

**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 程序执行时间与代码长度优化**

**实验时间： 2018-4-9，14：00-17：30 实验地点： 南一楼804室90号实验台**

**指导教师： 李海波**

**专业班级：校交 201601班**

**学 号： U201612696 姓 名： 陈淏睿**

**同组学生： 无 报告日期： 2018年 4 月 10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：2018.4.10

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 1](#_Toc515177486)

[2 实验内容 1](#_Toc515177487)

[3 实验过程 2](#_Toc515177488)

[3.1 任务1 2](#_Toc515177489)

[3.1.1 实验步骤 2](#_Toc515177490)

[3.1.2 流程图 2](#_Toc515177491)

[3.1.3 源程序 3](#_Toc515177492)

[3.1.4 实验记录与分析 7](#_Toc515177493)

[3.2 任务2 9](#_Toc515177494)

[3.2.1 实验步骤 9](#_Toc515177495)

[3.2.2 流程图 9](#_Toc515177496)

[3.2.3 源程序 10](#_Toc515177497)

[3.2.4 实验记录与分析 11](#_Toc515177498)

[4 总结与体会 13](#_Toc515177499)

[参考文献 15](#_Toc515177500)

# 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。

提示: （1）在进入调整后的功能三之前增加m次循环的初始化工作，在调整后的功能三结束之后增加m次循环的条件判断和转移语句。

（2）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与状态的不同而不同。

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。减少的越多，评价越高！

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，尽量采用32位寄存器寻址，能有更多的机会提高指令执行效率。

# 实验过程

## 任务1

### 实验步骤

1. 准备上机实验环境,将实验一任务四的拷贝重命名t1-,在此基础上修改程序。

2. 在进入调整后的功能三之前增加m次循环的初始化工作。新建存储器变量M作为循环计数器，在从功能二进入功能三时将一足够大的值存入M。在进入循环前调用计时程序TIMER开始计时。

3. 在调整后的功能三结束之后增加m次循环的条件判断和转移语句。从M中取出当前计数器值并自减，不为0则继续下一重循环，否则再次调用计时器TIMER结束计时。

4. 修改功能三循环体。先在商店一中找到BAG，判断已售数目是否小于进货数目，若非则已售数目自增，是则提示并直接返回程序起点。

5. 对商店一中的每个商品，在商店二中找到目标商品后，分别计算两商店中商品利润。需注意的是商店一中相应商品的利润偏移地址需单独保存为存储器变量，以方便计算总利润时从记录的位置去商店一中利润与之后将总利润存入商店一中相应位置。

6. 汇编，连接文件t1-。

7. 反复运行与TD单步调试，直到程序能正常退出循环并给出相应正确利润等级。

### 流程图

图3.1.1是任务1的程序流程图。

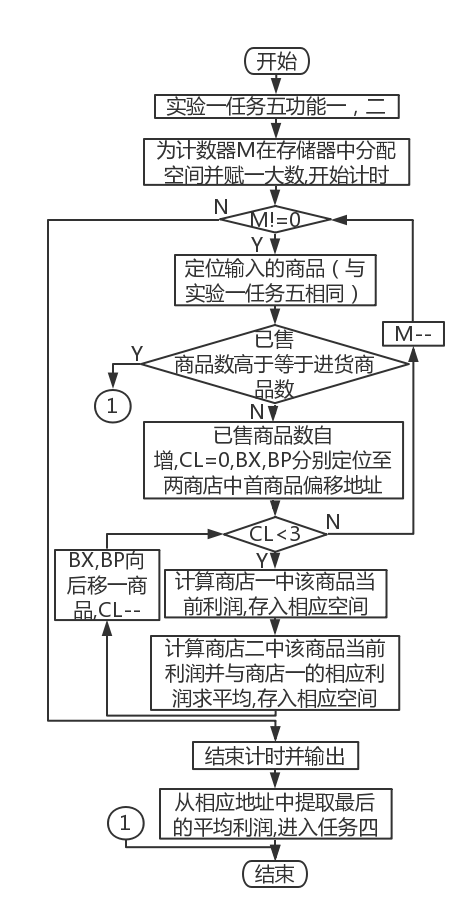


图3.1.1程序流程图

### 源程序

.386

┇ ；此行之前为实验一相关数据定义

GA3 DB 'BAG',7 DUP(0) ;商品名称

DW 1,3,19000,12,? ;利润率还未计算

GAN DB N-3 DUP( 'Temp-Value',15,0,20,0,30,0,2,0,?,?) ;除了2个已经具体定义了商品信息以外，其他商品信息暂时假定为一样的。

