

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验二 程序优化**

**实验时间： 2018-4-2，14：00-17：30 实验地点： 南一楼804室**

**指导教师：**

**专业班级：计算机科学与技术201706班**

**学 号： U201714761 姓 名： 胡澳**

**同组学生： 无 报告日期： 2018年 4 月 2 日**

**原创性声明**

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.4.2

成绩评定

|  |  |  |
| --- | --- | --- |
| 实验完成质量得分（70分）（实验步骤清晰详细深入，实验记录真实完整等） | 报告撰写质量得分（30分）（报告规范、完整、通顺、详实等） | 总成绩（100分） |
|  |  |  |

指导教师签字：

日期：

目 录

[1 实验目的与要求 2](#_Toc5118298)

[2 实验内容 2](#_Toc5118299)

[3 实验过程 4](#_Toc5118300)

[3.1 任务1 4](#_Toc5118301)

[3.1.1 设计思路 4](#_Toc5118302)

[3.1.2 流程图 4](#_Toc5118303)

[3.1.3 源代码 5](#_Toc5118304)

[3.1.4 实验步骤 8](#_Toc5118305)

[3.1.5 实验记录和分析 9](#_Toc5118306)

[3.2 任务2 10](#_Toc5118307)

[3.2.1 设计思路 10](#_Toc5118308)

[3.2.2 流程图 10](#_Toc5118309)

[3.2.3 源代码 10](#_Toc5118310)

[3.2.4 实验步骤 12](#_Toc5118311)

[3.2.5 实验记录和分析 12](#_Toc5118312)

[4 总结和体会 14](#_Toc5118313)

# 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4的背景为基础，只要有一个顾客访问网店中的商品，系统就需要计算一遍所有商品的推荐度(本次实验都要按照此需求计算推荐度），然后再处理顾客实际购买的商品的信息。现假设在双十一零点时，SHOP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4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4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) 提示用户输入要购买的商品名称(比如“Bag”)。【此后可插入计时、循环】

(2) 计算SHOP中所有商品的推荐度。

(3) 在SHOP中找到顾客购买的商品(比如“Bag”，若未能找到该商品，回到(1)重新输入。若只输入回车，则回到功能一(1))。

(4) 判断该商品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。 【循环控制，计时结束】

(5) 回到功能三(1)。

请按照上述设想修改实验一任务4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。另外，也可以把定义“Bag”的位置放在所有商品的最后，使得搜索它的时间变长），以得到较明显的效果。

提示: 学校汇编教学网站的软件下载中提供了显示当前时间“秒和百分秒”的子程序。若在m次循环前调用一下该子程序，m次循环执行完之后再调用一下该子程序，就能在屏幕上观察并感受到执行循环前后的时间差（时间差值需要自行手工计算，当然，你也可以选用网站上另一个计时程序，它是可以帮你计算好差值的）。注意，由于虚拟机环境下CPU会被分时调度，故该时间差值会因计算机运行环境、状态以及虚拟机的设置参数的不同而不同。

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（注意，在编写任务1的程序时，尽量不要考虑代码优化的问题）。

优化方法提示：首先是通过选择执行速度较快的指令来提高性能，比如，把乘除指令转换成移位指令、加法指令等；其次，内循环体中每减少一条指令，就相当于减少了m\*n条指令的执行时间，需要仔细斟酌；第三，在寻址方式中尽量把16位寄存器换成32位寄存器，能有更多的机会和技巧提高指令执行效率。

# 3 实验过程

## 3.1 任务1

### 3.1.1 设计思路

在实验一任务4的基础上，将商品数量增加至N(本实验中取1000)，并将最后一个商品设置为bag，将其数量设置为numOfBag(本实验中取1000)，其余商品全部设置成相同的商品。另外，将商品定义为结构体类型，便于重复定义相同的商品；将计算所有商品的推荐度定义为子程序，以便于在循环结构中调用。在读取到用户输入商品后，判断该商品是否为bag，若为bag，则执行numOfBag次循环，模拟相应次数的购买，否则，直接按照实验一任务4执行。另外，在读取到用户输入商品后开始计时，当程序执行完相应的操作后，终止计时并输出时间。

