

目录

1	实验目的与要求.....	1
2	实验内容.....	1
3	实验过程.....	2
3.1	任务 1	2
3.1.1	设计思想及存储单元分配	2
3.1.2	流程图	3
3.1.3	源程序	3
3.1.4	实验步骤	6
3.1.5	实验记录与分析	6
3.2	任务 2	10
3.2.1	设计思想及存储单元分配	10
3.2.2	流程图	10
3.2.3	源程序	10
3.2.4	实验步骤	11
3.2.5	实验记录与分析	12
4	总结与体会.....	15
	参考文献	16

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”）。
- (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。
- (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。

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）。

3 实验过程

3.1 任务 1

3.1.1 设计思想及存储单元分配

1) 设计思想

本实验要完成功能三的改写以及对 n 个货物循环 m 次的时间计算，设计思想主要如下：

1. 设计一个二重循环，外层为 m 个客户的循环，内层为 n 个商品推荐度的计算，总共需要的步骤为 $m*n$ 次。

2. 每次循环到要搜索的商品时，即内层循环最后一个元素时，比较已售数量和进货总数，若是前者大于后者则回到功能一，否则将已售数量加一。

3. 利用从实验网站上下载下来的计算时间的子程序 TIMER 实现。函数功能为，当调用函数，使得 BX 置 0 的时候，TS 存储当前时间，当调用程序使得 BX 置 1 的时候，用当前时间减去 TS 时间并输出时间差。时间单位为毫秒，精度也为毫秒级，精确到 1ms。