S2 DB 'SHOP2',0 ;网店名称，用0结束

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

GB3 DB 'BAG',7 DUP(0) ;商品名称

DW 20,35,42,35,? ;利润率还未计算

GBN DB N-3 DUP( 'Temp-Value',15,0,20,0,30,0,2,0,?,?) ;除了2个已经具体定义了商品信息以外，其他商品信息暂时假定为一样的。

TIP\_1 DB 'Enter ID:',0AH,0DH,'$'

TIP\_2 DB 'Enter PW:',0AH,0DH,'$'

TIP\_3 DB 'Input the commodity you want to look for:',0AH,0DH,'$'

TIP\_4 DB 'Input the commodity you want to look for again',0AH,0DH,'$'

TIP\_5 DB 'Login as visitor',0AH,0DH,'$'

TIP\_6 DB 'Login as manager',0AH,0DH,'$'

TIP\_7 DB 'ID incorrect',0AH,0DH,'$'

TIP\_8 DB 'password incorrect',0AH,0DH,'$'

TIP\_9 DB 0AH,0DH,'$' ;换行，回车

TIP\_10 DB 'item utterly sold out',0AH,0DH,'$'

PR1 DW ? ;商店一中的利润地址

M DW ? ;步骤三的循环次数

┇ ；此行之前为实验一相关数据定义

┇ ；此行之前为实验一任务五功能三相关代码

INC DI ;找到真实串长

CMP BL,[DI]

JE STEP1

MOV M,10000 ;每次由功能二进入功能三时初始化循环计数器

MOV AX,0

CALL TIMER

BRK17:

MOV CL,3 ;外循环计数器

LEA BP,GA1

LPB:

LEA BX,IN\_GOOD

INC BX

MOV DX,[BX]

MOV DH,0

LPBA: ;比较商店一中当前字符串与输入串

LEA BX,IN\_GOOD

INC BX

MOV AX,[BX]

MOV AH,0

SUB AX,DX

MOV SI,AX

MOV BX,DS:[BP+SI]

MOV BH,0

MOV DI,BX

LEA BX,IN\_GOOD

ADD BX,2

MOV AX,SI

XLAT

CMP DI, AX

JNE BRK1 ;当前字符不同时直接退出循环

DEC DX

JNE LPBA ;SI非0时继续下一重循环

BRK1:

CMP DX,0 ;SI不为0说明对输入字符串的遍历未到头，即字符串不同，应继续比较下一字符串；为0则跳出外循环

JNE BRK13

INC SI

MOV BX,DS:[BP+SI]

CMP BX,0

JE BRK2

BRK13:

ADD BP,20 ;BP移向商店一中下一货物

DEC CL

JNE LPB ;商品未查找完

BRK2:

CMP CL,0 ;CL不为0说明已找到商品

JNE BRK3

LEA DX,TIP\_4 ;显示提示:重新输入待查商品名称

MOV AH,9

INT 21H

JMP STEP3

BRK3:

CMP AUTH,1

JNE BRK6

MOV CX,DS:[BP+16] ;已售数量（用于比较）

CMP CX,DS:[BP+14] ;进货数量

JB BRK15 ;已售数量低于进货数量

LEA DX,TIP\_10 ;显示提示:已售数量高于进货数量

MOV AH,9

INT 21H

JMP STEP1

BRK15:

MOV CX,DS:[BP+16]

INC CX

MOV DS:[BP+16],CX ;已售数量加一

MOV CL,3 ;CL负责商店一的遍历

LEA BP,GA1

LPC: ;之前所有寄存器的值可作废

MOV DX,0

LPC\_A: ;获得商店一中当前商品串长，存入AX(AL)

CMP BYTE PTR DS:[BP+DX],0 ;//变址寄存器使用错误，应该为SI,DI之一

JE BRK18

INC SI

JMP LPC\_A

BRK18:

