

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验时段： 2018/04/09下午**

**指导教师： 左 琼**

**专业班级： CS1603**

**学 号： U201614577**

**姓 名： 龙 际 全**

实验报告成绩

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实验 序号 | 一 | 二 | 三 | 四 | 五 | 六 | 总评 |
| 成绩 |  |  |  |  |  |  |  |

指导教师签字：

                     日 期：

**计算机科学与技术学院**

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验一 程序执行时间与代码长度优化**

**实验时间： 2018.04.09 实验地点：南一楼804\_3实验室79试验台**

**指导教师： 左琼**

**专业班级：计科 201603班**

**学 号： U201614577 姓 名： 龙际全**

**同组学生： 无 报告日期： 2018年 04 月 09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

[1 实验目的与要求 1](#_Toc511219739)

[2 实验内容 1](#_Toc511219740)

[3 实验过程 2](#_Toc511219741)

[3.1任务1 2](#_Toc511219742)

[3.1.1源程序 2](#_Toc511219743)

[3.1.2实验步骤 7](#_Toc511219744)

[3.1.3实验记录与分析 8](#_Toc511219745)

[3.2任务2 9](#_Toc511219746)

[3.2.1任务1程序优化分析 9](#_Toc511219747)

[3.2.2实验步骤 10](#_Toc511219748)

[3.2.3实验记录与分析 10](#_Toc511219749)

[4 总结与体会 11](#_Toc511219750)

[5 注 12](#_Toc511219751)

[参考文献 13](#_Toc511219752)

# 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

## 3.1任务1

### 3.1.1源程序

.386

;--------------------------------------------------

DATA SEGMENT USE16

in\_name DB 11

DB ?

DB 11 DUP(0)

in\_pwd DB 7

DB ?

DB 7 DUP(0)

goods\_name DB 11

DB ?

DB 11 DUP(0)

APR DW 0

AUTH DB 0

COST1 DW 0;??

PROFIT1 DW 0;??

COST2 DW 0;??

PROFIT2 DW 0;??

cycle\_times dw 1000

BNAME DB 'LONG JQ',3 DUP(0);????

BPASS DB 'NOPASS';??

N EQU 30

goods\_offset dw 0

SHOP1 DB 'SHOP1',0;?????0??

ga1 DB 'PEN',7 DUP(0)

DW 35,56,30000,0,?

ga2 DB 'BOOK',6 DUP(0)

DW 12,30,30000,0,?

gaN DB N-2 DUP('TEMP-VALUE',15,0,20,0,30H,75H,0,0,?,?)

SHOP2 DB 'SHOP2',0;?????0??

gb1 DB 'PEN',7 DUP(0)

DW 35,50,30000,0,?

gb2 DB 'BOOK',6 DUP(0)

DW 12,28,30000,0,?

gbN DB N-2 DUP('TEMP-VALUE',15,0,20,0,30H,75H,0,0,?,?)

INPUT\_NAME\_MSG DB 0AH,0DH,'please input name(input q/Q to exit):$'

INPUT\_PASS\_MSG DB 0AH,0DH,'please input password:$'

LOGIN\_FAILED\_MSG DB 0AH,0DH,'Login failed! Please check the name or the password!$'

END\_MSG DB 0AH,0DH,'end of program, press any key to continue...$'

INPUT\_GOODS\_NAME\_MSG DB 0AH,0DH,'please input the name of the goods:$'

INPUT\_GOODS\_NAME\_AgaIN DB 0AH,0DH,'goods name input error,please input again:$'

goods\_sold\_out\_msg db 0ah,0dh,'sorry, the goods has sold out!$'

GRADE\_A\_MSG DB 0AH,0DH,'A$'

GRADE\_B\_MSG DB 0AH,0DH,'B$'

GRADE\_C\_MSG DB 0AH,0DH,'C$'

GRADE\_D\_MSG DB 0AH,0DH,'D$'

GRADE\_F\_MSG DB 0AH,0DH,'F$'

DATA ENDS

;---------------------------------------------------

STACK SEGMENT STACK

DB 200 DUP(0)

STACK ENDS

;---------------------------------------------------

CODE SEGMENT USE16

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

START:

MOV AX,DATA

MOV DS,AX

FUNCION1:

MOV SI,0;???

MOV CX,6;???

