

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实践

专业班级： _____

学 号： _____

姓 名： Jinhu

指导教师： _____

实验时段： 2020年3月19日~5月7日

实验地点： 武汉市洪山区

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：

报告日期：2020.5.10

实验报告成绩评定：

	1	2	3	4	5
实验完成质量（70%），报告撰写质量（30%），每次满分20分。					
合计（100分）					

备注：实验完成质量从实验目的达成程度，设计方案、实验方法步骤、实验记录与结果分析论述清楚等方面评价；报告撰写质量从撰写规范、完整、通顺、详实等方面评价。

指导教师签字：

日期：

汇编语言程序设计实验报告

目录

课程总体说明	- 3 -
0.1 课程目标	- 3 -
0.2 成绩构成	- 3 -
0.3 实验任务的总体描述	- 3 -
1 编程基础	1
1.1 实验目的与要求	1
1.2 实验内容	1
1.3 任务 1.1 实验过程	3
1.3.1 实验方法说明	3
1.3.2 实验记录与分析	3
1.4 任务 1.2 的实验过程	6
1.4.1 实验方法说明	6
1.4.2 实验记录与分析	7
1.5 任务 1.3 的实验过程	8
1.5.1 实验方法说明	8
1.5.2 实验记录与分析	9
1.6 任务 1.4 的实验过程	11
1.6.1 设计思想及存储单元分配	11
1.6.2 流程图	12
1.6.3 源程序	17
1.6.4 实验步骤	25
1.6.5 实验记录与分析	26
1.7 小结	30
2 程序优化	32
2.1 实验目的与要求	32
2.2 实验内容	32
2.3 任务 2.1 实验过程	32
2.3.1 实验方法说明	32
2.3.2 实验记录与分析	33
2.4 任务 2.2 实验过程	33
2.4.1 实验方法说明	33

汇编语言程序设计实验报告

2.4.2	实验记录与分析	34
2.5	小结	35
3	模块化程序设计	37
3.1	实验目的与要求	37
3.2	实验内容	37
3.3	任务 3.1 实验过程	38
3.3.1	设计思想及存储单元分配	38
3.3.2	流程图	39
3.3.3	源程序	40
3.3.4	实验步骤	44
3.3.5	实验记录与分析	45
3.4	任务 3.2 实验过程	50
3.4.1	实验方法说明	50
3.4.2	实验记录与分析	51
3.5	小结	53
4	中断与反跟踪	55
4.1	实验目的与要求	55
4.2	实验内容	55
4.3	任务 4.1 实验过程	56
4.3.1	设计思想及存储单元分配	56
4.3.2	流程图	57
4.3.3	源程序	58
4.3.4	实验步骤	61
4.3.5	实验记录与分析	62
4.4	任务 4.2 实验过程	64
4.4.1	实验方法说明	64
4.4.2	实验记录与分析	65
4.5	任务 4.3 实验过程	68
4.5.1	实验方法说明	68
4.5.2	实验记录与分析	69
4.6	小结	74
5	WIN32 程序设计	77
5.1	实验目的与要求	77
5.2	实验内容	77

汇 编 语 言 程 序 设 计 实 验 报 告

5.3 任务 5.1 实验过程.....	78
5.3.1 实验方法说明.....	78
5.3.2 实验记录与分析.....	79
5.4 小结	80
参考文献	82
附录 实验程序文件列表	83

汇编语言程序设计实验报告

课程总体说明

0.1 课程目标

下表是本课程的目标及与支撑的毕业要求指标点之间的关系。请大家关注下表中最后一列“实验中的注意事项”的内容，以便更有针对性的满足课程目标的要求。

课程目标	支撑的毕业要求指标点	实验中的注意事项
掌握汇编语言程序设计的全周期、全流程的基本方法与技术，通过程序调试、数据记录和分析，了解影响设计目标和技术方案的多种因素。	3.1 掌握与计算机复杂工程问题有关的工程设计和软硬件产品开发全周期、全流程的基本设计/开发方法和技术，了解影响设计目标和技术方案的多种因素。	不能只写代码完成功能，还要有设计、调试、记录、分析等部分的内容。
掌握编写、调试汇编语言程序的基本方法与技术，能根据实验任务要求，设计出较充分利用了汇编语言优势的软件功能部件或软件系统。	3.2 能为计算机复杂工程问题解决方案设计满足特定需求的软/硬件模块。	要思考与运用汇编语言的优势编写某些程序。
熟悉支持汇编语言开发、调试以及软件反汇编的主流工具的功能、特点与局限性及使用方法。	5.1 了解计算机专业常用的现代仪器、信息技术工具、工程工具和模拟软件的使用原理和方法，并理解其局限性。	熟悉实验中使用的工具，把对工具的看法记录在案。

0.2 成绩构成

实验课程综合成绩由实验过程成绩和实验报告成绩二部分构成。**实验过程成绩**：30%。主要考察各实验完成过程中的情况，希望大家做到预习准备充分，操作认真熟练，在规定的时间内完成实验任务，结果正确，积极发现和提出问题，交流讨论时描述问题准确、清晰。**实验报告成绩**：70%。主要考核报告体现的实验完成质量(含问题的分析、设计思想与程序、针对问题的实验方法与步骤、实验记录、实验结果分析等方面)和报告格式规范等撰写质量方面的内容。

0.3 实验任务的总体描述

本课程安排了 8 次 4 学时的课内实验课时，将实现一个具有一定复杂程度的系统。对该系统的相关要求被划分成了 **5 个主题**：1) 搭建原型系统；2) 在原型系统基础上探索程序指令级别的优化；3) 通过模块化调整与优化原型系统的程序结构；4) 通过中断、内存数据和地址操纵、跟踪与反跟踪、加密等措施增强系统安全性；5) 程序在不同平台上的移植。

针对这 5 个主题，对应地布置了 5 次实验。**实验 1 (编程基础)** 安排 8 个课内学时熟悉汇编语言程序设计的基本方法、技术与工具，设计实现指定原型系统的主要功能。针对原型系统的搭建，

汇编语言程序设计实验报告

实验报告中要有全周期、全流程的描述。**实验2（程序优化）**安排4个课内学时探索如何通过选择不同的指令及组合关系来优化程序的性能或代码长度。**实验3（模块化程序设计）**安排8个课内学时，利用子程序、模块化程序设计方法、与C语言混合编程等，调整与优化程序结构。**实验4（中断与反跟踪）**安排8个课内学时，通过利用中断机制、内存数据和地址操纵技术、跟踪与反跟踪技巧、加密等措施增强系统安全性。**实验5（WIN32 程序设计）**安排4个课内学时，熟悉在不同操作系统平台上移植实现已有系统功能的基本方法。每次实验的侧面有所不同，但都会涉及到课程目标的三个方面，因此，需要大家在实验过程中以及实验报告中有所注意和体现。

本次课程所涉及的原型系统是一个网店商品信息管理系统。下面描述该系统的基本需求，后续每次实验都是以这个基本需求为背景而展开的。

有一个老板在网上开了1个网店SHOP，网店里n种商品销售。每种商品的信息包括：商品名称（最长名称9个字节，其后加一个数值0表示名称结束），折扣（字节类型，取值0~10；0表示免费赠送，10表示不打折，1~9为折扣率；实际销售价格=销售价*折扣/10），进货价（字类型），销售价（字类型），进货总数（字类型），已售数量（字类型），推荐度【=（进货价/实际销售价格+已售数量/（2*进货数量））*128，字类型】。老板管理网店信息时需要输入自己的名字（最长名字9个字节，其后加一个数值0表示结束）和密码（最长密码6个字节，其后加一个数值0表示结束），老板登录后可查看商品的全部信息；顾客（无需登录）可以查看网店中每个商品除了进货价以外的信息，可以对指定商品下单预定。

该系统被执行后，首先显示一个菜单界面，菜单界面信息包括：

当前用户名：（老板名称或顾客）

当前浏览商品名称：（没有时空缺）

请输入数字1...9选择功能：

1. 登录/重新登录
2. 查找指定商品并显示其信息
3. 下订单
4. 计算商品推荐度
5. 排名
6. 修改商品信息
7. 迁移商店运行环境
8. 显示当前代码段首址
9. 退出

当用户输入某一个有效数字后，就进入到指定的功能中执行，执行完之后再回到该菜单界面。如果选择的是退出功能，则程序退出。该菜单中每项菜单的具体功能要求详见每次的实验任务描述。

汇编语言程序设计实验报告

1 编程基础

1.1 实验目的与要求

- (1) 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具 TD 的使用；
- (2) 理解数、符号、寻址方式等在计算机内的表现形式；
- (3) 理解指令执行与标志位改变之间的关系；
- (4) 熟悉常用的 DOS 功能调用；
- (5) 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
- (6) 加深对转移指令及一些常用的汇编指令的理解；
- (7) 掌握设计实现一个原型系统的基本方法。

1.2 实验内容

任务 1.1: 《80x86 汇编语言程序设计》教材中 P31 的 1.14 题。要求：

- (1) 直接在 TD 中输入指令，完成两个数的求和、求差的功能。求和/差后的结果放在(AH)中。
- (2) 请事先指出执行指令后(AH)、标志位 SF、OF、CF、ZF 的内容。
- (3) 记录上机执行后的结果，与（2）中对应的内容比较。

任务 1.2: 《80x86 汇编语言程序设计》教材中 P45 的 2.3 题。要求：

- (1) 分别记录执行到“MOV CX, 10”和“INT 21H”之前的(BX),(BP),(SI),(DI)各是多少。
- (2) 记录程序执行到退出之前数据段开始 40 个字节的內容，指出程序运行结果是否与设想的一致。

任务 1.3: 《80x86 汇编语言程序设计》教材中 P45 的 2.4 题的改写。要求：

- (1) 实现的功能不变，但对数据段中变量访问时所用到的变址寄存器采用 32 位寄存器。
- (2) 记录程序执行到退出之前数据段开始 40 个字节的內容，检查程序运行结果是否与设想的一致。
- (3) 在 TD 代码窗口中观察并记录机器指令代码在内存中的存放形式，并与 TD 中提供的反汇编语句及自己编写的源程序语句进行对照，也与任务 1.2 做对比。（相似语句记录一条即可，重点理解机器码与汇编语句的对应关系，尤其注意操作数寻址方式的编码形式，比如寄存器间接寻址、变址寻址、32 位寄存器与 16 位寄存器编码的不同、段前缀在代码里是如何表示的等）。
- (4) 观察连续存放的二进制串在反汇编成汇编语言语句时，从不同字节位置开始反汇编，结果怎样？理解 IP/EIP 指明指令起始位置的重要性。

任务 1.4 设计实现一个网店商品信息管理系统。

该系统的基本需求见本报告中的[“实验任务的总体描述”](#)。

根据系统的基本需求，可以制定如下的数据段的定义（供参考）：

汇编语言程序设计实验报告

BNAME DB 'ZHANG SAN',0 ; 老板姓名 (本实验要求必须是自己名字的拼音)

BPASS DB 'test', 0, 0, 0 ; 密码

AUTH DB 0 ; 当前登录状态, 0 表示顾客状态

GOOD DB/DW ... ; 当前浏览商品名称或地址 (自行确定)

N EQU 30

SNAME DB 'SHOP',0 ; 网店名称, 用 0 结束

GA1 DB 'PEN', 7 DUP(0), 10 ; 商品名称及折扣

DW 35, 56, 70, 25, ? ; 推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0), 9 ; 商品名称及折扣

DW 12, 30, 25, 5, ? ; 推荐度还未计算

GAN DB N-2 DUP('TempValue',0,8, 15, 0, 20, 0, 30, 0, 2, 0, ?, ?);除了 2 个已经具体定义了的商品信息以外, 其他商品信息暂时假定为一样的。

本次实验主要是利用分支、循环程序的结构, 实现该系统的基本功能, 并能熟悉全周期、全流程地设计实现一个原型系统的基本方法。本次实验要具体实现的功能要求如下:

0.主菜单界面

完整显示“实验任务的总体描述”中给出的界面信息。等待用户输入数字 (可使用 1 号 DOS 系统功能调用)。对用户输入的字符进行判断, 看是否是 1~9 的数字; 是的话就转移到对应功能的程序标号, 不是的话就提示错误, 回到主菜单界面。

1.登录/重新登录

(1) 先后分别提示用户输入姓名和密码 (可使用 9 号 DOS 系统功能调用)。

(2) 分别获取输入的姓名和密码 (可使用 10 号 DOS 系统功能调用)。输入的姓名字符串放在以 in_name 为首址的存储区中, 密码放在以 in_pwd 为首址的存储区中。

(3) 若输入姓名时只是输入了回车, 则将 0 送到 AUTH 字节变量中, 回到主菜单界面。

(4) 进行身份认证:

(a) 使用循环程序结构, 比较姓名是否正确。若不正确, 则跳到 (c)。

(b) 若正确, 再比较密码是否相同, 若相同, 跳到 (d)。

(c) 若名字或密码不对, 则提示登录失败, 并转到“(3)”的位置。

(d) 若名字和密码均正确, 则将 1 送到 AUTH 变量中, 回到主菜单界面。

提示: 字符串比较时, 当采用输入串的长度作为循环次数时, 若因循环次数减为 0 而终止循环, 则还要去判断网店中定义的字符串的下一个字符是否是结束符 0, 若是, 才能确定找到了 (这样做是为了避免输入的字符串仅仅是数据段中所定义字符串的子集的误判情况)。

2.查找指定商品并显示其信息

(1) 提示用户输入商品名称。

(2) 在商店中寻找是否存在该商品。

(3) 若存在, 则将商品名称或地址记录到 GOOD 字段中。商品信息的显示暂时不做。返回到主菜单界面。

(4) 若没有找到, 提示没有找到, 返回到主菜单界面。

3.下订单

(1) 判断当前浏览商品是否有效 (GOOD 不为空), 若有效, 判断其剩余数量是否为 0, 不为

汇编语言程序设计实验报告

0 则将已售数量加 1，重新计算所有商品的推荐度（目前不是用子程序实现的，所以，跳转之前，要把返回地址送到指定变量中），返回主菜单界面。

（2）若无效或剩余数量为 0，则提示错误，回到主菜单界面。

4. 计算商品推荐度

按照给出的公式计算所有商品的推荐度，返回到指定的位置（JMP 含返回地址的指定变量）。要求尽量避免溢出。结果只保留整数部分。

5. 排名

暂不实现，直接返回主菜单界面。

6. 修改商品信息

暂不实现，直接返回主菜单界面。

7. 迁移商店运行环境

暂不实现，直接返回主菜单界面。

8. 显示当前代码段首址

将当前代码段寄存器 CS 里面的内容按照 16 进制的方式显示到屏幕上，返回主菜单界面。

9. 退出

退出本系统（可使用 4CH 号 DOS 系统功能调用）

1.3 任务 1.1 实验过程

1.3.1 实验方法说明

1. 准备上机实验环境，对实验用到的软件进行安装、运行，通过试用初步了解软件的基本功能、操作等。

2. 在 TD 的代码窗口中的当前光标下输入第一个运算式对应的两个 8 位数值对应的指令语句 MOV AH, 01001101B; MOV AL, -01110010B; ADD AH, AL; 观察代码区显示的内容与自己输入字符之间的关系；然后确定 CS:IP 指向的是自己输入的第一条指令的位置，单步执行三次，观察寄存器内容的变化，记录标志寄存器的结果。