MOV AX,SI

MOV CH,3 ;CH负责商店二的遍历

LEA BX,GB1

LPC\_B:

CALL CMPST ;比较两商品串是否相同,AH应为0

OR AH,AH

JNZ BRK20

ADD BX,20 ;BX移向商店二中下一商品，继续比较

DEC CH

JNZ LPC\_B

JMP STEP1 ;商店二中商品已遍历完毕，仍未找到

BRK20: ;更新各商品均利润率（CL,BX,BP被占用）

MOV SI,DS:[BP+10] ;进货价

IMUL SI,DS:[BP+14] ;进货数量

MOV AX,DS:[BP+12] ;销售价

IMUL AX,DS:[BP+16] ;已售数量

SUB AX,SI

MOV DI,100

IMUL DI

IDIV SI

MOV DS:[BP+18],AX ;商店一中的利润按字形式保存

MOV SI,DS:[BX+10] ;进货价

IMUL SI,DS:[BX+14] ;进货数量

MOV AX,DS:[BX+12] ;销售价

IMUL AX,DS:[BX+16] ;已售数量

SUB AX,SI

IMUL DI

IDIV SI

MOV DS:[BX+18],AX ;商店二中的利润按字形式保存

ADD AX,DS:[BP+18] ;两利润相加

CMP AX,0

JL BRK11

MOV DX,0 ;AX为正

JMP BRK12

BRK11:

MOV DX,0FFFFH ;AX为负

BRK12:

MOV DI,2

IDIV DI ;求平均利润

MOV DS:[BP+18],AX ;平均利润存入SHOP1的利润字段

DEC CL

OR CL,CL

JZ BRK5

ADD BP,20 ;BP移向商店一中下一商品

JMP LPC

BRK5:

LEA BX,M

┇ ；此行之后为实验一任务五功能三相关代码

┇ ；此行之后为实验一任务五功能四相关代码

TIMER PROC

PUSH DX

PUSH CX

PUSH BX

MOV BX, AX

MOV AH, 2CH

INT 21H ;CH=hour(0-23),CL=minute(0-59),DH=second(0-59),DL=centisecond(0-100)

MOV AL, DH

MOV AH, 0

IMUL AX,AX,1000

MOV DH, 0

IMUL DX,DX,10

ADD AX, DX

CMP BX, 0

JNZ \_T1

MOV CS:\_TS, AX

\_T0: POP BX

POP CX

POP DX

RET

\_T1: SUB AX, CS:\_TS

JNC \_T2

ADD AX, 60000

\_T2: MOV CX, 0

MOV BX, 10

\_T3: MOV DX, 0

DIV BX

PUSH DX

INC CX

CMP AX, 0

JNZ \_T3

MOV BX, 0

\_T4: POP AX

ADD AL, '0'

MOV CS:\_TMSG[BX], AL

INC BX

LOOP \_T4

PUSH DS

MOV CS:\_TMSG[BX+0], 0AH

MOV CS:\_TMSG[BX+1], 0DH

MOV CS:\_TMSG[BX+2], '$'

LEA DX, \_TS+2

PUSH CS

POP DS

MOV AH, 9

INT 21H

POP DS

JMP \_T0

\_TS DW ?

DB 'Time elapsed in ms is '

\_TMSG DB 12 DUP(0)

TIMER ENDP

CMPST PROC ;比较字符串函数

MOV SI,0

LPE:

MOV DH,DS:[BX+SI]

CMP DH,DS:[BP+SI]

JNE BRK19

INC SI

CMP SI,AX

JNE LPE

INC SI

CMP BYTE PTR DS:[BX+SI],0 ;判断BX是否同样遍历完成

JNE BRK19

MOV AH,1 ;字符串相同

RET

BRK19:

MOV AH,0 ;字符串不同

RET

CMPST ENDP

┇ ；此行之后代码段结束

### 实验记录与分析

1. 实验环境条件：Intel® Core™ i5-3230M CPU 2.60GHz，2.86G内存；WINDOWS 7下DOSBox0.74； notepad++ 7.55;MASM.EXE 6.0;LINK.EXE 5.2;TD.EXE 5.0。