SET\_PWD\_ALL\_ZERO:

MOV AL,0

MOV in\_pwd[SI+2],AL

INC SI

DEC CX

JNZ SET\_PWD\_ALL\_ZERO

MOV SI,0

MOV CX,10

SET\_NAME\_ALL\_ZERO:

MOV AL,0

MOV in\_name[SI+2],AL

INC SI

DEC CX

JNZ SET\_NAME\_ALL\_ZERO

INPUT\_NAME:

LEA DX,INPUT\_NAME\_MSG

MOV AH,9

INT 21H

LEA DX,in\_name

MOV AH,10

INT 21H

MOV SI, WORD PTR in\_name[1]

AND SI,00FFH

MOV in\_name[SI+2],0

MOV AL,0

CMP AL,in\_name[3]

JE AUTH\_FAIL

INPUT\_PWD:

LEA DX,INPUT\_PASS\_MSG

MOV AH,9

INT 21H

LEA DX,in\_pwd

MOV AH,10

INT 21H

MOV SI, WORD PTR in\_pwd[1]

AND SI,00FFH

MOV in\_pwd[SI+2],0

MOV SI,0;???

MOV CX,6;???

FUNCTION2\_PWD:

MOV AL,in\_pwd[SI+2]

CMP AL,BPASS[SI]

JNE PRINT\_LOGIN\_FAILED

INC SI

DEC CX

JNZ FUNCTION2\_PWD

MOV SI,0

MOV CX,10

FUNCTION2\_NAME:

MOV AL,in\_name[SI+2]

CMP AL,BNAME[SI]

JNE PRINT\_LOGIN\_FAILED

INC SI

DEC CX

JNZ FUNCTION2\_NAME

AUTH\_SUCCESS:

MOV BH,1

MOV BYTE PTR AUTH,BH

JMP START\_CYCLES

AUTH\_FAIL:

MOV AL,'q'

CMP AL,in\_name[2]

JE EXIT

MOV AL,'Q'

CMP AL,in\_name[2]

JE EXIT

MOV AL,0

CMP AL,in\_name[2]

JNE INPUT\_PWD

MOV BH,0

MOV BYTE PTR AUTH,BH

JMP START\_CYCLES

PRINT\_LOGIN\_FAILED:

LEA DX,LOGIN\_FAILED\_MSG

MOV AH,9

INT 21H

JMP FUNCION1

;???????1?2????

;???????3????

START\_CYCLES:

mov cx,10

mov si,0

reset\_goodsname:

mov goods\_name[si+2],al

inc si

dec cx

jnz reset\_goodsname

LEA DX,INPUT\_GOODS\_NAME\_MSG

MOV AH,9

INT 21H

LEA DX,goods\_name

MOV AH,10

INT 21H

MOV SI, WORD PTR goods\_name[1]

AND SI,00FFH

mov goods\_name[SI+2],0

mov cx,30

SET\_SOLD\_NUM\_ZERO:

mov si,cx

dec si

imul si,20

mov word ptr ga1[si+16],0

mov word ptr gb1[si+16],0

dec cx

jnz SET\_SOLD\_NUM\_ZERO

mov cycle\_times,1000

mov ax,0

call TIMER

FUNCTION3:

mov goods\_offset,0

is\_goods\_cycle:

mov cx,10

mov si,0

mov bx,goods\_offset

is\_goods\_cmp:

mov al,ga1[bx+si]

cmp al,goods\_name[si+2]

jne next\_goods

inc si

dec cx

jnz is\_goods\_cmp

mov goods\_name[bx+si+2],'$'

mov ax,word ptr ga1[bx+14]

cmp ax,word ptr ga1[bx+16]

jle GOODS\_SOLD\_OUT

add word ptr ga1[si+16],1

mov ax,word ptr gb1[bx+14]

cmp ax,word ptr gb1[bx+16]

jle GOODS\_SOLD\_OUT

add word ptr gb1[si+16],1

CALCULATE\_PR:

MOV AX,WORD PTR ga1[bx+10];???

IMUL ga1[bx+14];AX??cost1

MOV COST1,AX

MOV AX,WORD PTR ga1[bx+12];???

IMUL ga1[bx+16]

SUB AX,COST1;AX??profit1

MOV PROFIT1,AX

MOV AX,WORD PTR gb1[bx+10];???

IMUL gb1[bx+14];AX??cost2

MOV COST2,AX

MOV AX,WORD PTR gb1[bx+12];???

