滴水信息

滴水逆向培训

基础教程

昆山滴水信息技术有限公司

（内部学习资料）

地址：昆山市巴城镇学院路 88 号
邮编：215311
TEL：0512-57882866
官网：www.dtdishui.com
论坛：www.dtdebug.com
EMAIL：kunshandishui@163.COM

前言

在写这篇前言之前，首先非常感谢多年以来一直关心和支持滴水的朋友们，同时也非常感谢那些不停的鞭策我们勇往直前的批评者。在你们的掌声与谩骂声中我们一路走来。

2008 年滴水公司成立之日起，我们就面临着一个非常严峻的现实，巨大的工作量无人可以分担。既招不到合适的程序员，也找不到愿意同行的伙伴。在孤独与寂寞中我们慢慢的爬行。。。。。。

国内有不计其数的计算机专业毕业生，但是真正的程序员却少之又少。大量院校开办计算机专业，而真正酷爱编程的学习者却无处求学。只有少数人通过自学成为真正合格的程序员。业内的很多人其实都知道原因。大学缺乏合格的计算机专业老师，只会照本宣科、没有工程实践的老师永远也教不出合格的程序员。而真正经验丰富、适合做老师的程序员又没有教书育人的机会。现实需要有人能够改变这种不合理的现状。

2011 年经过深思与熟虑，滴水决定暂时放下产品开发，开办逆向培训。我们要做一些有意义的事情。即使我们是大海中的一滴水，也可以荡起一点点的涟漪。知行合一，立刻行动才是我们所需要的！

首先介绍一下滴水逆向培训的主讲老师-唐老师。2000 年毕业于西北一所不知名的大学，贸易经济专业毕业—似乎资深的程序员大多非计算机专业出身。有着 10 年以上的编程经验，取得过计算机软件发明专利，业内公认的杰出程序员。唐老师是滴水硬件调试器和 VT 调试器的开发者，成功逆向虚拟机 VMWare、加壳软件 Themida 和 VMProtect，并即将完成滴水动态变形壳的开发。

接下来说明一下本教程的编写历程。滴水逆向培训基础教程是唐老师自学成才的学习历程以及 10 多年软件开发经验的总结，并经过滴水逆向培训前几期的讲课实践整理编辑而成。以此让更多无法参加实地培训又渴望学习的人一起分享我们的经验。

在此需要说明的是此教程不同于我们以往常见的教材。为了尽可能的保持唐老师教学的原貌，本教程采用语录+章节体的风格编辑整理而成。先引述唐老师的讲课内容，然后再添加学生实际学习中整理、消化吸收的历程。为读者尽可能的创造一种身临其境、置身其中的学习氛围，共同探讨和学习本教程的内容。由于个人天赋的差异，以及文字表达的局限性，每个读者可能都会有不同的理解，甚至有些比较深奥的地方难以理解。这都是正常的，即使

参加实地培训的学员相互之间也存在着巨大的差距。在此只能靠每个读者细心的领悟。同时，由于编写的时间过于仓促，教程本身可能存在诸多错误，希望读者多多批评指正，我们将在后续版本中不断的加以纠正和完善。

最后，我们需要特别强调学习本教程的注意事项：

适合的读者：本教程适合于任何有志于从事编程的读者，包括零基础在内。零基础可能会比较吃力，但有志者事竟成！

教学的目的：帮助零基础的人跨过计算机门槛，为有志于从事计算机编程工作的读者打下坚实的基础。培养独立的自学能力。

学习效果：打下坚实的基础知识，可以独立学习任何自己感兴趣的东西。

学习的前提条件：一是数学要好；数学知识的掌握程度反映一个人的逻辑思维能力，这是学习编程的必备条件，否则会在后续的学习中非常吃力，难以达到预期的学习效果。如果你初中数学优异基本就可以证明具备一定的逻辑思维能力了。二是有兴趣；兴趣是最好的老师。如果这不是你终身希望做的事，学习本教程将是一件非常枯燥无味的事，最终会放弃的。三是肯用功；即使你的智商超过爱因斯坦也不能确保你一定成绩优异，天才+勤奋才是成功的秘诀。补充说明一点，学习编程和英语水平没有必然联系。因为编程涉及的英语单词有限，看的次数多了自然记得。

学习方法：勤学+苦练。练习到手都麻了为止。反复的练习是最佳的捷径，偷懒是你最大的敌人。任何人概莫能外，切记！我们将会在每一章节安排必要的练习，千万不可以偷懒。

教学内容及顺序：几乎所有教材的知识内容大致都是相同的，但是最后学习的效果却有天壤之别，很大一部分原因在于学习的顺序。正确的教学顺序才能让我们在计算机浩如烟海的知识体系中找到正确的方向，循序渐进的消化和吸收才可以真正掌握所学习的内容。大学计算机教育的失败很大一部分原因在于教学内容和顺序的安排错误。表面上所有的内容都已经学过了，但是学完之后全部都忘了，一无所获。过多的学习内容只会让学生望而却步、疲于应付。在没有很好的消化吸收前段学习内容的情况下，继续填鸭式的学习新的内容，割裂了各个知识点之间的联系，最终导致学习失败。所以我们要求，在没有充分掌握前一段知识的情况下，不要继续学习后续的内容。希望能够引以为戒。本教程的教学内容抛弃了知识体系中大量多余的内容，只讲述必要的内容，并进行有效的穿插，帮助读者把每个知识点前后联系起来，并最终牢牢的记住。这是我们所坚持的正确的教学安排，特意为之。学习完本教程，打下一定的基础之后，就可以根据自己的爱好和工作要求自学完成本教程之外暂时没有教授的知识了。所以教程没有涉及过多的应用知识，这是我们有意安排的。学完本教程，打

好基础之后，其他的应用知识可以自学完成。基础决定一切。挖多深的地基才能盖多高的楼。这就是我们为什么从汇编开始的原因。只有从底层的汇编开始学起，才是唯一正确的方法。汇编基础打好之后，其他知识就会学的越来越轻松。

此外，由于时间的限制，我们将会在编辑教程中引用其他优秀教材的部分内容，在此向他们表示诚挚的敬意！

最后再次感谢支持和鞭策滴水的朋友们，感谢滴水员工胡雨和陶捷为编写本教程所付出的辛勤工作。我们将在实际教学中不断完善本教程，对于一些自学能力相对较弱的读者，欢迎在情况允许的情况下报名参加滴水逆向实地培训。如果在学习本教程过程中遇到难以理解的问题也可以登录滴水论坛与大家一起探讨，我们将尽可能的给予解答。滴水官网地址：www.dtdishui.com;论坛地址：www.dtdebug.com；谢谢！