### 3.1.2 流程图

程序流程图如图2-1所示。



图2-1 程序流程图

### 3.1.3 源代码

由于本任务基于实验一任务4，因此本节中仅展示与实验一任务4源程序中不同的部分。如下为在实验一基础上增加和修改的部分代码，未改动代码使用“……”表示。

goodStruct struct

    good\_name   DB  10 dup(0)

    good\_len    DB  0

    good\_sale   DB  10

    good\_in\_price   DW  ?

    good\_out\_price  DW  ?

    good\_all\_count  DW  ?

    good\_sale\_count DW  ?

    good\_recommamd  DW  ?

goodStruct ends

…………. ; 此处为实验一任务4中的代码

    ga1     goodStruct  <'pen$', 3, 10, 35, 56, 70, 25, ?> ; 将商品使用结构体重新定义

    ga2     goodStruct  <'book$', 4, 9, 12, 30, 25, 5, ?>

    goodStruct  N-3 dup(<'egg$', 3, 7, 35, 56, 70, 25, ?>)

    bag     goodStruct  <'bag$', 3, 10, 20, 50, numOfBag, 0, ?>

    goodLen EQU $ - bag

………… ; 此处为实验一任务4中的代码

    mov ax, 0 ; 添加计时和循环部分

    call TIMER

    call findgood

    cmp ch, -1

    jz printNotFound

    mov ax, [bx].good\_all\_count

    mov cl, auth[0]

    cmp cl, 1

    jz authOK

    jmp authNotOK

authOK:

    mov dx, bx

    mov ah, 9

    int 21h

    lea dx, outenter[0]

    mov ah, 9

    int 21h

    mov ax, 1

    call TIMER

    lea dx, outenter[0]

    mov ah, 9

    int 21h

    jmp start

authNotOK:

    call calIndex

    cmp ax, numOfBag

    jnz func4

buygood:

    mov cx, [bx].good\_sale\_count

    add cx, 1

    mov [bx].good\_sale\_count, cx

    call calIndex

    sub ax, 1

    cmp ax, 0

    jnz buygood

    jmp func4

func4:

    mov ax, 1

    call TIMER

    mov dl, [bx + 20]

    mov ah, 2

    int 21H

    jmp returnStart

returnStart:

    lea dx, outenter[0]

    mov ah, 9

    int 21h

    jmp start

printNotFound:

    mov ax, 1

    call TIMER

    lea dx, notFound[0]

    mov ah, 9

    int 21h

    jmp recommandGood

…………. ; 此处为实验一任务4中的代码

calIndex proc ; 增加计算推荐度的函数

    pusha

    lea bx, ga1[0]

    sub bx, goodLen

    mov di, N

    inc di

calLoop:

    dec di

    add bx, goodLen

    cmp di, 0

    jz calEnd

    mov ax, [bx].good\_in\_price

    mov cx, 1280

    mul cx

    push dx

    push ax

    mov al, byte ptr [bx].good\_out\_price

    mov cl, [bx].good\_sale

    mul cl

    mov cx, ax

    pop ax

    pop dx

    div cx

    push ax

    mov ax, [bx].good\_sale\_count

    mov cx, 64

    mul cx

    mov cx, [bx].good\_all\_count

    div cx

    mov dx, ax

    pop ax

    add ax, dx

    cmp ax, 100

    jns recommA

    cmp ax, 50

    jns recommB

    cmp ax, 10

    jns recommC

    jmp recommF

recommA:

    push ax

    mov ax, 'A'

    mov [bx + 20], al

    pop ax

    jmp calLoop

recommB:

    push ax

    mov ax, 'B'

    mov [bx + 20], al

    pop ax

    jmp calLoop

recommC:

    push ax

    mov ax, 'C'

    mov [bx + 20], al

    pop ax

    jmp calLoop

recommF:

    push ax

    mov ax, 'F'

    mov [bx + 20], al

    pop ax

    jmp calLoop

calEnd:

    popa

    ret

calIndex endp

…………

### 3.1.4 实验步骤

使用visual studio code编辑代码，对实验一任务4的代码进行相应的修改，并使用dosbox虚拟机中对其进行编译、链接、执行和调试。

### 3.1.5 实验记录和分析

1. 当用户输入除bag外的其他商品或登录后老板查看各商品时，程序与实验一中相同，此时执行时间极短，难以被记录，因此输出执行时间为0。其测试如图2-2和图2-3所示。

