汇 编 语 言 程 序 设 计 实 验 报 告

目录

汇 编 语 言 程 序 设 计 实 验 报 告

1 实验目的与要求

(1) 了解程序计时的方法以及运行环境对程序执行情况的影响。

(2) 熟悉汇编语言指令的特点，掌握代码优化的基本方法。

2 实验内容

任务 1. 观察多重循环对 CPU 计算能力消耗的影响

应用场景介绍：以实验一任务四的背景为基础，只要顾客买走了网店中的一件商品，老板就需要重新获得全部商品的平均利润率。现假设在双十一零点时，SHOP1 网店中的“Bag”商品共有 m 件，有 m 个顾客几乎同时下单购买了该商品。请模拟后台处理上述信息的过程并观察执行的时间。

上述场景的后台处理过程，可以理解为在同一台电脑上有 m 个请求一起排队使用实验一任务四的程序。为了观察从第 1 个顾客开始进入购买至第 m 个顾客购买完毕之间到底花费了多少时间，我们让实验一任务四的功能三调整后的代码重复执行 m 次，通过计算这 m 次循环执行前和执行后的时间差，来感受其影响。功能三之外的其他功能不纳入到这 m 次循环体内（但可以保留不变）。

调整后的功能三的描述：

（1）在 SHOP1 中找到“Bag”商品，判断已售数量是否大于等于进货总数，若是，则回到功能一（1），否则将已售数量加 1。

（2）刷新全部商品的平均利润率。首先计算 SHOP1 中第一个商品的利润率 PR1，然后在 SHOP2 网店中寻找到该商品，也计算其利润率 PR2。最后求出该商品的平均利润率 APR=(PR1+PR2)/2，并保存到 SHOP1 的利润率字段中。重复上述步骤，依次将每个商品的平均利润率计算出来。

请按照上述设想修改实验一任务四的程序，并将 m 和 n 值尽量取大（比如大于 1000，具体数值依据实验效果来改变，逐步增加到比较明显的程度，比如秒级的时间间隔），以得到较明显的效果。

任务 2. 对任务 1 中的汇编源程序进行优化

优化工作包括代码长度的优化和执行效率的优化，本次优化的重点是执行效率的优化。请通过优化 m 次循环体内的程序，使程序的执行时间尽可能减少 10%以上。减少的越多，评价越高！

