本章的学习难点
(1) 无条件转移指令的灵活运用
(2) 条件转移指令的正确选择
(3) 分支出口的安排与汇合
(4) 循环程序的结构和控制循环的方法
(5) CALL与RET指令的执行过程，堆栈操作
(6) 综合应用前几章的内容，编写和调试程序

4.1 概述

汇编语言程序设计的一般步骤:
（1）分析问题，选择合适的解题方法。
（2）根据具体问题，确定输入输出数据的格式。
（3）分配存贮区并给变量命名(包括分配寄存器)。
（4）绘制程序的**流程图**，即将解题方法和步骤用程序流程图的形式表示出来。
（5）**根据流程图编写程序。**
（6）**静态检查**与动态调试

> 写流程图是还比较方便后面编写程序
>
> 静态检查就是走查，能找出许多问题。

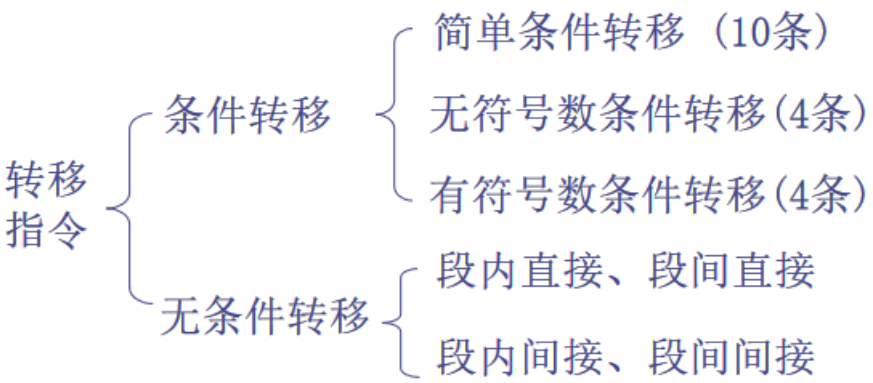
几种流程框图的说明见书P99

4.2 顺序程序设计

这里就不讲了，上节就讲过了。
书 P145, 作业 4.1

第二次上机题:
目的：复习、巩固第三章学习的指令

4.3 分支程序设计



4.3.1 简单条件转移（10条）

根据单个标志位 CF、ZF、SF、OF、PF的值确定是否转移。

语句格式： **[标号:] 操作符 短标号**

> 短标号是一个标号。该标号代表条件成立时，想转移到的目的地址。若该目的地址与当前(IP)之间的字节距离在-128 和 127之间，则称该标号为短标号。
>
> 比如 JE NEXT

指令名称	助记符	转移条件
相等/等于0转	JZ / JE	ZF=1时，转移
不相等/不等于0转	JNZ / JNE	ZF=0时，转移
为负转	JS	SF=1时，转移
为正转	JNS	SF=0时，转移
溢出转	JO	OF=1时，转移
未溢出转	JNO	OF=0时，转移
进位转	JC	CF=1时，转移
无进位转	JNC	CF=0时，转移
偶转移	JP / JPE	PF=1时，偶转移
奇转移	JNP / JPO	PF=0时，奇转移

> 例如，JZ / JE　ZF=1时，转移
>
> 结果相等，即A-B=0，即ZF=1时，即结果为0时。跳转
> 这个逻辑又和bool判断不同。
>
> 划线部分对有符号数。