昆山滴水信息技术有限公司

2013 年 1 月 8 日

《滴水逆向培训基础教程》电子版仅供参考，书面教程更为详尽。

目录

第 3 章 C语言

引言

在了解了汇编语言后，我们开始学习 C 语言，虽然两者看上去没什么关系，但是所有编译器都是先将高级语言（包括 C）转换成汇编，再由汇编转换成二进制。而且汇编语言语法简单，可以解释所有 C 语言中的语法。有人可能疑惑：既然汇编语言语法简单，为什么我们还要学习高级语言，直接使用汇编写程序不就可以了？当然可以，前提是你写的程序只有你一个人看，并且不打算运行在其他电脑上。这个前提诠释了汇编语言的诟病：很难阅读并且可移植性差。以往我们学习 C 语言总是和汇编扯不上半毛钱关系，这样理解的人大错特错，接下来我们通过汇编来揭开 C 语言神秘的面纱。

本章必须要掌握的知识点：

1．汇编与 C 的联系

2．函数的格式

3．条件执行语句

4．循环语句

5．表达式

6．数组

7．结构体

8．指针

本章常犯的错误：

1．内存的增长方向

2．一维数组和多位数组的区别

3．结构体定义和申明变量的使用、

4．指针和地址

3.1 C的汇编表示

本节主要内容：

1．观察内存变化

2．汇编与 C 的联系

老唐语录：

1>在我们之前的代码里总是出现 int main(),现在在 main 函数的上方写代码。

```
__declspec(naked) int abc(int a,int b)
{
  push ebx
  push esi
  push edi
  pop edi
  pop esi
  pop ebx
  ret
```

```
}
```

在 int main（）中添加

```
int r;
r=abc(2,3);
printf ("%d\n",r);
```

在 r=abc(2,3);处下断点然后按 F5，再按 F11 单步观察指令运行。

如果看懂每条指令的话，将每条指令改过的内存单元的编号记下来，写在纸上。

比如表 3-1：

表 3-1：内存地址书写格式

内存编号	内容
0x12ff2c	0x00000003
0x12ff28	0x00000002
0x12ff24	0x00401081
……	……

内存编号可以从小到大，也可以从大到小排列，每 4 个字节为一个单位，没有被改过的内存内容不用写，只记下被修改过的内容。

2>现在修改__declspec(naked) int abc(int a,int b)里面的内容如下：

```
push ebx
push esi
push edi
//在此处添加三条语句
mov eax,[esp+0x10]
```

```
mov ecx,[esp+0x14]
add eax,ecx
```

然后单步调试并观察变化。为什么最终打印的结果是 5，并将修改的内存画在纸上。

3>修改__declspec(naked) int abc(int a,int b)大括号里面的内容如下：

```
push ebp
mov ebp,esp
sub esp,4
push ebx
push esi
push edi
mov eax,[ebp+8]
add eax,[ebp+0xc]
mov [ebp-4],eax
pop edi
pop esi
pop ebx
mov eax,[ebp-4]
mov esp,ebp
pop ebp
ret
```

将修改的内存画在纸上。

修改 abc(2,3)为其他整数比如，abc(10,2)，观察结果变化。

问题：从 r=abc(2,3)开始到 printf 指令之前共有几条指令改变了内存单元？

回答：共九个。

4>修改__declspec(naked) int abc(int a,int b)大括号里面的内容如下：

```
push ebp
mov ebp,esp
sub esp,0x44
push ebx
push esi
push edi
mov eax,0xcccccccc
mov ecx,0x11
lea edi ,[esp+0xc]
rep stosd
mov eax,[ebp+8]
add eax,[ebp+0xc]
mov [ebp-4],eax
pop edi
pop esi
pop ebx
mov eax,[ebp-4]
mov esp,ebp
pop ebp
ret
```

将修改的内存画在纸上。

现在在 main()上面写下如下内容：

```
int abc(int a,int b,int c)
/*回车的地方都可以用空格代替，空格的地方可以用回车代替。也就是说，int a,int b,int c 写在一行和写在三行是一回事。*/
{
  int r;//每一句以分号结尾
  r=a+b+c;
  return r;
}
```

打比方，内存相当于算盘，但是算盘太多了，你根本就搞不清楚，给个地址，不好记，所以说，给每一个算盘取一个名字，比如：int a，int b，int c，int r，无论你取什么名字，都会分配给你四个字节的内存，然后在这个算盘上赋值，赋值很简单。我们称 a，b，c 为变量，也可以称为内存单元，赋值格式如下：

a	=	2	;
变量（内存单元）	等于	一个数，也可以是任意表达式	分号

不需要看 C，看汇编如何显示。

圆括号和大括号里面定义的变量都可以赋值和使用，也可以不使用。

课后理解：

当在对一个数进行运算前，需要先规定其宽度，在 C 语言中我们称之为申明。

申明语句格式如下：

数据类型 变量名；

其中数据类型即数据宽度，C 语言定义了以下几种数据类型：

int：32 位

short：16 位

char：8 位

注：C 语言除了十六进制数外，其他语句都区分大小写，这一点区别于汇编。

变量名也有要求：第一个字符必须是字母或下划线。可供使用的有小写字母、大写字母、数字和下划线。

我们现在用实例来说明一个完整的 C 程序：

使用 VC6 打开并创建一个控制台的 HelloWorld 程序，VC6 自动为我们生成代码如下：

```
#include <stdio.h>
int main(void)/*程序入口*/
{
    int num;/*申明一个名为 num 的变量*/
    num = 1;/*赋值语句：num 值为 1*/
    printf("Hello World!\n");/*调用一个库函数*/
    return 0;/*返回*/
}
```

解析：

1〉首先我们要了解的是函数入口，即我们的程序从哪里开始运行：main 函数。注意一个新名词：函数。函数的英文名是 function，它的原意是功能、作用，所以“main 函数”又可以译为“主功能”。

2〉那我们第一条执行的指令是：int num;申明语句，该条语句给了我们三条信息：该变量名字是 num；宽度是 32 位；C 语言的每一条语句后都要以”;”结尾。