2) 存储单元分配

BP：外层循环计数器，即客户数量计数器。

DI：内层循环计数器，从商品数量计数器。

SI：用于指示货物在数据段的地址。

AX, BX, CX, DX：用于暂存数据。

AL, BH, BL, BH：用于暂存数据。

汇编语言程序设计实验报告

3.1.2 流程图

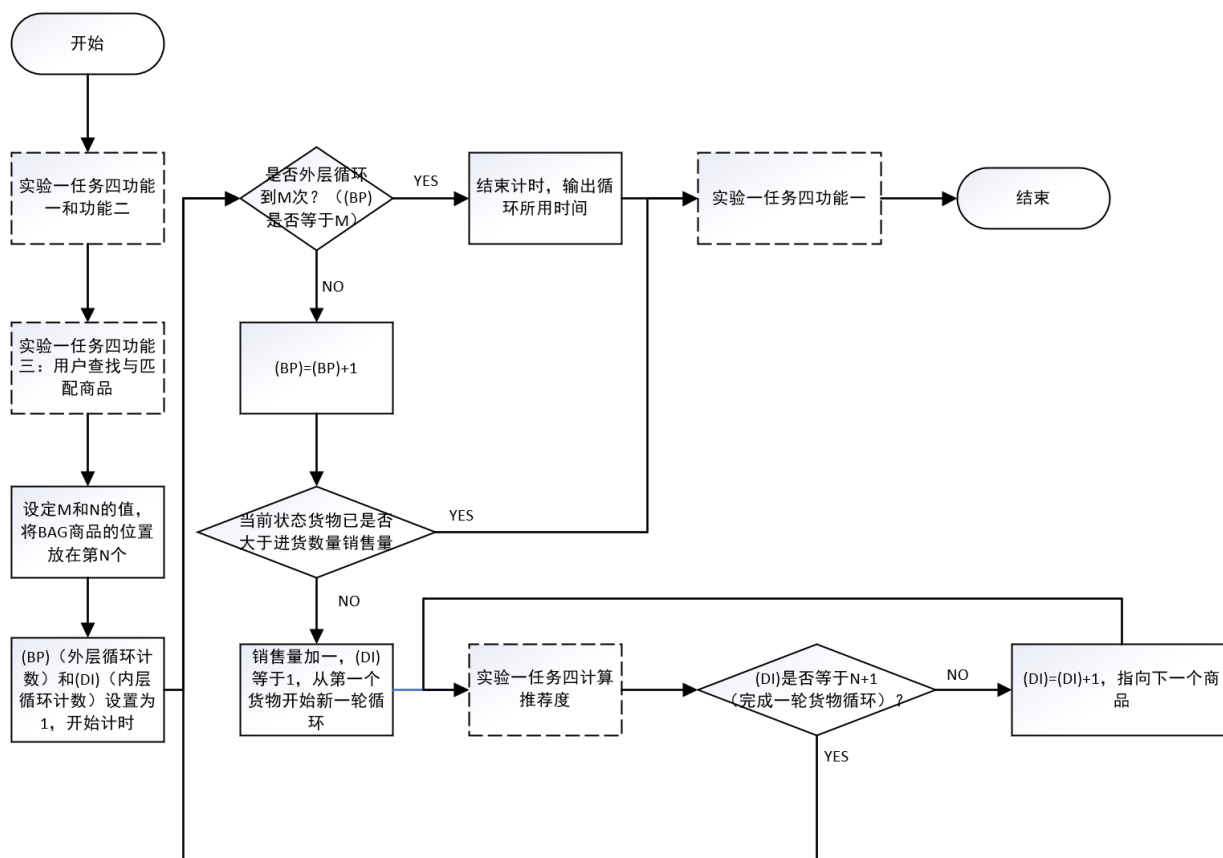


图 3-1 任务一流程图

3.1.3 源程序

```

:
:
ITEMC: ;之前为用户输入货物查找的步骤
MOV DI, 0
CMP AUTH, 1
JE PRINT
MOV BX, 0
MOV SP, DX ;将当前商品的位置存下来
MOV BP, 1 ;设置外层循环计数器, 即客户数量
MOV AX, 0
CALL TIMER ;开始计时
JMP LOPD

LOPD:
CMP BP, M ;比较当前循环次数是否到 M 次
JGE FINISHCUS ;若完成循环, 输出时间并回到功能一
INC BP ;若不是, 执行下一次客户输入
```

汇编语言程序设计实验报告

```
MOV SI, SP ;将货物首地址传到 SI
MOV AX, 0
MOV BX, 0
MOV AX, WORD PTR [SI]+17 ;已经销售数量
MOV BX, WORD PTR [SI]+15 ;货物总量
CMP AX, BX
JGE INNAME ;如果已经销售数量大于货物总量，回到功能一
MOV BX, 0
INC AX ;已经销售数量加一
MOV WORD PTR [SI]+17, AX ;将更新后的销售数量传回数据段
MOV AX, 0
MOV DI, 1 ;内层循环计数器，指示从第一个开始的货物，刷新推荐度
MOV SI, OFFSET GA1 ;从第一个货物开始更新推荐度
JMP FINDSUC
```

FINDSUC:

```
;LCMP AUTH, 1 ;判断登录状态，若是若已经登陆则显示该商品名称，若不是则计算推荐度
;JE PRINT
;MOV SI, DX
MOV AL, [SI+10] ;折扣
MOV AH, 0
MOV BX, [SI+13] ;销售价
MUL BX ;销售价乘以折扣
MOV BX, 10
MOV DX, 0
DIV BX ;销售价乘以折扣除以 10，实际售价
MOV CX, AX
MOV AX, [SI]+11 ;进货价
MOV BX, 128
MUL BX ;进货价乘以 128
MOV DX, 0
DIV CX ;进货价乘以 128 除以销售价
MOV TEMP1, AX
MOV AX, [SI]+17 ;已售数量
MOV BX, 64
MUL BX ;已售数量乘以 64
MOV BX, [SI]+15 ;进货数量
MOV DX, 0
DIV BX ;已售数量乘以 64 除以进货数量
ADD AX, TEMP1 ;两部分推荐度相加，存入 AX
CMP DI, 0
JZ GRADE
CMP DI, N+1 ;比较当前的内层循环次数是否到 N，若没到，继续循环
JGE LOPD ;如果全部货物的推荐度刷新完，循环下一个客户
INC DI ;否则下一个更新下一个商品推荐度
ADD SI, 21
JMP FINDSUC ;计算下一个商品推荐度
```

FINISHCUS:

```
LEA DX, REMINDED
MOV AH, 9H
INT 21H
MOV DL, 0AH;换行符
MOV AH, 2H
INT 21H
MOV AX, 1
CALL TIMER ;结束计时
```

汇编语言程序设计实验报告

```
JMP INNAME ;之后为输出等级的步骤
:
:
;时间计数器(ms), 在屏幕上显示程序的执行时间(ms)
;使用方法:
;      MOV  AX, 0 ;表示开始计时
;      CALL TIMER
;      ...    ;需要计时的程序
;      MOV  AX, 1
;      CALL TIMER ;终止计时并显示计时结果(ms)
;输出: 改变了 AX 和状态寄存器
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
```