3. 尝试按照自己想的其他语句及输入格式等进行操作，积累更多的经验。

1.3.2 实验记录与分析

1. 实验环境条件：i7 1.8GHz；WINDOWS10 下 DOSBox0.74；TD.EXE 5.0。

2. 实验结果记录与分析

(1) $x_1 = + 0110011B$ $x_2 = + 1011010B$

$[x_1]_{补} = 00110011B$

$[x_2]_{补} = 01011010B$

$[x_1]_{补} + [x_2]_{补} = 10001101B$ （无进位，有溢出）

$[x_1]_{补} - [x_2]_{补} = 11011001B$ （有进位，无溢出）

预判：执行两数相加指令后 (AH) = 8DH, SF=1, OF=1, CF=0, ZF=0

执行两数相减指令后 (AH) = 0D9H, SF=1, OF=0, CF=1, ZF=0

汇编语言程序设计实验报告

实际执行过程：确定 CS:IP 指向自己输入的第一条指令的位置，如图 1.1 所示。

[CPU 80486]			1=[] []		
cs:0100	B433	mov ah,33	ax	3EF1	c=1
cs:0102	B05A	mov al,5A	bx	0000	z=0
cs:0104	02E0	add ah,al	cx	0000	s=0
cs:0106	1E	push ds	dx	0000	o=0
cs:0107	8ED8	mov ds,ax	si	0000	p=0
cs:0109	8E4608	mov es,[bp+08]	di	0000	a=0
cs:010C	8CC3	mov bx,es	bp	0000	i=1
cs:010E	B80600	mov ax,0006	sp	0000	d=0
cs:0111	CD31	int 31	ds	0AA0	
cs:0113	7216	jb 012B	es	0AA0	
cs:0115	83F910	cmp cx,0010	ss	0AA0	
cs:0118	7311	jnb 012B	cs	0AA0	
cs:011A	C1EA04	shr dx,04	ip	0100	

图 1.1 执行程序前 CS:IP 指向自己输入的第一条指令

实际执行结果：单步执行三次后寄存器 AX 和标志寄存器中的内容如图 1.2 所示。

[CPU 80486]			1=[] []		
cs:0100	B433	mov ah,33	ax	8D5A	c=0
cs:0102	B05A	mov al,5A	bx	0000	z=0
cs:0104	02E0	add ah,al	cx	0000	s=1
cs:0106	1E	push ds	dx	0000	o=1
cs:0107	8ED8	mov ds,ax	si	0000	p=1
cs:0109	8E4608	mov es,[bp+08]	di	0000	a=0
cs:010C	8CC3	mov bx,es	bp	0000	i=1
cs:010E	B80600	mov ax,0006	sp	0000	d=0
cs:0111	CD31	int 31	ds	0AA0	
cs:0113	7216	jb 012B	es	0AA0	
cs:0115	83F910	cmp cx,0010	ss	0AA0	
cs:0118	7311	jnb 012B	cs	0AA0	
cs:011A	C1EA04	shr dx,04	ip	0106	

[CPU 80486]			1=[] []		
cs:0100	B433	mov ah,33	ax	D95A	c=1
cs:0102	B05A	mov al,5A	bx	0000	z=0
cs:0104	2AE0	sub ah,al	cx	0000	s=1
cs:0106	1E	push ds	dx	0000	o=0
cs:0107	8ED8	mov ds,ax	si	0000	p=0
cs:0109	8E4608	mov es,[bp+08]	di	0000	a=1
cs:010C	8CC3	mov bx,es	bp	0000	i=1
cs:010E	B80600	mov ax,0006	sp	0000	d=0
cs:0111	CD31	int 31	ds	0AA0	
cs:0113	7216	jb 012B	es	0AA0	
cs:0115	83F910	cmp cx,0010	ss	0AA0	
cs:0118	7311	jnb 012B	cs	0AA0	
cs:011A	C1EA04	shr dx,04	ip	0106	

图 1.2 执行相加指令后寄存器 AH 中的值、各标志寄存器的值和事先预想结果完全一致

(2) $x_1 = -0101001B$ $x_2 = -1011101B$

$[x_1]_{补} = 11010111B$

$[x_2]_{补} = 10100011B$

$[x_1]_{补} + [x_2]_{补} = 01111010B$ (有进位, 有溢出)

$[x_1]_{补} - [x_2]_{补} = 00110100B$ (无进位, 无溢出)

预判：执行两数相加指令后 (AH)=7AH, SF=0, OF=1, CF=1, ZF=0

执行两数相减指令后 (AH)=34H, SF=0, OF=0, CF=0, ZF=0

实际执行过程：确定 CS:IP 指向自己输入的第一条指令的位置，如图 1.3 所示。

汇编语言程序设计实验报告

CPU 80486			1=1111		
cs:0100 B4D7	mov	ah,D7	ax	34A3	c=0
cs:0102 B0A3	mov	al,A3	bx	0000	z=0
cs:0104 02E0	add	ah,al	cx	0000	s=0
cs:0106 1E	push	ds	dx	0000	o=0
cs:0107 8ED8	mov	ds,ax	si	0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di	0000	a=0
cs:010C 8CC3	mov	bx,es	bp	0000	i=1
cs:010E B80600	mov	ax,0006	sp	0080	d=0
cs:0111 CD31	int	31	ds	0AAD	
cs:0113 7216	jb	012B	es	0AAD	
cs:0115 83F910	cmp	cx,0010	ss	0AAD	
cs:0118 7311	jnb	012B	cs	0AAD	
cs:011A C1EA04	shr	dx,04	ip	0100	

图 1.3 执行程序前 CS:IP 指向自己输入的第一条指令

实际执行结果：单步执行三次后寄存器 AX 和标志寄存器中的内容如图 1.4 所示。

CPU 80486			1=1111		
cs:0100 B4D7	mov	ah,D7	ax	7AA3	c=1
cs:0102 B0A3	mov	al,A3	bx	0000	z=0
cs:0104 02E0	add	ah,al	cx	0000	s=0
cs:0106 1E	push	ds	dx	0000	o=1
cs:0107 8ED8	mov	ds,ax	si	0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di	0000	a=0
cs:010C 8CC3	mov	bx,es	bp	0000	i=1
cs:010E B80600	mov	ax,0006	sp	0080	d=0
cs:0111 CD31	int	31	ds	0AAD	
cs:0113 7216	jb	012B	es	0AAD	
cs:0115 83F910	cmp	cx,0010	ss	0AAD	
cs:0118 7311	jnb	012B	cs	0AAD	
cs:011A C1EA04	shr	dx,04	ip	0106	

CPU 80486			1=1111		
cs:0100 B4D7	mov	ah,D7	ax	34A3	c=0
cs:0102 B0A3	mov	al,A3	bx	0000	z=0
cs:0104 2AE0	sub	ah,al	cx	0000	s=0
cs:0106 1E	push	ds	dx	0000	o=0
cs:0107 8ED8	mov	ds,ax	si	0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di	0000	a=0
cs:010C 8CC3	mov	bx,es	bp	0000	i=1
cs:010E B80600	mov	ax,0006	sp	0080	d=0
cs:0111 CD31	int	31	ds	0AAD	
cs:0113 7216	jb	012B	es	0AAD	
cs:0115 83F910	cmp	cx,0010	ss	0AAD	
cs:0118 7311	jnb	012B	cs	0AAD	
cs:011A C1EA04	shr	dx,04	ip	0106	

图 1.4 执行程序后寄存器 AH 中的值，各标志寄存器的值和事先预想结果完全一致

(3) $x_1 = +1100101B$ $x_2 = -1011101B$

$[x_1]_{补} = 01100101B$

$[x_2]_{补} = 10100011B$

$[x_1]_{补} + [x_2]_{补} = 00001000B$ (有进位，无溢出)

$[x_1]_{补} - [x_2]_{补} = 11000010B$ (有进位，有溢出)

预判：执行两数相加指令后 (AH)=08H, SF=0, OF=0, CF=1, ZF=0

执行两数相减指令后 (AH)=0C2H, SF=1, OF=1, CF=1, ZF=0

实际执行过程：确定 CS:IP 指向自己输入的第一条指令的位置，如图 1.5 所示。

汇编语言程序设计实验报告

CPU 80486			1=[]		
cs:0100 B465	mov	ah,65	ax	08A3	c=1
cs:0102 B0A3	mov	al,A3	bx	0000	z=0
cs:0104 02E0	add	ah,al	cx	0000	s=0
cs:0106 1E	push	ds	dx	0000	o=0
cs:0107 8ED8	mov	ds,ax	si	0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di	0000	a=0
cs:010C 8CC3	mov	bx,es	bp	0000	i=1
cs:010E B80600	mov	ax,0006	sp	0000	d=0
cs:0111 CD31	int	31	ds	0AAD	
cs:0113 7216	jb	012B	es	0AAD	
cs:0115 83F910	cmp	cx,0010	ss	0AAD	
cs:0118 7311	jnb	012B	cs	0AAD	
cs:011A C1EA04	shr	dx,04	ip	0100	

图 1.5 执行程序前 CS:IP 指向自己输入的第一条指令

实际执行结果：单步执行三次后寄存器 AX 和标志寄存器中的内容如图 1.6 所示。

CPU 80486			1=[]		
cs:0100 B465	mov	ah,65	ax	08A3	c=1
cs:0102 B0A3	mov	al,A3	bx	0000	z=0
cs:0104 02E0	add	ah,al	cx	0000	s=0
cs:0106 1E	push	ds	dx	0000	o=0
cs:0107 8ED8	mov	ds,ax	si	0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di	0000	a=0
cs:010C 8CC3	mov	bx,es	bp	0000	i=1
cs:010E B80600	mov	ax,0006	sp	0000	d=0
cs:0111 CD31	int	31	ds	0AAD	
cs:0113 7216	jb	012B	es	0AAD	
cs:0115 83F910	cmp	cx,0010	ss	0AAD	
cs:0118 7311	jnb	012B	cs	0AAD	
cs:011A C1EA04	shr	dx,04	ip	0106	

CPU 80486			1=[]		
cs:0100 B465	mov	ah,65	ax	C2A3	c=1
cs:0102 B0A3	mov	al,A3	bx	0000	z=0
cs:0104 2AE0	sub	ah,al	cx	0000	s=1
cs:0106 1E	push	ds	dx	0000	o=1
cs:0107 8ED8	mov	ds,ax	si	0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di	0000	a=0
cs:010C 8CC3	mov	bx,es	bp	0000	i=1
cs:010E B80600	mov	ax,0006	sp	0000	d=0
cs:0111 CD31	int	31	ds	0AAD	
cs:0113 7216	jb	012B	es	0AAD	
cs:0115 83F910	cmp	cx,0010	ss	0AAD	
cs:0118 7311	jnb	012B	cs	0AAD	
cs:011A C1EA04	shr	dx,04	ip	0106	

图 1.6 执行程序后寄存器 AH 中的值，各标志寄存器的值和事先预想结果完全一致

1.4 任务 1.2 的实验过程

1.4.1 实验方法说明

1. 准备上机实验环境。
2. 使用编辑程序 EDIT.EXE 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；观察提示信息，若出错，则检查是否有录入错误并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 文件连接成执行文件。
即 LINK TASK2；
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。

汇编语言程序设计实验报告

4. 执行该程序。即在命令行提示符后输入 TASK2 后回车，观察执行现象。
5. 使用 TD.EXE 观察 TASK2 的执行情况。即 TD TASK2.EXE 回车
 - (1) 观察 CS、IP、SP、DS、ES、SS 的值。
 - (2) 单步执行开始第 1 条和第 2 条指令，观察 DATA 的实际值，以及 AX 和 DS 的改变情况。
 - (3) 观察 SS: 0 至 SS: SP 区域的数据值。
 - (4) 观察 DS: 0 开始数据区，找到各变量在数据段中的位置 and 值。
 - (5) 观察第 3 至 6 条语句中源操作数的值，是否依次和 BUF1,BUF2,BUF3,BUF4 变量的偏移地址相同，以及 SI,DI,BX,BP 寄存器中的变化。
 - (6) 执行第 7 条指令，观察 CX 的值是否为变化为立即数 0AH。
 - (7) 执行到 MOV AL,[SI]，观察源操作数的具体值，是否和 BUF1 变量中第一个字节的值相同。
 - (8) 执行 MOV [DI], AL，观察目的操作数的形式。到数据段中观察 BUF2 变量中第一个字节中的值是否和 AL 寄存器中的值相等。
 - (9) 执行到 INC AL，观察 AL 寄存器中的值，是否增加了 1。
 - (10) 执行到 MOV [BX],AL，观察目的操作数的形式。到数据段中观察 BUF3 变量中第一个字节中的值，是否和 AL 寄存器中的值相等。
 - (11) 执行到 ADD AL, 3，观察 AL 寄存器中的值，是否增加了 3。
 - (12) 执行到 MOV DS:[BP], AL，观察目的操作数的形式。到数据段中观察 BUF4 变量中第一个字节中的值，是否和 AL 寄存器中的值相等。
 - (12) 执行第 14 至 17 条指令，依次观察寄存器 SI,DI,BP,BX 中的值是否增加了 1。
 - (13) 执行到 DEC CX，观察寄存器 CX 中的值，是否减少 1。
 - (14) 观察 CX 中的值是否为 0，并观察程序执行的下一条指令，是否回到了第 (7) 步。
 - (15) 循环执行 (7) 至 (14) 步中的步骤，直至寄存器 CX 中的内容变为 0，观察程序是否跳脱了循环。
 - (16) 程序退出之前，观察变量 BUF1,BUF2,BUF3 和 BUF4 中的值，是否满足 $BUF2[i]=BUF1[i]$, $BUF3[i]=BUF1[i]+1$, $BUF4[i]=BUF1[i]+4$ 。
 - (16) 执行到 MOV AH, 4CH，观察 AH 寄存器中的内容，是否变为 4CH。
 - (17) 执行到 INT 21H，观察程序是否成功退出，以及退出值是否是 AL 寄存器中的值。
6. 将程序运行中的执行结果和自己预想的运行结果作比较，若有不正确，弄清楚问题出在哪，并解决问题。

1.4.2 实验记录与分析

1. 实验环境条件: i7 1.8GHz; WINDOWS10 下 DOSBox0.74; TD.EXE 5.0。
2. 实验结果记录与分析
程序执行过程中段寄存器 SS,DS 和 CS 中的值如图 1.7 所示。

