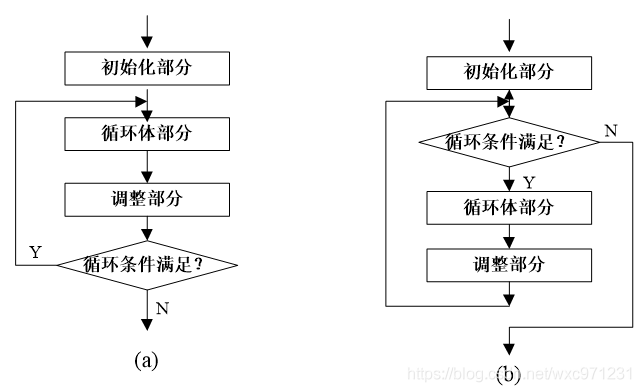
## 一、循环程序设计

## （1）循环程序设计示例

### 1. 两种循环结构



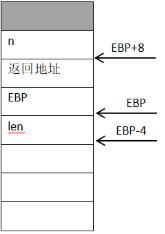
### 2. 简单循环示例

* 有简单循环程序

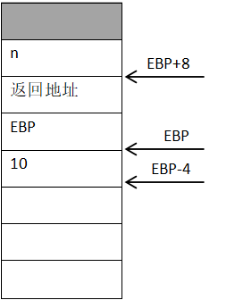
//统计无符号整数n作为十进制数时的位数  
int cf320(unsigned int n)  
{  
 int len = 0;  
 do {  
 len++;  
 n = n/10;  
 } while (n != 0);  
 return len ;   
}

* 现在把它反汇编（关闭优化）

//堆栈传参数，eax传返回值  
push ebp  
 mov ebp, esp  
 push ecx ;在堆栈，安排局部变量len  
 mov DWORD PTR [ebp-4], 0 ; len=0;  
LN3cf320: ; do {  
 ; len++;  
 mov eax, DWORD PTR [ebp-4]  
 add eax, 1  
 mov DWORD PTR [ebp-4], eax  
 ; n = n/10;  
 mov eax, DWORD PTR [ebp+8]  
 xor edx, edx ;因n是无符号数，用XOR指令清0  
 mov ecx, 10  
 div ecx  
 mov DWORD PTR [ebp+8], eax  
 cmp DWORD PTR [ebp+8], 0  
 jne SHORT LN3cf320  
 ; return len ;  
 mov eax, DWORD PTR [ebp-4] ;准备返回值  
 ;}  
 mov esp, ebp ;撤销局部变量len  
 pop ebp ;撤销堆栈框架  
 ret

* 简单分析
  1. 堆栈示例  
     
  2. 32位数除法，先把被除数扩展到64位，这里是无符号数，所以直接0扩展就行。使用64位无符号数除法div OPDR，被除数放在edx:eax中，除数OPDR这里是ecx，商存在eax，余数在edx
  3. 没优化，改一个数的值要三步：从堆栈取到寄存器，改寄存器值，存回堆栈。几乎所有数据计算之后都要先在堆栈更新，要用时再从堆栈取
* 现在再打开优化反汇编一次

push ebp  
 mov ebp, esp  
 ;ECX作为len  
 xor ecx, ecx ;len=0;  
 push esi ;在使用ESI之前，保护之  
LL3cf320: ;do {  
 ; len++;  
 ; n = n/10;  
 mov eax, DWORD PTR [ebp+8]  
 push 10 ;准备借助堆栈送到ESI  
 xor edx, edx ;使得EDX=0  
 pop esi ;使得ESI=10  
 div esi  
 inc ecx  
 mov DWORD PTR [ebp+8], eax  
 test eax, eax ;测试n是否为0？  
 jne SHORT LL3cf320  
 ; return len ;  
 mov eax, ecx ;准备返回值  
 pop esi ;恢复ESI  
 ;}  
 pop ebp  
 ret

