

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验三** 程序执行时间与代码长度优化

**实验时间： 2016-4-5，14：30-17：30 实验地点： 南一楼804室102号实验台**

**指导教师： 张勇**

**专业班级：计算机科学与技术201409班**

**学 号： U2014414808 姓 名： 王林**

**同组学生： 无 报告日期： 2016年 4 月 5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： 王林

日期： 2016.4.5

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](#_Toc447898637)

[实验过程 1](#_Toc447898638)

[3.1任务1 1](#_Toc447898639)

[3.1.1设计思想及存储单元分配 1](#_Toc447898640)

[3.1.2流程图 2](#_Toc447898641)

[3.1.3源程序 2](#_Toc447898642)

[3.1.4实验步骤 6](#_Toc447898643)

[3.1.5实验记录 6](#_Toc447898644)

[3.2任务2 6](#_Toc447898645)

[3.2.1设计思想及存储单元分配 6](#_Toc447898646)

[3.2.2流程图 7](#_Toc447898647)

[3.2.3源程序 7](#_Toc447898648)

[3.2.4实验步骤 11](#_Toc447898649)

[3.2.5 实验记录 11](#_Toc447898650)

[3.3 任务3 12](#_Toc447898651)

[3.3.1设计思想及存储单元分配 12](#_Toc447898652)

[3.3.2 流程图 12](#_Toc447898653)

[3.3.3 源程序 12](#_Toc447898654)

[3.3.4 实验步骤 15](#_Toc447898655)

[3.3.5 实验记录 15](#_Toc447898656)

[体会 17](#_Toc447898657)

[参考文献 18](#_Toc447898658)

实验目的与要求

(1) 熟悉汇编语言指令的特点，掌握代码优化的基本方法；

(2)理解高级语言程序与汇编语言程序之间的对应关系。

实验内容

任务1. 观察多重循环对CPU计算能力消耗的影响

若有m个用户在同一台电脑上排队使用实验二的程序，想要查询“ZHANGSAN\_C”这个学生的平均成绩，那就相当于将实验二的程序执行了m次。为了观察从第一个用户开始进入查询至第m个用户查到结果之间到底延迟了多少时间，我们让实验二的功能二和功能三的代码重复执行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。由于功能一和功能四需要输入、输出，速度本来就较慢，所以，没有纳入到这m次循环体内（但可以保留不变）。请按照上述设想修改实验二的程序，并将m值尽量取大（建议m>=1000），以得到较明显的效果。

提示: 在进入功能二之前增加m次循环的初始化工作，在功能三结束之后增加m次循环的条件判断和转移语句。

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）。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。

任务3. 观察用C语言实现的任务1的程序与汇编语言实现的程序的差异。

实验过程

3.1任务1

任务1. 观察多重循环对CPU计算能力消耗的影响

3.1.1设计思想及存储单元分配

在功能2和功能3之前加一个外层循环，循环m次，用于模拟进行了大规模的m个用户进行查询事件。同时在头尾添加时间计算子程序，用于计算程序执行时间。

3.1.2流程图



**图 3-1 任务一流程图**

3.1.3源程序

;lab-3-old

;source code

;@author Strawberrylin

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

BUF DB 15

DB ?

DB 15 DUP(0)

STU DB 997 DUP('zhangsan\_x',0,0,0,0)

DB 'ZHANGSAN\_A',100,90,90,0

DB 'ZHANGSAN\_B',80,70,60,0

DB 'ZHANGSAN\_C',0,10,40,0

BUF1 DB 'Input Student Name:$'

BUF2 DB 'No This Student...$'

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

LEA DX, BUF1

MOV AH, 9 ;9TH FUNCTION CALLS

INT 21H

LEA DX, BUF ;INPUT STUDENT NAME

MOV AH, 10 ;10TH FUNCTION CALLS

INT 21H

MOV EBP, 100000

MOV AX, 0

CALL TIMER

LOAP:

MOV DX, 1000

LEA DI, STU

LOAPI:

JMP FIGURE

CON:

LEA SI, BUF+2

MOV CX, 10

LOAPJ:

MOV AL, [SI]

MOV BL, [DI]

CMP AL, BL

JNZ NEXT

DEC CX

JZ EXIT

INC DI

INC SI

JMP LOAPJ

NEXT:

DEC DX

JZ REINPUT

ADD CX, 4

ADD DI, CX

JMP LOAPI

FIGURE:

MOV AL, [DI+10]

MOV AH, 0

IMUL AX, 4

MOV BL, [DI+11]

MOV BH, 0

IMUL BX, 2

ADD AX, BX

MOV BL, [DI+12]

MOV BH, 0

ADD AX, BX

MOV BH, 0

MOV BL, 7

DIV BL

MOV [DI+13], AL

JMP CON

REINPUT:

LEA DX, BUF2

MOV AH, 9

INT 21H

JMP START

EXIT:

DEC EBP

JNZ LOAP

MOV AX, 1

CALL TIMER

MOV AH, 4CH

INT 21H

;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

POP DS

JMP \_T0

\_TS DW ?