汇编语言程序设计实验报告

[]=CPU 80486			1=[]		
cs:0000	B8D40A	mov ax,0AD4	ax	0AD4	c=0
cs:0003	8ED8	mov ds,ax	bx	0000	z=0
cs:0005	BE0000	mov si,0000	cx	0000	s=0
cs:0008	BF0A00	mov di,000A	dx	0000	o=0
cs:000B	BB1400	mov bx,0014	si	0000	p=0
cs:000E	BD1E00	mov bp,001E	di	0000	a=0
cs:0011	B90A00	mov cx,000A	bp	0000	i=1
cs:0014	8A04	mov al,[si]	sp	00C8	d=0
cs:0016	8805	mov [di],al	ds	0AD4	
cs:0018	FEC0	inc al	es	0AB7	
cs:001A	8807	mov [bx],al	ss	0AC7	
cs:001C	0403	add al,03	cs	0AD7	
cs:001E	3EB84600	mov ds:[bp],al	ip	0005	

图 1.7 段寄存器 SS,DS 和 CS 中的值

由图可知在此程序运行过程中堆栈段的首址为 0AC70H，数据段首址为 0AD40H，代码段首址为 0AD7H，经计算可得堆栈段首址与数据段首址之间相差 256 个字节，数据段与代码段之间相差 48 个字节。由源程序中的定义可知，堆栈段占据 200 个字节，数据段占据 40 个字节。且段首址只能是 16 的倍数，可知在此程序运行过程中堆栈段、数据段、代码段是根据堆栈段和数据段的大小相邻排列的。

刚把 DATA 段首址送到 DS 寄存器之后数据段前 40 个字节的内容如图 1.8 所示。

ds:0000	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
ds:0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ds:0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

图 1.8 程序执行前数据段前 40 个字节内容

程序执行后，退出前数据段前 40 个字节的内容如图 1.9 所示。

ds:0000	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
ds:0010	06 07 08 09 01 02 03 04 05 06 07 08 09 0A 01 05
ds:0020	06 07 08 09 0A 0B 0C 0D 00 00 00 00 00 00 00 00

图 1.9 程序执行后数据段前 40 个字节内容

经比对可知，完全符合实验前的预想，即

```
BUF2[i]=BUF1[i]
BUF3[i]=BUF1[i]+1
BUF4[i]=BUF1[i]+4
```

1.5 任务 1.3 的实验过程

1.5.1 实验方法说明

1. 准备上机实验环境。
2. 使用编辑程序 EDIT.EXE 录入源程序，存盘文件名为 TASK3.ASM。使用 MASM 6.0 汇编源文件。即 MASM TASK3；观察提示信息，若出错，则检查是否有录入错误并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 TASK3.OBJ 文件连接成执行文件。
即 LINK TASK3；
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. 执行该程序。即在命令行提示符后输入 TASK3 后回车，观察执行现象。
5. 使用 TD.EXE 观察 TASK2 的执行情况。即 TD TASK3.EXE 回车
(1) 观察 CS、IP、SP、DS、ES、SS 的值。

汇编语言程序设计实验报告

(2) 单步执行开始第 1 条指令, 观察 DATA 的实际值, 以及把 DATA 送到 32 位寄存器中的反汇编语句, 关注段前缀 DATA 在代码中的表示和 32 位寄存器 AX 的编码。

(3) 执行到 MOV DS, AX, 观察 16 位寄存器 AX 和 DS 的编码, 比较第 1 条指令和第 2 条指令的机器码中的 AX 编码是否有差异。

(4) 单步执行第 3 条和第 4 条指令, 观察 32 位寄存器 EDI 和 16 位寄存器 CX 的编码以及立即数放到不同字节长度的寄存器中时的编码差异。

(5) 执行到 MOV AL, [EDI], 观察 8 位寄存器 AL 的编码并比较其与 16 位寄存器 AX 编码的差异, 观察寄存器间接寻址方式[EDI]的反汇编语句。

(6) 执行到 MOV BUF2[EDI], AL, 观察变址寻址方式的反汇编语句和机器码。

(7) 执行到 JNE CX, 观察条件转移指令 JNE 的反汇编语句、机器码以及转移的目的代码段位置。

(8) 循环执行 (7) 至 (14) 步中的步骤, 此时重点在于比较与任务 1.2 中反汇编语句的异同, 记录相似的语句。循环执行到寄存器 CX 中的内容变为 0, 此时跳脱循环

(9) 程序退出之前, 观察变量 BUF1,BUF2,BUF3 和 BUF4 中的值, 是否和任务 1.2 中程序执行后的结果相同, 以判断是否完成程序改写目标。

(10) 执行到 INT 21H, 观察请求中断指令的反汇编语句。

6.分析以上记录结果, 总结各种寻址方式的编码特点以及反汇编语句的特点。

7.从不同字节位置开始执行反汇编语句, 观察程序执行情况, 关注是否出现异常。

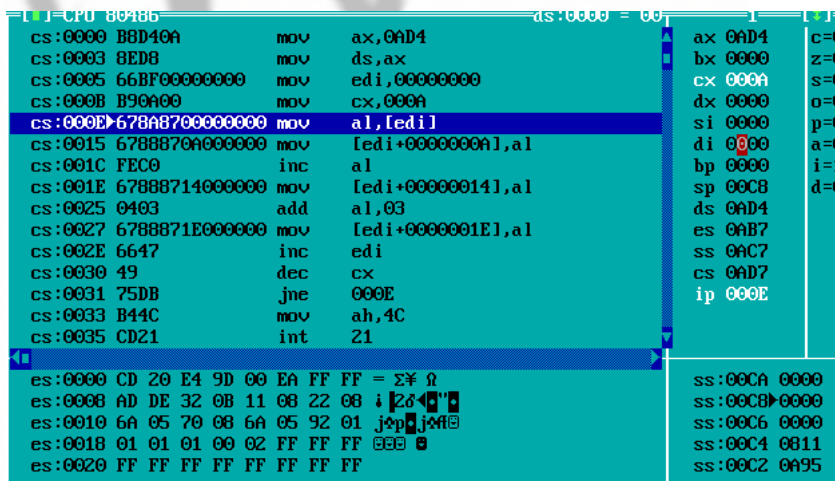
8.根据 7 中的执行结果体会 IP 和 EIP 指明程序起始位置的重要意义。

1.5.2 实验记录与分析

1.实验环境条件: i7 1.8GHz; WINDOWS10 下 DOSBox0.74; TD.EXE 5.0。

2.关于反汇编语句和机器码

任务 3 中程序的部分反汇编语句和机器码如图 1.10 所示。



```
cs:0000 B8D40A mov ax,0AD4
cs:0003 8ED8 mov ds,ax
cs:0005 66BF00000000 mov edi,00000000
cs:000B B90A00 mov cx,000A
cs:000E 678AB700000000 mov al,[edi]
cs:0015 6788B70A000000 mov [edi+0000000A],al
cs:001C FEC0 inc al
cs:001E 6788B714000000 mov [edi+00000014],al
cs:0025 0403 add al,03
cs:0027 6788B71E000000 mov [edi+0000001E],al
cs:002E 6647 inc edi
cs:0030 49 dec cx
cs:0031 75DB jne 000E
cs:0033 B44C mov ah,4C
cs:0035 CD21 int 21

es:0000 CD 20 E4 9D 00 EA FF FF = Σ¥ Ω
es:0008 AD DE 32 0B 11 08 22 08 ; 2d 0"
es:0010 6A 05 70 08 6A 05 92 01 jmp 0,j240
es:0018 01 01 01 00 02 FF FF FF 000 0
es:0020 FF FF FF FF FF FF FF FF

ss:00CA 0000
ss:00CB 0000
ss:00C6 0000
ss:00C4 0B11
ss:00C2 0A95
```

图 1.10 任务 3 中程序的部分机器码

对比任务 2 中的部分反汇编语句和机器码, 如图 1.11 所示。

汇编语言程序设计实验报告

```

CPU 80486
cs:0000 B8D40A mov ax, 0AD4
cs:0003 8ED8 mov ds, ax
cs:0005 BE0000 mov si, 0000
cs:0008 BF0A00 mov di, 000A
cs:000B BB1400 mov bx, 0014
cs:000E BD1E00 mov bp, 001E
cs:0011 B90A00 mov cx, 000A
cs:0014 8A04 mov al, [si]
cs:0016 8B05 mov [di], al
cs:0018 FEC0 inc al
cs:001A 8B07 mov [bx], al
cs:001C 0403 add al, 03
cs:001E 3E884600 mov ds: [bp], al
cs:0022 46 inc si
cs:0023 47 inc di

ds:0000 CD 20 E4 9D 00 EA FF FF = Σ¥ Ω
ds:0008 AD DE 32 0B 11 08 22 08 i 2d 4
ds:0010 6A 05 70 08 6A 05 92 01 j&p j&f
ds:0018 01 01 01 00 02 FF FF FF 000 0
ds:0020 FF FF FF FF FF FF FF FF

ss:00CA 0000
ss:00CB 0000
ss:00C6 FFFF
ss:00C4 0000
ss:00C2 0000
    
```

图 1.11 对比任务 2 中程序的部分机器码

由以上两张截图可以得到以下几点规律：

- (1) 立即数的编码长度由寄存器的字节长度或者 PTR 的类型决定。
- (2) 16 位寄存器的直接和间接寻址的编码长度均为 1 个字节，32 位寄存器的直接寻址的编码长度为 2 个字节。
- (3) 变址寻址和基址加变址寻址的反汇编语句会将变量转化为对应的偏移地址，因为这个偏移地址在程序运行前根据数据段中的定义就能确定。
- (4) 转移指令的反汇编语句会将标号转化为标号后的第一条指令在代码段中的偏移地址。

3.关于 IP/EIP 寄存器的重要意义

若通过更改 IP 寄存器值的方式使程序跳过 MOV CX, 10 这一指令，且 CX 初始值为 0000H 经过一次循环体后 CX 中的值变为 0FFFFH。显然，程序不能正确运行。这一步实验如图 1.12 所示。

```

CPU 80486
cs:000E 678A8700000000 mov al, [edi]
cs:0015 6788870A000000 mov [edi+0000000A], al
cs:001C FEC0 inc al
cs:001E 67888714000000 mov [edi+00000014], al
cs:0025 0403 add al, 03
cs:0027 6788871E000000 mov [edi+0000001E], al
cs:002E 6647 inc edi
cs:0030 49 dec cx
cs:0031 75DB jne 000E
cs:0033 B44C mov ah, 4C
cs:0035 CD21 int 21
cs:0037 0306B426 add ax, [26B4]
cs:003B 0306B626 add ax, [26B6]
cs:003F A3B026 mov [26B0], ax
cs:0042 BB0500 mov bx, 0005

ax 0A05 c=0
bx 0000 z=0
cx FFFF s=0
dx 0000 o=0
si 0000 p=0
di 0002 a=0
bp 0000 i=1
sp 00C8 d=0
ds 0AD4
es 0AB7
ss 0AC7
cs 0AD7
ip 0030

es:0000 CD 20 E4 9D 00 EA FF FF = Σ¥ Ω
es:0008 AD DE 32 0B 11 08 22 08 i 2d 4
es:0010 6A 05 70 08 6A 05 92 01 j&p j&f
es:0018 01 01 01 00 02 FF FF FF 000 0
es:0020 FF FF FF FF FF FF FF FF

ss:00CA 0000
ss:00CB 0000
ss:00C6 0000
ss:00C4 0B11
ss:00C2 0A95
    
```

图 1.12 修改 IP 值后的程序运行

根据以上运行结果可知 IP/EIP 始终指向下一条即将执行的代码的地址，对程序的正确运行有至关重要的保证。

3.更改寻址方式后的程序运行结果

任务 1.3 中的程序的最终运行结果如图 1.13 所示。

```

ds:0000 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
ds:0010 06 07 08 09 01 02 03 04 05 06 07 08 09 0A 04 05
ds:0020 06 07 08 09 0A 0B 0C 0D 0E 0F 00 00 00 00 00 00
    
```

图 1.13 任务 1.3 中的程序最终运行结果

汇编语言程序设计实验报告

由图 1.13 可知,此结果和任务 1.2 中的运行结果相同,说明在一定条件下,各种寻址方式之间可以灵活转化,我们可以在多种寻址方式之间尽量选择编码短的、运行效率高的寻址方式。

1.6 任务 1.4 的实验过程

1.6.1 设计思想及存储单元分配

任务 1.4 中所涉及的功能主要是 DOS 功能调用实现输入输出,以及字符串的匹配和二进制数转化为任意进制输出以及功能三和功能四的转移这三个方面的实现。以下就字符串的匹配和二进制数转化为任意进制输出以及功能三四之间的转移说明设计思想、存储单元分配、寄存器分配的问题。

1. 字符串的匹配

(1) 设计思想:对于两个已知长度的字符串,首先比较两者长度是否相等,若不相等直接判定两个字符串不同,若相等,将这两个字符串的相同长度放入 CX 寄存器中,然后开始循环,依次比较每个字符是否相等。若至少一个字符串的长度未知,也是依次比较每一字符,但需要将循环出口设置为遇 0 跳出循环并去检查另一个字符串中的这个位置是否也是 0 或者是 '\$',若是,则匹配成功,否则两字符串不相等。

(2) 存储单元分配

STRING1:字符串 1。

STRING2:字符串 2。

(3) 寄存器分配

SI:初始化为 STRING1 的偏移地址,使 SI 指向字符串 1,每循环一次自增 1。

DI:初始化为 STRING2 的偏移地址,使 DI 指向字符串 2,每循环一次自增 1。

2. 二进制数转化为任意进制输出

(1) 设计思想:采用除 k 取余法,每次的余数压入堆栈,全部除尽后再依次出栈,出栈的同时将其转化成对应的 ASCII 码。

(2) 存储单元分配

OUT:输出缓冲区。

(3) 寄存器分配

DX:AX 存放要转化的数。

BX:存放转化成的进制数。

CX:计数,记录压入堆栈中多少个字。

SI:指向内存中的输出缓冲区。

3. 功能三(下订单)中使用功能四(计算商品推荐度)部分代码的跳转方法

(1) 设计思想:在跳转到功能四中之前,用\$将将来要返回的指令地址保存在寄存器中(保证功能四不会修改这个寄存器的值),在计算完商品推荐度之后,再判断这是否是用户选择功能三的副效应,若是,则通过条件转移指令转移到一个中转处,在中转处通过无条件转移指令跳转到 SI 指向的指令地址。

