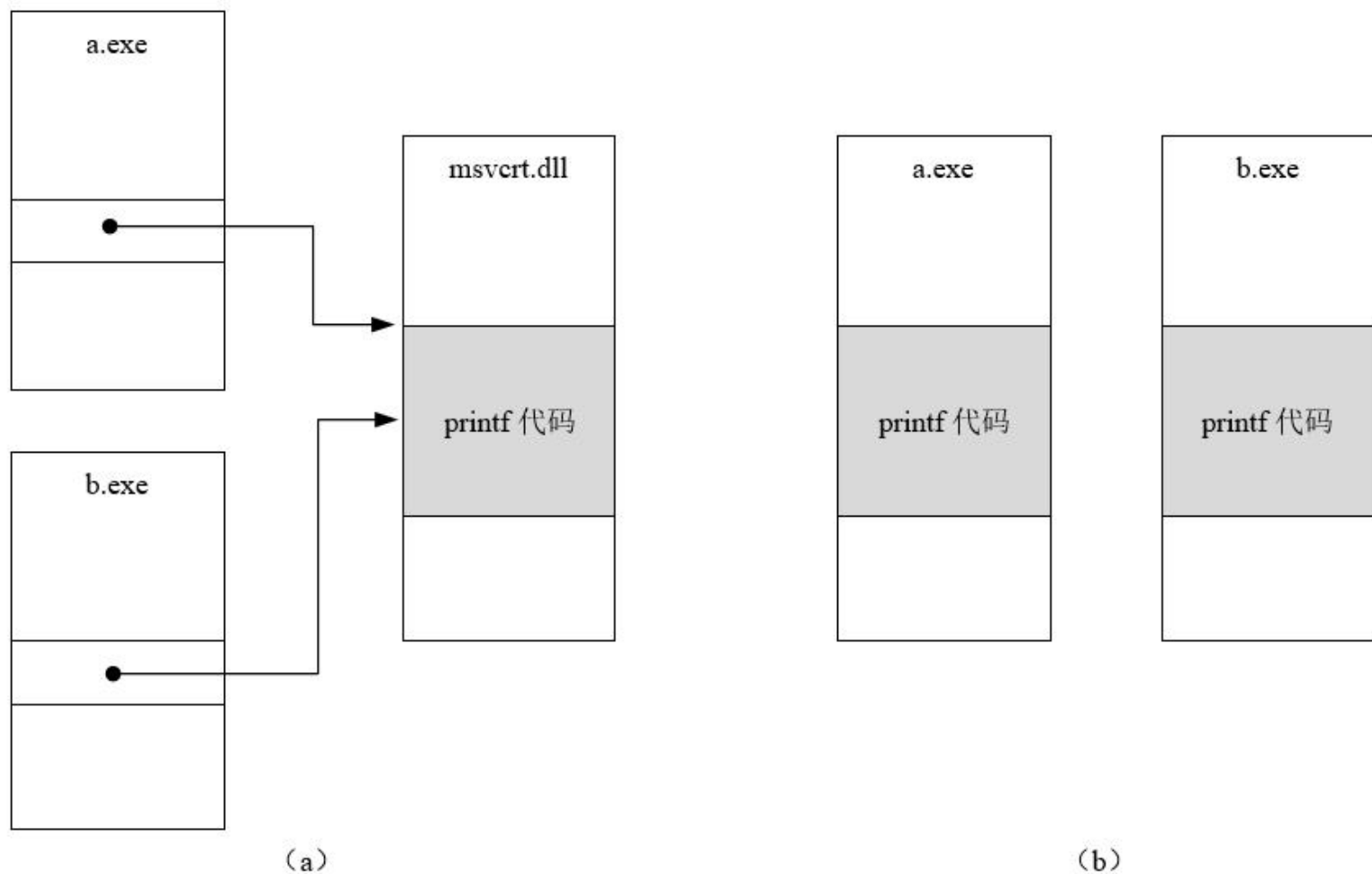


图 4-3 动态连接和静态连接

(a) 动态连接; (b) 静态连接

3. 函数声明

- 函数名称 PROTO [调用规则] : [第一个参数类型] [, : 后续参数类型]
- 在汇编语言中，用ptr sbyte代表const char *
- printf PROTO C : ptr sbyte, : VARARG
- 函数声明后，就可以用INVOKE伪指令来调用

4.3.2 控制台界面的汇编源程序

➤ 4. include语句

- include语句格式: include 文件名
- include kernel32. inc
- include user32. inc

➤ 5. 数据和代码部分

- 程序中的数据部分从.data语句开始定义, 代码部分从.code语句开始定义, 所有的指令都必须写在代码区中。
- 在编程时, 不能把那些需要修改的变量放到.code部分。



4.3.2 控制台界面的汇编源程序

➤ 6. 程序结束

- 与DOS程序相同，Win32程序在遇到end语句时结束。end语句后面跟的标号指出了程序执行的入口点，即装入执行的第一条指令的位置，表示源程序结束。
- 语句格式：END [过程名]

➤ 7. 跨行

```
invoke      MessageBox, \  
            NULL, \ ;HWND hWnd  
            offset szMsg, \ ;LPCSTR lpText  
            offset szTitle, \ ;LPCSTR lpCaption  
            MB_OK;UINT uType
```



4.3.2 控制台界面的汇编源程序

➤ 8. 程序中的数据归类

- (1) 可读可写的初始变量

- (2) 可读可写的未初始变量:

 - ⑩ `buffer` `byte` `65536 dup (?)`

- (3) 常量数据

 - ⑩ `szMsg` `byte` `"Hello World!"`, `0ah`, `0`

➤ 9. `invoke`伪指令---相当于`Call`

- 格式: `invoke` 函数名[, 参数1] [, 参数2]...

