

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 编程基础**

**实验时间： 2018-4-13，18:30-21:50 实验地点： 南一楼804室17号实验台**

**指导教师： 李海波 专业班级：计算机科学与技术201607班**

**学 号： U201614700 姓 名： 王亚宁**

**同组学生： 无 报告日期： 2018年4月13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：

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**

[实验目的与要求 1](#_Toc511569821)

[实验内容 1](#_Toc511569822)

[实验过程 1](#_Toc511569823)

[任务1 1](#_Toc511569824)

[1. 设计思想及存储单元分配 1](#_Toc511569825)

[2. 流程图 2](#_Toc511569826)

[3. 源程序 2](#_Toc511569827)

[4. 实验步骤 2](#_Toc511569828)

[5. 实验记录 9](#_Toc511569829)

[体会 10](#_Toc511569830)

[参考文献 10](#_Toc511569831)

# 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。

# 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！

# 实验过程

## 任务1

### 设计思想及存储单元分配

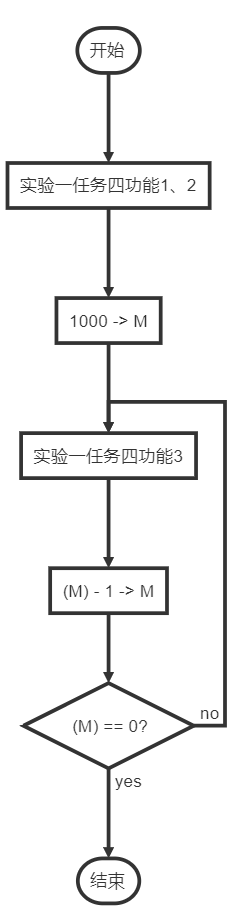
设计思想：本次实验是对第一次实验的代码改动加入计时功能，添加的计时函数我的想法是使用80x86的时间指令获取毫秒级别的时间戳，在func3运行前和运行结束后分别得到时间戳，后减前就可以得出函数运行的时间，最后使用PRINT\_NUM函数输出时间。

内存分配：所有分配基本相同，只是多了个存储时间的变量，可以将时间存储在一个双字中，直接使用时间戳，两时间相减得出程序运行时间。

时间戳获取的函数的内存分配：共使用了EAX, EBX, ECX, EDX寄存器，使用前先将其压栈保存环境，调用系统时间函数后ECX和EDX中存放的是返回值，EBX用来存放临时的CL, CH, DL, DH等寄存器值，这样就可以使用32位寄存器EAX计算时间毫秒了。

输出数字函数的内存分配：函数参数存放在栈中，函数中使用了BP，EAX, EBX, ECX, EDX寄存器，同样使用前先压栈保存环境，使用时EAX保存参数，RBX记录数字位数，ECX作除数，EDX存放余数。

### 流程图



### 源程序

实验.386

STACK SEGMENT USE16 STACK

DB 400 DUP(0)

STACK ENDS

DATA SEGMENT USE16

NOTICE\_NAME DB 'Please enter user name: $', 5 DUP(0)

NOTICE\_PWD DB 'Please enter user password: $', 2 DUP(0)

BNAME DB 'yaning', 4 DUP(0); 用户名

BPASS DB 'passwd', 4 DUP(0); 密码

IN\_NAME DB 20, ?, 20 DUP(0)

IN\_PWD DB 20, ?, 20 DUP(0)

G\_CNT = 60

S1 DB 'SHOP1', 5 DUP(0)

BAG1 DB 'BAG', 7 DUP(0)

DW 12, 30, 1000, 5, ? ; 利润率未计算

G\_SIZE = $ - BAG1

GA1 DB 'PEN', 7 DUP(0)

DW 35, 56, 70, 25, ? ; 利润率未计算

GA2 DB 'BOOK', 6 DUP(0)

DW 12, 30, 25, 5, ? ; 利润率未计算

GAN DB G\_CNT - 3 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?); 其他商品暂时未知