* 简单分析
  1. 堆栈示例  
     
  2. 仍使用64位无符号数除法div OPDR，但除数OPDR用了源变址寄存器esi，可能因为本质是指针寄存器，所以这里利用堆栈给它赋值，esi=0x0A
  3. 优化
     1. 每轮循环一开始就把n取到寄存器，循环结束时才存回，减少堆栈操作。
     2. 使用test指令判断等于0

## （2）循环指令

1. **循环指令的说明**
   1. 类似于条件转移指令，**段内转移，相对转移**方式。
   2. 通过在**指令指针寄存器EIP上加一个地址差的方式实现转移**。
   3. 只**用一个字节（8位）表示地址差，转移范围仅在-128至+127之间**。
   4. 在保护方式（32位代码段）下，以**ECX作为循环计数器**。在实方式下，以**CX作为循环计数器**。
   5. **不影响**各标志。
2. **计数循环指令LOOP**

| 名称 | LOOP（计数循环指令） |
| --- | --- |
| 格式 | LOOP LABEL |
| 动作 | 令使寄存器ECX的值减1，如果结果不等于0，则转移到标号LABEL处，否则顺序执行LOOP指令后的指令 |
| 注意 | **用于循环次数已知的**循环，如for循环 |
|  | 计数器必须用ecx先设置计数器ECX初值，即循环次数。 |
|  | 由于首先进行ECX减1操作，再判结果是否为0，所以最多可循环 2 32 2^{32} 232遍。 |

;统计寄存器EAX中位是1的个数  
  
 XOR EDX, EDX ;清EDX  
 MOV ECX, 32 ;设置循环计数  
LAB1: SHR EAX, 1 ;右移1位（最低位进入进位标志CF）  
 ADC DL, 0 ;统计（实际是加CF）  
 LOOP LAB1 ;循环

1. **等于/全零循环指令LOOPE/LOOPZ**

| 名称 | LOOPE/LOOPZ（等于/全零循环指令） |
| --- | --- |
| 格式 | LOOPE（LOOPZ） LABEL |
| 动作 | 指令使寄存器ECX的值减1，如果**结果不等于0**，并且零标志**ZF等于1（表示相等**），则**转移到标号LABEL处**，否则顺序执行。 |
| 注意 | 适用于循环比较直到找到**相等字符**的情况 |
|  | 同一条指令，有两个助记符 |
|  | 指令本身实施的ECX减1操作不影响标志 |
|  | 可以在循环开始前把ecx设为-1，相当于最大循环FFFFFFFFH-1次，退出循环后用not ecx把ecx按位取反，即可得LOOPE执行次数 |

//在一个字符数组中查找第一个非空格字符，假设字符数组buff的长度为100：  
  
 LEA EDX, buff ;指向字符数组首  
 MOV ECX, 100 ;  
 MOV AL, 20H ;空格字符  
 DEC EDX ;为了简化循环，先减1  
LAB2:  
 INC EDX ;调整到指向当前字符  
 CMP AL, [EDX] ;比较  
 LOOPE LAB2

1. **不等于/非零循环指令LOOPNE/LOOPNZ**

| 名称 | LOOPNE/LOOPNZ（等于/全零循环指令） |
| --- | --- |
| 格式 | LOOPNE（LOOPNZ） LABEL |
| 动作 | 指令使寄存器ECX的值减1，如果结果**不等于**0，并且零标志**ZF等于0（表示不相等）**，则**转移到标号LABEL**处，否则顺序执行。 |
| 注意 | 适用于循环比较直到找到**不相等字符**的情况 |
|  | 同一条指令，有两个助记符 |
|  | 指令本身实施的ECX减1操作不影响标志 |
|  | 可以在循环开始前把ecx设为-1，相当于最大循环FFFFFFFFH-1次，退出循环后用not ecx把ecx按位取反，即可得LOOPNE执行次数 |

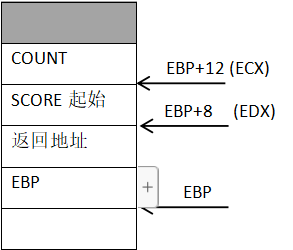
//演示LOOPNE指令的使用：嵌入汇编代码形式，测量由用户输入的字符串之长度  
  
#include <stdio.h>  
int main( )  
{ char string[100]; //用于存放字符串  
 int len; //用于存放字符串长度  
 printf("Input string:"); //由用户输入一个字符串  
 scanf("%s",string);  
   