(2) 存储单元分配

不需要分配存储单元。

汇编语言程序设计实验报告

(3) 寄存器分配

SI: 保存将来需要返回的指令地址。

1.6.2 流程图

1. 字符串比较过程的流程图（避免后续流程图过于繁琐）

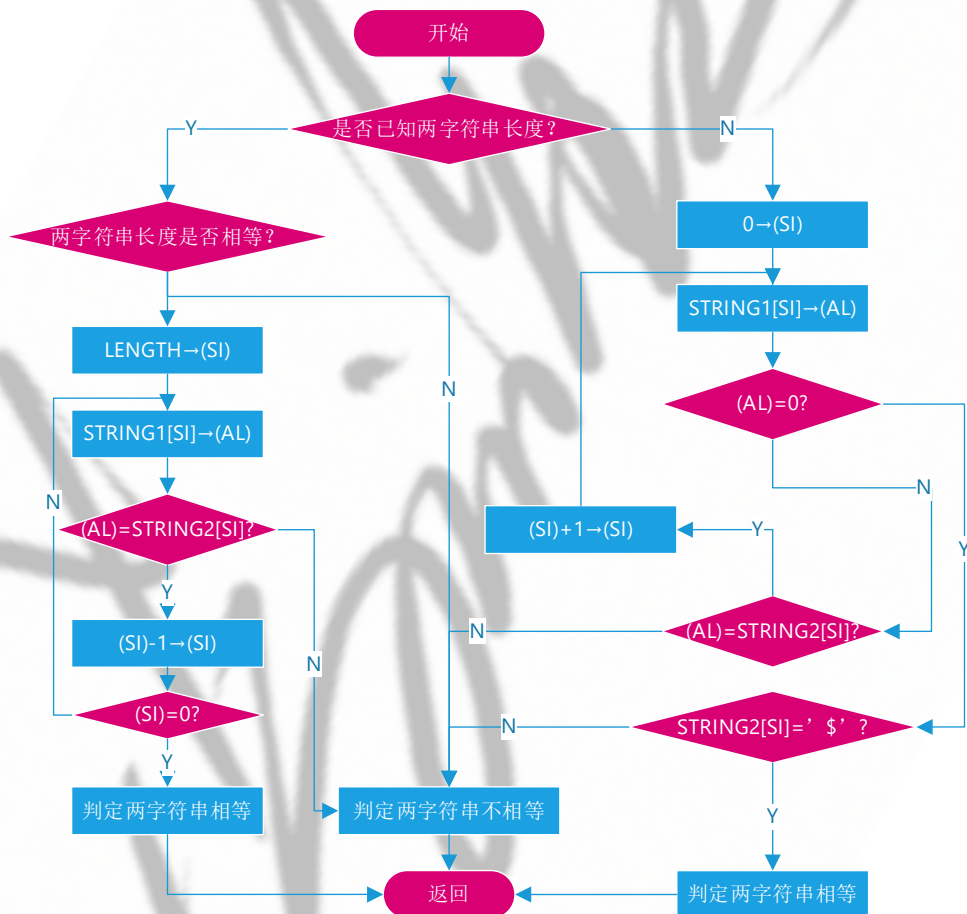


图 1.14 字符串比较过程的流程图

汇编语言程序设计实验报告

2. 二进制数转化为任意进制数的字符串（RADIX 子程序）的流程图

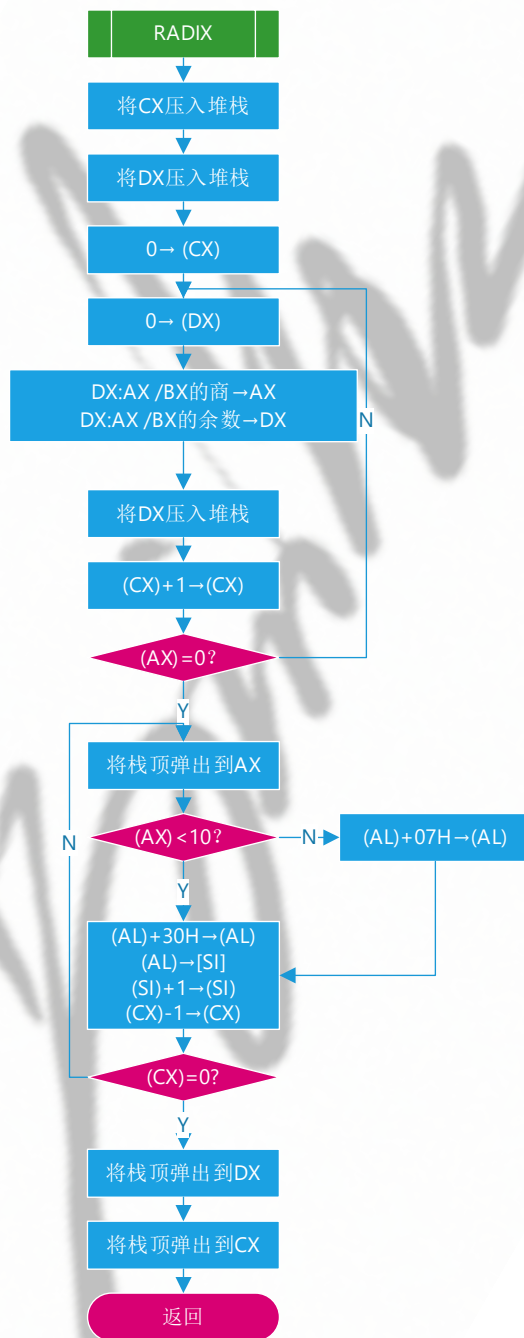


图 1.15 RADIX 子程序的流程图

汇编语言程序设计实验报告

3. 系统整体设计的流程图

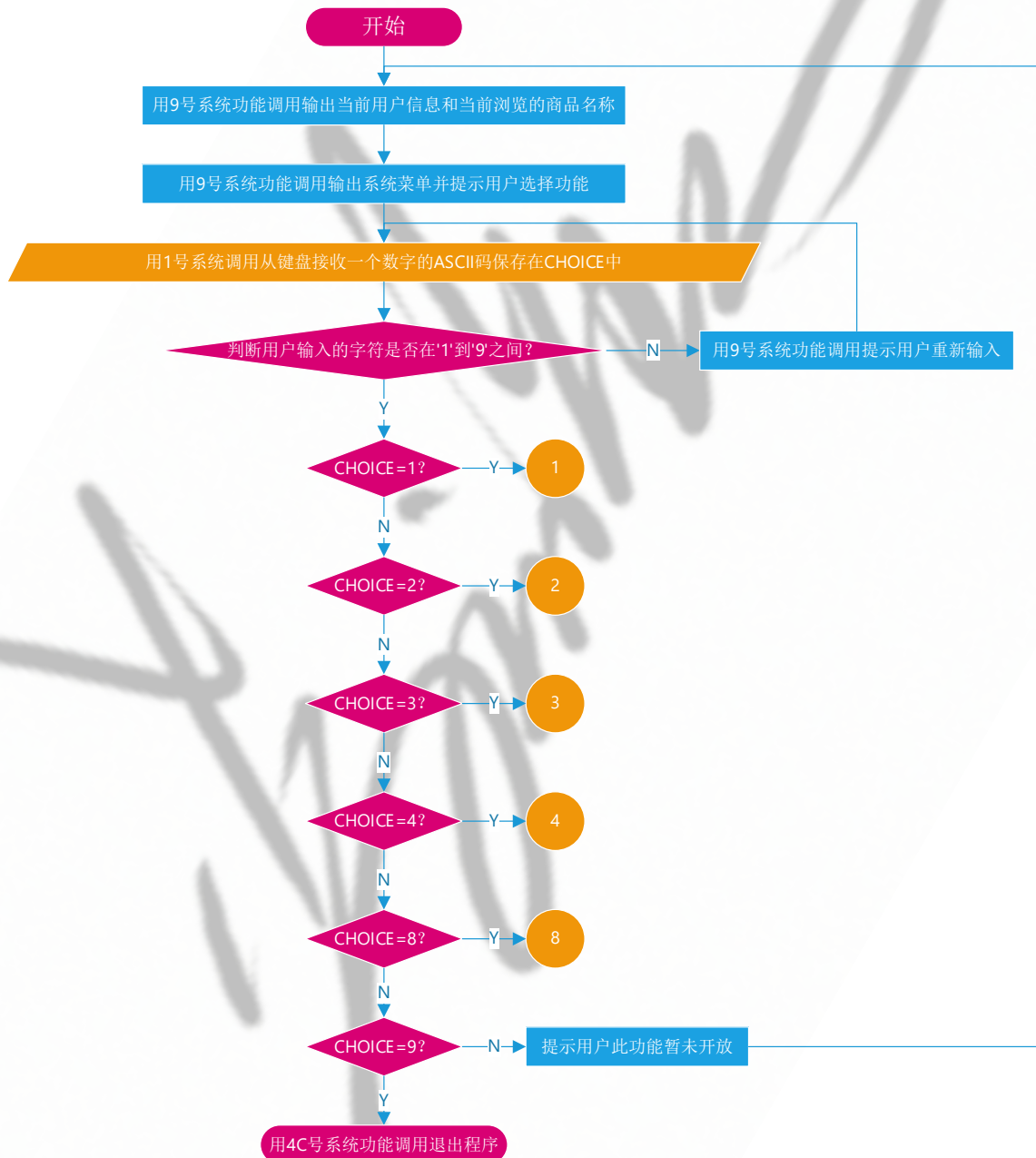


图 1.16 系统整体设计的流程图

汇编语言程序设计实验报告

4. 功能1（老板登录）的流程图

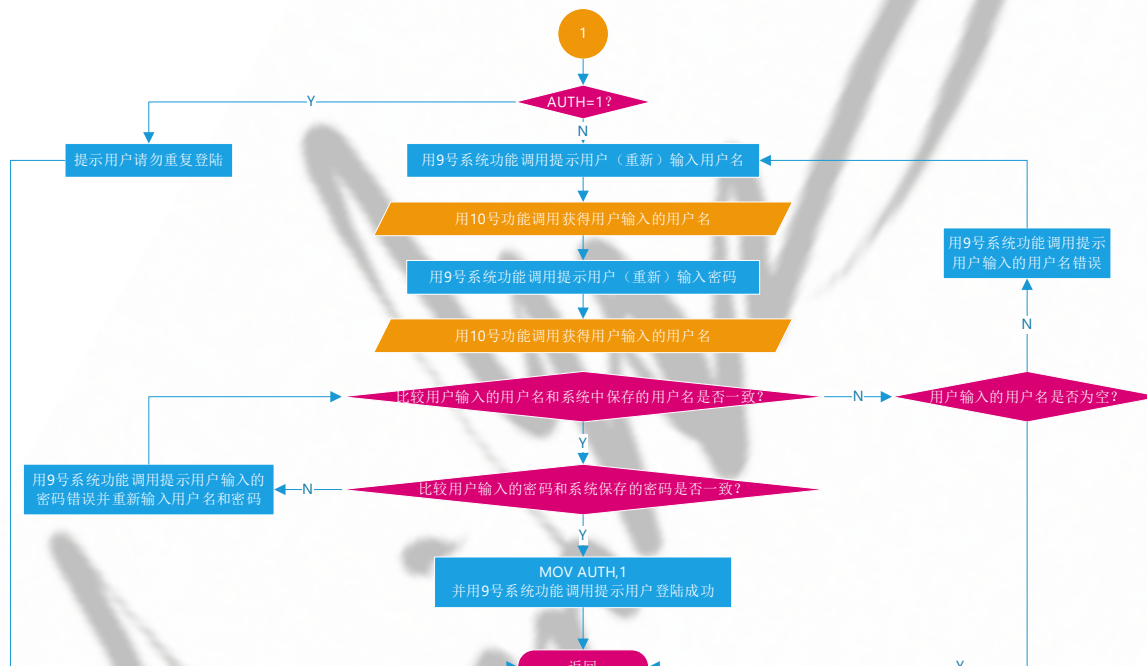


图 1.17 登录功能的流程图

5. 功能二（搜索商品）的流程图

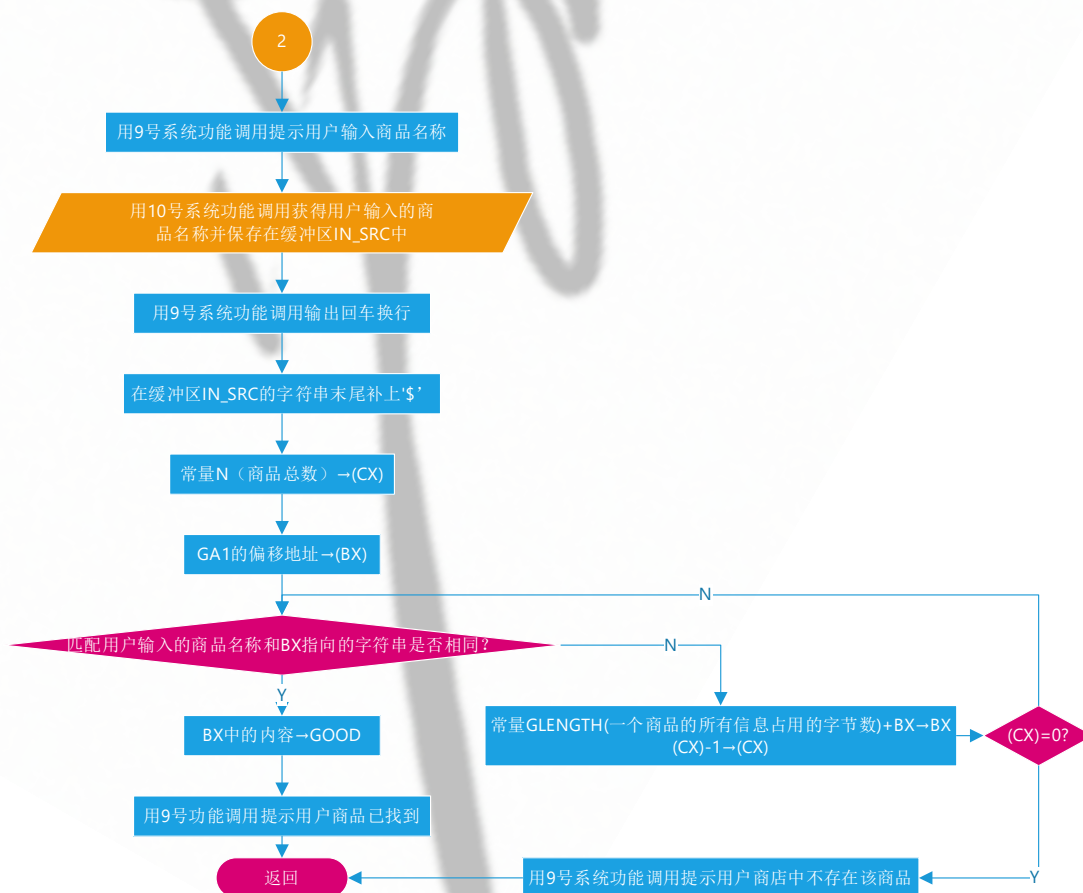


图 1.18 搜索商品功能的流程图

汇编语言程序设计实验报告

6. 功能三（下订单）和功能四（计算商品推荐度）的流程图

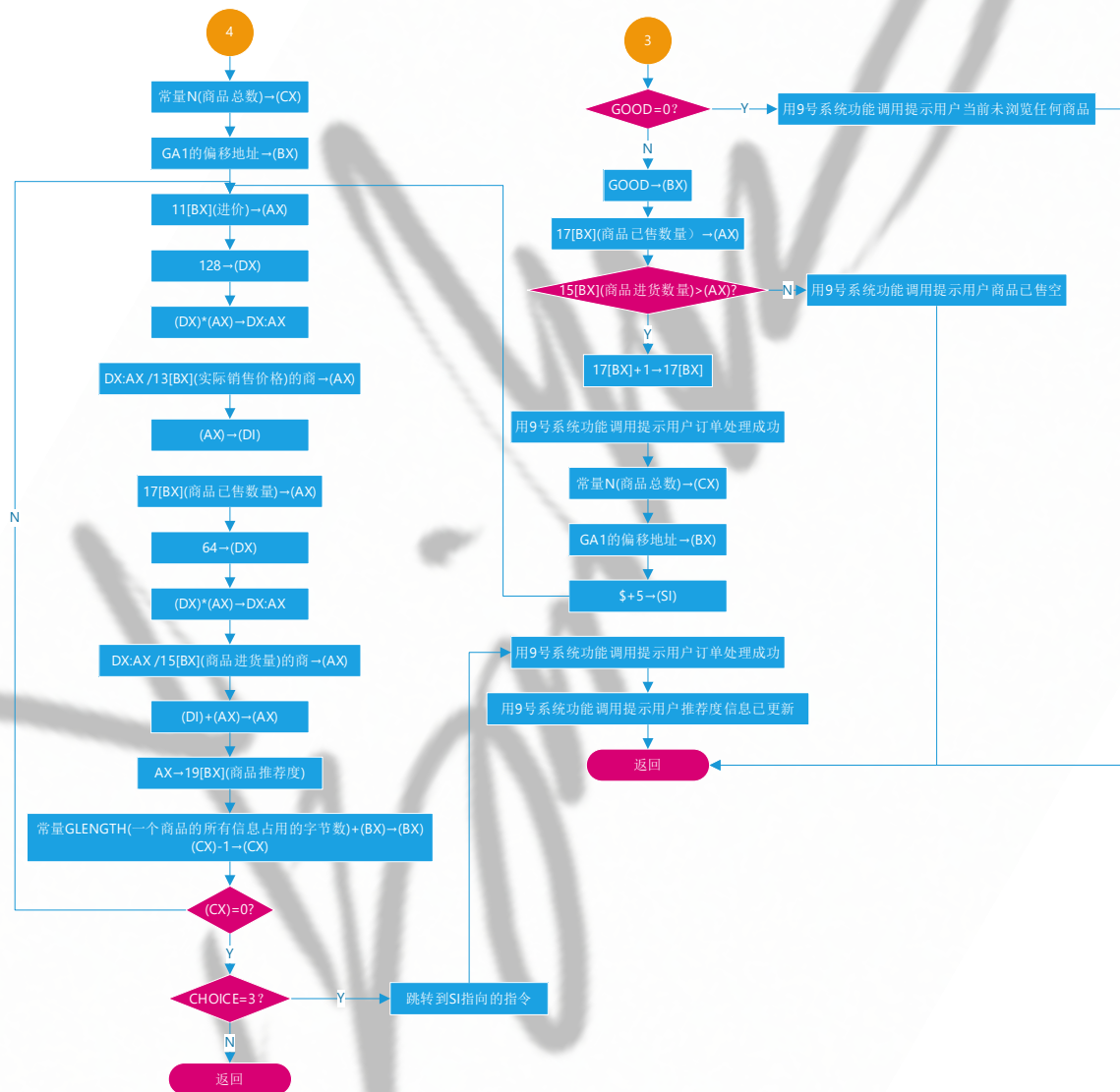


图 1.19 下订单功能和计算商品推荐度功能的流程图

汇编语言程序设计实验报告

7. 功能八（以 16 进制输出代码段首址）的流程图

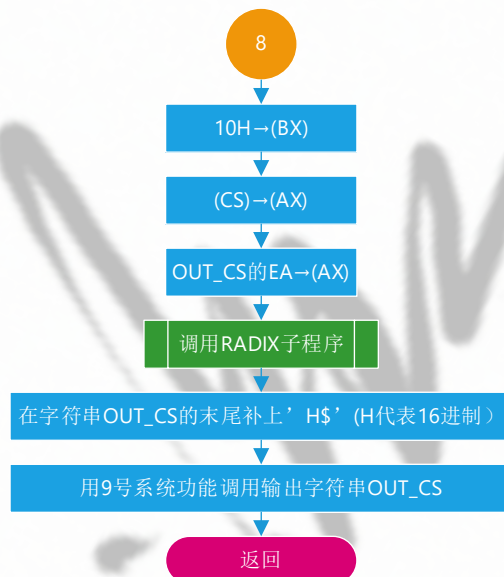


图 1.20 输出代码段首址的流程图

1.6.3 源程序

```
.386
STACK SEGMENT USE16    STACK
    DB 200 DUP(0)
STACK ENDS

DATA SEGMENT USE16
BNAME DB 'LI YONGKANG',0
NLENGTH EQU 11
BPASS DB 'LOVECS',0
PLENGTH EQU 6
AUTH DB 0
GOOD DW 0
N EQU 30
SNAME DB 'ASMSHOP',0
GA1 DB 'EDITOR', 4 DUP(0), 10
    DW 35, 56, 70, 25, ?
GA2 DB 'COMPILER', 2 DUP(0), 9
    DW 12, 30, 25, 5, ?
GA3 DB 'IDE', 7 DUP(0), 8
    DW 50, 70, 20, 10, ?
GAN DB N-3 DUP('TEMPVALUE', 0, 8, 15, 0, 20, 0, 30, 0, 2, 0, ?, ?)

GLENGTH EQU 21

CRLF DB 0DH, 0AH, '$'
FORMAT DB ' ', 0AH
    DB '-----Welcome to ASMSHOP-----', 0AH, '$'
CSTM DB 'Current User: Customer', 0AH, '$'
BOSS DB 'Current User: LI YONGKANG', 0AH, '$'
BRSING DB 'You are browsing ', '$'
MENU DB '-----', 0AH
    DB ' | MAIN MENU |', 0AH
```

汇编语言程序设计实验报告

```

DB ' |                                     |', 0AH
DB ' |             1.log in                |', 0AH
DB ' |             2.search commodities    |', 0AH
DB ' |             3.place an order        |', 0AH
DB ' |             4.compute recommendation |', 0AH
DB ' |             5.update rankings       |', 0AH
DB ' |             6.update information    |', 0AH
DB ' | 7.migrate to operation environment |', 0AH
DB ' | 8.display the offset address of CS |', 0AH
DB ' |             9.exit the system      |', 0AH
DB ' -----', 0AH, '$'

CHOICE DB 0
DB 'Please choose a function:', '$'
LOGIN  DB 'You have not logged in yet.', 0AH
DB 'Please log in first.', 0AH, '$'
NAMEH  DB 'Please input your name:$'
IN_NAME DB 25
DB ?
DB 25 DUP(0)
PWDH   DB 'Please input your password:$'
IN_PWD DB 40
DB ?
DB 40 DUP(0)
SRCH   DB 'Please input your name of the commodity you are going to search:$'
IN_SRC DB 20
DB ?
DB 20 DUP(0)

SUCCESS DB 'You have logged in successfully!', 0AH, '$'
RPT     DB 'You have already logged in. Do not repeate.', 0AH, '$'
SORRY   DB 'Sorry, this function is waiting for implementation.', 0AH, '$'
AGAIN   DB 'Invalid input, please make your choice again:', '$'
WNH     DB 'Wrong Username, unable to log in.', 0AH
DB 'Please log in again.', 0AH, '$'
WPH     DB 'Wrong Password, unable to log in.', 0AH
DB 'Please log in again.', 0AH, '$'
NFOUND  DB 'Sorry, we do not have this.', 0AH, '$'
FOUND   DB ' is found.', 0AH, '$'
BRNS    DB 'You have not chosen any product.', 0AH
DB 'Search a commodity before making an order.', 0AH, '$'
ORMD    DB 'Your order is processed successfully.', 0AH, '$'
CMPTED  DB 'All recommendation computation is done.', 0AH, '$'
UPDATED DB 'All recommendation information is updated.', 0AH, '$'
SOLD    DB ' is sold out. We are sorry about this.', 0AH, '$'

OUT_CS DB 6   DUP(0)
DATA   ENDS

CODE   SEGMENT USE16
ASSUME CS:CODE, DS:DATA, SS:STACK
START: MOV AX, DATA
        MOV DS, AX

MAIN:   ;dos 功能调用输出主菜单格式
        LEA DX, FORMAT
        MOV AH, 09H
        INT 21H

```


汇编语言程序设计实验报告

```
;判断当前用户是顾客还是老板，以输出当前用户信息
CMP AUTH, 0
JNE OBOSS
;dos 功能调用输出当前用户信息（顾客）
LEA DX, CSTM
MOV AH, 09H
INT 21H
JMP OGOOD
;dos 功能调用输出当前用户信息（老板）
OBOSS: LEA DX, BOSS
MOV AH, 09H
INT 21H
;dos 功能调用输出当前浏览商品信息
OGOOD: LEA DX, BRSING