汇编语言程序设计实验报告

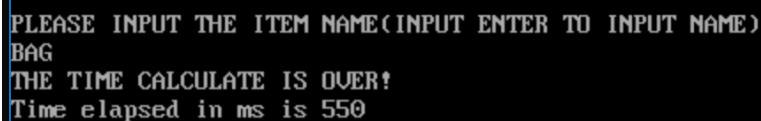
```
POP    DS
JMP     _T0
_TS DW    ?
      DB    'Time elapsed in ms is '
_TMSG DB    12 DUP(0)
TIMER ENDP ;后续为程序退出的代码
      :
```

3.1.4 实验步骤

1. 使用编辑程序 EDIT.EXE 录入源程序，存文件名为 shop1.asm。
2. 汇编源程序，即 MASM shop.asm。
3. 若报错，思考解决出现的报错，可通过查阅手册弄清问题。
4. 链接源程序，即 LINK shop1.obj。
5. 得到 shop1.exe 之后测试其功能是否正常，若不正常找出错误。
6. 测试程序，输入货物名称，观察循环需要的运行时间。
 - a) 测试程序时，在标准环境下（BAG 位置为最后一个，且为非 td 环境），以 M*N 规模测以下几组值 1000*30, 3000*30, 3000*100。
 - b) 测试程序时，改变 BAG 的位置为第一个或者第十个，测试 3000*30 规模的运行时间。
 - c) 改变运行环境，例如在 td 环境下，测试 1000*30, 3000*30, 3000*100 规模的运行时间。
7. 在循环体中插入二号调用或者九号调用，探究其对执行时间的影响, 操作如下：
 - a) 在程序中每一次执行循环体时，用二号调用输出字符 A，比较其运行时间与标准时间。
 - b) 在程序中每一次执行循环体时，用二号调用输出字符串“temp sign”，比较其运行时间与标准时间，以及将其与二号调用时间进行比较。
 - c) 比较二者时间，给出运用二号调用、九号调用对程序执行时间影响的结论。

3.1.5 实验记录与分析

1. 汇编源程序和连接的过程没有出现异常
2. 程序功能测试
 - 1) 标准状况测试(BAG 位置在最后一个，环境为非 td 环境)
 - a) M*N 规模为 1000*30 的循环体测试(单位是毫秒)



```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 550
```

图 3-2 规模为 1000*30 的循环体的测试

可以看出，运行时间为 550ms。

汇编语言程序设计实验报告

b) M*N 规模为 3000*30 的循环体测试(单位是毫秒)

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1540
```

图 3-3 规模为 3000*30 的循环体的测试

可以看出,运行时间为 1540ms。

c) M*N 规模为 3000*100 的循环体测试(单位是毫秒)

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 5220
```

图 3-4 规模为 3000*100 的循环体的测试

可以看出,运行时间为 5520ms。

2) 将 BAG 的位置放在第一个,测试 3000*30 规模的运行时间

将 BAG 放在数据段 GA1 的位置测试,重新编译连接并运行,截图如下:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1260
```

图 3-5 将 BAG 位置放在第一个测试

可以看出,运行时间为 1260ms。

我们可以看出,把 BAG 的位置从第 30 个放到第 1 个,时间有减少,但是也只减少了 1/3 (从 1540ms 减少到 1050ms),所以我们可以看出,在循环过程中,搜索占的时间比例不是很高,而时间主要消耗在刷新推荐度。

3) 在 td 环境下,测试运行时间

运用 td 运行,直接输入 Ctrl+F9 执行。我们会发现,在标准情况下 cpu speed 的值为 3000 cycles,而在 td 调试的情况下 cpu speed 的值为 max 100% cycles,如下图所示:

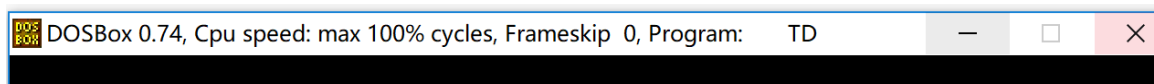


图 3-6 td 运行情况下 cpu speed 的值

下面我们在这个环境下测试 1000*30, 3000*30, 3000*100 规模的运行时间:

a) 1000*30 的样例测试

汇编语言程序设计实验报告

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 50
```

图 3-7 td 环境下 1000*30 样例测试

可以看出，运行时间为 50ms。

b) 3000*30 样例测试:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 110
```

图 3-8 td 环境下 3000*30 样例测试

可以看出，运行时间为 110ms。

c) 3000*100 样例测试:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 390
```