- 功能: 调用函数或子程序。





4.3.3 Windows界面的汇编源程序

```
.386
.model flat, stdcall
option casemap:none
MessageBoxA PROTO :dword, :dword, :dword, :dword
MessageBox equ <MessageBoxA>
Includelib user32.lib
NULL equ 0
MB_OK equ 0
.stack 4096
.data
SzTitle byte 'Hi!', 0
SzMsg byte 'Hello World!', 0
.code
start:
    invoke MessageBox,
        NULL, ; HWND hWnd
        offset szMsg, ; LPCSTR lpText
        offset szTitle, ; LPCSTR lpCaption
        MB_OK ; UINT uType
    ret
end start
```

《汇编语言与接口技术》讲义/张华平



4.3.4 输入输出有关的Windows API函数

1、printf

`include lib msvcrt.lib`

`printf PROTO C :ptr sbyte,:vararg`

`printf PROTO C :dword,:vararg`

`invoke printf, offset szOut, x, n, p`

其中，szOut要在数据区中定义，例如：

`szOut byte 'x=%d n=%d x(n)=%d', 0ah, 0`

其效果等价于：

`printf ("x=%d n=%d x(n)=%d\n" , x, n, p);`



4.3.4 输入输出有关的Windows API函数

2、scanf

includelib msvcrt.lib

scanf PROTO C :dword,:vararg

szInFmtStr byte '%d %c %d', 0

invoke scanf, offset szInFmtStr, offset a, offset b, offset d

其中，第1个参数是格式字符串szInFmtStr的地址，第2、3、4个参数分别是a、b、d的地址。其效果等价于：

scanf("%d %c %d", &a, &b, &d);



3、 MessageBoxA

includelib user32.lib

```
#define WINAPI __stdcall
```

```
int
```

```
WINAPI
```

```
MessageBoxA(
```

```
    HWND hWnd ,
```

```
    LPCSTR lpText,
```

```
    LPCSTR lpCaption,
```

```
    UINT uType);
```

它的调用规则和参数类型说明为:

```
MessageBoxA PROTO :DWORD, :DWORD, :DWORD, :DWORD
```

//窗口句柄

//消息框正文的指针

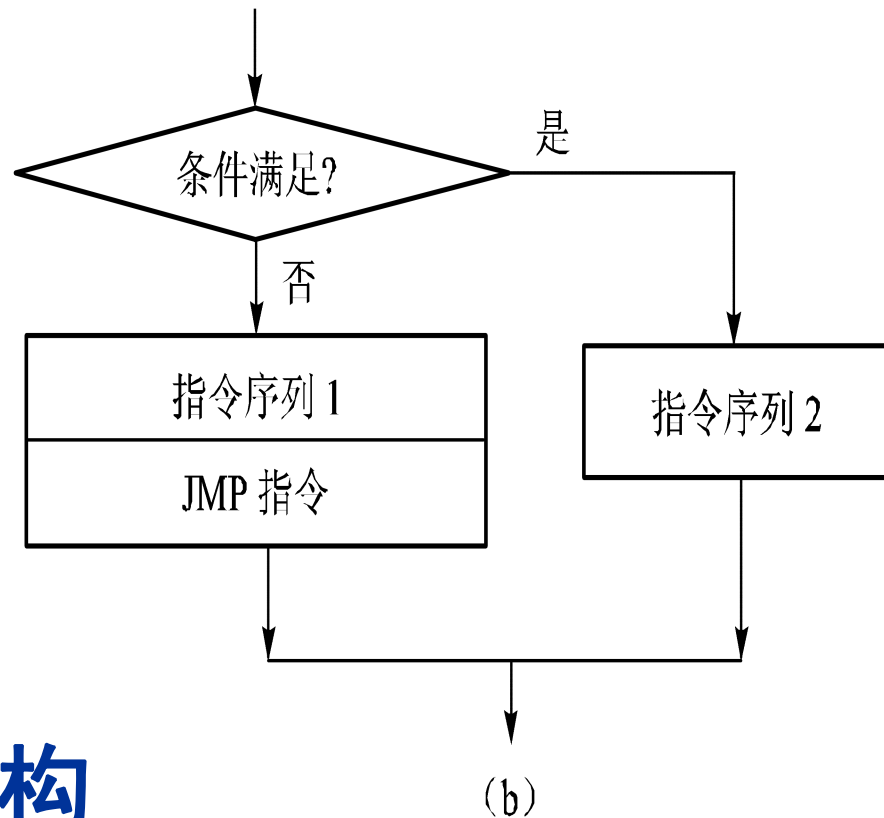
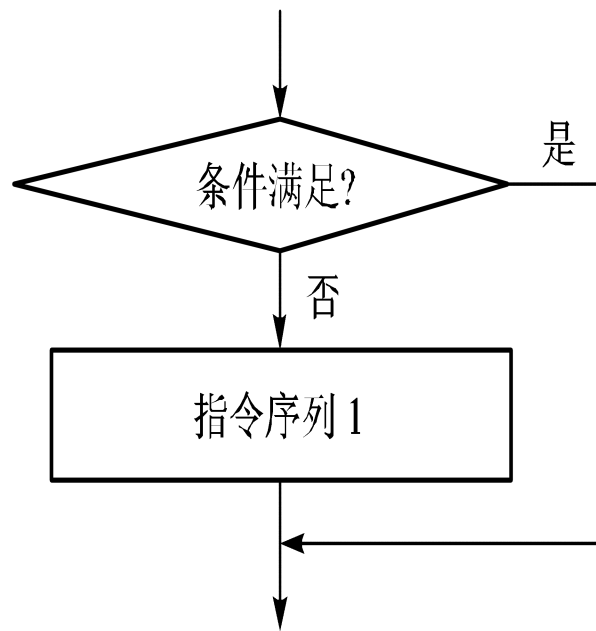
//消息框窗口标题的指针