3〉第二条语句：num = 1;给该变量赋 1。既然 num 是变量，那么它的值可以再次被

修改，这个留给大家测试。

4〉使用（专业术语是调用）一个 printf 函数，printf 译为打印，所以我们可以说调用一个打印功能。

5〉return 0;返回 0。该功能完成后，返回上一级。类似于我们去一个机关办事，办完事后，要从该机关出来才能去做下一件事。

课后疑问：

__declspec(naked)代表什么？

回答:代表裸函数，不要编译器帮我们构造函数框架，用户自己构造。

课后总结：

申明变量（比如 int a）就是给内存取名。

课后练习：

抄写汇编函数框架。

3.2 函数

本节主要内容：

1. C 语言函数的定义
2. 参数和变量的内存变化

老唐语录：

```
int abc(int e,int f,int c,int a,int b)
/*像前几讲一样 int 必须写，不需要理解他的意思，后面取一个函数的名字，这个名字只要是由字母数字或下划线组成并不以数字打头都可以，长度不受限制，只要不和别人冲突就可以。后面是空格，然后是左圆括号，右圆括号，中间是 int，然后空格（几个空格都可以，一般位 1 个），取个名字，作为标识符，可以取一排变量，用逗号隔开，最后没有分号。*/
//然后大括号
{
  int a;//int 后面是变量名，可以任意取，只要不和上面冲突就可以
  int x;//可以取一排，用分号隔开
  int y;
  a=b+y;
  x=1;
  y=2;
```

```
  x=a+y+e+f+c+a+b;
  return x+y+a;
}
```

如果前面是个 int，后面是个名称、左圆括号和右圆括号的话，我们称之为函数。就是因为这个导致产生一个 CALL。然后在调用的时候可以发现，他会往内存里面写参数，比如你在括号里写一个 3，他会 push 3。在括号中定义多少个，他就会在往内存单元写多少个数，和他对应的是，从右边往左边开始，先给右边的赋值，所以在高地址，因为 push 指令会令会导致 esp-4，所以最右边的是地址是最高的内存单元。比如：

调用 abc(1,2,3)的汇编程序如下：

```
push    3
push    2
push    1
call    abc
```

每个单元 4 个字节。在圆括号里面定义的是 int 变量，每多定义一个，ebp 多减去 4 个字节，比如定义一个 int 变量，sub esp，0x44，定义两个 int 变量，sub esp，0x48。每一个名字（变量）都对应一个 4 字节的内存单元。比如：

int abc(int e,int f,int c,int a,int b)汇编程序如下：

```
push    ebp
mov     ebp,esp
sub     esp,40h
push    ebx
push    esi
push    edi
```

```
lea         edi,[ebp-40h]
mov         ecx,10h
mov         eax,0CCCCCCCCh
rep          stosd
```

注意看内存，里面有很多0xCC，也就是说给一片单元，每四个字节取一个名字。无论是圆括号的里面定义的名字，还是大括号里面定义的名字，我们都把他叫做变量（variable）。给变量赋值是最基础的。无论做什么之前，都要对变量赋值：变量名=表达式；表达式可以是一个数。

变量名	等于号	表达式	分号
a	=	3	;

C语言中的回车符只是为了便于阅读。将所有语句并做一行，代码仍然可以运行，只是不便于阅读。

练习：

将C语言对应的汇编和内存变化写在纸上。

将r=abc(1,2,3);修改为：

```
__asm
{
        sub esp,0xc
        mov dword ptr ds:[esp+0],1
        mov dword ptr ds:[esp+4],2
        mov dword ptr ds:[esp+8],3
        call abc
        add esp,0xc
```

```
        mov r,eax
    }
```

再画出内存变化并运行。

课后理解：

下面简单讲一下函数的架构：

返回值 函数名（传入参数 1，传入参数 2，……）

```
{
    函数主体
}
```

比如：

```
int Add(int a,int b)
{
   int c;
   c = a+b;
   return c;
}
```

说明：这个函数实现了两数相加的功能，返回的是两个数的和。我们可以这样调用它：

```
int result;
result = Add(34,56);
```

这样就实现了 34+56 的和，并把这个结果传给了 result。

课后疑问：

函数的参数可以定义任意多个吗？

回答：理论上可以。一般情况下不要多余四个，这样便于阅读。

课后总结：

函数格式：前面是个 int，后面是个名称、左圆括号和右圆括号。

课后练习：

1．更改参数查看内存变化

2．更改变量查看内存变化

3.3 内存结构

本节主要内容：

1. 变量和参数在内存中的存放格式
2. 函数中嵌套多个函数

老唐语录：

```
int abc(int p1,p2,p3)
{
  int a;
  int b;
  int c;
  a=p1+p2+p3;
  b=p1-p2-p3;
  c=p1-p2+p3;
  return a+b+c;
}
```

写的多了，自然就清楚了，在汇编里写一个 call，或者在 C 语言里，main 函数中写一个函数名，左圆括号，参数，右圆括号，然后 call 指令地址加上本指令的长度就是下一条指令的首地址，放在内存里面去，如图 3-1：