图 3-9 td 环境下 3000*100 样例测试

可以看出，运行时间为 390ms。

我们可以看出，在 td 环境下，程序运行时间大成都减少，因为在 td 条件下程序的 cpu speed 为 max 100% cycles，故程序拥有更快的运行速度。

3. 编写程序验证如果循环体中每一次都有信息显示的代码（2 号或 9 号调用），对程序的影响

1) 二号调用测试

我们在 3000*30 (m*n) 的循环中，每一次都加入输出字符，采用二号调用，将代码加入到内层循环的末尾，即程序要输出 3000*30 次该字符，执行后的程序截图以及耗费时间如下（注：原程序的运行时间为 1540ms）：

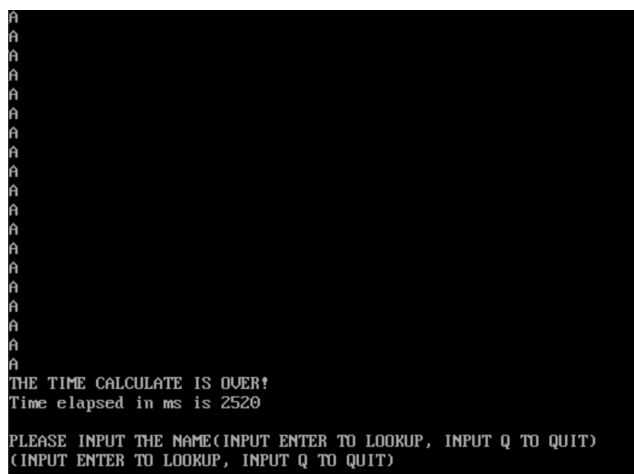


图 3-10 循环体中有 2 号调用结果截图

可以看出, 程序运行时间为 2520ms, 比原来的运行时间 1540ms 多 63.6%。

2) 九号调用测试

我们在 3000*30(m*n)的循环中，每一次都加入输出的字符串“temp sign”，采用九号调用，将代码加入到内层循环的末尾，即程序要输出 3000*30 次该字符串，执行后的程序截图以及耗费时间如下（注：原程序的运行时间为 1540ms）：

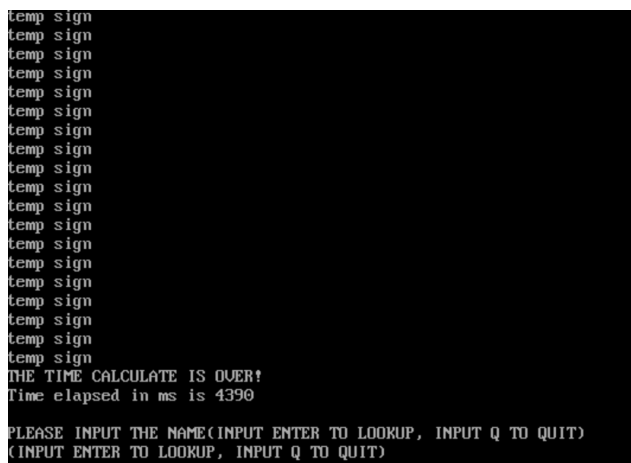


图 3-11 循环体中有 9 号调用结果截图

可以看出, 程序运行时间为 4390ms, 比原来的运行时间 1540ms 多 183%。

3) 循环体中插入二号调用或者九号调用对程序循环运行时间的影响总结

1. 程序显示字符串（字符）需要的时间实际上是不可忽略的，而且在运行时占有着相当的时间，无论是再循环体中插入二号调用（插入字符）还是插入 9 号调用（插入字符串），多会对程序运行时间产生显著性的提高，其中插入字符是程序运行时间多出大约 60%，插入字符串对程序运行时间多出大约 180%。

2. 循环体中插入 9 号调用(插入字符串)所用的时间明显大于插入 2 号调用(插入字符), 这证明程序输出的运行时间和输出的串长度有着直接关系, 输出的串越长, 程序需要的时间越多。

汇编语言程序设计实验报告

3.2 任务 2

3.2.1 设计思想及存储单元分配

1) 设计思想

本任务要做到的是优化程序，减少循环体执行的时间，程序设计思想如下：

1. 在任务一中，我们推荐度计算时我们用的都是 MUL 乘法，而对于乘以 2 的倍数来说，移位具有更高的效率，所以我们将乘以 2 的倍数指令全部改为左移位指令，在计算的时候有时候也会有多余的移动指令，将其去除。