MOV AH, 09H
INT 21H
MOV SI, 0
CMP GOOD, 0
JE MN
MOV BX, GOOD
GNAME: MOV DL, [BX][SI]
CMP DL, 0
JE MN
MOV AH, 02H
INT 21H
INC SI
JMP GNAME
;dos 功能调用输出功能选择菜单
MN: MOV DL, 0AH
MOV AH, 02H
INT 21H
LEA DX, MENU
MOV AH, 09H
INT 21H

;dos 功能调用提示用户选择功能
LEA DX, CHOICE+1
MOV AH, 09H
INT 21H

CHOOSE: ;dos 功能调用获得用户想选择的功能
MOV AH, 01H
INT 21H
MOV CHOICE, AL
LEA DX, CRLF
MOV AH, 09H
INT 21H
;判断是否需要退出程序
CMP CHOICE, '9'
JE EXIT

;判断用户输入的有效性
CMP CHOICE, '1'
JB INVALID
CMP CHOICE, '9'
JB VALID
```

汇编语言程序设计实验报告

INVALID: ;用户输入的功能不合法, 提示用户重新输入

```
LEA DX, AGAIN
MOV AH, 09H
INT 21H
JMP CHOOSE
```

VALID: ;根据用户选择功能进行跳转

;判断用户是否选择了功能一

```
CMP CHOICE, '1'
JNE F2
```

F1: ;判断用户是否已经登陆过, 若是, 提示用户请勿重复登录

```
CMP AUTH, 1
JNE LOG
```

```
LEA DX, RPT
```

```
MOV AH, 09H
```

```
INT 21H
```

```
JMP MAIN
```

;dos 功能调用提示用户输入用户名

LOG: LEA DX, NAMEH

```
MOV AH, 09H
```

```
INT 21H
```

;dos 功能调用获得用户姓名

```
LEA DX, IN_NAME
```

```
MOV AH, 0AH
```

```
INT 21H
```

```
LEA DX, CRLF
```

```
MOV AH, 09H
```

```
INT 21H
```

```
MOV BL, IN_NAME+1
```

```
MOV BH, 0
```

```
MOV BYTE PTR IN_NAME+2[BX], '$'
```

;dos 功能调用提示用户输入密码

```
MOV DX, OFFSET PWDH
```

```
MOV AH, 09H
```

```
INT 21H
```

;dos 功能调用获得用户密码

```
LEA DX, IN_PWD
```

```
MOV AH, 0AH
```

```
INT 21H
```

```
LEA DX, CRLF
```

```
MOV AH, 09H
```

```
INT 21H
```

```
MOV BL, IN_PWD+1
```

```
MOV BH, 0
```

```
MOV BYTE PTR IN_PWD+2[BX], '$'
```

```
CMP IN_NAME+1, 0
```

```
JNE CMPNAME
```

```
MOV AUTH, 0
```

```
JMP MAIN
```

CMPNAME: ;验证用户输入的姓名是否正确

汇编语言程序设计实验报告

```
;不正确则跳转到 WRONGN  
;正确则跳转到 CMPWD (继续比较密码)  
CMP IN_NAME+1, NLENGTH  
JNE WRONGN  
MOV CX, NLENGTH  
CNL:  MOV SI, CX  
      MOV AL, BNAME-1[SI]  
      CMP AL, IN_NAME+1[SI]  
      JNE WRONGN  
      DEC CX  
      JNZ CNL  
      JMP CMPPWD
```

```
WRONGN: ;提示用户输入的姓名错误  
        LEA DX, WNH  
        MOV AH, 09H  
        INT 21H  
        JMP F1
```

```
CMPPWD: ;验证用户输入的密码是否正确  
        ;不正确则跳转到 WRONGP  
        ;正确则跳转到 RIGHT (登录成功)  
        CMP IN_PWD+1, PLENGTH  
        JNE WRONGP  
        MOV CX, PLENGTH  
CPL:    MOV SI, CX  
        MOV AL, BPASS-1[SI]  
        CMP AL, IN_PWD+1[SI]  
        JNE WRONGP  
        DEC CX  
        JNZ CPL  
        JMP RIGHT
```

```
WRONGP: ;提示用户输入的密码错误  
        LEA DX, WPH  
        MOV AH, 09H  
        INT 21H  
        JMP F1
```

```
RIGHT:  ;dos 功能调用提示用户登录成功  
        LEA DX, SUCCESS  
        MOV AH, 09H  
        INT 21H  
        MOV AUTH, 1  
        CMP CHOICE, '1'  
        JE MAIN
```

```
F2:      ;判断用户是否选择功能二  
        CMP CHOICE, '2'  
        JNE F3  
        ;dos 功能调用提示用户输入商品名称  
        MOV DX, OFFSET SRCH  
        MOV AH, 09H  
        INT 21H  
  
        ;dos 功能调用获得商品名称  
        LEA DX, IN_SRC
```

汇编语言程序设计实验报告

```
MOV AH, 0AH
INT 21H
LEA DX, CRLF
MOV AH, 09H
INT 21H
MOV BL, IN_SRC+1
MOV BH, 0
MOV BYTE PTR IN_SRC+2[BX], '$'

;依次检查每一个商品的名称
;是否能和用户输入的名称匹配
MOV CX, N
MOV BX, OFFSET GA1
GOODS: MOV DI, 0
CMPG:  MOV AL, [BX][DI]
        CMP AL, 0
        ;检查到商品名称的末尾
        ;去检查输入缓冲区此处是否是'$'
        JE MARGIN
        CMP AL, IN_SRC+2[DI]
        JNE NMATCH
        INC DI
        JMP CMPG
NMATCH: ADD BX, GLENGTH
        LOOP GOODS

MARGIN: CMP BYTE PTR IN_SRC+2[DI], '$'
        ;两个字符串同时结束且完全匹配
        JE ISFOUND

        ;dos 功能调用提示用户未找到商品
        LEA DX, NFOUND
        MOV AH, 09H
        INT 21H
        JMP MAIN

        ;dos 功能调用提示用户找到商品
        ;并更改数据段中 GOOD 的值
ISFOUND: MOV GOOD, BX
        LEA DX, IN_SRC+2
        MOV AH, 09H
        INT 21H
        LEA DX, FOUND
        MOV AH, 09H
        INT 21H
        JMP MAIN

F3:      ;判断用户是否选择了功能三
        CMP CHOICE, '3'
        JNE F4
        ;判断当前是否有正在浏览的商品
        CMP GOOD, 0
        JNE CTN
        LEA DX, BRSN
        MOV AH, 09H
        INT 21H
        JMP MAIN
```