//消息框类型



4.4 分支与循环程序设计

➤ 需要根据不同条件进行不同的处理，此时需要设计分支程序。



IF_THEN_ELSE分支结构

(a) IF结构; (b) IF_THEN_ELSE结构



➤ 1. IF结构程序举例（单分支结构）

➤ 例5.1 求带符号数A和B的较大值MAXAB=MAX（A， B）。
;PROG0501

;PROG0501

MOV EAX, A

CMP EAX, B

JGE AlsLarger; 如果 $A \geq B$ ，跳转到

AlsLarger标号处

MOV EAX, B

AlsLarger:

MOV MAXAB, EAX



例5.3 求带符号数X的符号，如果 $X \geq 0$ ，SIGNX置1，否则置为-1。

;PROG0503



4.4 分支与循环程序设计

;PROG0503

X

SDWORD -45

SIGNX

SDWORD

?

MOV

SIGNX, 0

CMP

X, 0

JGE

XisPostive

; X \geq 0, 跳转

MOV

SIGNX, -1

JMP

HERE

; 跳过“MOV SIGNX, 1”语句

XisPostive:

;CMP X, 0

JE XisZero

MOV

SIGNX, 1

JMP HERE

XisZero:

MOV SIGNX, 0

HERE:



4.4 分支与循环程序设计

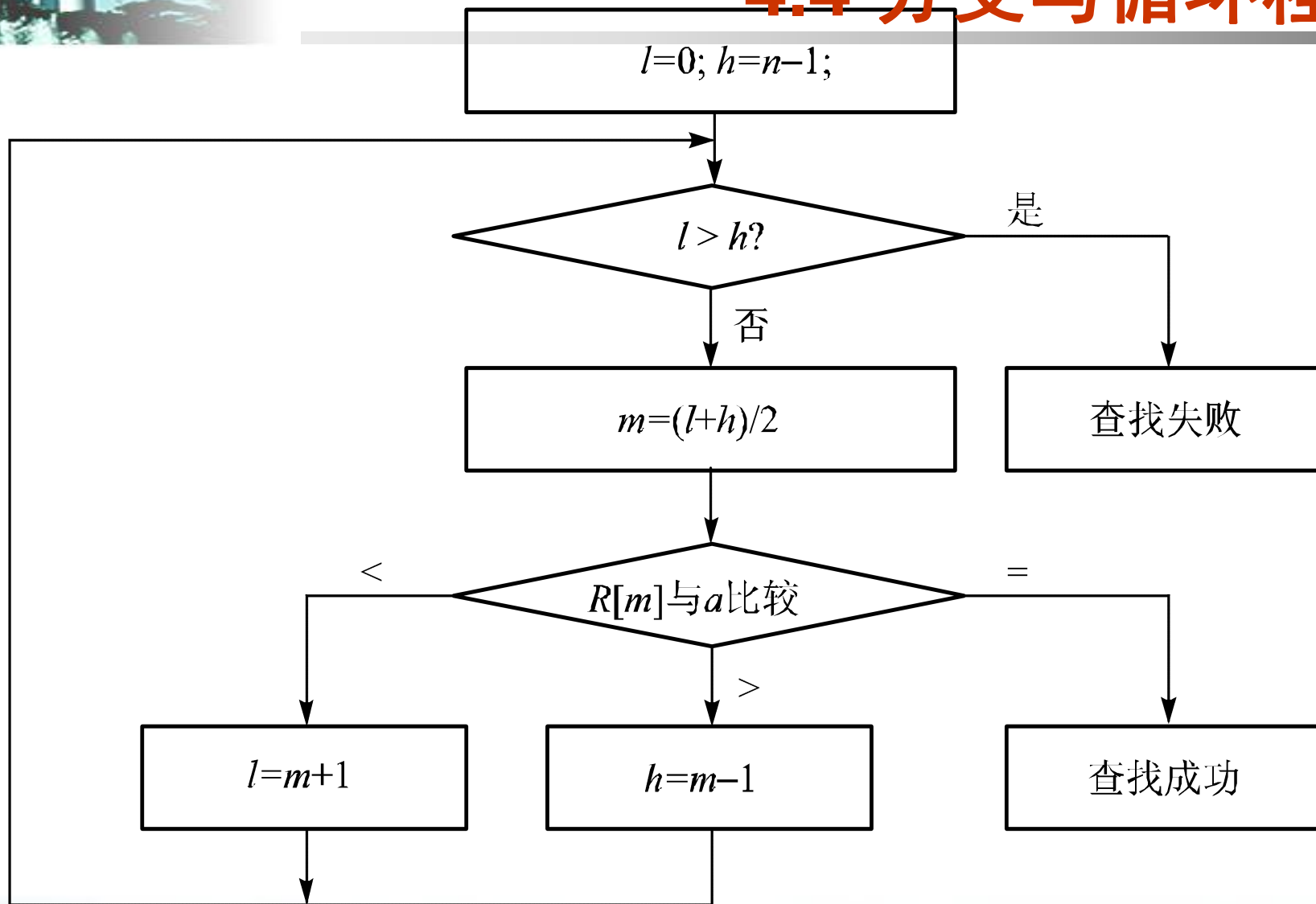
例5.5 在升序数组中查找指定数。

➤ 算法分析：在有序数组中查找元素，使用折半查找的效率最高，平均比较次数为 $\log_2 n$ （顺序查找的次数 $n/2$ ）。本例使用折半查找法。设：

- 字节型数组为R
- 元素个数为n
- 要查找的数为a
- 查找范围，以下界l和上界h表示
 - ✎ l和h是数组下标
 - ✎ 初始时下界为0上界为n-1，即整个数组
- m是下界l和上界h的中点
- 查找结束条件： $l > h$ ，意味无指定数