2. 在任务一的代码中，我们有许多的置零指令是不必要的，我们将其删除，任务一的代码也有许多重复的赋值与移动指令，将其调整删除。

2) 存储单元分配

BP: 外层循环计数器，即客户数量计数器

DI: 内层循环计数器，从商品数量计数器

SI, SP: 用于存储地址

其他通用寄存器用于暂时存储数据

3.2.2 流程图

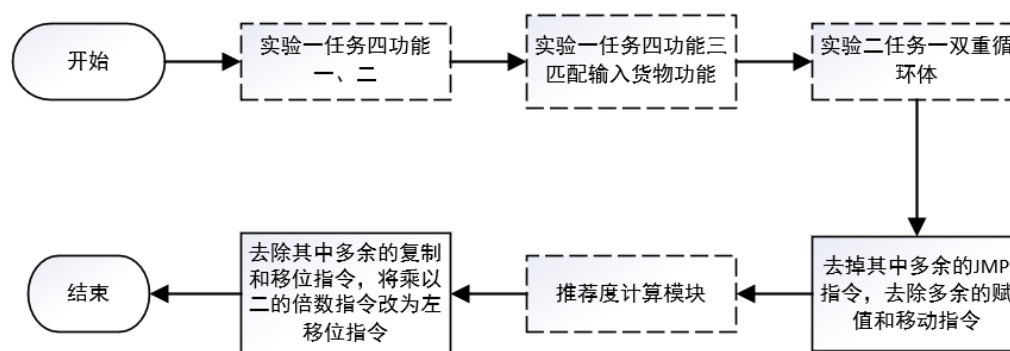


图 3-12 任务二流程图

3.2.3 源程序

```

      :
      :
ITEMC: ;前面为判断输入货物是或否匹配
      MOV DI, 0
      CMP AUTH, 1
      JE PRINT
```

汇编语言程序设计实验报告

```
MOV BX, 0
MOV SP, DX ;将当前商品的位置存下来
MOV BP, 1 ;设置外层循环计数器, 即客户数量
MOV AX, 0
CALL TIMER ;开始计时
```

LOPD:

```
CMP BP, M ;比较当前循环次数是否到 M 次
JGE FINISHCUS ;若完成循环, 输出时间并回到功能一
INC BP ;若不是, 执行下一次客户输入
MOV SI, SP ;将货物首地址传到 SI
MOV AX, WORD PTR [SI]+17 ;已经销售数量
MOV BX, WORD PTR [SI]+15 ;货物总量
CMP AX, BX
JGE INNAME ;如果已经销售数量大于货物总量, 回到功能一
INC AX ;已经销售数量加一
MOV WORD PTR [SI]+17, AX ;将更新后的销售数量传回数据段
MOV DI, 1 ;内层循环计数器, 指示从第一个开始的货物, 刷新推荐度
MOV SI, OFFSET GA1 ;从第一个货物开始更新推荐度
```

FINDSUC:

```
MOV AL, [SI+10] ;折扣
MOV AH, 0
MOV BX, [SI+13] ;销售价
MUL BX ;销售价乘以折扣
MOV BX, 10
MOV DX, 0
DIV BX ;销售价乘以折扣除以 10, 实际售价
MOV CX, AX
MOV AX, [SI]+11 ;进货价
SAL AX, 7 ;左移 7 位, 乘以 128
MOV DX, 0
DIV CX ;进货价乘以 128 除以销售价
MOV TEMP1, AX
MOV AX, [SI]+17 ;已售数量
SAL AX, 6 ;左移 6 位, 乘以 64
MOV BX, [SI]+15 ;进货数量
MOV DX, 0
DIV BX ;已售数量乘以 64 除以进货数量
ADD AX, TEMP1 ;两部分推荐度相加, 存入 AX
CMP DI, 0 ;后面接等级判断
⋮
⋮
```

3.2.4 实验步骤

1. 使用编辑程序 EDIT.EXE 录入源程序, 存文件名为 shop2.asm。
2. 汇编源程序, 即 MASM shop2.asm。
3. 若报错, 思考解决出现的报错, 可通过查阅手册弄清问题。
4. 链接源程序, 即 LINK shop2.obj。
5. 得到 shop2.exe 之后测试其功能是否正常, 若不正常找出错误
6. 测试程序, 观察循环所需要的时间, 和任务一中的程序进行对比, 算出优化率。这里要测