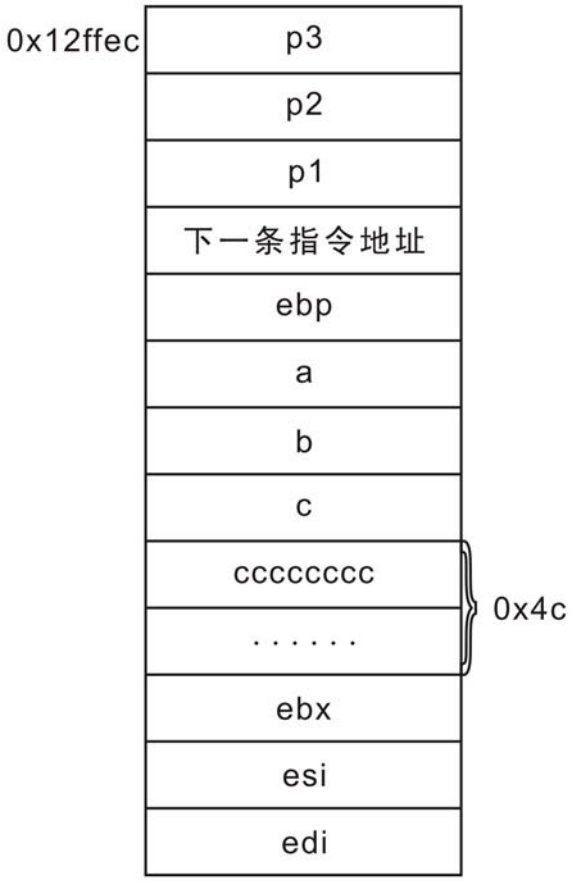

图 3-1：调用函数内存结构

练习：

```
int abc19(int a,int b,int c,int d,int e)
{
  return a+b+c+d+e;
}
……
int abc5(int a,int b,int c,int d,int e)
```

```
{
  int r;
  int v;
  v = abc6(a+5,b+5,c+5,d+5,e+5);
  r = a+b+c+d+e+v;
  return r;
}
int abc4(int a,int b,int c,int d,int e)
{
  int r;
  int v;
  v = abc5(a+5,b+5,c+5,d+5,e+5);
  r = a+b+c+d+e+v;
  return r;
}
int abc3(int a,int b,int c,int d,int e)
{
  int r;
  int v;
  v = abc4(a+5,b+5,c+5,d+5,e+5);
  r = a+b+c+d+e+v;
  return r;
}
```

```
int abc2(int a,int b,int c,int d,int e)
{
  int r;
  int v;
  v = abc3(a+5,b+5,c+5,d+5,e+5);
  r = a+b+c+d+e+v;
  return r;
}
int abc1(int a,int b,int c,int d,int e)
{
  int r;
  int v;
  v = abc2(a+5,b+5,c+5,d+5,e+5);
  r = a+b+c+d+e+v;
  return r;
}
int abc(int a,int b,int c,int d,int e)
{
  int r;
  int v;
  v = abc1(a+5,b+5,c+5,d+5,e+5);
  r = a+b+c+d+e+v;
  return r;
```

```
}
```

main 函数中的调用语句：

```
int r= abc(1,2,3,4,5);
printf(“%d\n”,r);
```

写出内存变化。

课后理解：

函数中嵌套有多个函数时，它的执行顺序是依据函数之间调用的关系由内到外的执行，其中用于运算的寄存器的顺序是 EAX，ECX，EDX，注意：在中间的函数的执行中，整个子程序从 PUSH 开始到最后的 RETURN 语句结束，再将返回值写入 EAX 传递给下一函数使用。函数的执行都是有始有终的。

课后疑问：

本节没有疑问。

课后总结：

函数调用都会引发 CALL 指令的执行。

课后练习：

练习嵌套函数并抄写反汇编语句。

3.4 条件执行

本节主要内容：

1. Jcc、SETcc、CMOVcc 指令
2. if 语句

老唐语录：

我们学习汇编到现在，其实就是学习了 mov 指令，在“寄存器和寄存器”或者“寄存器和内存”之间移动数据，之前学过很多指令影响标志位：ADD，ADC，AND，OR，XOR……

MOV eax,ebx 是无条件将 ebx 移动到 eax 里面去,现在学习有条件移动指令：CMOVcc，后面两个 cc 表示可以替换。比如 CMOVC eax，edx 表示当 CF 位为 1 时，数据从 edx 到 eax。如果 CF 为 0 时，这条指令什么都不做，也就是说在移动数据时，首先要判断标志位；而 CMOVNC 是 CF 为 0 时，数据从 edx 到 eax。如果 CF 位 1 时，不移动。除了 c、nc 之外还有 p、np，z、nz。“CMOVcc eip,”等于“Jcc”，Jcc 也有 16 种写法。所有条件见表 3-2：

表 3-2：Jcc 指令

指令	条件	描述
有符号条件跳转		
JA/JNBE	(CF+ZF) = 0	高于/不低于等于
JAE/JNB	CF = 0	高于或等于/不低于
JB/JNAE	CF = 1	低于/不高于等于

JBE/JNA	(CF+ZF) = 1	低于或等于/不高于
JC	CF = 1	进位
JE/JZ	ZF = 1	等于/零
JNC	CF = 0	不进位
JNE/JNZ	ZF = 0	不等于/非零
JNP/JPO	PF = 0	奇数
JP/JPE	PF = 1	偶数
JCXZ	CX = 0	寄存器 CX 为零
JECXZ	ECX = 0	寄存器 ECX 为零
无符号条件跳转		
JG/JNLE	((SF^OF)+ZF) = 0	大于/不小于等于
JGE/JNL	(SF^OF) = 0	大于或等于/不小于
JL/JNGE	(SF^OF) = 1	小于/不大于等于
JLE/JNG	((SF^OF)+ZF) = 1	小于或等于/不大于
JNO	OF = 0	不溢出
JNS	SF = 0	非负
JO	OF = 1	溢出
JS	SF = 1	负数

SETcc AL 是将条件赋给 AL，比如 SETC AL，将 CF 位赋给 AL，当 CF=1 时，AL=1，当 CF=0 时，AL=0。

练习：

将 CMOVcc 中的每一个条件都试一下。

注：cc 代表十六种条件，十六种写法。

练习：

画出内存变化：

```
int fun10(int a,int b,int c,int d,int e)
{
  int sum;
  int stub;
  if(a>=0x65)
  return 0;
  stub = fun10(a+10,b+10,c+10,d+10,e+10);
  sum = stub+a+b+c+d+e+(a+5+b+5+c+5+d+5+e+5);
  return sum;
}
```

调用语句：`result = fun10(1,2,3,4,5);`

课后理解：

cmovc edx,eax ：如果 cf=1，等同于 mov edx,eax 如果 cf=0,不处理

cmovnc edx,eax ：如果 cf=0，等同于 mov edx,eax 如果 cf=1,不处理

setcc oprd：oprd 可以是 8bit 寄存器、内存，不可为立即数

setz al:如果 zf=1,置 al 为 1， 如果 zf=0,不处理

setnz al:如果 zf=0,置 al 为 1， 如果 zf=1,不处理

课后疑问：

为什么 jcc [eax]不执行？

回答：vc 编译器不识别 jcc [寄存器]指令，但 test 编译器识别。

课后总结：

条件执行指令是根据 EFLAG 寄存器中相应的标志位为判断条件。

课后练习：

写程序练习条件执行语句。

3.5 移位指令

本节主要内容：

1．算数移位与逻辑移位指令

2．循环移位指令

老唐语录：

计算机的运算指令除了 ADD、SUB、ADC、SBB、OR、XOR、AND 之外，总会有其他指令，比如说操作数的取反指令 NOT。还有 CMP 指令，其实和 SUB（减指令）是一样的，只是不改变目标操作数，比如 SUB eax，edx 是 eax 的值减去 edx，结果放到 eax 里面去，影响标志位，而 cmp 只影响标志位，运算结果忽略，不影响 eax。

我们今天来学习移位指令。

逻辑移位：

SHR AL，2 向右面移两个位，如果 AL 值为 00101010，则向右移动两位后变成 00001010，简单的理解是，先移一位变成 00010101，再移一位变成 00001010，可移动的最大值为 0x1F，如果不论输入的值为多少，结果都要与 0x1F。也就是说只有低五位有效，如图 3-2：