4.4 分支与循环程序设计



折半查找算法的流程



4.4 分支与循环程序设计

- 执行过程如下：
- (1) 先设定一个查找范围，以下界 l 和上界 h 表示。 l 和 h 是数组下标。初始时，下界为 0，上界为 $n-1$ ，即查找范围是整个数组。
- (2) 如果下界 l 大于上界 h ，则查找范围为空，查找结束。在这种情况下，数组中没有 a ，算法结束。
- (3) 取下界 l 和上界 h 的中点 m , $m = (l+h)/2$ 。
- (4) 从数组的中点 m 处取出一个数 $R[m]$ ，和 a 进行比较。
- (5) 如果 $R[m]$ 等于 a ，则在数组中找到 a ，下标为 m 。算法结束。
- (6) 如果 $R[m]$ 大于 a ，中点上的数比 a 大，从中点到上界中的所有数都比 a 大，修改上界 h 为 $m-1$ 。然后跳转到第 (2) 步。
- (7) 如果 $R[m]$ 小于 a ，中点上的数比 a 小，从下界到中点中的所有数都比 a 小，修改下界 l 为 $m+1$ 。然后跳转到第 (2) 步。
- 每经过一次比较，查找范围就缩小一半。缩小查找范围的过程如图 5-4 所示。

;PROG0505





4.4 分支与循环程序设计

SWITCH_CASE结构分支程序设计

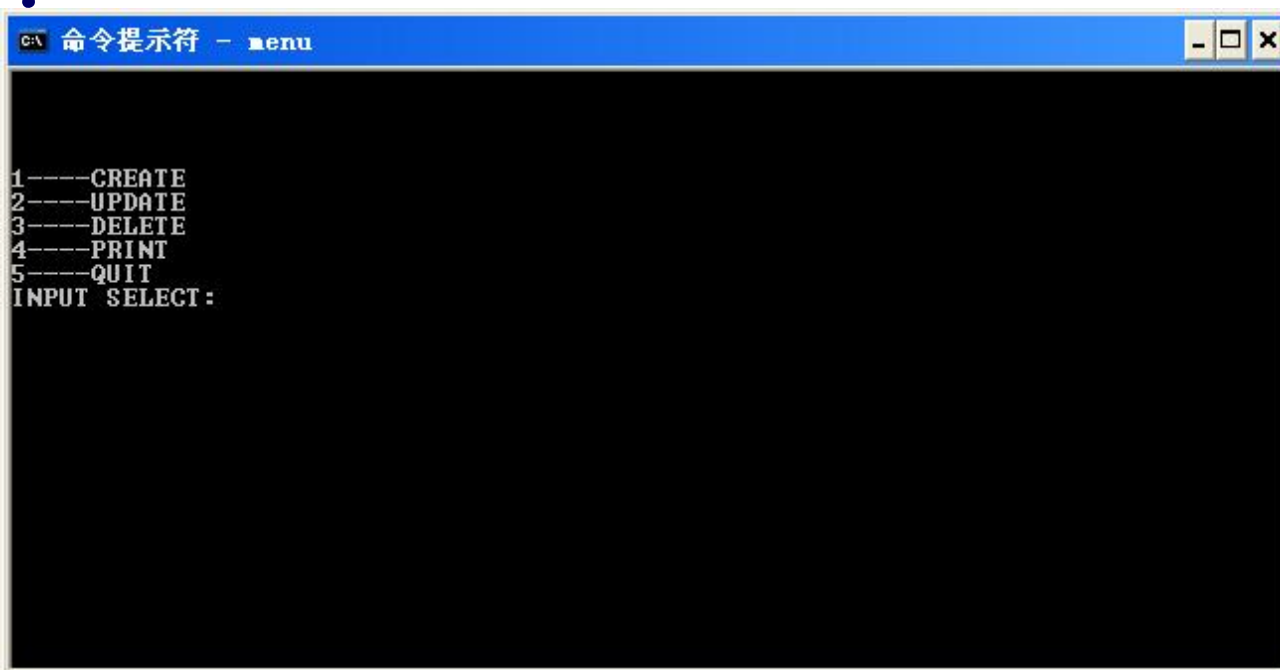
- 分支较少的IF_THEN_ELSE结构使用条件转移指令实现
- 分支较多时使用条件转移指令实现分支程序变得较复杂，不易读懂、扩充。多分支的SWITCH_CASE结构可用基于跳转表的方法实现
 - ✎ 实现的关键是先要构成跳转表
 - ✎ 然后用无条件间接转移指令实现跳转
- SWITCH_CASE结构可用于主控程序设计等



4.4 分支与循环程序设计

➤ 例. 编制一个管理文件的菜单程序，要求能够实现建立文件、修改文件、删除文件、显示文件和退出应用程序5个主控功能。首先在屏幕上显示5种功能，然后从键盘上输入数字1~5即可转入相应的功能，而输入其他字符则提示输入非法。若选择退出功能，则能正确返回；若选择其他功能，应能返回到主菜单。