汇编语言程序设计实验报告

```
CTN:      ;判断当前商品是否有剩余
MOV BX, GOOD
MOV AX, 17[BX]
CMP 15[BX], AX
JA ORDER
;提示用户商品已售空
MOV BX, GOOD
MOV SI, 0
CNAME: MOV DL, [BX][SI]
CMP DL, 0
JE SOLDH
MOV AH, 02H
INT 21H
INC SI
JMP CNAME
SOLDH: LEA DX, SOLD
MOV AH, 09H
INT 21H
JMP MAIN

ORDER: INC WORD PTR 17[BX]
MOV CX, N
MOV BX, OFFSET GA1
MOV SI, $+5 ;将“输出订单处理成功的提示信息”的指令地址保存在 SI 中
JMP ALLG ;跳转到功能四中去计算商品推荐度

;输出订单处理成功的提示信息
LEA DX, ORMD
MOV AH, 09H
INT 21H
;dos 功能调用提示用户推荐度信息已经更新
LEA DX, UPDATED
MOV AH, 09H
INT 21H

JMP MAIN ;返回到主菜单

F4:      ;判断用户是否选择了功能四
CMP CHOICE, '4'
JNE F5
MOV CX, N ;CX 中存放商品总数
MOV BX, OFFSET GA1 ;BX 指向第一个商品
ALLG: MOV AX, WORD PTR 11[BX] ;商品的进价->AX
MOV DX, 128
MUL DX ;DX*AX->DX:AX //MUL 只接一个操作数
DIV WORD PTR 13[BX] ;DX: AX / 售价 的商->AX //DIV 只接一个操作数
MOV DI, AX;
MOV AX, WORD PTR 17[BX] ;商品的已售数量->AX
MOV DX, 64
MUL DX ;DX*AX->DX:AX //MUL 只接一个操作数
DIV WORD PTR 15[BX] ;DX: AX / 进货数量 的商->AX //DIV 只接一个操作数
ADD AX, DI ;两次计算的商相加既是推荐度
MOV 19[BX], AX ;将计算结果保存在内存中的对应单元
ADD BX, GLENGTH ;使 BX 指向下一商品
DEC CX
JNZ ALLG
```

汇编语言程序设计实验报告

```
CMP CHOICE, '3'          ;判断这次推荐度计算是否是由功能三引起的
JE MEDIA                 ;//条件转移指令不能直接接 SI，要做一个中转
;dos 功能调用提示用户推荐度信息计算完成
LEA DX, CMPTED
MOV AH, 09H
INT 21H
JMP MAIN

MEDIA: JMP SI

F5:      ;判断用户是否选择了功能五
CMP CHOICE, '5'
JNE F6
;dos 功能调用提示用户此功能暂未开放
LEA DX, SORRY
MOV AH, 09H
INT 21H
JMP MAIN

F6:      ;判断用户是否选择了功能六
CMP CHOICE, '6'
JNE F7
;dos 功能调用提示用户此功能暂未开放
LEA DX, SORRY
MOV AH, 09H
INT 21H
JMP MAIN

F7:      ;判断用户是否选择了功能七
CMP CHOICE, '7'
JNE F8
;dos 功能调用提示用户此功能暂未开放
LEA DX, SORRY
MOV AH, 09H
INT 21H
JMP MAIN

F8:      ;判断用户是否选择了功能八
CMP CHOICE, '8'
JNE EXIT
;调用 RADIX 子程序计算输出的字符串
MOV BX, 10H             ;转化的进制数(16)->BX
MOV AX, CS              ;代码段首址->AX
LEA SI, OUT_CS          ;输出缓冲区首址->SI
CALL RADIX
MOV BYTE PTR [SI], 'H'  ;在输出缓冲区末尾增加 16 进制后缀
INC SI
MOV BYTE PTR [SI], '$'  ;在输出缓冲区末尾添加'$'
;dos 功能调用输出代码段首址
LEA DX, OUT_CS
MOV AH, 09H
INT 21H
JMP MAIN

EXIT:    ;4CH 号系统功能调用退出程序
MOV AH, 4CH
INT 21H
```

汇编语言程序设计实验报告

```
RADIX  PROC
        PUSH CX    ;现场保护
        PUSH DX

        MOV CX, 0    ;记录压入堆栈中的字节数
LOP1:   MOV DX, 0
        DIV BX        ;DX:AX / BX 的余数->DX
        PUSH DX
        INC CX
        CMP AX, 0     ;判断 DX:AX / BX 的商是否为 0
        JNE LOP1
        ;将刚刚压入堆栈中的数依次弹出并转化为对应字符
LOP2:   POP AX
        CMP AL, 0AH
        JB NUM
        ADD AL, 07H
NUM:    ADD AL, 30H
        MOV [SI], AL
        INC SI
        LOOP LOP2

        POP DX        ;恢复现场
        POP CX
        RET
RADIX  ENDP

CODE   ENDS
        END START
```

1.6.4 实验步骤

1. 准备上机实验环境。
2. 使用编辑程序 EDIT.EXE 录入源程序，存盘文件名为 TASK4.ASM。使用 MASM 6.0 汇编源文件。即 MASM TASK4.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 TASK4.OBJ 文件连接成执行文件。
即 LINK TASK4;
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 TASK4.EXE 文件。
4. 执行该程序。即在命令行提示符后输 TASK4E 后回车。测试系统的各种功能是否都能正常运行。
5. 使用 TD.EXE 观察 TASK4 的执行情况。即 TD TASK4.EXE 回车。
 - (1) 观察程序中的各个分支结构的具体执行情况。
 - (2) 观察程序中的各个循环结构的具体执行情况。
 - (3) 重点观察功能三中跳转到功能四后能否回到功能三中该回去的地方，这个过程观察 SI 寄存器（用来存放指令地址）中值的变化。
6. 测试各种特殊情况（比如输入的字符超过缓冲区大小、正确的用户名和密码是输入的用户

汇编语言程序设计实验报告

名或密码的一个子集、商品已售数量不小于进货量等), 检验程序是否能正常运行。

1.6.5 实验记录与分析

1. 实验环境条件: i7 1.8GHz; WINDOWS10 下 DOSBox0.74; TD.EXE 5.0。
2. 汇编源程序时, 汇编程序报了 5 个错。它们是:
TASK4.asm(363): error A2008: syntax error :,
TASK4.asm(364): error A2008: syntax error :,
TASK4.asm(368): error A2008: syntax error :,
TASK4.asm(369): error A2008: syntax error :,
TASK4.asm(376): error A2077: instruction does not allow NEAR indirect addressing
按照提示, 对对应行仔细检查, 发现了错误原因。
3. 连接过程没有发生异常
4. 执行之后按照预期输出了当前用户信息、当前浏览商品和主菜单, 如图 1.21 所示。

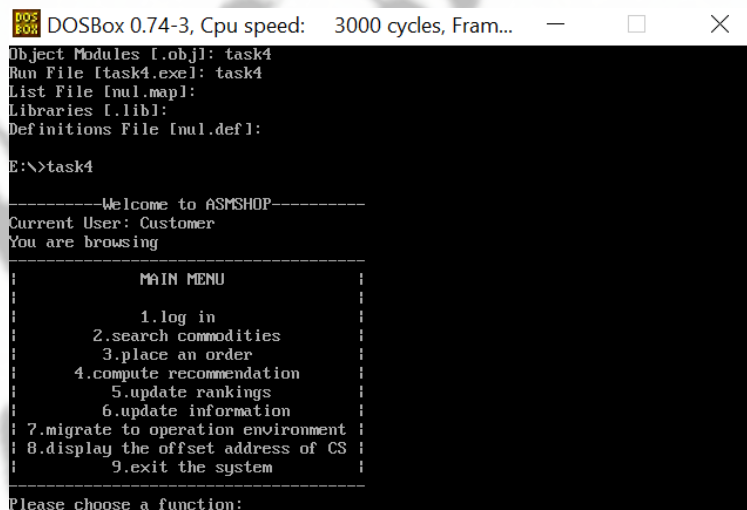


图 1.21 刚执行程序的界面

5. 选择功能时输入非 ‘1’ – ‘9’ 之间的字符
测试结果如图 1.22 所示。

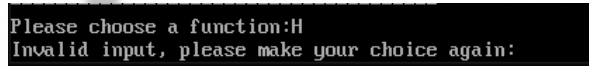


图 1.22 输入非法功能键

5. 登录功能测试
输入正确用户名和正确密码, 测试结果如图 1.23 所示。

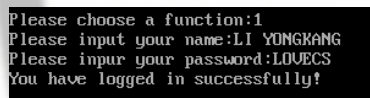


图 1.23 输入正确的用户名和密码

- 输入错误的用户名和正确的密码, 测试结果如图 1.24 所示。

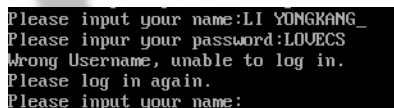


图 1.24 输入错误的用户名和正确的密码

汇编语言程序设计实验报告

输入正确的用户名和错误的密码，测试结果如图 1.25 所示。

```
Please choose a function:1
Please input your name:LI YONGKANG
Please input your password:LOUECS_
Wrong Password, unable to log in.
Please log in again.
Please input your name:_
```

图 1.25 输入正确的用户名和错误的密码

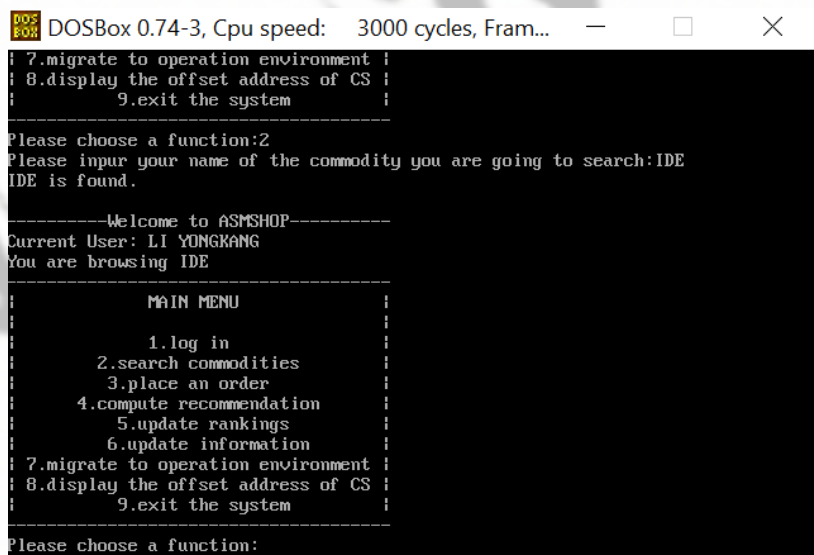
登录成功后再次请求登录，测试结果如图 1.26 所示。

```
Please choose a function:1
You have already logged in. Do not repeate.
```

图 1.26 重复登陆

6. 搜索商品功能测试

搜索商店中存在的商品 IDE，测试结果如图 1.27 所示。



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...
7.migrate to operation environment :
8.display the offset address of CS :
9.exit the system :
-----
Please choose a function:2
Please input your name of the commodity you are going to search:IDE
IDE is found.
-----Welcome to ASMSHOP-----
Current User: LI YONGKANG
You are browsing IDE
-----
MAIN MENU
-----
1.log in
2.search commodities
3.place an order
4.compute recommendation
5.update rankings
6.update information
7.migrate to operation environment :
8.display the offset address of CS :
9.exit the system :
-----
Please choose a function:
```

图 1.27 搜索 IDE

搜索商店中不存在的商品 watermelon，测试结果如图 1.28 所示。

```
Please choose a function:2
Please input your name of the commodity you are going to search:watermelon
Sorry, we do not have this.
```

图 1.28 搜索 watermelon

7. 下订单功能测试

未浏览商品时下订单，测试结果如图 1.29 所示。

```
Please choose a function:3
You have not chosen any product.
Search a commodity before making an order.
```

图 1.29 未浏览商品时下订单

有正在浏览的商品时下订单，测试结果如图 1.30 所示。

```
Please choose a function:3
Your order is processed successfully.
All recommendation information is updated.
```

图 1.30 有正在浏览的商品时下订单

当重复购买一件商品超出库存数量，测试结果如图 1.31 所示。