DB 'Time elapsed in ms is '

\_TMSG DB 12 DUP(0)

TIMER ENDP

CODE ENDS

END START3.1.4

3.1.4实验步骤

1、按照实验要求修改代码，并且添加子程序调用不能分代码。

2、编译、连接、输出可执行程序

3、执行lab-3-old.exe文件,屏幕直接输出程序执行时间。

3.1.5实验记录

1、实验环境条件：P3 1GHz，256M内存；WINDOWS XP命令行窗口；Sublime Text 2；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。

2、通过实验发现，在DOS上程序每次执行 相差不大，而在Windows和Ubuntu上出现每次执行时间波动较为明显。

3、少量的改动程序循环次数对CPU资源消耗影响不大

|  |  |  |  |
| --- | --- | --- | --- |
| 次数 | 1 | 2 | 3 |
| 时间(ms) | 1040 | 990 | 1040 |
| 截图 |  |  |  |

**表 3-1 任务一记录表**

3.2任务2

对任务1中的汇编源程序进行优化。

3.2.1设计思想及存储单元分配

对以前的源程序进行优化，是执行时间减少，代码精简。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.然后删除冗余代码，减少程序执行时间。

3.2.2流程图



**图 3-2 任务二流程图**

3.2.3源程序

;lab-3-new

;source code

;@author Strawberrylin

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

BUF DB 15

DB ?

DB 15 DUP(0)

STU DB 997 DUP('zhangsan\_x',0,0,0,0)

DB 'ZHANGSAN\_A',100,90,90,0

DB 'ZHANGSAN\_B',80,70,60,0

DB 'ZHANGSAN\_C',0,10,40,0

BUF1 DB 'Input Student Name:$'

BUF2 DB 'No This Student...$'

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

LEA DX, BUF1

MOV AH, 9 ;9TH FUNCTION CALLS

INT 21H

LEA DX, BUF ;INPUT STUDENT NAME

MOV AH, 10 ;10TH FUNCTION CALLS

INT 21H

MOV EBP, 100000

MOV AX, 0

CALL TIMER

LOAP:

MOV DX, 1000

LEA DI, STU

LOAPI:

JMP FIGURE

CON:

LEA SI, BUF+2

MOV CX, 10

LOAPJ:

MOV AL, [SI]

MOV BL, [DI]

CMP AL, BL

JNZ NEXT

DEC CX

JZ EXIT

INC DI

INC SI

JMP LOAPJ

NEXT:

DEC DX

JZ REINPUT

ADD CX, 4

ADD DI, CX

JMP LOAPI

FIGURE:

MOV AL, [DI+10]

MOV AH, 0

SAL AX, 2

MOV BL, [DI+11]

MOV BH, 0

SAL BX, 1

ADD AX, BX

MOV BL, [DI+12]

MOV BH, 0

ADD AX, BX

MOV BH, 0

MOV BL, 7

DIV BL

MOV [DI+13], AL

JMP CON

REINPUT:

LEA DX, BUF2

MOV AH, 9

INT 21H

JMP START

EXIT:

DEC EBP

JNZ LOAP

MOV AX, 1

CALL TIMER

MOV AH, 4CH

INT 21H

;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

POP DS

JMP \_T0

\_TS DW ?

DB 'Time elapsed in ms is '

\_TMSG DB 12 DUP(0)

TIMER ENDP

CODE ENDS

END START

3.2.4实验步骤

1、对3.1的程序进行修改：对计算平均成绩部分将乘法换成位移运算，删除冗余代码。

2、在姓名匹配部分，通过测试字符串指令和循环的出最佳方法。

3、执行lab-3old.exe文件,屏幕直接输出程序执行时间

3.2.5 实验记录

1、实验环境条件：P3 1GHz，256M内存；WINDOWS XP命令行窗口；Sublime Text 2；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。

2、汇编语言可以通过减少代码、采用时间少的指令替换，比如 用加法和位移替换乘法

3、关键指令在于循环内部语句，每一条语句通过循环后都有很大影响。其他关键语句如乘法指令，比较慢。

4、优化后出现执行效率明显提高。

5、字符串操纵指令对程序影响较大。

|  |  |  |  |
| --- | --- | --- | --- |
| 次数 | 1 | 2 | 3 |
| Old |  |  |  |
| New |  |  |  |
| 优化比 | 15.38% | 12.15% | 15.38% |