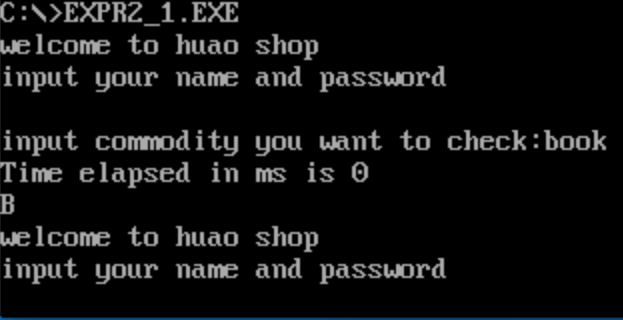


图2-2 顾客查看除bag外的其他商品

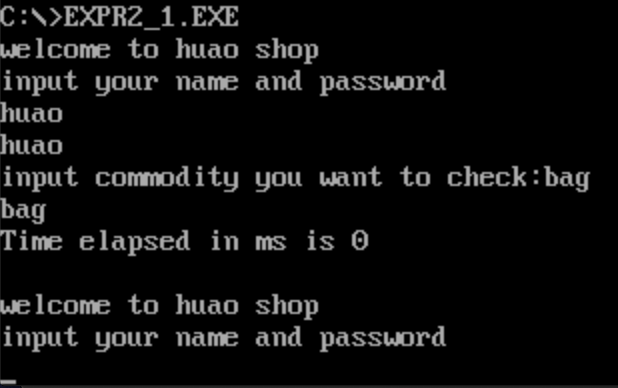


图2-3 老板查看商品信息

2. 顾客查看商品bag时，程序将模拟抢购情况，此时，程序根据商品种类数量以及bag商品的数量的设置情况的不同，将执行次数较多的循环，因此会消耗较长的时间。如图2-4所示。

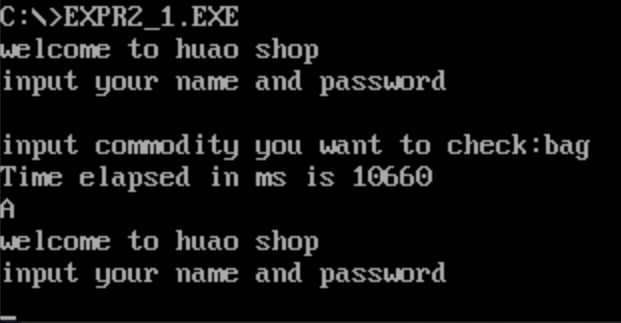


图2-4 顾客查看商品bag

## 3.2 任务2

### 3.2.1 设计思路

由于不同的汇编指令的实际执行效率不同、寄存器的读写效率远高于内存的读写效率等影响因素，使用效率较高的指令、高效的使用寄存器、减少内存读写等操作均可以减少程序的运行时间，提高运行效率。