 \_asm   
 {  
 LEA EDI, str //使得EDI指向字符串  
 XOR ECX, ECX //假设字符串“无限长”  
 XOR AL, AL //使AL=0（字符串结束标记）  
 DEC EDI //为了简化循环，先减1  
 LAB3: INC EDI //指向待判断字符  
 CMP AL, [EDI] //是否为结束标记  
 LOOPNE LAB3 //如果不是结束标记，继续循环  
 NOT ECX //据ECX，推得字符串长度  
 MOV len, ECX  
 }  
 printf("len=%d\n",len); //显示为len=12  
 return 0;  
}

## （3）计数器转移指令

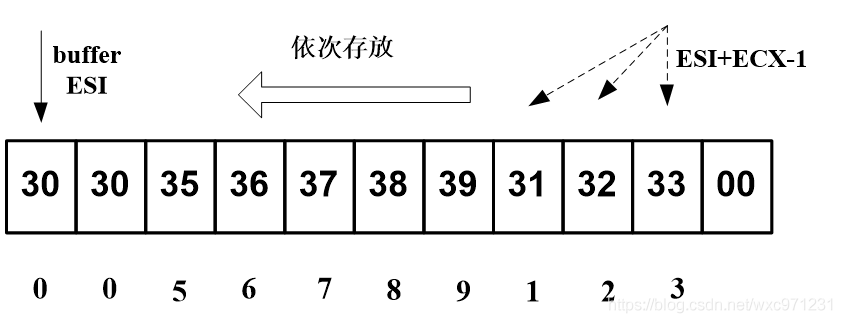
* 上面的第一条LOOP指令，提供了一种指定循环次数的方法，但它有一个问题：由于是先将ecx减一再判断，当设定循环次数为0时，实际上会循环FFFFFFFFH次。为了解决这个问题，IA32专门提供了一条用**ECX是否为0**作为判断条件的条件转移指令JECXZ/JCXZ

| 名称 | JECXZ/JCXZ（计数器转移指令） |
| --- | --- |
| 格式 | JECXZ（JCXZ） LABEL |
| 动作 | 指令实现当寄存器**ECX（CX）的值等于0时转移到标号LABEL**处，否则顺序执行。 |
| 注意 | **通常在上面几条循环指令之前使用，这样当循环次数为0时，就可以跳过循环体** |
|  | JECXZ对应判断ECX值；JCXZ对应判断CX值 |

//计算由用户输入的若干成绩的平均值  
//演示堆栈传递参数调用子程序和JECXZ指令的使用：  
//注意JECXZ和LOOP配合  
  
#include <stdio.h>  
#define COUNT 5 //假设成绩项数  
int main()  
{  
 int score[COUNT]; //用于存放由用户输入的成绩  
 int i, average;  
 for (i=0; i < COUNT; i++)  
 { //由用户从键盘输入成绩  
 printf("score[%d]=", i);  
 scanf("%d", &score[i]);  
 }  
 //调用子程序计算成绩平均值  
 \_asm {  
 LEA EAX, score  
 PUSH COUNT //把数组长度压入堆栈  
 PUSH EAX //把数组起始地址压入堆栈  
 CALL AVER //调用子程序  
 ADD ESP, 8 //平衡堆栈  
 MOV average, EAX  
 }  
 printf("average=%d\n",average);  
 return 0;  
  
  
 \_asm {  
 AVER: //子程序入口  
 PUSH EBP  
 MOV EBP, ESP  
 MOV ECX, [EBP+12] //取得数组长度  
 MOV EDX, [EBP+8] //取得数组起始地址  
 XOR EAX, EAX //将EAX作为和sum  
 XOR EBX, EBX //将EBX作为下标i  
 JECXZ OVER //如数组长度为0，不循环累加  
 NEXT:  
 ADD EAX, [EDX+EBX\*4] //累加  
 INC EBX //调整下标i  
 LOOP NEXT   
   