2. 将实验一task5副本重命名为t1-，按实验步骤中的思路修改功能三，M值初始设定为1000。

3. 汇编连接中报错，如图3.1.2。经仔细检查排除了上述错误，修改方式见源代码。

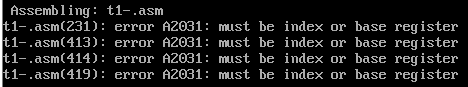


图3.1.2汇编报错

4.重新汇编并连接，无异常情况，如图3.1.3所示。

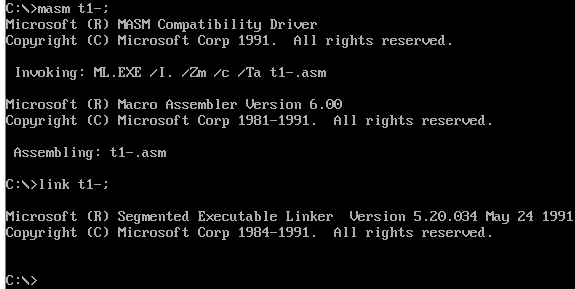


图3.1.3汇编连接成功

4. 执行t1-,发现总利润等级正确，但执行时间过短，如图3.1.4所示，不便于任务二观察，因此考虑放大M为5000。

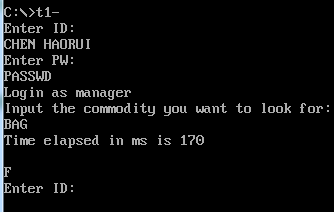


图3.1.4 M为1000时的总利润评级及执行时间

5. 将M改为5000，重新汇编，连接，执行，发现执行时间依然过小，如图3.1.5所示。将M放大至10000。



图3.1.5 M为5000时的总利润评级及执行时间

6.将M改为10000，经计算可知总利润评级正确，同时执行时间已可供任务二使用。

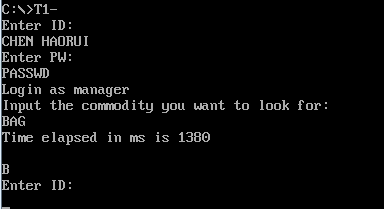


图3.1.6 M为10000时的总利润评级及执行时间

## 任务2

### 实验步骤

1. 准备上机实验环境。

2. 将t1-的副本重命名为t2-，在此基础上将在商店一中查找商品的循环以保存商品在存储器位置中的方式提出订货循环。

3. 设法通过新建存储器变量，空间换时间，减少通过寄存器对立即数的间接操作。

4. 将循环中所有16位寄存器改为32位寄存器。

5. 汇编，连接文件t2。

6. 反复运行与TD单步调试，直到程序能正常退出循环并给出相应正确利润等级。

### 流程图

图3.2.1是任务2的程序流程图。

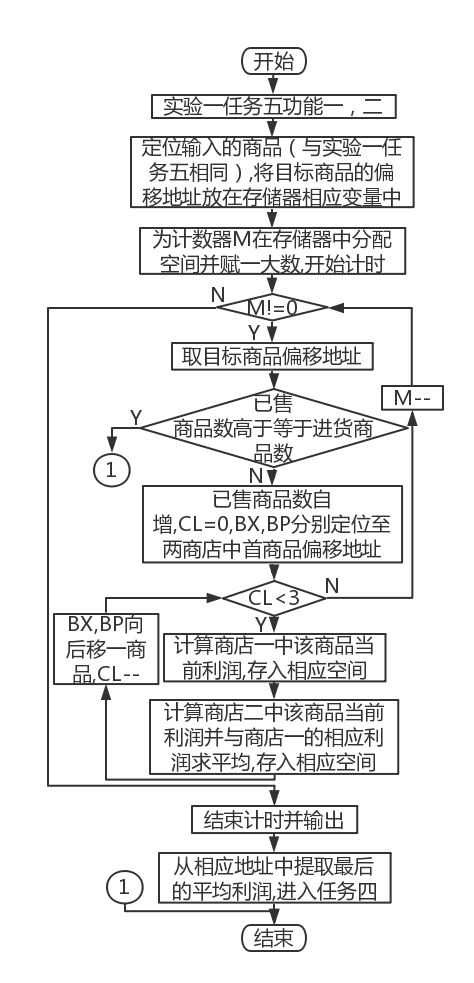


图3.2.1程序流程图

### 源程序

.386

┇ ；此行之前为任务一相关数据段

AUTH DB ? ;判断登录方式

HKU DW 100

┇ ；此行之后数据段结束

┇ ；此行之前为任务一相关代码

SUB AX,SI

IMUL HKU

IDIV SI

┇ ；此行之后为任务一相关代码

┇ ；此行之前为任务一相关代码

SUB AX,SI

IMUL HKU

IDIV SI

┇ ；此行之后为任务一相关代码

┇ ；此行之前为任务一相关代码

SUB AX,SI

ADD AX,DS:[EBP+18] ;两利润相加

CWD