IMUL gb1[bx+16]

SUB AX,COST2;AX??profit2

MOV PROFIT2,AX

mov dx,0

MOV AX,PROFIT1

IMUL AX,WORD PTR 10

cwd

IDIV COST1

mov word ptr ga1[bx+18],ax;?????

mov dx,0

MOV AX,PROFIT2

IMUL AX,WORD PTR 10

cwd

IDIV COST2

mov word ptr ga2[bx+18],ax;?????

next\_goods:

add goods\_offset,20

cmp goods\_offset,600

jne is\_goods\_cycle

sub cycle\_times,1

jnz FUNCTION3;FUNCTION3????

mov ax,1

call TIMER

jmp FUNCION1

GOODS\_SOLD\_OUT:

jmp next\_goods

EXIT:

LEA DX,END\_MSG

MOV AH,9

INT 21H

MOV AH,1

INT 21H

MOV AH,4CH

INT 21H

;?????(ms),?????????????(ms)

;????:

; MOV AX, 0 ;??????

; CALL TIMER

; ... ... ;???????

; MOV AX, 1

; CALL TIMER ;???????????(ms)

;??: ???AX??????

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 0AH,0DH,'Time elapsed in ms is '

\_TMSG DB 12 DUP(0)

TIMER ENDP

disptime proc ;???????????55ms?(???ax???)

local timestr[8]:byte ;0,0,'"',0,0,0dh,0ah,'$'

push cx

push dx

push ds

push ss

pop ds

mov ah,2ch

int 21h

xor ax,ax

mov al,dh

mov cl,10

div cl

add ax,3030h

mov word ptr timestr+2,ax

mov timestr+4,'"'

xor ax,ax

mov al,dl

div cl

add ax,3030h

mov word ptr timestr+5,ax

mov word ptr timestr,0D0Ah

mov timestr+7,'$'

lea dx,timestr

mov ah,9

int 21h

pop ds

pop dx

pop cx

ret

disptime endp

CODE ENDS

END START

### 3.1.2实验步骤

1. 准备上机实验环境。

2. 使用记事本编辑录入源程序，存盘文件名为TASK1.ASM。使用MASM 6.0汇编源文件。即masm TASK1.ASM观察提示信息，若出错，则用编辑程序修改错误，存盘后重新汇编，直至不再报错为止。

3. 使用连接程序LINK.EXE将汇编生成的TASK1.OBJ文件连接成可执行文件。即link TASK1.OBJ；若连接时报错，则依照错误信息修改源程序。之后重新汇编和连接，直至不再报错并生成TASK1.EXE文件。

4. 使用调试器TD.EXE对生成的文件进行调试。

### 3.1.3实验记录与分析

1. 实验环境条件：AMD A8-8600P 1.6GHz，8G内存；WINDOWS 10下DOSBox0.73；EDIT.EXE 2.0；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。
2. 编译链接正常，如下图3.1.1所示：