S\_SIZE = $ - S1

S2 DB 'SHOP2', 5 DUP(0) ;网店名称，用0结束

BAG2 DB 'BAG', 7 DUP(0)

DW 12, 30, 2000, 2, ? ; 利润率未计算

GB1 DB 'PEN', 7 DUP(0) ;商品名称

DW 35, 50, 30, 24, ? ;利润率还未计算

GB2 DB 'BOOK', 6 DUP(0) ; 商品名称

DW 12, 28, 20,15, ? ;利润率还未计算

GBN DB G\_CNT - 3 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?)

AUTH DB 0; 标志是否通过验证

GG\_INDEX DD 2 DUP(0); 一共两家店,存放查询的物品的地址在内存中

G\_INDEX DD 0

SHIFTLINE DB 13, 10, '$'

S\_CNT = 2

S\_TEMP\_CNT DB 0

G\_TEMP\_CNT DB 0

SG\_INDEX DD 0

S\_INDEX DD 0

G\_COST DD 0

G\_PRO DD 0

TIME DD 0

; proc

;output

\_ADD DW 0

NUM DD 0

DATA ENDS

CODE SEGMENT USE16

ASSUME CS:CODE, SS:STACK, DS:DATA, ES:DATA

START:

MOV AX, DATA

MOV DS, AX

; code after this

FUNC1: ; 功能1开始

PUSH OFFSET NOTICE\_NAME

CALL OUTPUT

PUSH OFFSET IN\_NAME

CALL INPUT

CALL CRLF

PUSH OFFSET NOTICE\_PWD

CALL OUTPUT

PUSH OFFSET IN\_PWD

CALL INPUT

CALL CRLF

FUNC2: ;功能2开始

; 用户名输入长度为0？

MOV AUTH, 0

CMP IN\_NAME[1], 0

; 调到功能3

JZ FUNC3

; 长度为1，检测是否是q

CMP IN\_NAME[1], 1

; 不是1跳check\_name

JG CHECK\_NAME

CMP IN\_NAME[2], 'q'

; 输入为q，退出

JZ QUIT

CHECK\_NAME:

; 检查用户名

; 已知用户名长度为6，作为已知量使用

CMP IN\_NAME[1], 6

; 不为6错误

JNZ FUNC1

MOV CX, 6

LEA EBX, OFFSET IN\_NAME + 2

LEA EDX, OFFSET BNAME

LOOP\_NAME: ; 偷懒从高到低检查

MOV AH, [EBX]

MOV AL, [EDX]

CMP AH, AL

JNZ FUNC1

INC EBX

INC EDX

LOOP LOOP\_NAME

; 密码也是6位

CMP IN\_PWD[1], 6

JNZ FUNC1

MOV CX, 6

LEA EBX, OFFSET IN\_PWD

ADD EBX, 2

LEA EDX, OFFSET BPASS

LOOP\_PWD:

MOV AH, [EBX]

MOV AL, [EDX]

CMP AH, AL

; 密码不同跳功能1

JNZ FUNC1

INC EBX

INC EDX

LOOP LOOP\_PWD

; 通过用户名和密码检测

MOV AUTH, 1

; 进入功能3

CALL GET\_MS

MOV TIME, EAX

FUNC3:

CMP AUTH, 1

JNZ FUNC1

LEA SI, OFFSET S1

ADD SI, 10; 指向物品地址

; BAG我放在了第一个物品处

MOV DX, [SI + 14];进货数

MOV BX, [SI + 16];卖的数量

CMP DX, BX

JLE FUNC1

ADD WORD PTR [SI+ 16], 1

MOV CX, G\_CNT

NEXT\_G:

PUSH SI; SI里面放的就是当前物品地址

CALL CALCUG\_PRO

MOV [SI + 18], AX

ADD SI, G\_SIZE

LOOP NEXT\_G

; 检测M是否为0

LEA SI, OFFSET S1

ADD SI, 10; 指向物品地址

; BAG我放在了第一个物品处

DEC DX;进货数

;MOV BX, [SI + 16];卖的数量

CMP DX, BX

JE FUNC4

JMP FUNC3

FUNC4:

; 时间计算一下

CALL GET\_MS

SUB EAX, TIME

; print参数压栈

PUSH EAX

; 显示运行时间

CALL PRINT\_NUM

CALL CRLF

; 原来的出栈

MOV AX, [SI + 18]

; 检测利润率范围

CMP AL, 90

JGE SHOWA

CMP AL, 50

JGE SHOWB

CMP AL, 20

JGE SHOWC

CMP AL, 0

JGE SHOWD

JMP SHOWF

SHOWA:

MOV DL, 'A'

JMP SHOW

SHOWB:

MOV DL, 'B'

JMP SHOW

SHOWC:

MOV DL, 'C'

JMP SHOW

SHOWD:

MOV DL, 'D'

JMP SHOW

SHOWF:

MOV DL, 'F'

SHOW:

MOV AH, 2

INT 21H

; 换行

CALL CRLF

JMP FUNC1

QUIT:

;stop here

MOV AH, 4CH

INT 21H

; 函数：获取时间戳，以ms为单位，

; 参数: 无

; 返回值：EAX:

GET\_MS PROC

PUSH EBX

PUSH ECX

PUSH EDX

MOV AH, 2CH

INT 21H

XOR EAX, EAX

XOR EBX, EBX

; 获取小时

MOV BL, CH

ADD EAX, EBX

;; 转为分钟

IMUL EAX, 60

; 获取分钟

MOV BL, CL

ADD EAX, EBX

; 转为s

IMUL EAX, 60

; 获取10ms

MOV BL, DH

ADD EAX, EBX

IMUL EAX, 100

MOV BL, DL

ADD EAX, EBX

; 转为ms

IMUL EAX, 10

POP EDX

POP ECX

POP EBX

RET

GET\_MS ENDP

; 输出函数：输出参数中的字符串

; 参数：要输出的字符串的地址：字

; 返回值：无

OUTPUT PROC

;local \_ADD DW

POP BP

POP \_ADD

PUSH EAX

PUSH EDX

MOV DX, \_ADD

MOV AH, 9

INT 21H

POP EDX

POP EAX

PUSH BP

RET

OUTPUT ENDP

; 输入函数：输入一个字符串到缓冲区

; 参数：要输入的缓冲区的地址：字

; 返回值：无

INPUT PROC

POP BP

POP \_ADD

PUSH EAX

PUSH EBX

MOV DX, \_ADD

MOV AH, 10

INT 21H

POP EBX

POP EAX

PUSH BP

RET

INPUT ENDP

; 换行函数

; 参数：无

; 返回值：无

CRLF PROC

PUSH EAX

PUSH EDX

; 获取换行字符串

LEA DX, OFFSET SHIFTLINE

MOV AH, 9

INT 21H

POP EDX

POP EAX

RET

CRLF ENDP

; 函数：计算一个商品的所有利润率

; 参数：第一个商店里的物品的偏移地址

; 返回值：利润率AX

CALCUG\_PRO PROC

POP BP

POP WORD PTR \_ADD

PUSH EBX

PUSH ECX

PUSH EDX

PUSH SI

MOV G\_PRO, 0

MOV SI, WORD PTR \_ADD

MOV CX, S\_CNT

L1:

PUSH WORD PTR G\_PRO

XOR EAX, EAX

XOR EBX, EBX

XOR EDX, EDX

MOV AX, [SI + 10]

MOV BX, [SI + 14]

IMUL AX, BX

; 80x86低位在前

MOVSX EAX, AX; 拓展AX符号位

MOV G\_COST, EAX

MOV AX, [SI + 12]