```
Please choose a function:3
IDE is sold out. We are sorry about this.
```

图 1.31 商品已售空

汇编语言程序设计实验报告

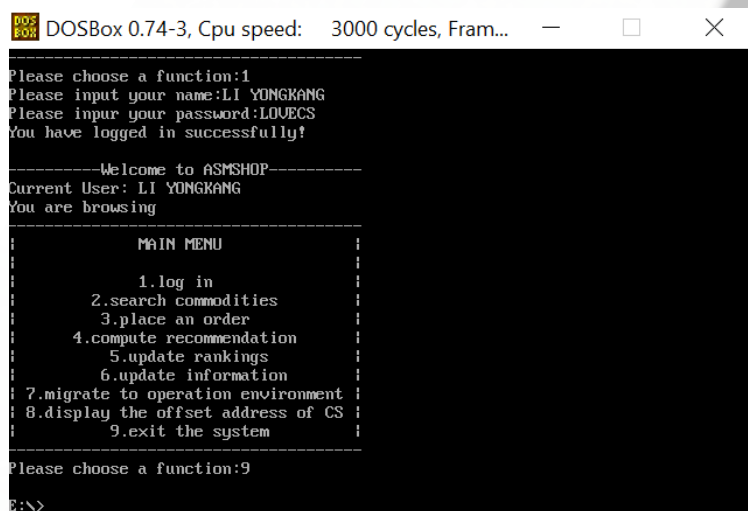


图 1.36 退出系统

11. 功能三在计算商品推荐度之后返回功能三

SI 寄存器指向准备返回的指令，如图 1.37 所示。

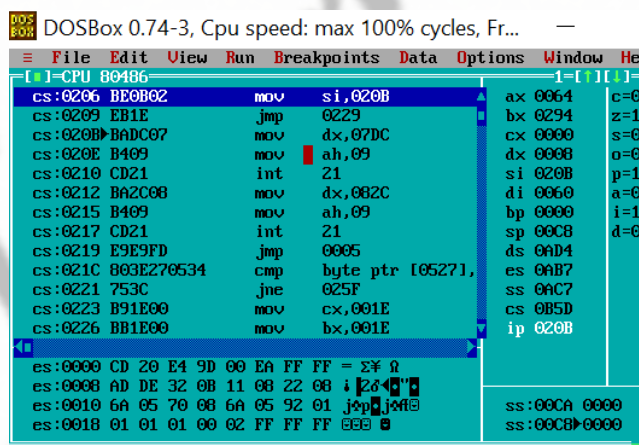


图 1.37 SI 指向将要返回的指令位置

可以看出 SI 指向的指令就是计算完商品推荐度后返回到功能三应该执行的下一条指令。

12. 10 号系统功能调用时，输入的长度超出缓冲区长度

当输入超出缓冲区长度时，超出的部分不会在屏幕上显示，此时按下回车键也不能结束输入，必须删掉一个字符以保留结束字符所需要的空间，测试结果如图 1.38 所示。

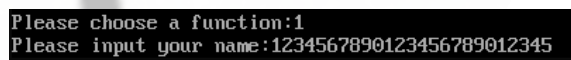


图 1.38 输入超出缓冲区长度

13. 在 9 号功能调用时，待显示字符串的结尾没有“\$”结束符

程序会将后面的字节依次以字符形式输出，直至遇到‘\$’结束符，测试结果如图 1.39 所示。

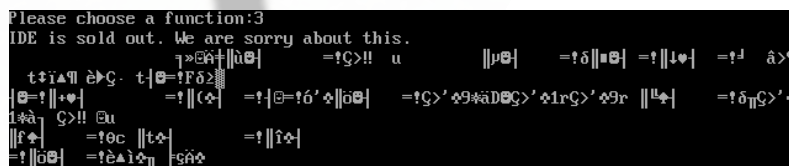


图 1.39 待显示的字符串没有‘\$’结束符

14. 在 9 号功能调用前，未对 DS 赋值

汇编语言程序设计实验报告

在刚进入 TD 调试工具，还未对 DS 赋值的时候，记下来 DS 中内容为 0AB7，赋值后 DS 中内容为 0AD4，数值上二者相差 29，乘上 16 后二者相差 464 个字节。在提示用户选择功能前将 DS 手动修改为 0AB7，然后进行 9 号系统功能调用，根据计算，应该从前 464 个字节开始输出，直至结束符停止，根据数据段的定义，应该会从菜单中间某一位置开始重新输出一遍菜单。测试结果如图 1.40 所示。

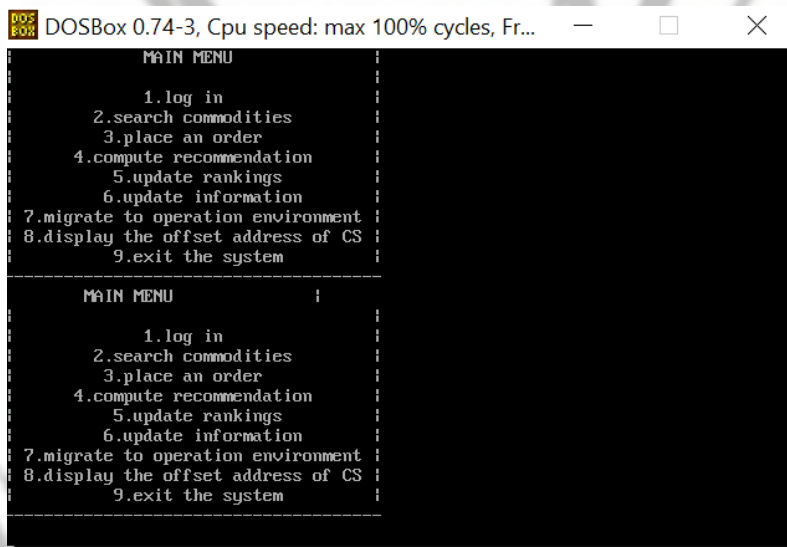


图 1.40 9 号系统功能调用时未给 DS 赋值

可见测试结果和理论结果一致。

15. 选取特殊的值，观察计算结果溢出的情况。

手动修改 DX 和 AX 寄存器中的值为 0FFFFH，再执行 DIV 指令，会跳转到非 task4 的指令，然后进入一个死循环，测试结果如图 1.41 所示。

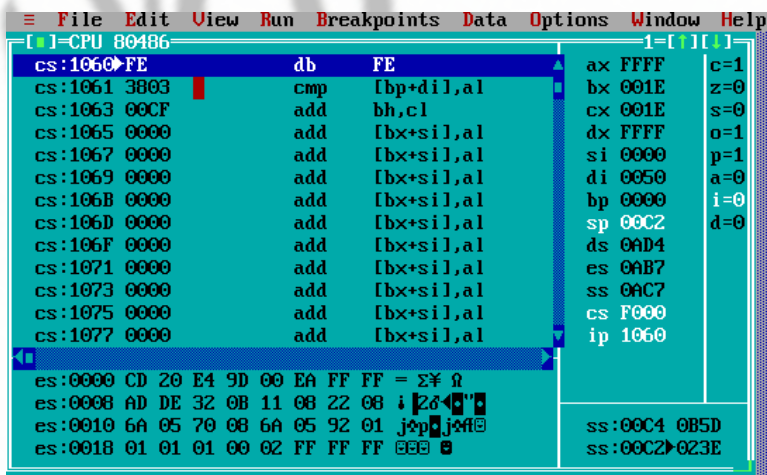


图 1.41 计算结果溢出

1.7 小结

第一周的上机实验是汇编程序设计实验的第一次实验，最重要的意义在于学会了如何在 DOS 系统下汇编源文件、如何生成可执行文件，如何用 TD 来调试等。

在任务一中，学会了直接在 TD 中输入指令并按下 F8 键单步执行，并且直接可以观察或者修改

汇编语言程序设计实验报告

各寄存器和标志位的值，除此以外，还理解了指令执行和标志位改变之间的关系，并深刻体会到了溢出和进位的差异；在任务二中，学会了如何查看数据段和堆栈段中的数据，也学会了如何设置断点并且运行到断点，还体会到在程序一开始将 DATA 段首址送到 DS 寄存器中的重要意义；在任务三中，通过反汇编指令可以探寻到计算机在汇编源文件时的蛛丝马迹，也知道了计算机是通过 IP/EIP 保证程序按照指令正确运行的，除此以外，还体会到我们在编写源程序时需要灵活使用寻址方式，而且这些寻址方式之间也可以灵活地进行转换。

这次实验过程中也存在一些不足，在以后的实验过程中还需要不断改进。一个很明显的问题就是对汇编语言的各种指令不是很熟，阅读代码不是很熟练，这一点上还需要多多改善，毕竟熟能生巧。除此以外，对于调试工具的使用也不是很熟练，一开始都不知道如何查看数据段和堆栈段的数据，也不知道如何单步运行，好在后来慢慢地也清楚了如何使用 TD 调试工具。

总体来看，第一周的上机实验过程中收获了很多，也基本达到了实验目的和要求，即掌握了各种工具的使用、理解了数、符号、寻址方式等在计算机内的表达形式以及理解了指令执行与标志位改变的关系等，而实验过程中出现的问题和不足也要在后面的实验过程中多加克服。

而在第二周的上机实验中，我们编写了自己的第一个汇编语言源程序，让我经历了用汇编语言编写源程序，然后汇编、连接生成可执行程序的全流程，这个过程中让我熟悉了 DOS 系统各种常见的功能调用和汇编语言程序设计的基本框架，尤其是分支结构和循环结构，熟悉了这两个结构，自然也加深了对各种转移指令的理解。除此以外，这次上机也让我对汇编语言程序设计有了更深刻的认识。首先，对于一些基本指令要多记，记不住就要多查。拿本次实验来说，在实现计算商品推荐度的时候我想当然地以为 MUL 和 DIV 的指令格式与 ADD 和 SUB 的指令格式一样，在编译的时候就报了语法错误，然后查了书上的附录才知道这两个指令的具体用法。除此以外，对于一个指令影响的标志位也要多查表，尤其是记不太清楚的时候，否则的话，虽然汇编和连接没问题，但是程序可能会有逻辑错误并导致运行错误。其次，汇编语言程序设计有很多细节需要特别注意。拿本次实验来说，在用 \$ 从功能四返回到功能三时，直接用了 JE SI 指令，这很显然会报错，后来记起来老师上课讲过，条件转移指令不能这样做，需要用一个标号做一个中转，这样就解决了这个问题。

总的来说，第二周的上机实验也是收获颇丰，虽然过程一波三折，但正是实验过程中出现的各种问题让我们有了更深刻的体会和感悟，在后续的实验课程要继续再接再厉，摸清楚汇编语言程序设计中的各种门道。

2 程序优化

2.1 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。

2.2 实验内容

任务 2.1 观察多重循环对 CPU 计算能力消耗的影响。

请通过适当修改任务 1.4 的程序，完成如下研究：

1. 请描述并实现对一段代码的执行时间进行测量的方法。该方法应能观察到程序中一条指令发生修改时，程序完成同样功能时的执行时间的变化。

2. 通过在不同软硬件运行环境下运行同一个程序，观察程序执行时间是否会随之发生变化。

对于任务 1.4，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任务 1.4 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，我们让任务 1.4 的“下订单”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。

调整后的下订单的描述：

【前提条件】用户查询到了商品“Bag”，在主菜单界面下选择了“3”对应的下订单功能。

《下订单》

(1) 判断当前浏览商品是否有效（GOOD 不为空），若有效，判断其剩余数量是否为 0，不为 0 则将已售数量加 1，重新计算所有商品的推荐度。

返回主菜单界面

(2) 若无效或剩余数量为 0，则提示错误，回到主菜单界面。

任务 2.2 对任务 2.1 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 2.1 的程序时，尽量不要考虑代码优化的问题）。

2.3 任务 2.1 实验过程

2.3.1 实验方法说明

1. 准备上机环境，编辑、汇编、连接文件 TASK1。
2. 运行程序，测试下订单操作耗时。
3. 修改 m 和 n 的值，测试在不同的 m 和 n 的值的条件下下订单操作的耗时。

汇编语言程序设计实验报告

4. 在下订单操作中增加 2 号和 9 号调用，测试其对 CPU 资源的消耗情况。
5. 在不同的软硬件环境下运行同样的程序（即重复 2、3、4 步），测试下订单操作耗时。

2.3.2 实验记录与分析

1. 实验环境条件：i7 1.8GHz；WINDOWS10 下 DOSBox0.74；TD.EXE 5.0。
2. 调试程序后运行程序，程序正确运行，并返回总耗时，如图 2.1 所示。

```
Please choose a function:3
Time elapsed in ms is 2250
```

图 2.1 m=1000, n=30 时程序耗时

3. 修改 m 和 n 的值，测试程序耗时，测试结果如表 2.1 所示。

表 2.1 m 和 n 的值不同时程序耗时（单位：ms）

n \ m	10000	20000	30000	60000
30	2250	4560	6760	13570
60	4450	8960	13410	26750
120	8840	17680	26590	53170

分析：由上表可知，在误差允许范围内，下订单操作的耗时基本与 $m*n$ 成正比。

4. 测试在不同软硬件环境下运行同样的程序的耗时。

对比组实验环境条件：Kirin659 2.36GHz；Android 8.0 下 Dosbox Turbo for Android。
m=10000, n=30 时测试结果如图 2.2 所示。

```
6.update information
7.migrate to operation environment
8.display the offset address of CS
9.exit the system
-----
Please choose a function:3
Time elapsed in ms is 160
```

图 2.2 不同软硬件条件下相同程序运行耗时对比

分析：在不同的软硬件环境下重复之前的测试步骤可以知道，程序的运行环境也对程序的运行效果有很大影响。经过对比可知，在安卓虚拟机上程序跑得比在 Windows 下 Dosbox 中快得多。个人认为原因可能有两点：一方面，Kirin659 的主频比 i7 的主频大一点；另一方面，Windows 上还同时运行很多其他程序，操作系统需要在多个程序之间进行调度，增加了程序的耗时。

2.4 任务 2.2 实验过程

2.4.1 实验方法说明

优化方法：

- （1）每次循环结束后直接将 BX 寄存器中的内容修改为下一商品的首址，减少了 CX 寄存器的使用，也去掉了一条冗余的代码。
- （2）将乘以 64 的指令改为移位指令。
- （3）由于每次除法结果都不会超过 65536，EAX 寄存器的高 16 位一定为 0，因而在下一次除法操作之前不需要将 EAX 寄存器清零，直接将 WORD 类型的数据移动到 AX 寄存器中即可，这样又删除了两条冗余指令。

汇编语言程序设计实验报告

(4) 返回 3 号功能时不再使用寄存器跳转，而是直接使用标号跳转，提供了更多可用的寄存器，不再需要出栈入栈操作。

实验步骤：

1. 准备上机环境，编辑、汇编、连接文件 TASK2。
2. 运行程序，测试下订单操作耗时，并计算优化率。
3. 将循环体修改为使用子程序，再汇编、连接文件 TASK2_，然后运行程序，并测试下订单操作的耗时。
4. 对于在 TASK1 源文件的基础上的优化进行逐条测试，测量下订单操作的耗时，并计算优化率，然后分析哪些指令的优化对于程序的运行时间有比较大的影响。

2.4.2 实验记录与分析

1. 实验环境条件：i7 1.8GHz；WINDOWS10 下 DOSBox0.74；TD.EXE 5.0。
2. 为减小优化率计算误差，将 m 设为 60000，n 设为 30，优化后的程序测试结果见图 2.3。

```
Please choose a function:3
Time elapsed in ms is 11040
```

图 2.3 优化后的程序运行结果

分析：在 m=60000，n=30 的条件下，优化前的程序运行耗时为 13570ms，优化后的程序运行耗时为 11040ms，经过计算，优化率为 18.6%。

3. 将计算推荐度改为子程序，程序运行结果如图 2.4 所示。

```
Please choose a function:3
Time elapsed in ms is 11260
```

图 2.4 将计算推荐度改为子程序的运行结果

考虑到计算推荐度用的都是此时不影响其他部分程序的寄存器，去掉子程序中保护现场和恢复现场的代码部分，测试结果如图 2.5 所示。

```
Please choose a function:3
Time elapsed in ms is 11040
```

图 2.5 去掉保护现场和恢复现场后的运行结果

分析：可以看出，更改为子程序后程序耗时略有增加，但影响不大，去掉保护现场和恢复现场后程序耗时和改为子程序前完全一样，说明将程序改为子程序对运行效率的影响仅仅在于保护现场和恢复现场，而 PUSH 和 POP 指令相对比较高效，因而将程序改为子程序对程序运行效率的影响不大。

4. 对之前进行的优化操作分开实验，利用控制变量的思想判断哪些优化对程序运行效率有显著提升。（以下都是在 m=10000，n=30 的条件下，优化前程序耗时 2250ms，也可见表 2.1）

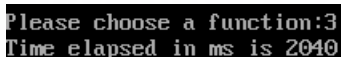
- (1) 仅将*64 改为向左移 6 位后，测试结果如图 2.6 所示。

```
Please choose a function:3
Time elapsed in ms is 2140
```

图 2.6 乘法指令改为移位指令的优化结果

- (2) 仅不再使用 CX 作为循环变量，直接用商品地址，内循环中可减少一条指令，并且多了一个寄存器可以使用，保存中间结果不再需要出栈入栈，又减少了两条指令，测试结果如图 2.7 所示。

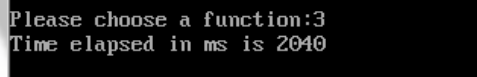
汇编语言程序设计实验报告



```
Please choose a function:3
Time elapsed in ms is 2040
```

图 2.7 不再使用 CX 作为循环变量的优化结果

(3) 由于商不可能超过 65535, 仅去掉 MOV EAX, 0 的两条冗余指令, 测试结果如图 2.8 所示。