SAR AX,1 ;求平均利润

MOV DS:[EBP+18],AX ;平均利润存入SHOP1的利润字段 IDIV SI

┇ ；此行之后为任务一相关代码

┇ ；此行之前为任务一相关代码

CMP BYTE PTR DS:[EBX+ESI],0 ;判断BX是否同样遍历完成

JNE BRK19

INC AH ;字符串相同

BRK19:

RET

┇ ；此行之后代码段结束

### 实验记录与分析

1. 实验环境条件：Intel® Core™ i5-3230M CPU 2.60GHz，2.86G内存；WINDOWS 7下DOSBox0.74； notepad++ 7.55;MASM.EXE 6.0;LINK.EXE 5.2;TD.EXE 5.0。

2. 将订货循环中的查找商品部分提出循环，目标商品偏移地址存入存储器，在订货循环中直接提取其地址进行比较。

3. 汇编与连接过程没有发生异常。

4. 运行时输入BAG，发现几乎立即跳出“商品已售空”提示，如图3.2.2所示。然而程序中订单数目为10000，与商店一中已售数目25之和远小于进货数目19000，因此判断发生逻辑错误。由于目前只在t1的基础上修改了这一处，因此需检查偏移地址的存取过程。

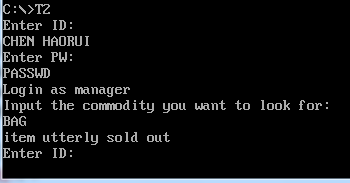


图3.2.2程序执行出错

5.执行LEA BP,PR1后，发现赋给BP的不是PR1内的目标商品的偏移地址，而是PR1的偏移地址，如图3.2.3所示。于是意识到LEA使用出错，此处应改为MOV BP,PR1。在本次实验单步执行过程中，由于一点点翻找待检查的代码太过困难，因此我开始研究CS段窗口右键菜单，从而发现选search后可输入待查语句附近有比较明显特征的语句时可直接定位至对应语句，向上至下寻找便可定位目标语句，从而提高检查效率。

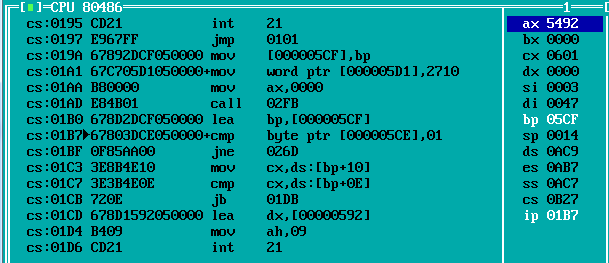


图3.2.3向BP的赋值出现问题

6. 重新汇编，连接并执行，可看到程序正常完成了循环并给出了正确结果，同时运行时间大幅减少，如图3.2.4所示。

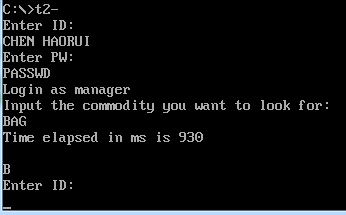


图3.2.4运行时间大幅减少

7. 新建寄存器变量HKU并存放100，将MOV DX,100和IMUL DX用IMUL HKU代替，将求平均利润根据前AX的符号决定DX的值的分支结构直接用CWD代替，将MOV DI,2和IDIV DI直接用SAR AX,1代替。重新编译，连接，运行，结果如图3.2.5所示，发现循环时间再次减少。