3 实验过程

3.1 任务 1

3.1.1 设计思想及存储单元分配

本次实验基于实验一任务五之上完成。设计思想和存储单元分配与之相同。由于本次实验的要

求，将之前的功能三改进了，根据要求每次都要将所有商品的利润率刷新（这部分属于嵌套循环），将改进后的代码重复执行 m（m≥1000）次。

3.1.2 流程图

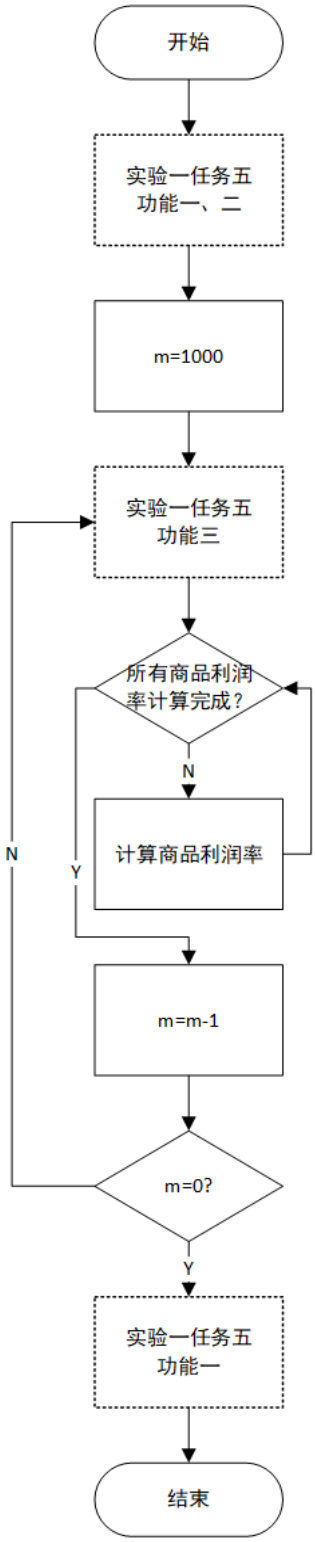

图 3.1.1 改进后功能 3 流程图

3.1.3 源程序

由于本次实验只对功能三有改进，所以在这里贴出任务 1 中要求的改进后的功能三的代码：

```
f3:     lea dx,tip4
        mov ah,9
        int 21h
        lea dx,in_itemname                          ;display tip4
        mov ah,0ah
        int 21h
        mov bl,in_itemname+1                        ;get length of string
        mov bh,0
        mov byte ptr in_itemname+2[bx],0
        mov bx,offset in_itemname+2                 ;to see whether the string is 0dh or not
        mov bx,[bx]
        mov bh,0
        mov ax,0dh
        cmp bx,ax
        je  f1

        mov ax,0
        call timer
f3_6:
        mov ax,offset in_itemname+2                 ;get the input of string
        mov bx,offset ga1
        mov cx,30 ;loop 30 times
cmpitem:
        push cx
        push bx
        mov cx,5
cmpitemname:
        push bx
        mov bx,ax
        mov dx,[bx]
        pop bx
        mov di,[bx]
        cmp dx,di ;compare by word
        jne incorrectitem
        add ax,2
        add bx,2
        loop cmpitemname
        mov byte ptr itemfind,1 ;put the flag about itemfind to 1
        jmp short ok3
incorrectitem:
        pop bx
        add bx,20 ;next
        pop cx
        loop cmpitem
ok3:    mov bx,offset itemfind ;get the flag
        mov bx,[bx]
        mov bh,0
        cmp bx,0
        je f3_5
        mov bx,offset auth
```

```
        mov bx,[bx]
        mov bh,0
        cmp bx,0
        je f3_4

        pop si ;the position had been found

        mov ax,14[si] ;compare the sold and total
        cmp 16[si],ax
        jnb exit33

        mov ax,16[si] ;increase the sold
        inc ax
        mov 16[si],ax

        mov si,offset ga1
        mov cx,30
f3_3:   push cx
        push si
        mov al,12[si]
        mov bl,16[si]
        imul bl
        push ax ;in order to save data
        mov al,10[si]
        mov bl,14[si]
        imul bl ;prime price
        pop di
        push ax
        sub di,ax
        mov ax,di
        mov bx,100
        imul bx
        pop di
        idiv di                 ;put profit to ax
        mov word ptr apr1,ax

        mov ax,si
        mov bx,offset gb1
        mov cx,30
cmpitem1:
        push cx
        push bx
        mov cx,5
cmpitemname1:
        push bx
        mov bx,ax
        mov dx,[bx]
        pop bx
        mov di,[bx]
        cmp dx,di ;compare by word
        jne incorrectitem1
        add ax,2
        add bx,2
        loop cmpitemname1
        jmp short ok33
incorrectitem1:
        pop bx
```

```
        add bx,20 ;next
        pop cx
        loop cmpitem1


ok33:   pop si ;item had been found in shop2
        mov al,12[si]
        mov bl,16[si]
        imul bl ;total income
        push ax
        mov al,10[si]
        mov bl,14[si]
        imul bl ;prime price
        pop di
        push ax
        sub di,ax
        mov ax,di
        mov bx,100
        imul bx
        pop di
        idiv di ;put profit to ax
        mov bx,offset apr1
        mov bx,[bx]
        add ax,bx
        cwd
        mov bx,2
        idiv bx ;get average
        mov ah,0
        pop si
        pop si
        mov word ptr apr,ax
        mov 18[si],ax
        mov ax,word ptr apr
        add si,20
        pop cx
        dec cx
        jnz f3_3

        jmp f3_6
        jmp short f4
exit33: mov ax,1
        call timer
        jmp f1
```

3.1.4 实验步骤

1.准备上机实验环境，编辑，汇编，连接文件 2code_1。

2.输入老板姓名，密码，进入功能三，等待执行完毕。

3.观察并计算这 m 次循环执行前和执行后的时间差，感受多重循环对 CPU 计算能力消耗的影响。

3.1.5 实验记录与分析

1. 实验环境条件：P3 1GHz，256M 内存；WINDOWS 10 下 DOSBox0.73；EDIT.EXE 2.0；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。

2. 汇编源程序时，汇编程序没有报错，进入连接。

3. 连接过程没有发生异常。

4. 程序中的逻辑错误如下：（由于当时没有截图，现只在下面说明情况）

（1）一开始调试程序时，发现当 m 设置的值过大时，程序会陷入死循环，经过单步调试发现是由除法溢出导致的，当循环一定次数后，会出现除数为 0 的情况，此时，从 TD 单步调试的情况来看，程序会在除法指令后执行两步程序后再次跳到当初的除法指令那里去，导致死循环。