➤ 屏幕显示为：



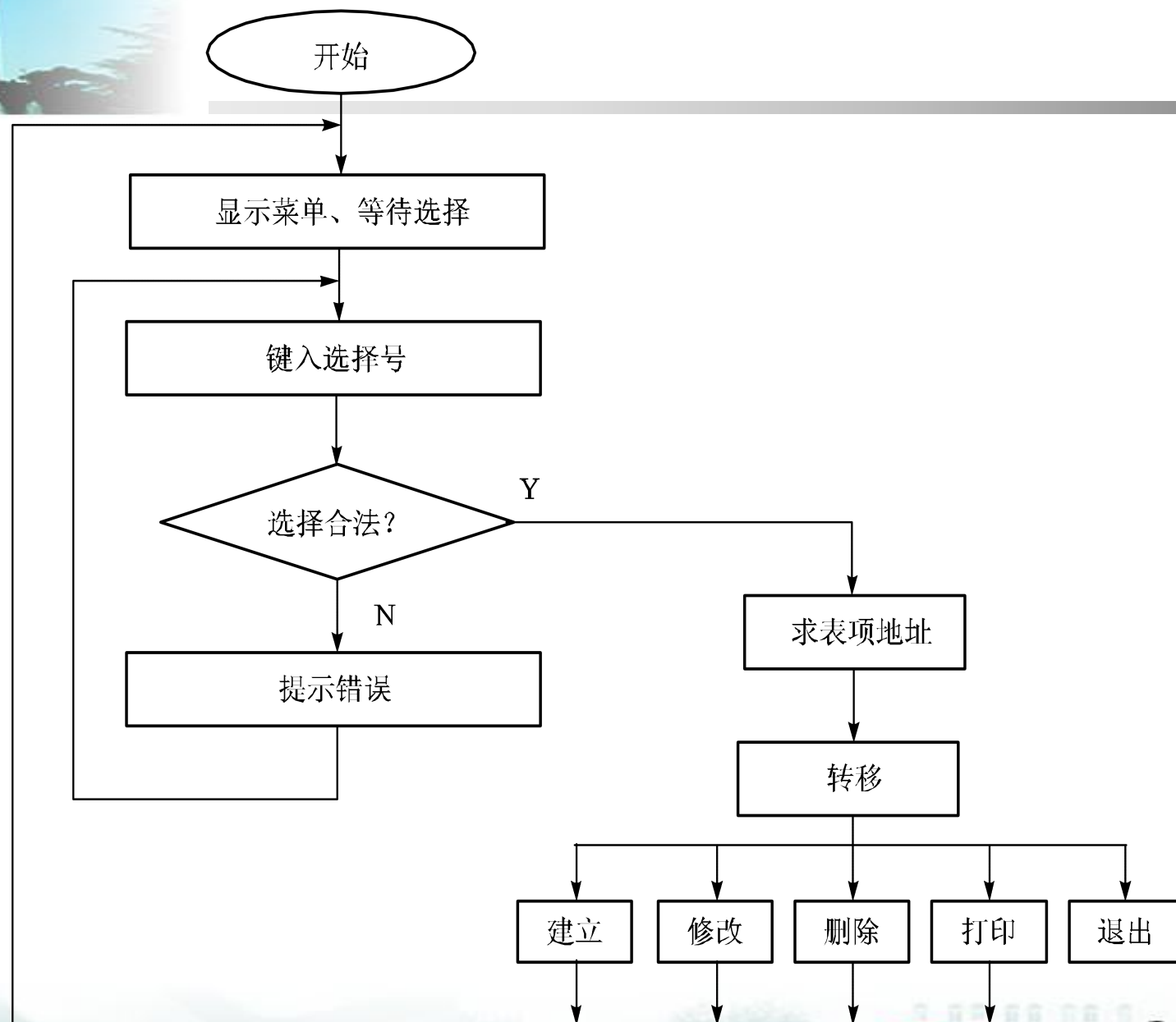
```
C:\ 命令提示符 - menu
1---CREATE
2---UPDATE
3---DELETE
4---PRINT
5---QUIT
INPUT SELECT:
```

4.4 分支与循环程序设计

➤ 算法分析：要能够转移到不同分支，必须提供各个分支的入口地址。对于SWITCH_CASE结构，由于分支众多，可以把各分支入口地址集中在一起构成一个地址表，把这个地址表称为跳转表。设建立文件分支入口标号为CR，修改文件分支入口标号为UP，删除文件分支入口标号为DE，显示文件分支入口标号为PR，退出分支入口标号为QU，则该跳转表如下所示：

```
➤      JMPTAB  DD  OFFSET CR           ;跳转表
➤              DD  OFFSET UP
➤              DD  OFFSET DE
➤              DD  OFFSET PR
➤              DD  OFFSET QU
```





菜单程序流程图



说明：位移量是跳转表中所选项与表基址的距离。把所有功能号连续排列，设选择了K号功能，则：

⑩ 索引号 = K - 一起始功能号（例如功能号为1, 2, 3...N，则索引号 = K - 1）

⑩ 位移量 = 索引号 × 每项入口地址占用的字节数。

⑩ 表项地址 = 表基址 + 位移量。一旦得到了表项地址，使用无条件间接转移指令实现转移即可





◎用无条件间接转移指令实现SWITCH_CASE转

- 在32位程序中，地址表用DD定义，用段内间接转移指令实现跳转
- 在16位程序中，若跳转表是用DW定义的，则用段内间接转移指令。地址表用DD定义，则用段间间接转移指令