4.3.2 无符号数条件转移指令（4条）

Above,Blow

JA / JNBE

a>b转移
则a-b>0，即a-b一定无进位，且大于0。

当 CF=0 且 ZF=0时，转移

JAE / JNB

a>=b转移
则a-b一定无进位，有可能a-b=0。

当 CF=0 或者 ZF=1时，转移

JB / JNAE

a<b转移
则a-b<0，因为是无符号数变成负数了，故是有进位的。且a-b!=0

> 顺带一提，无符号数才有进位，看CF，有符号数才有溢出，看OF

当 CF=1 且 ZF=0时，转移

JBE / JNA

a<=b转移
则a-b<=0，变负数，有进位，可能为0。

当 CF=1 或者 ZF=1时，转移

4.3.3 有符号数条件转移指令

Great,Little

JG / JNLE 短标号

将(AX),(BX)中的数据当成有符号数，
执行 (AX) – (BX)>0, 则SF、OF会相等，ZF=0

> 为什么说SF=OF
> 正常情况下 SF=OF=0，无溢出，为正。
> 特殊情况，例如(AX)>0,(BX)<0 ，结果一定是大于0的，但可能发生溢出了，导致结果变成负的。
> 所以SF=OF=1。实际上(AX)>(BX)是成立的。

当 SF=OF 且 ZF=0时，转移

JGE / JNL 短标号

可能相等。

当 SF=OF 或者 ZF=1时，转移

JL / JNGE 短标号

将(AX),(BX)中的数据当成有符号数，
执行 (AX) – (BX)< 0
此时若SF=1,那么铁定无溢出 OF=0
另一种情况(AX)<0,(BX)>0，这时候可能会溢出，反而会使结果成正数，即OF=1,SF=0

当 SF≠OF 且 ZF=0时，转移

JLE / JNG 短标号

当 SF≠OF 或者 ZF=1时，转移

4.3.4 无条件转移指令

格式	名称	功能	可用寻址方式
JMP 标号	段内直接	(IP/EIP)+位移量 → IP/EIP	立即方式
JMP OPD	段内间接	(OPD) → IP/EIP	寄存器、存储器方式
JMP 标号	段间直接	标号的EA → IP/EIP 段首址 → CS	立即方式
JMP OPD	段间间接	(OPD) → IP/EIP (OPD+2/4) → CS	存储器方式

设在数据段中有：

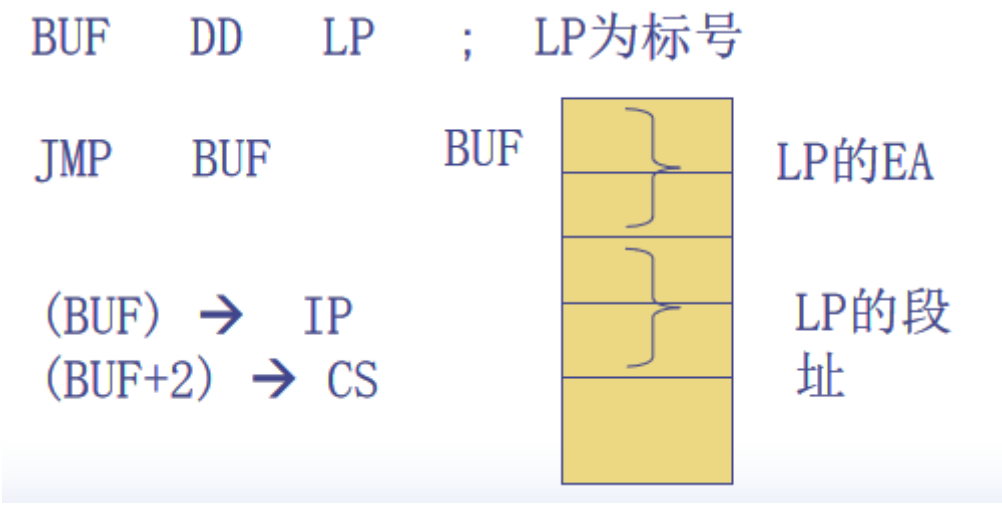
```
BUF   DW   L1   ; L1为标号
```

功能等价的转移指令：

1. JMP L1
2. JMP BUF
3. LEA BX , BUF
   JMP WORT PTR [BX]
4. MOV BX , BUF
   JMP BX

无条件转移指令的转移范围不受限制。程序段A,B的长度均大于128个字节都没事。

关于段间间接转移



所以需要定义标号为双字，分别存段址和EA
(OPD) → IP/EIP (OPD+2/4) → CS

4.3.5 条件转移伪指令

条件控制流伪指令 P323

书P253也有介绍

```
.IF  条件表达式
     语句序列
.ELSEIF  条件表达式
         语句序列
……
.ELSE
      语句序列
.ENDIF
```