汇编语言程序设计实验报告

试的例子与任务一的实验对应，并算出优化率，具体如下：

a) 测试程序时，在标准环境下（BAG 位置为最后一个，且为非 td 环境），以 M*N 规模测以下几组值 1000*30, 3000*30, 3000*100。

b) 测试程序时，改变 BAG 的位置为第一个或者第十个，测试 3000*30 规模的运行时间

c) 改变运行环境，例如在 td 环境下，测试 1000*30, 3000*30, 3000*100 规模的运行时间。

7. 编写程序，测试以下几种情况：

a) 仅仅将 MUL 改为左移，看时间能优化多少。

b) 仅仅去掉程序中 3 个不必要的 JMP 指令，看时间能优化多少。

c) 仅仅去掉程序中有一些不必要的置 0 的指令，将去掉一些重复的移位指令。即缩短程序长度，看时间能优化多少。

上述程序在 3000*30 的规模且在标准环境下执行（BAG 位置为最后一个，且为非 td 环境

最后总结不同类别的优化措施对效率的影响程度。

3.2.5 实验记录与分析

1. 汇编源程序和连接的过程没有出现异常

2. 程序功能测试

1) 标准状况优化测试(BAG 位置在最后一个，环境为非 td 环境)

a) 规模为 1000*30 的循环体优化（单位为 ms）

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 490
```

图 3-13 规模 1000*30 的循环体优化结果如下

对比上一节中 1000*30 的循环体运行时间为 550ms，可以算出优化率为
 $(550-490)/550*100\%=10.9\%$

b) 规模为 3000*30 的循环体优化（单位为 ms）

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1370
```

图 3-14 规模 3000*30 的循环体优化结果如下

对比上一节中 3000*30 的循环体运行时间为 1540ms，可以算出优化率为
 $(1540-1370)/1540*100\%=11.03\%$

汇编语言程序设计实验报告

c) 规模为 3000*100 的循环体优化 (单位为 ms):、

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 4610
```

图 3-15 规模 3000*100 的循环体优化结果如下

对比上一节中 3000*100 的循环体运行时间为 5220ms, 可以算出优化率为:
 $(5220-4610)/5220*100\%=11.7\%$

2) BAG 位置在第一个, 规模为 3000*30 的优化测试

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1150
```

图 3-16 BAG 位置在第一个, 规模为 3000*30 优化测试

对比上一节中 BAG 在第一个位置运行时间为 1260ms, 可以算出优化率为:
 $(1260-1150)/1260*100\%=8.7\%$

3) td 环境下, 优化测试

a) td 环境下 1000*30 规模的优化测试:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 30
```

图 3-17 td 环境下 1000*30 规模优化测试

对比上一节中 td 下运行 1000*30 的规模的时间 50ms, 可以算出优化率为:
 $(50-30)/50*100\%=20\%$

b) td 环境下 3000*30 规模的优化测试:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 100
```

图 3-18 td 环境下 3000*30 规模优化测试

对比上一节中 td 下运行 1000*30 的规模的时间为 110ms, 可以算出优化率为:
 $(110-100)/110*100\%=9.1\%$

汇编语言程序设计实验报告

c) td 环境下 3000*100 规模的优化测试:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 350
```

图 3-19 td 环境下 3000*100 规模优化测试

对比上一节中 td 下运行 1000*30 的规模的时间为 390ms, 可以算出优化率为:
 $(390-350)/390*100\%=10.3\%$

通过以上情况的测试, 对比没有优化前的程序运行时间, 发现大多数情况下, 程序运行时间都能减少 10%甚至更多, 可见优化效果达到标准。

3. 测试不同优化情况对于程序执行时间的影响

注意: 以下程序均在非 td 环境下, 3000*30 的规模下进行。原程序运行时间为 1540ms, 而将以下所列出的优化项目全部解决运行时间为 1370ms, 优化时间为 170ms。

1) 仅仅将推荐度计算中的 MUL 指令改为 SHL (左移) 指令。

我们将乘以 64 改为左移六位, 乘以 128 改为左移 7 位, 得到执行时间截图如下:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1480
```