MOV BX, [SI + 16]

IMUL AX, BX

MOVSX EAX, AX

MOV G\_PRO, EAX

MOV EAX, G\_PRO

MOV EBX, G\_COST

SUB EAX, EBX

IMUL EAX, 100

CDQ

IDIV EBX; 得出结果

POP WORD PTR G\_PRO

ADD G\_PRO, EAX

ADD SI, S\_SIZE

LOOP L1

; 恢复现场并返回

MOV EAX, G\_PRO; AX为低位

POP SI

POP EDX

POP ECX

POP EBX

PUSH BP

RET

CALCUG\_PRO ENDP

; 输出函数：输出参数的十进制表示

; 参数：一个双字的数值

; 返回值：无

; 注意：

PRINT\_NUM PROC

POP BP

POP DWORD PTR NUM

PUSH EAX

PUSH EDX

PUSH ECX

PUSH EBX

MOV EBX, 10

MOV EAX, NUM

XOR ECX, ECX

CDQ

CMP EAX, 0

JGE L2

PUSH EAX

PUSH EDX

; 输出负号，正数不输出

MOV DL, '-'

MOV AH, 2

INT 21H

POP EDX

POP EAX

NEG EAX

L2:

IDIV EBX

ADD DL, '0'

PUSH DX

; 还原EDX的实际情况

CDQ

INC ECX

CMP EAX, 0

JG L2

; 栈里一共ECX个数字可用

L3:

POP DX

MOV AH, 2

INT 21H

LOOP L3

; 恢复环境

POP EBX

POP ECX

POP EDX

POP EAX

PUSH BP

RET

PRINT\_NUM ENDP

CODE ENDS

END START

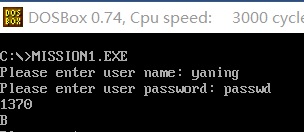
### 步骤

1. 修改代码：主要是加入计时功能。
2. 计时功能的设计：查书找到系统时间函数的定义，返回值的存放情况，编写函数得出毫秒级时间戳，返回值放置在EAX中。
3. 代码优化：避免不必要的循环，减少运算中的冗余部分，多使用寄存器，少使用内存。
4. 对比优化前后运行时间。

### 实验记录

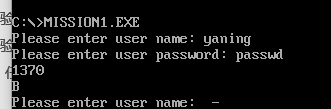
1. 计时代码的编写：调用系统函数后将返回值转换为毫秒，放在EAX中，这个函数没有遇到什么困难。
2. 数字输出的代码的编写：使用EAX存放参数，EDX存放商，EBX存放除数，ECX存放数字位数，将每次的余数加上’0’后入栈，最后使用一个LOOP循环将数字输出。
3. 本次实验的主要浪费时间的地方是函数的编写，函数编写完成后需要进行测试，但是汇编语言下面，几乎不可能像高级语言一样写测试，只能在调试时手动debug，效率很低，而且不容易排查出错误。
4. 代码编写时遇到的问题主要还是死循环问题，有时候用
5. 代码修改完后运行如图所示：

用户名和密码，enter之后会自动运行func3：



可见运行时间时1370ms，计算后得出bag的所处等级时B级，利润率大于等于80%小于90%。

1. 代码优化：代码优化有很多地方，我先对几个函数进行了优化，减少了内存的使用，赋值0的操作使用XOR代替，减少不必要的寄存器使用，修改后的代码运行时间：



看起来优化似乎没有什么作用。

# 体会

本次的实验前刚好学习了函数的定义，正好派上了用场，通过函数的使用，我可以少写很多重复代码，而且代码的可读性也得到了提高，代码在修改的时候也比较方便。函数的引入使汇编语言有了一点高级语言的样子，接下来的学习估计也会变得简单一些。

# 参考文献

[1] 王元珍等.80x86汇编语言程序设计.版本(第1版)

[2] 王晓虹等.汇编语言程序设计教程.版本(第1版)