4.3.6 分支程序设计举例

下面这个程序用于计算0-9的立方值，之前有错误，INT 21H总不记得加H，这样导致EXIT语句没有成功退出DOS调用，继续跳到ERR去了

```
.386
DATA SEGMENT USE16
INPUT DB 'Please input X(0~9):$'
TAB DW 0,1,8,27,64,125,216,343,512,729
X DB ?
```

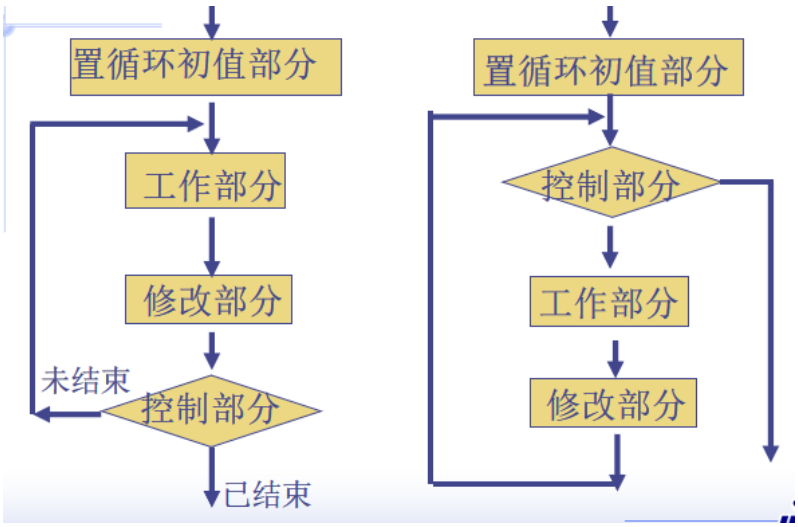
```
XXX DW ?
INERR DB 0AH,0DH,'Input error! try again',0AH,0DH,'$'
DATA ENDS

STACK SEGMENT USE16 STACK
      DB 200 DUP(0)
STACK  ENDS

CODE   SEGMENT USE16
       ASSUME CS:CODE,DS:DATA,SS:STACK
BEGIN: MOV AX,DATA
       MOV DS,AX
NEXT:  MOV DX,OFFSET INPUT
       MOV AH,9
       INT 21H
       MOV AH,1
       INT 21H
       CMP AL,'0'
       JB ERR
       CMP AL,'9'
       JA ERR
       AND AL,0FH;因为输入的是Ascii码值，如输入3，实际上是33H，要^0FH之后变成03H，才是数字3
       MOV X,AL
       XOR EBX,EBX
       MOV BL,AL
       MOV AX,TAB[EBX*2]
       MOV XXX,AX
EXIT:  MOV AH,4CH
       INT 21H
ERR:   MOV DX,OFFSET INERR
       MOV AH,9
       INT 21H
       JMP NEXT
CODE   ENDS
       END BEGIN
```

4.4 循环程序设计

1. 循环程序的结构



就像do while以及 while语句一样，第一个一定会先执行一次

2. 循环控制方法

（1）计数控制法

1. LOOP 标号

(CX / ECX) -1 → CX / ECX
若 （CX / ECX）**不为0**, 则转标号处执行
基本等价于： DEC CX / ECX
JNZ 标号

（LOOP指令**对标志位无影响！**）

2. JCXZ 标号 / JECXZ 标号

若 （CX / ECX) 为0, 则转标号处执行。

3. LOOPE /LOOPZ 标号

(CX / ECX) -1 → CX / ECX
若 （CX / ECX）**不为0**, 且**ZF=1**,则转标号处执行。
（**等于或为0循环转移指令,** 本指令对标志位无影响）