5. 观察多重循环对 CPU 计算能力消耗的影响

设置 m 值为 1000，设置 n 值为 30，测试结果如下图：（间隔达到秒级）

图 3.1.2 多重循环程序所需要的时间

6. 思考题（在任务 2 中一起写）

3.2 任务 2

3.2.1 设计思想及存储单元分配

任务 2 要求将任务 1 的代码进行优化，优化工作包括代码长度的优化和执行效率的优化，在达到要求的基础上尽可能多地减少运行时间。

本次优化主要是寻址方式的优化，减少了寄存器的使用，减少汇编指令提高运行速度，同时将乘除指令尽可能改成移位指令。

3.2.2 流程图

由于本任务是基于任务 1 的代码基础上进行优化，在流程图上并没有区别，所以本流程图与任务 1 相同。

3.2.3 源程序

对于任务 2 的改进代码，提高效率，在这里贴出改进的地方：

```
…
f3_6: mov ax,offset in_itemname+2（此处去掉）
…
    mov cx,5 ;此之前为任务 1 功能 3 的代码
```

```
    mov di,offset in_itemname+2
cmpitemname:
    mov dx,[di]
    cmp dx,[bx] ;此之后为任务 1 功能 3 的代码

…
    ok3:mov bl,byte ptr itemfind  ;此之前为任务 1 功能 3 的代码
    cmp bl,0
    je f3_5
    mov bl,byte ptr auth
    cmp bl,0
    je f3_4
    pop si  ;此之后为任务 1 功能 3 的代码

…
cmpitemname1: ;此之前为任务 1 功能 3 的代码
    mov dx,[di]
    cmp dx,[bx]
    jne incorrectitem1 ;此之后为任务 1 功能 3 的代码

…
    idiv di ;此之前为任务 1 功能 3 的代码
    mov bx,word ptr apr1
    add ax,bx
    sar ax,1
    pop si
    pop si
    mov 18[si],ax
    add si,20
    pop cx
    dec cx
    jnz f3_3 ;此之后为任务 1 功能 3 的代码
…
```

3.2.4 实验步骤

1.优化代码，提高运行效率。

2.准备上机实验环境，编辑，汇编，连接文件 2code_2。

3.输入老板姓名，密码，进入功能三，等待执行完毕。

4.再次观察并计算这 m 次循环执行前和执行后的时间差，比较代码优化前后的时间差。

5.计算提高效率。

6.完成思考题。

3.2.5 实验记录与分析

1. 实验环境条件：P3 1GHz，256M 内存；WINDOWS 10 下 DOSBox0.73；EDIT.EXE 2.0；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。

2. 汇编源程序时，汇编程序没有报错，进入连接。

3. 连接过程没有发生异常。

4. 程序没有逻辑错误，正常运行。

5. 观察代码优化后的时间差

设置 m 值为 1000，设置 n 值为 30，测试结果如下图：

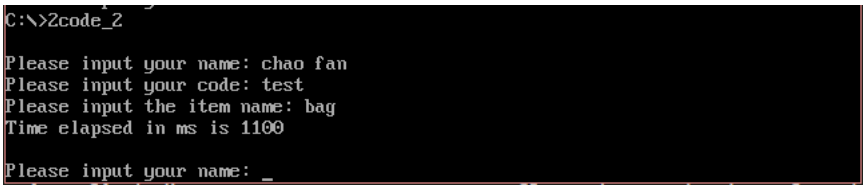

图 3.2.1 代码优化后多重循环程序所需要的时间

代码优化前的时间差如下图：

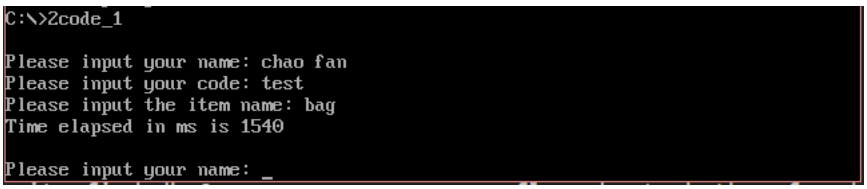

图 3.2.2 代码优化前多重循环程序所需要的时间

6.计算优化效率

优化前程序执行时间差为 1540ms，优化后程序执行时间差为 1100ms，优化了 440ms，对比优化前的时间差，优化效率为 28.6%。

7.思考题

对于本次实验思考题的回答如下：

（1）如何有效展现时间优化的效果？有多少种方法获取程序执行的时间？

在展现时间优化效果时，可以显示执行前的时间，执行结束后再显示一遍时间，这两个时间的差值就是时间差。另一种方法是直接计算时间差。这两种在汇编教学网站上有详细例子。

（2）改变循环程序的结构和循环次数，对 CPU 资源消耗的影响有多大？

改变循环程序的结构，循环次数，对 cpu 资源消耗影响是很大的。比如有一个嵌套循环，你将其改成了两个单个循环，这种优化就大大提高了效率，从几何型增长变成了线性增长。