◎本例使用以下指令得到表项地址，并实现转移

```
mov eax, JmpTab[ebx*4]
```

```
jmp eax
```

也可以使用jmp JmpTab[ebx*4]指令等实现



4.4 分支与循环程序设计

➤ 循环两种基本结构

● DO_WHILE结构

☞ 先判断后执行结构，把对循环控制条件的判断放在循环的入口，先判断控制条件，若满足控制条件就执行循环体，否则退出循环。

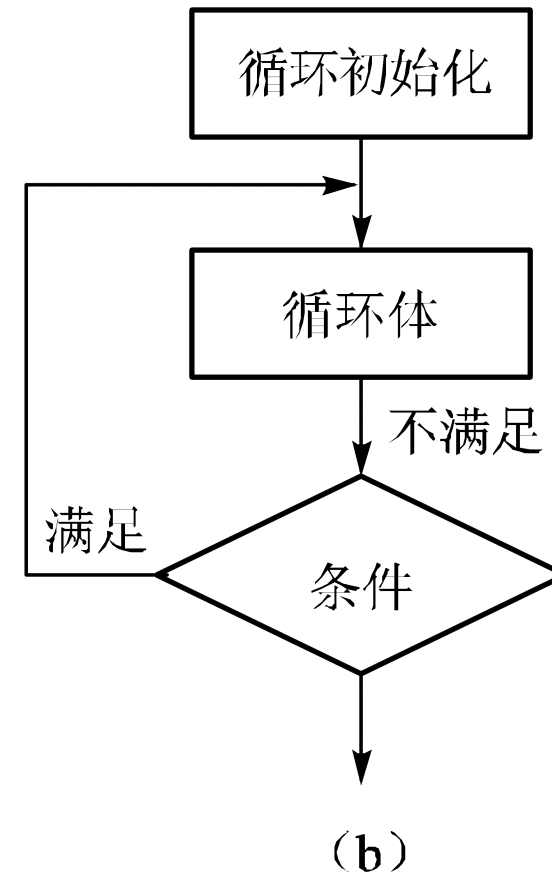
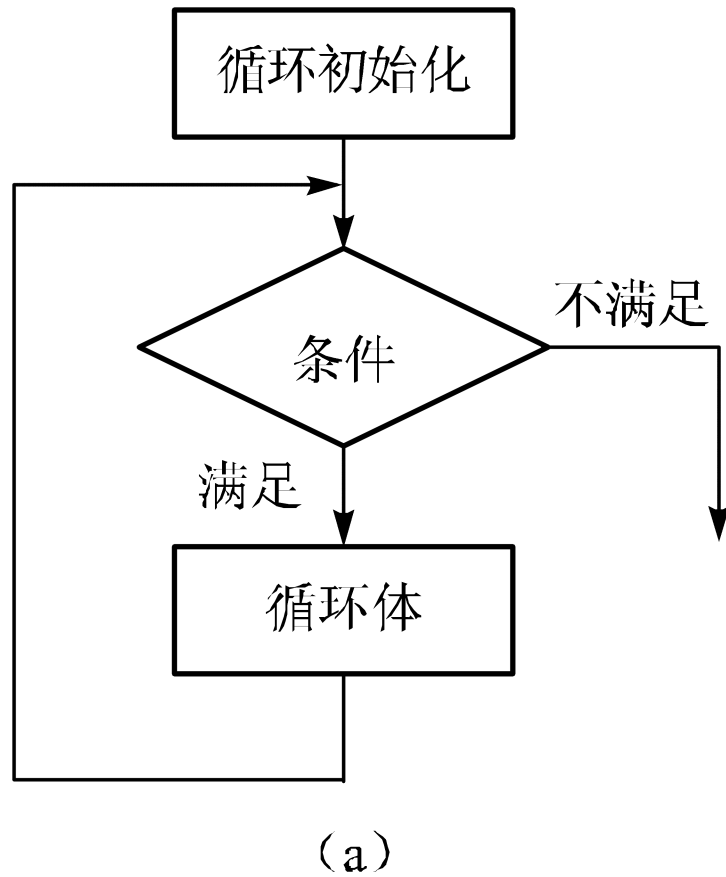
● DO_UNTIL结构

☞ 先执行后判断结构，先执行循环体然后再判断控制条件，若满足控制条件则继续执行循环体，否则退出循环

- 两种结构一般可以随习惯使用，但在初始循环次数可能为0的情况下则必须使用DO_WHILE结构



4.4 分支与循环程序设计



基本循环结构

4.4 分支与循环程序设计

- 无论使用哪种循环结构，循环程序一般应包括以下几个部分。
- (1) **循环初始化**。它包括设置循环次数的初始值、地址指针的初始设置等。
- (2) **循环体**。这是循环工作的主体，包括要重复执行的操作，以及循环的修改部分。修改部分包括地址指针的修改、循环控制条件的修改等。
- (3) **循环控制部分**。它是控制循环的关键，判断循环条件满足与否。例如判断循环次数是否为0等。





4.4 分支与循环程序设计

单重循环程序设计

➤ 例5.10 计算 n 的阶乘。

➤ 算法分析：阶乘 (factorial) 计算的公式为： $n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$ 。因此，需要循环 n 次，每次循环中完成一次乘法。

;PROG0510



4.4 分支与循环程序设计

;PR0G0510

. 386

.model flat, stdcall

option casemap:none

includelib

msvcrt.lib

printf

PROTO C :dword, :vararg

.data

Fact dword ?

N equ 6

szFmt byte 'factorial (%d)=%d', 0ah, 0;输出结果格式字符串

.code

start:

mov ecx, N;循环初值

mov eax, 1;Fact初值

e10:

imul eax, ecx;Fact=Fact*ECX

loop e10;循环N次

mov Fact, eax;保存结果

invoke printf, offset szFmt, N, Fact;打印结果

ret

end start





4.4 分支与循环程序设计

多重循环程序设计

- 有些复杂问题使用单重循环可能无法解决，此时就需要设计多重循环程序。
- 在多重循环的程序中，**内层循环嵌套于外层循环**，循环的嵌套层次没有限制。
- **各层循环都有各自的循环次数、循环体、循环结束条件，相互之间不能干扰、交叉。**





➤ DELAY

PROC

➤ MOV

BL, 20 ;置外循环次数

DELAY10:

MOV CX, 2801 ;置内循环次数

WT:

LOOP WT ;内循环体

DEC BL ;修改外循环次数

JNZ DELAY10 ;外循环控制

RET

➤ DELAY

ENDP





➤ 冒泡排序



4.5 浮点运算

➤ 4.5.1 浮点数的表述与存储

格式	说明
单精度	32位：1位符号位，8位阶码，23位为有效数字的小数部分。
双精度	64位：1位符号位，11位阶码，52位为有效数字的小数部分。
扩展精度	80位：1位符号位，15位阶码，1位为整数部分，63位为小数部分。

01000010 11101110 00011011 10100110

根据单精度的划分方式把32位划分成三部分：

- ①.符号位为0，为正数；
- ②.指数为 10000101（133），减去127得6；
- ③.尾数加上1后为1.11011100001101110100110，十进制表示为：1.86021876
尾数乘以2的6次方后可得结果为：119.05400（单精度7~8位有效数字）



4.5 浮点运算

➤ 4.5.2 浮点数寄存器

1. 浮点寄存器栈



图 4-13 浮点寄存器栈示意图

;PROG0409.asm

.586

.model flat, stdcall

option casemap:none

includelib msvcrt.lib

printf PROTO C :ptr sbyte, :VARARG

.data

szMsg byte "%f", 0ah, 0

a real8 3.2

b real8 2.6

m real8 7.1

f real8 ?