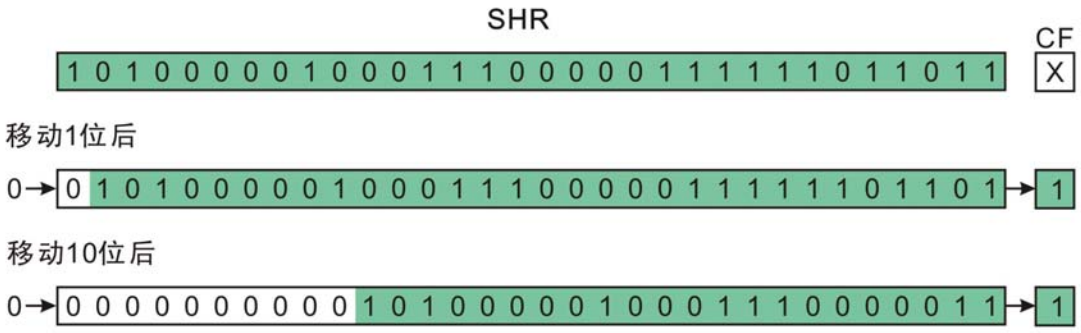

图 3-2：SHR

向左移也是一样：SHL ，只要会移动一位，就可以了，移动后的结果放在目标寄存器里面，当然该操作也会改变标志寄存器中的 CF，将移出来的位赋给 CF；右移也是一样，凡是移出来的那个位都赋给 CF，如图 3-3：

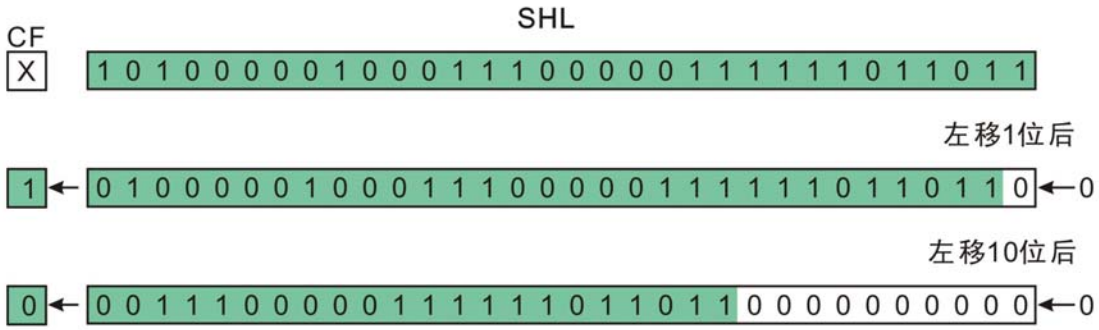

图 3-3：SHL

疑问：哪一个位赋给 CF

移动 N 位，相当于 N 次移动一位，也就是说，每移动一位，就将移出来的那一位赋给 CF，移动了 N 位，就赋给 CF 位 N 次。所以结果很明显，就是最后移动的那一位。

当然移位指令照样影响 SF 位，将移动后剩下的最高位赋给 SF。还要看结果是否全为零，如果全为零，ZF 置 1，否则置 0。也影响 PF 位，1 的奇偶性。不影响 OF 位。

移位指令的源操作数只能是 8 位的寄存器 CL 或立即数（ECX 中的 C 为 counter，计数）；目标操作数可以是字节/双字节/四字节，内存或寄存器。

算术移位：

算术移位和逻辑移位的区别：算术左移和逻辑左移没有区别；在右移时，高位移至低位，如果最高位补 0，是逻辑移位；最高位保持不变（符号），是算术移位，如图 3-4：

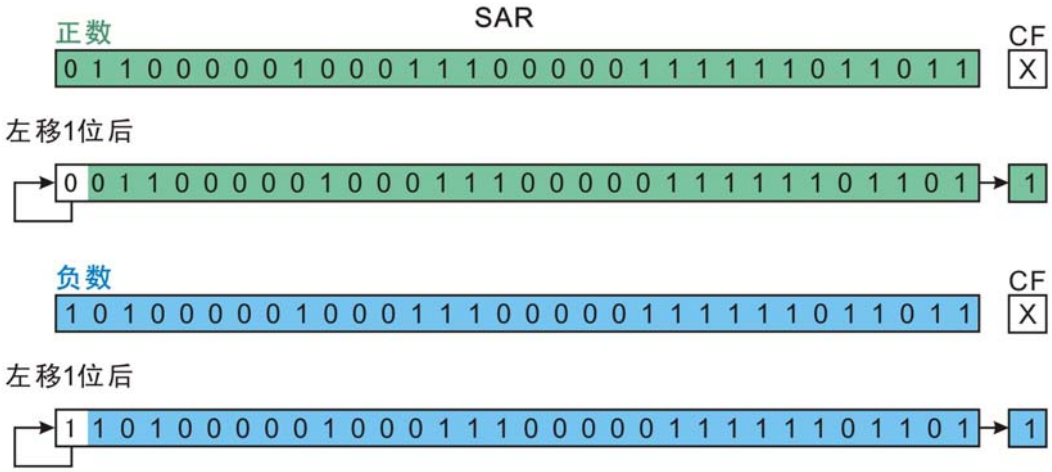

图 3-4：SAR

SHR：逻辑右移，SH 是 shift，R 代表 right，L 代表 left。

SAR：算术右移，A 是算术 arithmetic。

除了上述四条之外，还有循环移位：

循环右移：循环右移一位，31 位赋给 30 位，30 位赋给 29 位……最低位同时赋给最高位和 CF 位，导致最高位和 CF 位相同。当然也影响 SF，ZF，PF。如果移动五位看不懂，只需要移动一位，然后移动 5 次即可。