4. LOOPNE /LOOPNZ 标号

(CX / ECX) -1 → CX / ECX
若CX / ECX)≠0, 且ZF=0,则转标号处执行。

5.总结

1. 所有的循环指令对(cx/ecx)-1的操作**不影响标志位。**（所以ZF=1怎么来的？它如果说不影响除ZF以外的标志位，我比较理解）
2. 16位段中，默认使用CX寄存器，32位段默认用ECX寄存器。

（2）条件控制法

循环次数未知，相当于if(XX) break;

这个就要自己设计跳出条件

```
    MOV CL,0
L1: AND AX,AX
    JZ EXIT;AX为0时跳出
    SAL AX,1;将AX的最高位移到CF中
    JNC L;CF为0转L1
    INC CL
    JMP L1
;这个程序用来统计AX中1的个数。
```

3. 单重循环程序设计

见P116

- [x] 多看例题

4. 多重循环程序设计

见P121

- [x] 多看例题，把例题都看一遍

4.5 子程序设计

4.5.1 子程序的概念

将经常要使用的或者重复的程序段设计成可供反复调用的独立程序段，在需要时，用控制指令调用它，在执行完后，再返回调用它的程序中继续执行。这样的独立程序段称为子程序。

4.5.2 子程序的定义

```
主程序中   CALL 子程序名

子程序名   PROC   NEAR 或者 FAR
             ……
             ……
           RET ;返回
子程序名   ENDP
```

- NEAR可以省略
- FAR是在不同的段中需加的。
- proc代表——procedure 程序。

4.5.3 子程序的调用与返回

调用

段内直接　CALL 过程名
段间直接　CALL FAR PTR 过程名
段内间接　CALL **WORD** PTR OPD　；word是因为要纪录EA
段间间接　CALL **DWORD** PTR OPD ；DWORD是因为两个字分别记录段址和EA、

返回

返回指令：RET

CPU 要做的工作：

(1) 保存断点

直接：(IP / EIP) →↓ (SP / ESP)

间接：(CS) →↓(SP / ESP)
(IP / EIP) →↓(SP / ESP)

(2)将子程序的地址送 CS, IP

段内：子程序的入口偏移地址→ IP / EIP

段间：子程序的入口偏移地址→ IP /EIP
子程序的段地址→ CS

(3) 返回指令

段间返回：
(1) ↑(SP) → IP / EIP
(2) ↑(SP) → CS

段内返回：
↑(SP) → IP /EIP

4.5.4 子程序调用现场的保护方法

可以在主程序中做，也可以在子程序中做，一般在子程序里做。

可以用1.3.2小节，书P11的指令，PUSHA/POPA，PUSHAD/POPAD，这样来保护和恢复全部通用寄存器的内容。

4.5.5 主程序与子程序间的参数传递

到底用哪种参数传递法，视情况而定，有时混用

详见P134的4.5.6 子程序及其调用举例

1. 寄存器法

9号，10号调用就是用的此法。

就是子程序的入口参数和出口参数都放在约定的寄存器中。

优点：信息传递快，编程方便，节省内存单元。

缺点：只适用于要传递参数较少的状况
还要注意一点，出口参数的寄存器不需要现场保护，因为子程序要交给主程序的，不能现场保护和恢复后，把要交的结果给抵消了。

2. 约定单元法

把入口参数和出口参数都放在约定好的存储单元中

优点：不易出错，参数数量可多可少。

缺点：占用一定的存储单元，增加了编程的复杂性。

3. 堆栈法

放在公用的堆栈区

优点：较高的灵活性，参数数量可多可少

缺点：参数和子程序的返回地址混杂在一起，存取参数时必须小心的计算它在栈中的位置

- 0DH回车，0AH换行，这两个写的顺序无所谓。但要在一起。