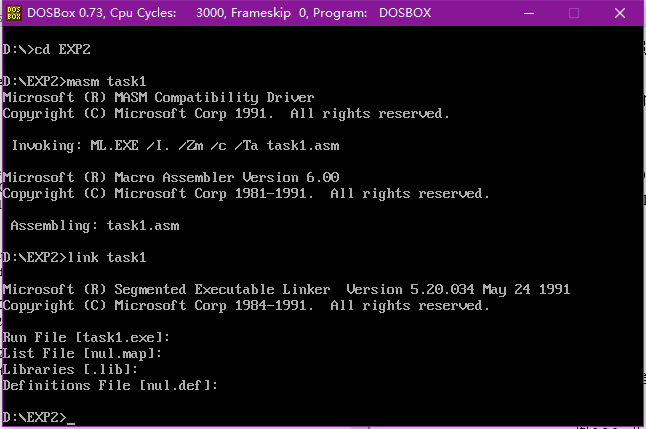


图3.1.1 正常编译连接

1. M=1000时，程序执行时间如下图3.1.2所示：

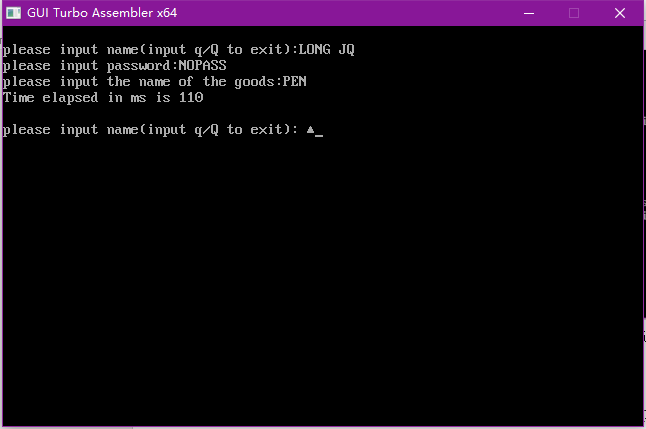


图3.2.2 循环次数1000程序执行时间

1. M=10000时，程序执行时间如下图3.1.3所示：

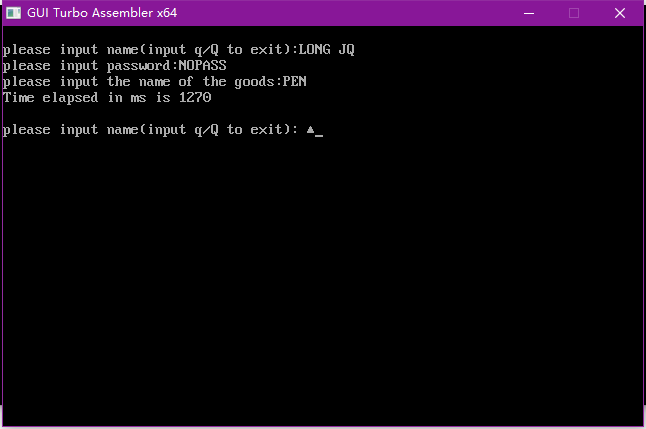


图3.2.3 循环次数10000时程序执行时间

1. M=30000时，程序执行时间如下图3.1.4所示：

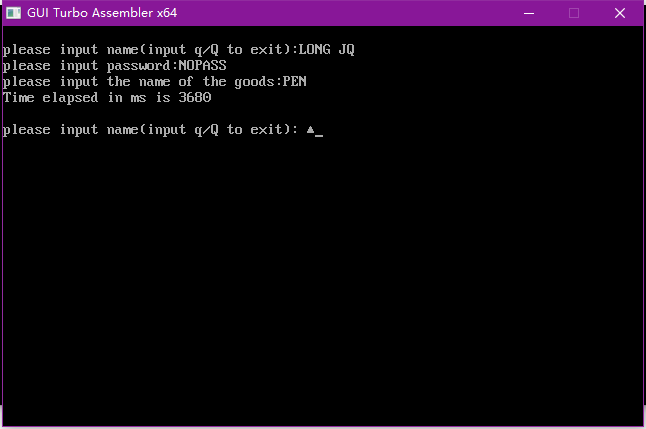


图3.2.4 循环次数30000时程序执行时间

观察上述执行结果可知，程序执行时间大致与循环的规模成正比，符合预期。

## 3.2任务2

### 3.2.1任务1程序优化分析

任务1程序理论上来说大致可以从以下几个方面优化：

1. 尽可能使用32位的寄存器，由于32位寄存器相比16寄存器上更加具有通用性和高效性，因此在双重循环中尽可能使用32位寄存器会提高程序运行性能。
2. 尽可能少的做比较判断，在程序需要跳转时，我们往往需要执行比较语句，而比较语句的实质是改变了标志寄存器的值，当我们尽可能少的使用比较语句的同时又能改变标志寄存器的值时，程序执行性能会有小幅度的提高。
3. 尽量使用移位运算和加减法运算代替乘法和除法运算，在CPU设计的底层就已经决定使用乘除法时间上的花销要远远大于加减法和移位运算的花销，因此当我们尽量使用移位和加减法运算时，程序执行性能会显著提高。
4. 第四个不是从细节上提高程序运行性能，而是从算法上提高运行性能，任务1task1.asm中我们每次找到该商品并卖出该商品后会重新算一次利润率以达到更新的效果，这样子意味着我们在双重循环反复提供了一些较大规模的指令，这样子时间上的花销是相当大的。我们退回去想一下，因为我们的程序意味着我们每次只卖出一件商品，而每卖出一件商品对利润率的贡献都是相同的，因此我们可以每次更新利润率的时候只把这部分多出来的利润率加上即可。
5. 综合考虑：由于程序框架构架在16位寄存器上，临时使用32位寄存器会打乱程序骨架，而且在该程序背景下，这种细节上的轻微改变不会对程序性能有显著的提高，因此不宜使用32位寄存器，但是也正是如此才为自己以后编写汇编语言程序提了个醒，提醒自己尽可能利用32位的寄存器。而第二种方法需要改变程序设计思路，也是相当于变相改变了程序框架，同样不可取，正确而良好的编程风范应是写代码之前就构思好程序的框架。第三种方法仍然不可取，当然，如果采用第三种方法，程序的性能相较前两种方法又会有显著的提高，然而缺点是乘除法操作中并不是常量也就是立即数在参与运算，这无疑大大增加了程序往移位和加减法运算优化的复杂性，而且最主要的一点就是我们有第四套方案，也就是从算法上改变。这样不光省掉了大量繁琐的乘除法运算大大减少了程序运行时间，而且在乘除法避免溢出上更加具备可靠性。