本实验中，使用inc、dec等指令替换add和sub指令，尽可能使用寄存器保存参数而不使用内存保存参数来优化程序运行效率。

### 3.2.2 流程图

本程序流程图与任务1相同，如图2-1所示。

### 3.2.3 源代码

本程序是对任务1中程序的修改和优化，因此本节中仅展示与任务1源程序中不同的部分。如下为在任务1基础上增加和修改的部分代码，未改动代码使用“……”表示。

本优化中主要针对计算推荐度的函数进行。

…………. ; 此处为任务2中的代码

calIndex proc ; 对计算推荐度的函数进行修改

    pusha

    lea bx, ga1[0]

    sub bx, goodLen

    mov di, N+1

calLoop:

    add bx, goodLen

    dec di

    jz calEnd

    mov ax, [bx].good\_in\_price

    mov cx, 1280

    mul cx

    mov bp, dx

    mov si, ax

    mov al, byte ptr [bx].good\_out\_price

    mov cl, [bx].good\_sale

    mul cl

    mov cx, ax

    mov dx, bp

    mov ax, si

    div cx

    mov bp, ax

    mov ax, [bx].good\_sale\_count

    mov cx, 64

    mul cx

    mov cx, [bx].good\_all\_count

    div cx

    add ax, bp

    cmp ax, 100

    jns recommA

    cmp ax, 50

    jns recommB

    cmp ax, 10

    jns recommC

    jmp recommF

recommA:

    mov bp, 'A'

    mov [bx + 20], bp

    jmp calLoop

recommB:

    mov bp, 'B'

    mov [bx + 20], bp

    jmp calLoop

recommC:

    mov bp, 'C'

    mov [bx + 20], bp

    jmp calLoop

recommF:

    mov bp, 'F'

    mov [bx + 20], bp

    jmp calLoop

calEnd:

    popa

    ret

calIndex endp

………… ; 此处为任务2中的代码

### 3.2.4 实验步骤

使用visual studio code编辑代码，对任务1的代码进行相应的修改，并使用dosbox虚拟机中对其进行编译、链接、执行和调试。

### 3.2.5 实验记录和分析

分别运行任务1中未优化的程序和任务2中优化后的程序，获得两者中作为顾客查看商品bag所需的执行时间，如图2-5和图2-6所示。

未优化的程序完成上述过程的时间为10660ms，优化后的程序完成相同过程的时间为9060ms，因此优化率为。由此可见，通过使用效率更高的汇编指令、尽可能使用寄存器存储数据而不是内存可以有效的提高汇编程序的运行效率。

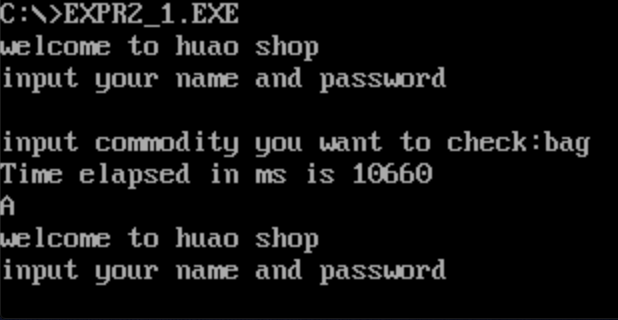


图2-5 未优化程序运行时间

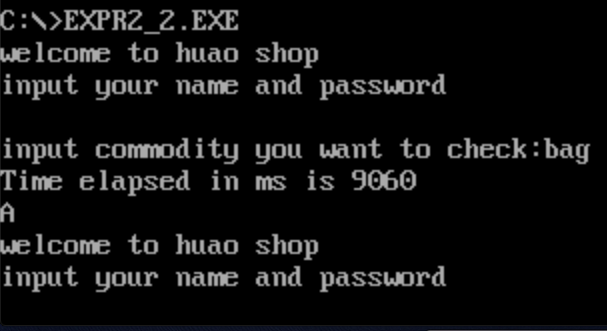


图2-6 优化后程序运行时间

# 4 总结和体会

本次实验中，我对汇编程序进行了一定的优化，并得到了较为明显的优化率。通过本次实验，我认识到了不同的汇编指令之间存在的效率的区别，例如乘除法与移位指令之间有时可以实现相同的功能，但是移位指令的效率要远高于乘除法指令。另外，本次实验让我认识到计算机不同的存储层次的读写效率的明显差距，cpu在处理寄存器中的数据的效率要远高于处理内存中数据的效率，由此可见，在编写汇编程序时，合理使用寄存器和内存来存储数据可以有效的提高程序的运行效率。