 CDQ //被除数符号扩展到64位，准备做除法  
  
 IDIV DWORD PTR [EBP+12]  
 OVER:  
 POP EBP //撤销堆栈框架  
 RET //返回  
 }  
}

* 说明：
  1. 堆栈示意  
     
  2. 32位有符号数除法，先用CDQ把EAX符号扩展到EDX:EAX

## 二、综合示例

1. 把二进制数转换为十进制数的ASCII码串
   1. 方法：
      1. 把一个整数除以10，所得的余数就是个位数。
      2. 把所得的商再除以10，所得的余数就是十位数。
      3. 继续把所得的商除以10，所得的余数就是百位数。
      4. 依次类推，就可以得到一个整数的各位十进制数字了。
   * 32位二[进制](https://so.csdn.net/so/search?q=%E8%BF%9B%E5%88%B6&spm=1001.2101.3001.7020)数能表示的最大十进制数只有10位，**循环地除上10次，就可以得到各位十进制数**，注意这样得到的结果**最前面有若干个0**
   1. 把一位十进制数转换为对应的ASCII码，只要加上数字符‘0’的ASCII码。
   2. 存放顺序：  
      由于先得到个位数，然后得到十位数，再得到百位数，**所以在把所得的各位十进制数的ASCII码存放到字符串中去时，要从字符串的尾部开始。**  
      

int main( )  
{  
 unsigned uintx = 56789123; //无符号整型变量  
 char buffer[11]; //用于存放ASCII码串的缓冲区  
 \_asm   
 {  
 LEA ESI, buffer ;获存放字符串的缓冲区首地址  
 MOV EAX, uintx ;取得待转换的数据  
 MOV ECX, 10 ;循环次数（十进制数的位数）  
 MOV EBX, 10 ;十进制的基数是10  
NEXT:  
 XOR EDX, EDX ;形成64位的被除数（无符号数除）  
 DIV EBX ;除以10，EAX含商，EDX含余数  
 ADD DL, '0' ;把析出十进制位转成对应的ASCII码  
 MOV [ESI+ECX-1], DL ;保存到缓冲区  
 LOOP NEXT ;计数循环  
 ;  
 MOV BYTE PTR [ESI+10],0 ;设置字符串结束标志  
}  
 printf("%s\n", buffer); //输出字符串  
 return 0;  
}

1. 改进上面的程序，（1）设二进制数是有符号的。如果负数，则所得字符串的第一个字符应该是负号。  
   （2）不需要前端可能出现的字符‘0’

int main( )  
{  
 int intx = -57312;  
char buffer[16]; //足够长  
//printf(“%d\n”,intx);  
  
 \_asm   
 {  
 LEA ESI, buffer ;置指针初值  
 MOV EAX, intx ;取得待转换的数据  
 CMP EAX, 0 ;判断待转换数据是否为负数  
 JGE LAB1 ;非负数，转  
 MOV BYTE PTR [ESI], '-' ;先保存一个负号  
 INC ESI ;调整指针  
 NEG EAX ;取相反数，得正数  
LAB1:  
 MOV ECX, 10 ;最多循环10次  
 MOV EBX, 10 ;每次除以10  
 MOV EDI, 0 ;置有效位数的计数器初值  
NEXT1:  
 XOR EDX, EDX  
 DIV EBX ;获得1位十进制数  
 ;  
 PUSH EDX ;把所得1位十进制数压入堆栈  
 INC EDI ;有效位数增加1  
 ;  
 OR EAX, EAX ;测试结果（商）  
 LOOPNE NEXT1 ;如结果不为0，考虑继续循环  
 MOV ECX, EDI ;置下一个循环的计数  
NEXT2:  
 POP EDX ;从堆栈弹出余数  
 ADD DL, '0' ;转成对应的ASCII码  
 MOV [ESI], DL ;依次存放到缓冲区  
 INC ESI  
 LOOP NEXT2 ;循环处理下一位  
 ;  
 MOV BYTE PTR [ESI], 0 ;设置字符串结束标志  
}  
 printf("%s\n", buffer); //输出字符串  
 return 0;  
}