### 3.2.2实验步骤

1. 准备上机实验环境。

2. 使用记事本编辑录入源程序，存盘文件名为TASK2.ASM。使用MASM 6.0汇编源文件。即masm TASK2.ASM观察提示信息，若出错，则用编辑程序修改错误，存盘后重新汇编，直至不再报错为止。

3. 使用连接程序LINK.EXE将汇编生成的TASK2.OBJ文件连接成可执行文件。即link TASK2.OBJ；若连接时报错，则依照错误信息修改源程序。之后重新汇编和连接，直至不再报错并生成TASK2.EXE文件。

4. 使用调试器TD.EXE对生成的文件进行调试。

### 3.2.3实验记录与分析

1. 实验环境条件：AMD A8-8600P 1.6GHz，8G内存；WINDOWS 10下DOSBox0.73；EDIT.EXE 2.0；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。
2. 编译连接正常，如图3.2.1所示

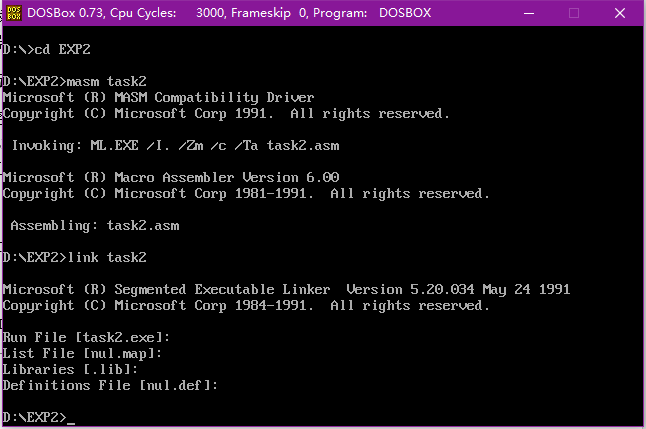


图3.2.1 正常编译连接

1. M=30000时，程序运行时间如图3.2.2所示：

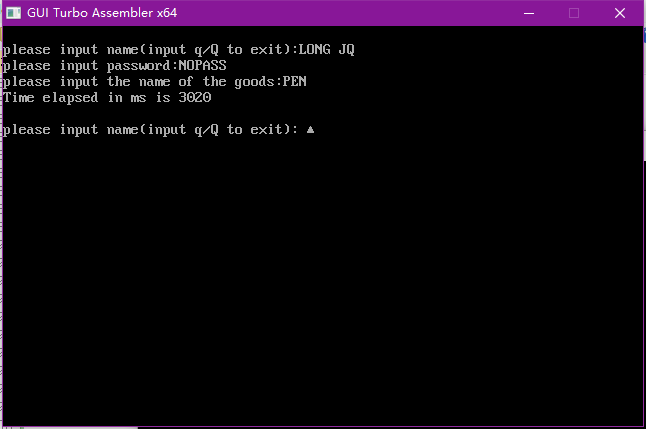


图3.2.2 循环次数30000时优化后程序执行时间

# 总结与体会

执行效率是一个程序的灵魂，而汇编语言已经是偏底层的语言，如果我们以后用汇编语言设计工具，我们能在汇编语言层面做出优化，那么这些工具成千上万次使用这些汇编语言程序时，效率必然会极高。

# 注

有个别实验代码较多较长，不适于直接在word中直接操作。因此本人将本次实验所有代码及二进制可执行文件上传至GitHub，具体网址为<https://github.com/DragonDriver/AssemblyExp/tree/master/Exp2>。

# 参考文献

[1] 王元珍 曹忠升 韩宗芬.《80x86汇编语言程序设计》“第二章 寻址方式”.华中科技大学出版社