.code

start:

finit ;finit为FPU栈寄存器的初始化

fld m ;fld为浮点值入栈

fld b

fmul st(0),st(1) ;fmul为浮点数相乘，结果保存在目标操作数中

fld a

fadd st(0),st(1) ;fmul为浮点数相加，结果保存在目标操作数中

fst f ;fst将栈顶数据保存到内存单元

invoke printf, offset szMsg, f

4.5.2 浮点寄存器

计算表达式 $f = a + b * m$ 的值



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

4.5.2 浮点寄存器



图 4-15 标志及存取示意图

状态寄存器 16 位, 表明浮点处理单元当前的各种操作状态, 每条浮点指令运算后都会更新状态寄存器以反映执行结果情况, 与整数处理单元的 EFLAGS 作用功能类似。如图 4-16 所示。

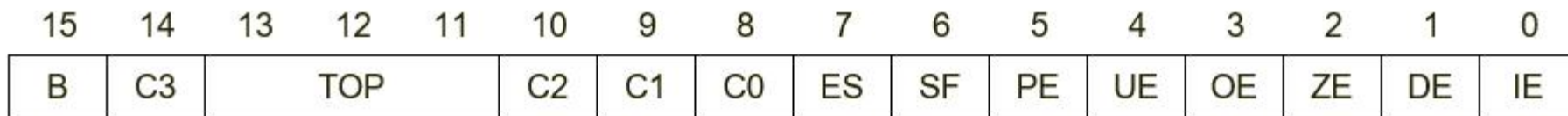


图 4-16 浮点状态寄存器



4.5.3 浮点指令及其编程

2. 数据定义

a	real8	3.2	;定义64位浮点数变量a，初始化为3.2
b	real10	100.25e9	;定义80位浮点数变量b，初始化为100.25e9
c	qword	3.	;定义64位浮点数变量c，初始化为3.0
d	qword	3	;定义64位整型变量d，初始化为3



4.5.3 浮点指令及其编程

3. 寻址方式

(1) 寄存器寻址：操作数保存在指定的数据寄存器栈中，用ST(i)表示。

例4.35 指令：fadd st(0), st(1)

将寄存器栈中的ST(0)和ST(1)相加，结果存储在ST(0)中。

(2) 存储器寻址：操作数在内存中，内存中的数据可以采用与数据有关的存储器寻址方式访问。

例4.36 指令：fld m

将在内存定义的变量m加载到浮点寄存器栈中，m保存在内存中，以直接寻址方式访问。



4.5.3 浮点指令及其编程

4. 指令

表 4-9 数据传送指令

指令	说明
FLD src	将源操作数 src (mem32/mem64/mem80/ST(i))加载到寄存器栈 ST(0)
FST dest	将寄存器栈 ST(0)保存到目标操作数 dest(mem32/mem64/mem80/ST(i))，ST(0)不出栈
FSTP dest	将寄存器栈 ST(0)保存到目标操作数 dest(mem32/mem64/mem80/ST(i))，ST(0)出栈



4.5.3 浮点指令及其编程

4. 指令

表 4-10 常数加载指令

指令	说明
FLD1	将常数 1.0 加载到寄存器栈 ST(0)
FLDZ	将常数 0.0 加载到寄存器栈 ST(0)
FLDPI	将常数 π 加载到寄存器栈 ST(0)
FLDL2T	将常数 $\log_2 10$ 加载到寄存器栈 ST(0)
FLDL2E	将常数 $\log_2 e$ 加载到寄存器栈 ST(0)
FLDLG2	将常数 $\log_{10} 2$ 加载到寄存器栈 ST(0)
FLDLN2	将常数 $\log_e 2$ 加载到寄存器栈 ST(0)



4.5.3 浮点指令及其编程

4. 指令

表 4-11 算术运算指令

指令	格式	说明
FADD	FADD	ST(0)加 ST(1)，结果暂存在 ST(1)中，ST(0)出栈，新的 ST(0)保存运算结果
	FADD src	将 src 与 ST(0)相加，结果保存在 ST(0)中
	FADD st(i),st(0)	ST(0)加 ST(i)，结果保存在 ST(i)中
	FADD st(0),st(i)	ST(0)加 ST(i)，结果保存在 ST(0)中
	FADDP st(i),st(0)	ST(0)加 ST(i)，结果保存在 ST(i)中，ST(0)出栈
FSUB	FSUB	ST(1)减去 ST(0)，结果暂存在 ST(1)中，ST(0)出栈，新的 ST(0)保存运算结果
	FSUB src	ST(0)减去 src，结果保存在 ST(0)中
	FSUB st(i),st(0)	ST(i)减去 ST(0)，结果保存在 ST(i)中
	FSUB st(0),st(i)	ST(0)减去 ST(i)，结果保存在 ST(0)中
	FSUBP st(i),st(0)	ST(i)减去 ST(0)，结果保存在 ST(i)中，ST(0)出栈
FMUL	FMUL	ST(0)乘 ST(1)，结果暂存在 ST(1)中，ST(0)出栈，新的 ST(0)保存运算结果
	FMUL src	ST(0)乘以 src，结果保存在 ST(0)中
	FMUL st(i),st(0)	ST(i)乘以 ST(0)，结果保存在 ST(i)中
	FMUL st(0),st(i)	ST(0)乘以 ST(i)，结果保存在 ST(0)中
	FMULP st(i),st(0)	ST(i)乘以 ST(0)，结果保存在 ST(i)中，ST(0)出栈
FDIV	FDIV	ST(1)/ST(0)，结果暂存在 ST(1)中，ST(0)出栈，新的 ST(0)保存运算结果
	FDIV src	ST(0)/src，结果保存在 ST(0)中
	FDIV st(i),st(0)	ST(i)/ST(0)，结果保存在 ST(i)中
	FDIV st(0),st(i)	ST(0)/ST(i)，结果保存在 ST(0)中