RCL 和 RCR：带进位的循环移位（C 代表 CF 位）：循环左移一位，将最高位给 CF，CF 值赋给最低位，最低位赋给次低位。

课后理解：

ROL/ROR 如图 3-5：

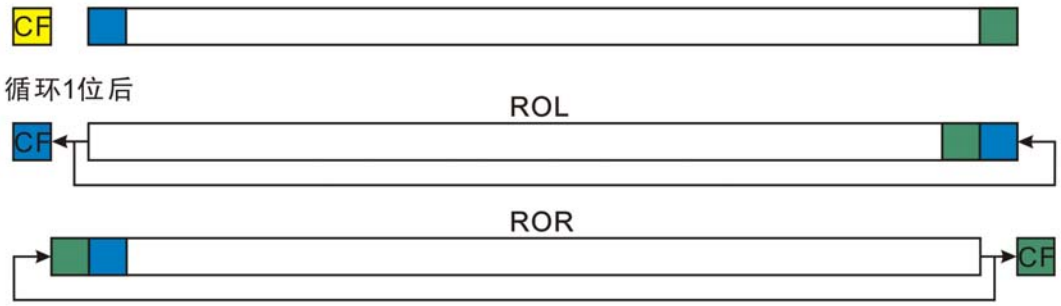

图 3-5：ROL/ROR

RCL/RCR 如图 3-6：

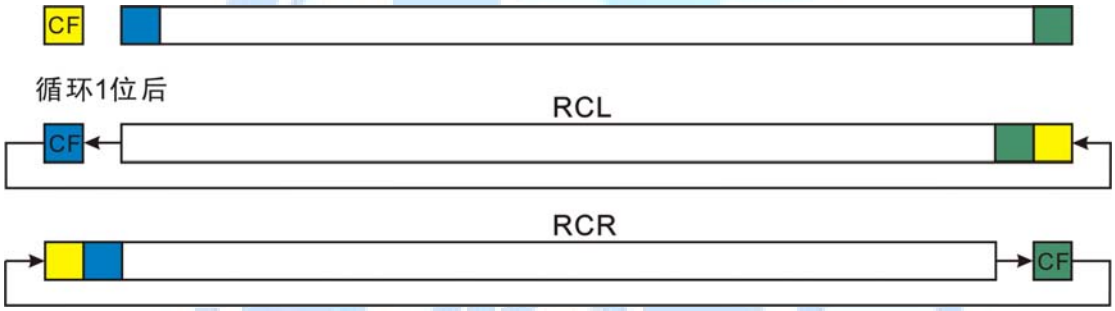

图 3-6：RCL/RCR

课后疑问：

C 语言中移位指令的书写格式？

左移<<，右移>>。

左移只会生成 SHL 指令，因为 SHL 和 SAL 等效

无符号右移（>>前的数）会生成 SHR，有符号数右移会生成 SAR

课后总结：

算术移位涉及符号位的运算，而逻辑移位不涉及。

课后练习：

画出下列指令执行时内存数据的变化情况：

```
int CumulativeHundred1(int a,int b,int c,int d,int e, int f,int g,int h,int i,int j);
int CumulativeHundred7(int a,int b,int c,int d,int e, int f,int g,int h,int i,int j)
{
 int sum;
 int stub;
 stub = (a>=90) ? 0:CumulativeHundred1 (a+20,b+20, c+20,d+20, e+20, f+20, g+20,h+20,i+20,j+20);
 sum = stub +g+h+i+j + ((a<<1)+10+(b<<1)+10+(c<<1) +10+(d<<1) +10+(e<<1) +10+(f<<1)+10+g+10+h+10+i+10+j+10);
 return sum;
}
int CumulativeHundred6(int a,int b,int c,int d,int e,int f,int g,int h,int i,int j)
{
 int sum;
```

```
    int stub;
    stub = (a>=90) ? 0:CumulativeHundred7(a+20,b+20, c+20,d+20, e+20,
f+20, g+20,h+20,i+20,j+20);
    sum = stub +f+g+h+i+j + ((a<<1)+10+(b<<1)+10+ (c<<1)+10+ (d<<1)
+10 +(e<<1)+10+f+10+g+10+h+10+i+10+j+10);
    return sum;
   }
   int CumulativeHundred5(int a,int b,int c,int d,int e, int f,int
g,int h,int i,int j)
   {
    int sum;
    int stub;
    stub = (a>=90) ? 0:CumulativeHundred6(a+20,b+20, c+20, d+20, e+20,
f+20, g+20,h+20,i+20,j+20);
    sum = stub +e+f+g+h+i+j + ((a<<1)+10+(b<<1) +10+(c<<1)+ 10+(d<<1)
+10 +e+10+f+10+g+10+h+10+i+10+j+10);
    return sum;
   }
   int CumulativeHundred4(int a,int b,int c,int d,int e, int f,int
g,int h,int i,int j)
   {
    int sum;
    int stub;
```

```
    stub = (a>=90) ? 0:CumulativeHundred5(a+20,b+20,c+20, d+20,e+20,
f+20, g+20,h+20,i+20,j+20);
    sum = stub +d+e+f+g+h+i+j + ((a<<1)+10+(b<<1)+10+ (c<<1)+10+d+
10+ e+ 10+f+10+g+10+h+10+i+10+j+10);
    return sum;
   }
   int CumulativeHundred3(int a,int b,int c,int d,int e, int f,int
g,int h,int i,int j)
   {
    int sum;
    int stub;
    stub = (a>=90) ? 0: CumulativeHundred4 (a+20,b+20, c+20,d+20, e+20,
f+20, g+20,h+20,i+20,j+20);
    sum = stub +c+d+e+f+g+h+i+j + ((a<<1) +10+ (b<<1) +10+c +10+d +10+e
+ 10 +f +10+g+10+h+10+i+10+j+10);
    return sum;
   }
   int CumulativeHundred2(int a,int b,int c,int d,int e, int f, int
g,int h,int i,int j)
   {
    int sum;
    int stub;
    stub = (a>=90) ? 0: CumulativeHundred3 (a+20, b+20, c+20, d+20,
```

```
e+20, f+20,g+20,h+20,i+20,j+20);
    sum = stub+b+c+d+e+f+g+h+i+j + ((a<<1) +10 +b +10 +c +10 +d +10
+e +10+f+10+g+10+h+10+i+10+j+10);
    return sum;
   }
   int CumulativeHundred1(int a,int b,int c,int d,int e, int f,int
g,int h,int i,int j)
   {
    int sum;
    int stub;
    stub = (a>=90) ? 0:CumulativeHundred2(a+20,b+20, c+20, d+20, e+20,
f+20, g+20,h+20,i+20,j+20);
    sum = stub +a+b+c+d+e+f+g+h+i+j + (a+10+b+10 +c+10+d +10+e+10
+f+10 +g+10+h+10+i+10+j+10);
    return sum;
   }
```