（3）观察不同环境下执行同一个程序的效率的差异。

还有在本次实验中，不同环境下执行同一程序，需要的时间可能会有细微的差别，这个可以经过多次执行程序来发现。

（4）汇编语言程序的优化可以从哪些方面进行？

程序的优化将在体会中提到，在此不做赘述。

（5）哪些指令是需要优化的关键性指令？

在优化的工作中，我发现了有些优化工作即使减少了代码长度，也不会对程序优化产生什么明显的效果，这可能是不同指令执行需要的时间不同，即使你减少了指令的数量，但是指令执行的时间可能增加了。比如利用 byte ptr 寻址时，即使减少了代码量，时间也似乎还是相同的。而将乘除指令转化为移位指令，减少寄存器的使用数量，寻址方式的正确使用，改变比较字符串的思路，这些优化措施对程序优化的影响程度是比较大的，一般程序优化工作也从这些方面着手。

（6）优化后的程序执行效率是否比优化前有明显提高？

有明显提高，具体效率见实验记录。

（7）总结不同类别的优化措施对效率的影响程度。

在体会以及思考题（5）中有描述，在此不赘述。

4 总结与体会

在本次实验任务 1 中，我了解到在计算量不大的情况下，计算机是可以很快得到执行结果的。但是在大量数据需要处理的情况下，程序执行的时间不仅仅取决于硬件的性能，还取决于你代码的质量，效率。在本次实验中，m 的值设定为 1000，我就已经感受到了“停顿”，其实这是计算机在计算，这个过程需要的时间达到了秒级，要知道计算机的计算能力是很强大的，秒级的时间差已经不小了。在之后自己测试的过程中，我自己将 m 的值设置到了万级，当 m 的值达到这个级别时，时间差已经很大了，达到了几十秒，在我测试的时候，时间差达到了 19s，即使是计算机这个强大的计算工具，在面对大量数据时，也会有很大消耗。所以在现在，不仅仅是硬件上要发展，在算法上也要发展，可能后者带来的效益会超过前者。因为硬件的发展是比较困难的。

任务 2 是一个优化代码的工作，其实在任务 1 修改功能 3 的时候，我就对功能 3 的代码进行了一定程度的修改，为了更好地实现嵌套循环。后来在优化代码的时候，发现了很多不必要的语句，然后我就将这些部分优化了。在数据量很小的情况下，代码的质量是看不出来的，然而当硬件有了限制，同时要处理的数据量比较大的时候，代码的优劣就很明显了，有的代码需要执行几个小时，而有的代码却只需要几分钟就可以实现相同的功能。在优化工作进行时，有时候会发现自己的优化并没有起到作用，有时候又会发现作用很明显。在这次优化工作中，我主要做了：（1）改变寻址方式，删去了一些不必要的寄存器，使用基址寄存器，能够直接取其中的内容，这在比较两个字符串是否相同时起到了很大的作用。（2）使用 byte ptr 加变量名的方式寻址，减少了代码长度。（3）将一些不起作用的代码删去，减少代码长度，减少执行时间。（4）将乘除指令改为移位指令，有利于提高效率。（5）还可以在比较字符串的时候先比较字符串的长度，一旦长度不一致，就可以直接进入下一个比较了。这样表面上看上去是多了一次比较，然而实际上却是一种很好的优化方法，尤其是面对大量数据时。在经过一些优化工作后，我发现程序执行时间减少了，虽然不是很多，但是应该将指令减少了几万次的执行，最终结果是程序时间优化了 28.6%。

在这次实验中，我发现改变循环程序的结构，循环次数，对 cpu 资源消耗影响是很大的。比如有一个嵌套循环，你将其改成了两个单个循环，这种优化就大大提高了效率，从几何型增长变成了线性增长。在优化的工作中，我发现了有些优化工作即使减少了代码长度，也不会对程序优化产生什么明显的效果，这可能是不同指令执行需要的时间不同，即使你减少了指令的数量，但是指令执行的时间可能增加了。比如利用 byte ptr 寻址时，即使减少了代码量，时间也似乎还是相同的。而将乘除指令转化为移位指令，减少寄存器的使用数量，寻址方式的正确使用，改变比较字符串的思路，这些优化措施对程序优化的影响程度是比较大的，一般程序优化工作也从这些方面着手。

本次上机理解了多重循环对 cpu 的消耗，主要的经验教训是在以后写代码的时候，要注意到写的代码是不是最优的，或者说是比较优秀的，这对我们养成良好的编程能力是非常有帮助的。同时要将已有的代码进行优化也不是一个简单的工作，普通的减少指令是很难将时间缩减一个数量级的，重要的是你编写程序时的思维要简洁，这对代码的优化才是影响程度最深的。

参考文献

[1] 王元珍、韩宗芬、曹忠升.《80X86 汇编语言程序设计》.华中科技大学出版社:2005 年 04 月