**表 3-2 任务二记录表**

5、在实验中发现将比较姓名字符串的内循环改为字符串比较指令后，程序运行时间反而变长。



修改为字符串比较指令后时间变长

3.3 任务3

观察用C语言实现的任务1的程序与汇编语言实现的程序的差异。

3.3.1设计思想及存储单元分配

阅读.c源程序，然后通过TD工具反汇编对应的exe文件，与自己写的汇编代码作比较。通过屏幕输出的Average函数的位置，用TD等工具反汇编执行程序ex3D.exe和ex3F.exe，找到对应函数调用入口，并且进入函数内部，观察对应的反汇编代码。

3.3.2 流程图

无

3.3.3 源程序

【.C源程序】

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <time.h>

/\*\*

N EQU 1000

BUF DB 'zhangsan', 0, 0, 100, 85, 80, ?

DB 'lisi', 6 DUP(0), 80, 100,70, ?

DB N-4 DUP( 'TempValue',0,80,90,95,?)

DB 'wangwu', 4 dup(0), 70, 60, 80, 0

DB 'xuxiaohua', 0, 40, 55, 61, 0

\*\*/

#define LEN 14 /\*\* 每个学生成绩表的长度 \*\*/

#define N 1000 /\*\* 学生人数 \*\*/

unsigned char buf[N\*LEN]; /\*\* 学生成绩缓冲区 \*\*/

unsigned char \*POIN; /\*\* 用于保存查找到的学生的成绩表起始地址 \*\*/

#define COUNT 1000 /\*\* 重复计算的次数 \*\*/

short int Average();

short int Search(char \*name);

int main(int argc, char \*argv[])

{

char name[10] = "wangwu"; /\*\* 需要查找学生的姓名 \*\*/

unsigned long c, addr;

int f;

clock\_t start, end;

/\*\*/

strcpy((char \*)(buf+0\*LEN),"zhangsan"); /\*\* 第1个学生 \*\*/

buf[0\*LEN+10] = 100;

buf[0\*LEN+11] = 85;

buf[0\*LEN+12] = 80;

/\*\*/

strcpy((char \*)(buf+(N-2)\*LEN),"wangwu"); /\*\* 倒数第2个学生 \*\*/

buf[(N-2)\*LEN+10] = 70;

buf[(N-2)\*LEN+11] = 60;

buf[(N-2)\*LEN+12] = 80;

/\*\*/

/\*\* 用于反汇编定位函数Average(),显示的值即为Average()的入口 \*\*/

addr = (unsigned long)(&Average);

printf("Average() address = %x \n",addr);

/\*\*/

start = clock();

for(c = 0; c < COUNT; c++)

{

Average();

f = Search(name);

}

end = clock();

printf("The time was: %dms\n", (end - start) \* 55);

/\*\*/

return( 0 );

}

/\*\*

功能: 计算所有学生的平均成绩((2\*语文A+数学B+英语C/2)/3.5)

参数: buf,N --- 学生成绩缓冲区和学生人数（2个都是全局变量）

返回: 无

\*\*/

short int Average()

{

int k;

unsigned short A,B,C,M;

for(k = 0; k < N; k++)

{

A = buf[k\*LEN+10];

B = buf[k\*LEN+11];

C = buf[k\*LEN+12];

M = (A \* 4 + B \* 2 + C) / 7;

buf[k\*LEN+13] = (unsigned char)M;

}

return( 0 );

}

/\*\*

功能: 在学生成绩表中查找某个学生的平均成绩

参数: buf,N --- 学生成绩缓冲区和学生人数（2个都是全局变量）

name ---- 需要查找的学生姓名

返回: 若找到,则返回1并且将该学生成绩表的起始地址保存到POIN中

否则,返回0

\*\*/

short int Search(char \*name)

{

int i,k;

unsigned char \*p;

int result = 0;

/\*\*/

for(k = 0; k < N; k++)

{

p = buf + k \* LEN;

for(i = 0; p[i] == name[i] && name[i] != 0 && i < 10; i++);

if( p[i] == name[i] && name[i] == 0 )

{

POIN = p;

result = 1;

break;

}

}

/\*\*/

return( result );

}

3.3.4 实验步骤

1、阅读C语言源程序。

2、TD反汇编。

3、观察、对比、记录。

3.3.5 实验记录

1、实验环境条件：P3 1GHz，256M内存；WINDOWS XP命令行窗口；Sublime Text 2；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。