4.5.3 浮点指令及其编程

4. 指令

表 4-12 浮点比较指令

指令	说明
FCOM	比较 ST(0)和 ST(1)的大小关系
FCOM src	比较 ST(0)和 src 的大小关系
FCOM st(i)	比较 ST(0)和 ST(i)的大小关系
FCOMP	比较 ST(0)和 ST(1)的大小关系，完成比较后 ST(0)出栈
FCOMP src	比较 ST(0)和 src 的大小关系，完成比较后 ST(0)出栈
FCOMP st(i)	比较 ST(0)和 ST(i)的大小关系，完成比较后 ST(0)出栈

表 4-13 根据比较结果视线转移指令

条件	C3	C2	C0	转移指令
ST(0)>操作数	0	0	0	JA/JNBE
ST(0)<操作数	0	0	1	JB/JNAE
ST(0)=操作数	1	0	0	JE/JZ
无序	1	1	1	(无)



4.5.3 浮点指令及其编程

4. 指令

表 4-14 超越函数指令

指令	说明
FSIN	计算 ST(0)的 sin 值, 结果保存在 ST(0)中
FCOS	计算 ST(0)的 cos 值, 结果保存在 ST(0)中
FPTAN	计算 ST(0)的 tan 值, 结果保存在 ST(0)中
FPATAN	计算 ST(0)的 arctan 值, 结果保存在 ST(0)中
F2XM1	计算 2 的 ST(0)次方, 减去 1, 结果保存在 ST(0)中

(5) FPU 控制指令

该类指令实现针对 FPU 的操作控制, 如表 4-15 所示。

表 4-15 FPU 控制指令

指令	说明
FINIT	初始化 FPU
FLDCW src	从 src 装入 FPU 的控制字
FSTCW dest	保存状态字的值到 dest
FCLEX	清除异常
FNOP	空操作



4.5.3 浮点指令及其编程

; PROG0410.asm 例4.37 输入圆的半径, 计算圆面积。

.data

```
szMsg1  byte          "%lf", 0
szMsg2  byte          "%lf", 0ah, 0
r        real8         ?;圆半径
S        real8         ?;圆面积
```

.code

start:

```
finit; finit为FPU栈寄存器的初始化
invoke scanf, offset szMsg1, offset r
fld      r
fld      r
fmulp st(1), st(0)
fldpi
fmulp st(1), st(0)
fst S;fst将栈顶数据保存到内存单元
invoke printf, offset szMsg2, S
ret
```

end start



4.6 程序优化

4.6.1. 运行时间优化

1. 选择执行速度快的指令

(1) 寄存器清零

00401277 B8 00 00 00 00

MOV EAX, 0

0040127C 2B C0

SUB EAX, EAX

0040127E 33 C0

XOR EAX, EAX

(2) 加减

EBX=EAX-30

00401225 8B D8

MOV EBX, EAX

00401227 83 EB 1E

SUB EBX, 1EH

0040122A 8D 58 E2

LEA EBX, [EAX-1Eh]





4.6.1 运行时间优化

(3) 乘除

SHR **EAX, 4; EAX=EAX/16**

SHL **EAX, 3; EAX=EAX*8**

LEA **EAX, [EBX+EBX*4]; EAX=EBX*5**



4.6.1 运行时间优化

2.操作的转化

除法指令比乘法指令的速度慢。如果程序中的除法操作中，除数为一个常数，那么可以将除法转换为乘法来进行，以提高程序执行的速度。

$$125 \div 25$$

```
MOV    EAX, 125
```

```
MOV    ESI, 0A3D70A4H ; ESI = (1000000000H + 24) / 25
```

```
MUL    ESI
```

```
; EDX = 5
```





4.6.1 运行时间优化

3.分支的转化

min (x, y) = x+ (((y-x) >>31) & (y-x)) **eax=min (eax, ebx)**

4.提高Cache命中率

for (i = 0; i < m; i++)
 for (j = 0; j < n; j++)
 A[i][j]++;

for (i = 0; i < n; i++)
 for (j = 0; j < m; j++)
 A[j][i]++;



```

unionBuf    UNION
fileBuf     BYTE 4096 DUP (?)
outputBuf   BYTE 2000 DUP (?)
unionBuf    ENDS

```

4.6.2 占用空间优化

1.短指令

```

0040127C 83 EC 04
0040127F 51

```

```

SUB    ESP, 4
PUSH   ECX

```

2.联合

```

unionBuf    UNION
fileBuffer  BYTE 4096 DUP (?)
outputBuffer BYTE 2000 DUP (?)
unionBuf    ENDS

```

```

fileBuffer  BYTE 4096 DUP (?)
outputBuffer BYTE 2000 DUP (?)

```





感谢关注聆听！



张华平

Email: kevinzhang@bit.edu.cn

微博: @ICTCLAS张华平博士

实验室官网:

<http://www.nlpir.org>



大数据千人会