图 3-20 仅改变 MUL 为左移指令运行截图

可以看出优化时间为 80ms, 占总优化比例为 $60/170*100\%=35.3\%$ 。

2) 仅仅去掉程序中多余的三个 JMP 指令。

因为程序为顺序执行的, 所以有些 JMP 指令是不必要的, 我们将其去除, 得到执行时间截图如下:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1530
```

图 3-21 仅仅去掉不必要跳转指令运行截图

可以看出程序优化时间为 10ms, 占总优化比例为 $10/170*100\%=5.9\%$

3) 去掉多余的置 0 指令以及重复赋值指令

去掉程序中多余的置零指令以及重复赋值指令 (不必要的代码), 一共 5 条指令, 得到执行时间截图如下:

```
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
BAG
THE TIME CALCULATE IS OVER!
Time elapsed in ms is 1450
```

图 3-22 进去掉多余置零指令与重复赋值指令运行截图

汇编语言程序设计实验报告

可以看出程序优化时间为 90ms，占总优化比例为 $90/170 \times 100\% = 52.9\%$

而且看出还剩下 10ms 的优化时间没给出，可能和当前系统状况（例如插电情况）有关，和程序每次编译运行的情况有关，10ms 的误差应该属于可接受范围。

4) 不同优化情况对于程序执行时间的影响的总结

1. 可以看出，对整个程序优化影响最大的因素是去掉多余的置零指令与重复赋值指令。这些指令都是 MOV 指令，去掉这些指令一定程度上减少了代码的长度，减少了执行指令的数量，在每一次循环中执行的命令都有减少，优化程度也较大。

2. 将 MUL 改为左移也是一种十分有效的减少运行时间的方法，移位指令比乘法指令拥有更快的运行速度，从一定程度上会减少程序的运行时间。

3. 去掉多余的 JMP 指令也可以减少一定的运行时间，但是收效很小，仅仅是在执行语句的时候少执行一句话，而且可以看出程序本身在执行 JMP 指令的时候肯定是十分迅速的，所以去掉了 JMP 指令程序缩短的时间没有特别长的缩短。

4 总结与体会

通过任务一的实验，我们对于怎样调用子程序有了实践性的认识，同时，任务一的二重循环加深了我们对于循环体的理解与设计的能力。任务一还让我们在不同的条件情况下执行同一个程序，观察其消耗的时间，我认识到，在 td 和非 td 条件下执行同一个程序所消耗的时间是有很大差别的。正常执行的情况下，cpu speed 是 3000 cycles，而在 td 执行的情况下，cpu speed 是 max 100% cycles。通过查阅 dosbox 的 option 文件我发现他对于 max 的定义是 will allocate as much cycles as your computer is able to handle，即分配给这个程序你的电脑所能承载的 cpu 速度的最大值。这样就能够解释为什么我在 td 调试的情况下程序运行的速度会比非 td 调试的情况下快速得多。

任务二中我主要对之前的程序进行了优化，在优化的过程中，我发现之前有许多指令是不需要的，是没有意义的赋值和跳转，我将这些语句进行了改动或者删除，同时意识到移位比乘法指令拥有更高的效率，将一些 2 的指数次方的乘法指令改成了移位指令。同时，我将这几种指令的优化程度进行了对比，发现 JMP 指令的执行所需要的时间很短，对于程序优化的时间占比很小，而去掉了一些不必要的重复赋值指令之后，能使每次循环执行的语句一定程度的减少，程序能执行的更快，这部分程序优化的更好。而将 MUL 指令改为移位指令之后，程序运行时间也能一定程度减少，证明移位指令比乘法指令拥有更高的执行速度。

由于这次程序是运用实验一任务四的程序进行改进，程序主体还是实验一的程序，所以我在实验中出现的主要问题是忽略了实验一的寄存器的分配，实验一有些寄存器是不能够被覆盖的，但是我在实验二中忽略了这个问题，重新分配了寄存器，导致有些关键值被覆盖，所以最后出现很多程序跑不出来的问题。这也就提醒我以后在更改程序的时候（给程序添加功能的时候），一定要注意程序寄存器的分配，不能出现在编写新功能的时候覆盖了原程序关键的寄存器内容的情况。

汇 编 语 言 程 序 设 计 实 验 报 告

参考文献

- [1] 王元珍 曹忠升 韩宗芬 编著. 80X86 汇编语言程序设计. 武汉: 华中科技大学出版社, 2005 年 4 月. 98 页-121 页
- [2] URL: <https://www.cnblogs.com/TaigaCon/p/7455443.html>