2、对ex3D.exe反汇编，找到average函数入口如下

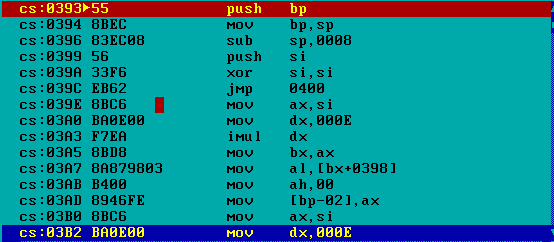
-average:--033D call 0393

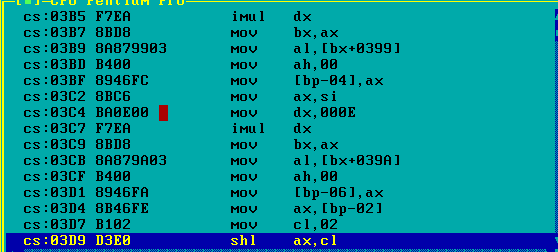


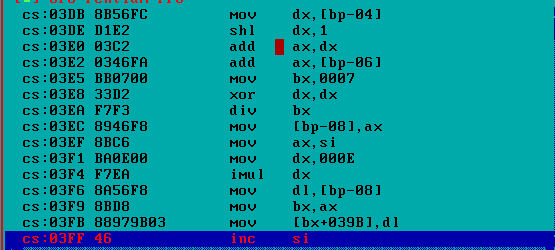
3、进入average函数内部，得到对应average函数开始部分：

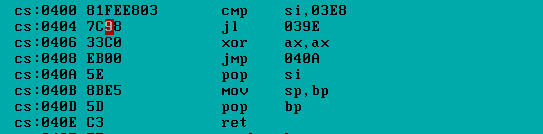


对应的反汇编代码如下：







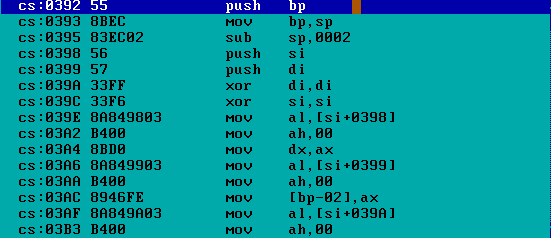


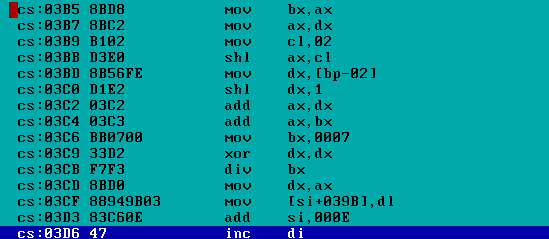
以上为直接通过TD反汇编后的截图，显示的为average部分的反汇编代码。可以发现代码部分比较长，对公式的计算通过将乘法转化为加法来优化，但是出现中还是有很多地方用到了乘法，导致程序称象运行时间变长。

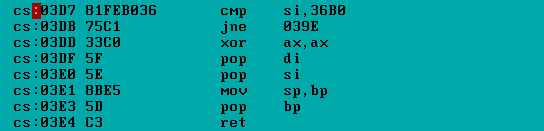
4、对ex3F.exe反汇编，找到average函数入口：  
 --033D call 0392



5、进入average函数内部，的到对应反汇编代码：







以上为直接通过TD反汇编后的截图，显示的为average部分的反汇编代码。显示的汇编代码为优化后的代码，通过对比，可以明显发现代码长度降低，并且所有的乘法运算都转化为了移位和加法指令，从而大大增加了程序运行效率。

5、通过对比，C程序反汇编生成汇编代码时，长度明显增加，但是汇编执行的效率却明显比c程序高。

6、c语言优化的关键代码在于代码的长度以及对乘法指令以及其他消耗时间的指令

体会

通过本次上机，更加熟悉了对TD工具的使用，通过对其他EXE文件的反汇编，观察反汇编代码。

本次上机的具体收获有：（1）通过对C程序的反汇编，熟悉程序转化为汇编程序的机理。（2）通过对汇编代码和C代码的对比，了解两种语言的特性，C语言比汇编代码简介，但是执行效率低，汇编代码的执行效率较高。

同时，对程序的执行机理还不够熟悉，需要进一步学习。

# 参考文献

[序号] 作者.书名.版本(第×版).译者.出版地:出版者,出版年：起页-止页

[序号] URL: 网络地址，如URL:http://www.cbs.dtu.dk/services