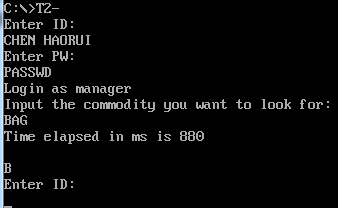


图3.2.5运行时间减少

8. 删除无关语句，如将商店二的商品利润保存；在单步调试CMPST函数过程中可知AH在循环中始终为0，因此只在判断字符串相等时INC AH，删除两条对AH的赋值语句。重新编译，连接，运行。结果如图3.2.6所示，发现循环时间再次减少；将所有用于间接寻址，基址寻址与基址加变址寻址的寄存器由16位改为32位，再次执行，发现程序运行时间没有变化。

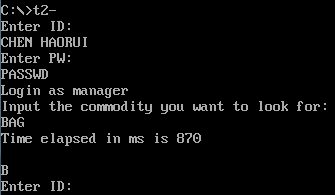


图3.2.6运行时间减少

# 总结与体会

在本次试验中我又一次意识到了细心审题的重要性。我在实验课前已基本完成任务一，同时任务二也已经找到了思路只待验证，然而上课后才得知自己对于任务一的理解出了偏差，不得不推倒重来，从而严重耽搁了实验进度。

任务1让我对CPU占用率随循环规模的增大而显著增长有了更深刻的领悟。在用C语言写循环体时，程序执行的时间受到环境的干扰较为严重，执行观察时显示的时间波动较为剧烈，从而难以对循环对执行时间的影响有较为深刻的领悟。而用汇编语言写的循环受环境影响较小，多次执行统一循环后其执行时间保持了相对稳定，从而达到了控制无关变量的效果。本任务在一定程度上模拟了网站对于一个密集时间段内同时到达的多个请求的反应方法，企业级的服务器数据处理能力虽然远大于笔记本电脑，但其面临的瞬时请求密度相对的也远大于任务假设。因此，即使再用高级语言编程时更要注意优化算法，从而简化便以形成的汇编原文件的代码长度，进一步缩短代码执行时间。

任务2使我明白，减小时间复杂度永远是算法研究的主要方向之一。在本任务中，只是单单将一个二重循环变为一重循环便显著地缩短了程序执行时间，而其它诸如减少寄存器与存储期间的数据传输，用移位代替乘除乃至换用32位寄存器寻址带来的优化合起来可能也不能与之匹敌。然而微小变量经过放大也能带来显著改变，因此我们也不能忽视代码细节，从指令下手也可起到优化代码的作用。

思考题：

任务1

1.首先应通过扩大循环次数让优化前的执行时间变得可用程序检测，同时因优化产生的时间缩短足够明显，其次应优化程序，通过优化前后的执行时间长度对比说明时间优化的效果。给出的两个时间检测程序代表了两种思路：一是分别给出程序执行前后的时间，手动计算时间差；二是在程序开始执行时计时，程序执行完成后结束计时，直接给出执行时间。

2.从任务1中可以看出，随着循环次数的增加，程序的运行时间明显提高，说明对CPU的占用率增大。

3.分别在DOSbox和WIN7 32位下运行t2-,结果分别如图4.1和图4.2所示，可见Win7 32位下程序执行速度明显快于DOSbox下。



图4.1 DOSbox下运行t2-

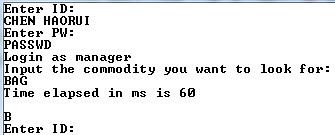


图4.2 WIN7 32位下运行t2-

任务2

1.减少循环嵌套层数；换用执行效率更高的语句；改良程序结构；减少寄存器与存储器的数据交互；换用32位寄存器寻址。

2.在循环体内的指令；可用较少指令替代的一段指令；可用移位运算替代的算术运算指令；数据交互指令。

4.时间复杂度的减少带来的优化最明显，其次是用移位运算替代算术运算；再次是其它指令。

# 参考文献

[1] 王元珍等.80x86汇编语言程序设计.版本(第1版)

[2] 徐向阳. 80x86汇编语言程序设计上机指南.版本(第1版)