3.6 表达式

本节主要内容：

1．表达式的格式

2．表达式的使用

老唐语录：

有些指令只操作两个寄存器，有些指令会操作几十甚至上百个寄存器。

cmp 和 SUB 指令功能是一样的，只是 cmp 不改变目标操作数的值，同样影响标志位。AND 的功能是按位与。TEST 与 AND 的功能相同，只是不改变目标操作数的值，也同样影响标志位。有些指令不能写立即数，有些指令可以全是内存，只要在书写时可编译通过，表示语法正确。

以后我们不需要写汇编，只需要看懂别人写的汇编代码即可。前面提到，定义一个 C 语言函数格式：int + 函数名 +（+定义的变量+……），可以称 C 语言是汇编语言的高级书写形式。

表达式的定义：**凡是由变量、常量和算术符号链接起来的都可以称为表达式。**

在 C 语言中，XOR 表示为“^”，OR 表示为“|”，AND 表示为“&”。数学里面的表达式有圆括号，中括号和大括号，在 C 语言中只有圆括号。

比如：

```
int _abcd(int ab,int cd,int name)
{
```

```
  int ba;//变量
  int dc;
  ba = ab+cd;//表达式
  return ab;//可以是一个值，也可以写表达式
}
```

练习：

用各种算术符号写表达式，越复杂越好，编译成功后，观察汇编。

如果看不到逻辑移位，只能看到算术移位，只需将 int 替换成 unsigned int。

注：int 是有符号整型，unsigned int 是无符号整型。

在表达式中，任意一个成员又可以表示成表达式，所以可以像嵌套一样无限膨胀。

练习：

```
int abc(int a,int b,int c)
{
  int var;
  int h;
  int j;
  var = 0;
  var = (var,0x5,var+0x5);//var = var+0x5;
  return 0;
}
```

“=”左边只能是个变量，不可以是常量或表达式。

逗号和等号也是算术运算符。

单目/双目/三目运算符：b=b+c 等价于 b+=c。

通过加圆括号和不加圆括号的运算来判断运算符的优先级。

练习：

乘法指令分为有符号乘法和无符号乘法：

计算机做有符号乘法（IMUL）时，首先判断最高位是否为 1，为 1（负数）时，将 1 提出来，变成 0 减去提出 1 后的数（此时为正数），乘法运算完成后，再用 0 减去结果。

当做乘法时，目标操作数默认为 EAX 或 EDX：通常乘法会使宽度扩大一倍，8 位的乘法需要两个字节保存，32 位的乘法需要 64 位保存。 1 个字节相乘时，结果放在 AX 里面，两个字节相乘时，结果放在 EAX 里面，4 个字节乘法时，结果放在 EAX 和 EDX 里面，其中 EAX 为高位。比如 MUL ecx；将 ecx 乘以 eax 并将结果放在 EAX 和 EDX 中。

除法里面，结果放在 EAX 里面，余数放在 EDX 里面。

```
V = ab%5+ba/7+ba*cd; // “%”：求余，“/”：除法，“*”：乘法
```

如果目标操作数不设定为 EAX，比如 IMUL ECX，EAX，则结果会溢出，没有意义。

IMUL 后面也可以含有 3 个操作数，比如 IMUL ECX，EAX，0x3；则第三个操作数必须是立即数。

比如：-5*-6 = -1*(0-(-5)) * -1*(0-(-6))。

练习：

写出下列 C 程序对应的汇编代码

```
int abc(int a,int b,int c)
{
  int v;
  int r;
  v = a+b+c;
  r = a|b&c;
```

```
  return v+r;
}
int ab(int a,int b,int c,int d)
{
  int v;
  v = abc(a+b,b+c,c+a)+3;
  return v+d+5;
}
```

调用函数的时候，函数的参数也可以是表达式。

观察函数的规律：

第一条指令都是：push ebp，然后是

```
push       ebp
mov        ebp,esp
sub        esp,40h
push       ebx
push       esi
push       edi
lea        edi,[ebp-40h]//没有定义变量为0x40
mov        ecx,10h
mov        eax,0CCCCCCCCh
rep stos    dword ptr [edi]
```

如果函数内部定义了变量，则[ebp-4]为第一个变量，[ebp-8]为第二个变量，[ebp+4]为函数返回后执行下一条指令的地址。

函数末尾结构如下：

```
pop        edi
pop        esi
pop        ebx
mov        esp,ebp
pop        ebp
ret
```

练习：

写一个函数，实现只有表达式由编译器生成汇编，其他部分由自己构造（汇编实现）。

```
v=a+b+c;r|=a&b|c;
```

说明：在写裸函数前需要在函数前加__declspec(naked)：

```
__declspec(naked) abc(int a,int b,int c)
{
  int v;int r;
  __asm
  {
    push ebp;
    mov ebp,esp
    sub esp,__LOCAL_SIZE(编译器的宏)//可以写成 0x48
  }
  v=a+b+c;
  r|=a&b|c;
  __asm
```

```
{
mov eax,r
mov esp,ebp
pop ebp
ret
}
}
```

高级语言的唯一方便之处就是表达式可以按数学思维书写，其他部分都只是固定的格式。

课后理解：

表达式可含有：

1.赋值符号： =

2.运算符号： +、-、*、/

3.不等号符号：==、>=、<=、!=、

课后疑问：

数字 3 是表达式吗？

回答：是的，属于常量表达式。

课后总结：

凡是由变量、常量和算术符号链接起来的都可以称为表达式。

课后练习：

找出下列对错

```
int #33;
int a;
int 2a;
int i;
int 3_Alafun()
{
  int v;
  int a;
  int b>a;
  v += i++++;
  v += ++++i;
  return 0;
}
```

3.7 if语句

本节主要内容：

1. if 语句的书写格式
2. if else 语句

老唐语录：

在函数调用时，用到的内存称为栈。无论在调用函数时，还是在函数内部，push 指令都要对应相应的 pop 指令，或者 esp 值最终都要被恢复到调用函数之前的值，我们称之为栈平衡。

函数框架：

```
int abc(int ab,int cd,int ef)//ebp+0:原 ebp 值；ebp+8：ab；ebp+0xc：cd；ebp+0x10：ef
{
  int v;//ebp-4
  int h;//ebp-8
  int r;
  ……
  return 0;//把表达式的值赋给 eax
}
```

C 语言中最常用的语句是 if 语句。

格式：

```
if ()//圆括号里面可以写任意表达式
{
//可以写任意多个表达式
}
```

既然 if 语句中间可以放任意语句，if 语句也属于语句，那么 if 语句中间也可以放 if 语句：

```
if ()
{
  if ()
  {
  }
}
```

if 语句的第二种格式：

```
if ()
{
}
else
{
  //也可以是任意语句
}
```

if 语句的第三种格式：

```
if ()
```

```
{
  //可以是任意语句
}
else if()
{
  //可以是任意语句
}
else if()
{
  //可以是任意语句
}

else if()
{
  //可以是任意语句
}
……//中间可以有任意多个 else if 语句
else //可以省略
{
  //可以是任意语句
}
```

其中圆括号里面都可以放任意表达式，且每个表达式之间没有任何关系。将 else if 格式省略掉，变成第二种格式；将 else if 格式和 else 格式都省略掉，变成第一种格式。

练习：

练习 if 语句的三种格式，并在圆括号和大括号里面写任意表达式，并用汇编查看。

说明：逗号表达式也属于表达式，比如 if(表达式 1，表达式 2，……)

Jcc lab10 指令等价于 CMOVcc eip，lab10//lab10 是标签

其实 C 语言中也含有功能等价于 jmp 的指令 goto：

goto lab10；等价于 jmp offset lab10

对于 if 语句，当圆括号里面的表达式值为非 0 时，执行大括号里面的程序。

课后理解：

```
if(表达式)
{
表达式;
}else if(表达式)
{
表达式;
}else
{
表达式;
}
```

其中表达式可以为空，数值，或判断语句。

课后疑问：

可以添加两个 else 语句，比如：

```
if ()
else
{
}
else
{
}
```

回答：不可以。因为 else 已经代表结束（包含了除上述情况外的所有情况），不需要再次结束。

课后总结：

if 语句又称选择语句，用于在可选择的几个动作之间进行选择。

课后练习：

写出下列函数中 FuncMove 函数每次执行完后变化内存的值

```
int FuncMove(int n,int nX,int nY)
{
 int nResult;
 int nTemp1;
 int nTemp2;
 int nTemp3;
 nResult = n;
 nTemp1 = 0;
```

```
 nTemp2 = 0;
 nTemp3 = 0;
 nTemp1 |= nY;
 nTemp2 = 0x3E2D2D00;
 nTemp2 |= nX;
 return nResult ;
}
int FuncHanoi(int n,int One,int Two,int Three)
{
 int nResult;
 nResult = 1;
 if (n==1)
 {
  FuncMove(n,One,Three);
 }
 else
 {
  FuncHanoi(n-1,One,Three,Two);
  FuncMove(n,One,Three);
  FuncHanoi(n-1,Two,One,Three);
 }
 return nResult;
}
```

3.8 循环语句

本节主要内容：

1. while 循环
2. for 循环

老唐语录：

在申明一个变量的时候没必要马上赋值。可以在使用的时候再进行初始化。

练习：

```
int v;
int i;
i=1;
v=0;
v+=i++++;//错误
(v=v+1)++;
(v=v+1)=(v=v+1)+1;
v+=+++++i;//编译器把落单的加号看作正号
v+=+++++++++i;//等价于 v=v+(++(++(++(++i))));
++i=i+1;//看汇编代码可知++优先级高于等号
```

总结：

```
5+i++//先进行 5+i，再运算 i+1
```

4+++i//先运算 i+1，再进行 4+i

程序往回转，可以通过 goto 语句实现。比如 goto lab（标签）；因为启用标号比较麻烦，所以 C 语言提供了新的语句 while。

格式：

while()/*圆括号里面可以是任意表达式，表达式的值成立，执行大括号里面的语句，执行完后，再次判断圆括号里面的表达式值是否成立，若成立，继续执行大括号里面的语句……直至判断圆括号里面的表达式值不成立，跳出循环*/

{

//可以是任意语句

}

练习：

练习 while 语句，并调试观察对应的汇编语句

当 while 语句中圆括号中值为 1 时，会永远执行，while 语句里面也可以嵌套 while 语句。

while 语句有两种形式，另外一种是 do while 语句：先执行，再判断，如果条件成立，继续执行：

do

{

}while();//圆括号里面不能为空

for 语句也是循环语句的一种，for 圆括号里面有三个表达式：

for (表达式 1;表达式 2;表达式 3)/*表达式可以不写，但是分号不能省略，表达式不写，则表示为 1 */

{

```
    //可以是任意语句
}
```

注：for 后面圆括号里面的表达式通常会改变某些变量的值，否则不执行或永远执行。表达式 1 只在开始时执行一次，表达式 2 和表达式 3 每次都执行。执行次序是：

1>表达式 1；

2>表达式 2；//表达式值成立

3>大括号里面内容；

4>表达式 3；

5>表达式 2；//表达式值成立

6>大括号里面内容；

7>表达式 3；

……

N>表达式 2；//表达式值不成立

N+1>跳出循环

练习：

构造 for 循环语句，观察其生成的汇编代码，并单步观察执行顺序

break 也是一条语句，通常放在 if，while 和 for 语句中间，作为中断使用，该功能可以由 goto 语句代替（需要设定标签）。如若 break 语句处于嵌套循环语句中，则只会跳出当前循环。

课后理解：

For 语句可以用 if 和 goto 语句实现，实现步骤如下：

1>for (i=begin,sum=0; end-i+1;i=i+1)

```
   { ... ... }
2>i=begin,sum=0;
   for ( ; end-i+1;i=i+1)
   { ... ... }
3>i=begin,sum=0;
   for ( ; end-i+1; )
   {
      ... ...
     i=i+1;
   }
4>i=begin,sum=0;
   lab:i=i+1;
   if ( end-i+1 )
   {
      ... ...
  }else
   {
  ... ...
     goto lab;
   }
```

以上步骤可以说明，c 语言中所有句型都可以用 if 和 goto 实现。

for 语句格式来源：

在 for 循环中加入 continue 语句，当 continue 执行后，程序跳转到下一循环

for(...; ...;...)的第二个";"处 （从括号中的第三条语句开始执行）。

课后疑问：

在嵌套循环如：

```
for(;;)
{
  for(;;)
  {
    break;
  }
  lab1:……
}
lab2:……
```

如果执行 break 语句，则跳转的位置是 lab1 还是 lab2？

回答：lab2，break 语句只跳出当前循环。

课后总结：

c 语言中所有句型都可以用 if 和 goto 实现。

课后练习：

画出下列指令执行时内存数据的变化情况

```
int fun ( int a , int b)
```

```
{
  int v;
  int i;
  int j;
  i=a;
  v=0;
  while (i-b-1)
  {
    v+=i++;
  }
  return v;
}
int main(int argc, char* argv[])
{
  int result;
  result = fun(1,100);  // 从这里开始画栈
  printf("%d \n",result);
  return 0;
}
```