```
Please choose a function:3
Time elapsed in ms is 2040
```

图 2.8 去掉两条冗余指令的优化结果

分析: 通过以上测试结果可知, 优化效果最明显的优化方法是——减少重复次数很多的那部分代码的数量。因而我们需要仔细分析重复次数很多的那部分代码是否存在冗余的代码, 或者通过一些灵活的方法去掉一些冗余的指令, 这样的优化立竿见影。优化的另一个关键点是使用尽可能少的寄存器, 这样我们在保留中间计算结果的时候不至于捉襟见肘, 还需要进栈出栈, 这样就会得不偿失。除此以外, 对于一些算术指令可以灵活处理, 尽量用效率高的指令代替效率低下的指令, 比如用移位指令代替乘除指令。

2.5 小结

经过前两周实验的磨合, 这一周的实验相对更加顺利, 同样也收获到了很多东西。

其中最直接的收获是彻底掌握了除法指令。由于这次实验涉及到程序优化, 我在反复斟酌如何通过调整计算顺序以及是用字除还是双字除来减少指令数量的过程中真正掌握了除法指令的用法。先考虑是用字除还是双字除, 然后把被除数放到指定的寄存器, 除完之后去指定的寄存器中取商和余数。个人认为, 双字除的情况下被除数一般都是乘法指令的结果, 这样程序写下来非常顺畅, 不需要再调整寄存器了。

然后, 最基本的收获是了解了程序计时的方法, 它最根本的原理是在计时起点得到当前时间, 然后在计时终点得到当前时间, 然后二者相减便是程序运行耗时。

在此基础上我们还研究了影响程序运行效率的各种因素, 其中最重要的因素是程序本身的时间复杂度。除此以外, 程序的运行快慢还跟运行的软硬件环境密不可分。当然, 程序运行的时间也跟问题的规模相关。

知道了影响程序运行效率的一个很重要的因素是程序本身之后, 我们还研究了如何优化程序, 也掌握了一些程序优化的基本方法。

其中, 最直接最有效的方法就是减少指令数量, 尤其是减少循环嵌套内部的指令数量, 这需要我们仔细观察, 深入思考, 找到那些冗余的指令, 或者在某些步骤用另一种解决思路, 看看能不能减少指令数量。

除此以外, 优化程序的另一个关键点在于那些相对更耗时的指令。由数字电路的知识我们知道, CPU 处理每一条指令所耗费的时间是不同的。有一些指令相对快速便捷, 比如移位; 另一些指令相对繁琐, 比如乘除。我们可以尝试用一些效率更高的指令去替代那些相对繁琐的指令, 这样也可以达到优化程序的目的。

除了上述收获以外, 经过这一周的汇编实验, 我们对汇编语言的指令也有了更深刻的认识。一方面, 汇编语言的指令基本上算是直接面向底层的, 因而在面向机器写程序时需要我们提高警惕性, 尤其是寄存器相关的, 哪些寄存器是已经用了的, 哪些还没用, 用了的那些都是干嘛的。

汇编语言程序设计实验报告

在这次实验中就是因为误用了当前另有它用的 BX 寄存器来当循环变量，导致程序进入了死循环。因而如果这一点处理不好，程序就很容易产生漏洞，而且很难排查。面向机器给我们带来巨大的便利性的同时也给我们带来了巨大的挑战，需要我们非常细心细致。另一方面，有一些汇编语言的指令有严格的规定，比如乘除指令，乘数、除数、被除数以及积、商、余数都在固定的寄存器中，不能随意瞎放。除了乘除以外，标志寄存器的置位也有详细的规定。我们在用到相关的指令的时候如果记不清楚一定要查表，绝不能不置可否，这样很容易导致程序出现严重的问题。

3 模块化程序设计

3.1 实验目的与要求

- (1) 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理；
- (2) 掌握宏指令、模块化程序的设计方法；
- (3) 掌握较大规模程序的开发与调试方法；
- (4) 掌握汇编语言程序与 C 语言程序混合编程的方法；
- (5) 了解 C 编译器的基本优化方法；
- (6) 了解 C 语言编译器的命名方法，主、子程序之间参数传递的机制。

3.2 实验内容

任务 3.1 宏与子程序设计

1. 把网店商品信息管理系统子功能尽量改成子程序的方式实现。
2. 将任务 1.4 中重复使用的程序段尽量改成宏（至少定义一个宏指令）或子程序的方式来实现。

3. 在网店商品信息管理系统中新增如下功能：

- 1) 在“2. 查找指定商品并显示其信息”的功能中，实现商品信息的显示功能。即：在找到指定商品之后，按照：“商品名称，折扣，销售价，进货总数，已售数量，推荐度”顺序显示该商品的信息。

2) 实现“6. 修改商品信息”的具体功能。

只有老板登录后可以使用本功能。若当前浏览商品无效，则返回；若有效，则按照：折扣，进货价，销售价，进货总数的次序，逐一先显示原来的数值，然后输入新的数值（若输入有错，则重新对该项信息进行显示与修改。若直接回车，则不修改该项信息）。

如：折扣：9》8 //符号“》”仅作为分隔符，也可以选择其他分隔符号

进货价：25》24

销售价：46》5A6 //输入了非法数值，下一行重新显示和输入

销售价：46》56

进货总数：30》 //直接回车时，对这项信息不做修改

当对这些信息都处理完毕后，回到主菜单界面。

4. 将本次新增功能的子程序放到另外单独的模块中，按照模块化程序设计的方法搭建系统。

任务 3.2：在 C 语言程序中调用汇编语言实现的函数

将网店商品信息管理系统中菜单界面的功能用 C 语言实现，其他子功能可以仍采用汇编语言实现。在 C 语言程序中调用汇编语言子程序。

3.3 任务 3.1 实验过程

3.3.1 设计思想及存储单元分配

1. 模块之间的通信方式

本次新增的功能放在 NEW.ASM 中，加上参考的两个模块 F2T10, F10T2 以及主模块 TASK1.ASM 共四个模块的通信方式如下（直接借鉴的程序不需要详细描述）。

（1）模块 NEW 中定义的子程序 DISPLAY、CHANGE 可供其他模块调用，同时调用其他模块中定义的子程序 F2T10、F10T2 以及变量 CRLF。

（2）主模块 TASK1 中定义的变量 CRLF 可供其他模块使用，同时调用其他模块中定义的 DISPLAY、CHANGE 子程序。

2. 相互关联的子程序名及其类型与参数、变量名及其类型

（1）子程序 PRINTS

类型：FAR

入口参数：要输出的串首址存放在 BX 中

出口参数：无

（2）子程序 DISPLAY

类型：FAR

入口参数：将商品信息首址存放在 BX 中

出口参数：无

（3）子程序 CHANGE

类型：FAR

入口参数：将商品信息首址存放在 BX 中

出口参数：无

（4）变量 CRLF

类型：BYTE

3. 段的定义要求

（1）四个模块中所有使用的数据段都具有相同的类别‘DATA’，所有的代码段也都具有相同的类别‘CODE’，所有的数据段和代码段的组合方式均为 PUBLIC。

（2）只有主模块中的一个堆栈段，组合方式为 STACK。

4. 新增功能的实现方法

（1）显示商品信息

调用 PRINTS 子程序（输出一个以 0 结尾的字符串）来显示商品名称；再分别调用 F2T10 程序来输出商品的其他信息。

（2）修改商品信息

dos 系统 9 号调用提示用户输入相关信息，dos 系统 10 号调用获得一个十进制数的字符串，设置好入口参数后调用 F10T2 程序将这个十进制数转化为 2 进制数并写到数据段即可。

汇编语言程序设计实验报告

3.3.2 流程图

1. 流程图

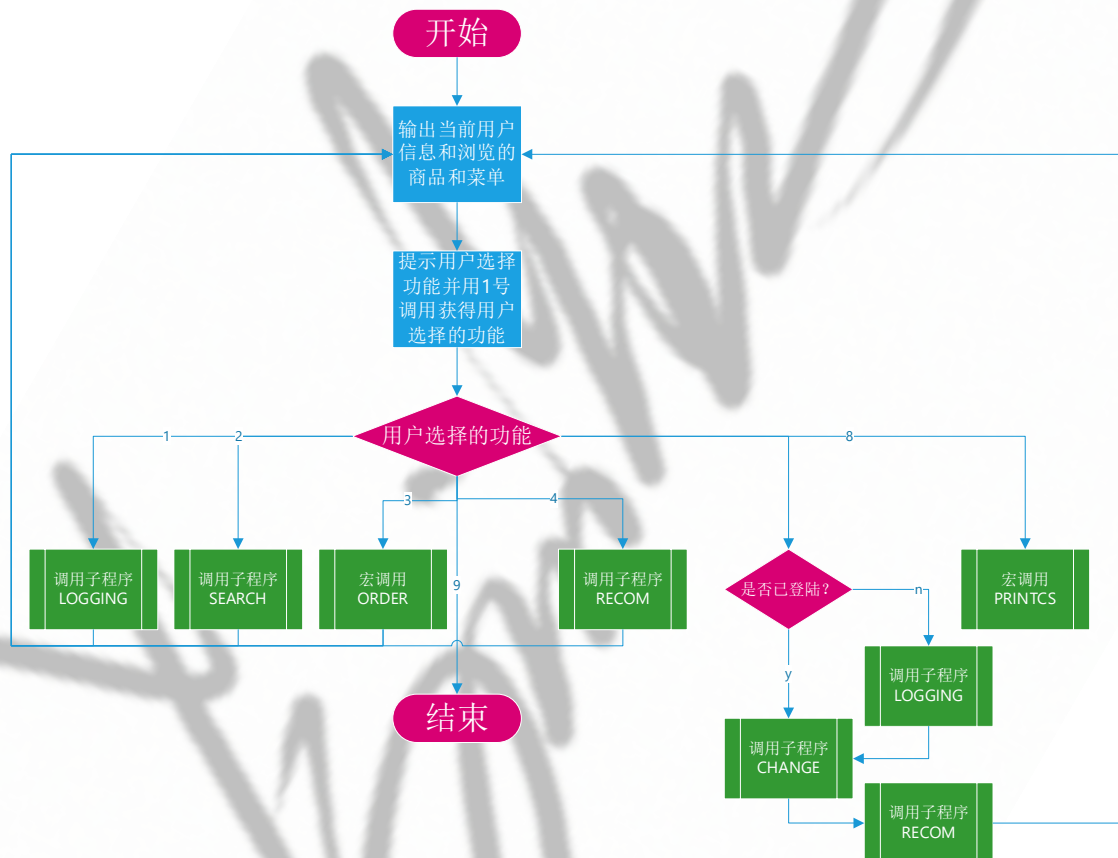


图 3.1 系统整体流程图

2. 模块结构

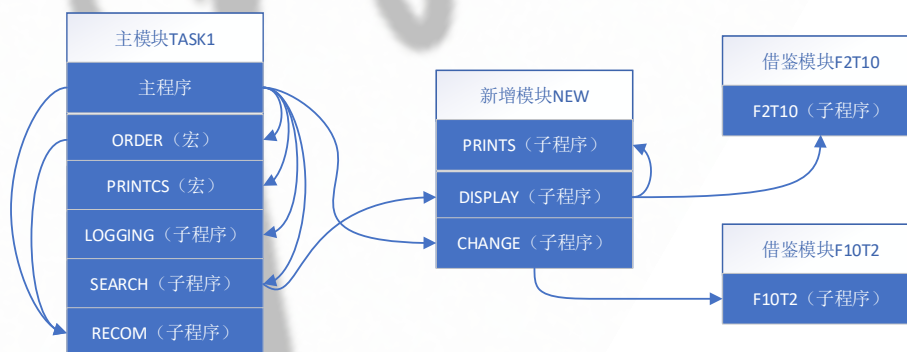


图 3.2 模块结构

汇编语言程序设计实验报告

3. 子程序调用关系图



图 3.3 函数调用关系图

3.3.3 源程序

新增模块的源程序如下。

;本模块完成功能：查找商品并显示信息、修改商品信息

;本模块的编写者：ACM1801 班李永康

EXTERN F2T10:FAR, F10T2:FAR, CRLF:BYTE

PUBLIC DISPLAY, CHANGE

.386

;子程序 PRINTS

;功能：输出一个以 0 结尾的字符串

;入口参数：要输出的串首址存放在 BX 中

;出口参数：无

;子程序 DISPLAY

;功能：显示商品信息

;入口参数：将商品信息首址存放在 BX 中

;出口参数：无

;子程序 CHANGE

;功能：修改商品信息

;入口参数：将商品信息首址存放在 BX 中

;出口参数：无

DATA SEGMENT USE16 PARA PUBLIC 'DATA'

NAMEH DB 'Product name: \$'

DISH DB 'Discount: \$'

PCH DB 'Purchased cost:\$'

PH DB 'Price: \$'

TSH DB 'Total stock: \$'

SAH DB 'Sold amount: \$'

RECH DB 'Recommendation:\$'

IN_TMP DB 10

DB ?

DB 10 DUP(0)

DATA ENDS

CODE SEGMENT USE16 PARA PUBLIC 'CODE'

ASSUME CS:CODE, DS:DATA

PRINTS PROC FAR

汇编语言程序设计实验报告

```
PUSH AX
PUSH BX

NEXT:  MOV DL, [BX]
        CMP AL, 0
        JE EN
        MOV AH, 02H
        INT 21H
        INC BX
        JMP NEXT

EN:     POP BX
        POP AX
        RET

PRINTS ENDP

DISPLAY PROC FAR
        PUSH AX
        PUSH DX

        LEA DX, NAMEH
        MOV AH, 09H
        INT 21H
        CALL PRINTS
        LEA DX, CRLF
        MOV AH, 09H
        INT 21H

        LEA DX, DISH
        MOV AH, 09H
        INT 21H
        MOV AX, 0
        MOV AL, 10[BX]      ;输出商品折扣
        MOV DX, 16
        CALL F2T10
        LEA DX, CRLF
        MOV AH, 09H
        INT 21H

        LEA DX, PH
        MOV AH, 09H
        INT 21H
        MOV AX, 13[BX]     ;输出商品销售价
        MOV DX, 16
        CALL F2T10
        LEA DX, CRLF
        MOV AH, 09H
        INT 21H

        LEA DX, TSH
        MOV AH, 09H
        INT 21H
        MOV AX, 15[BX]     ;输出进货总数
        MOV DX, 16
        CALL F2T10
        LEA DX, CRLF
        MOV AH, 09H
```

汇编语言程序设计实验报告

```
INT 21H

LEA DX, SAH
MOV AH, 09H
INT 21H
MOV AX, 17[BX] ;输出已售数量
MOV DX, 16
CALL F2T10
LEA DX, CRLF
MOV AH, 09H
INT 21H

LEA DX, RECH
MOV AH, 09H
INT 21H
MOV AX, 19[BX] ;输出商品推荐度
MOV DX, 16
CALL F2T10
LEA DX, CRLF
MOV AH, 09H
INT 21H

POP DX
POP AX
RET

DISPLAY ENDP

CHANGE PROC FAR
PUSH AX
PUSH CX
PUSH DX
PUSH SI

DISA: LEA DX, CRLF
MOV AH, 09H
INT 21H
LEA DX, DISH
MOV AH, 09H
INT 21H
MOV AX, 0
MOV AL, 10[BX] ;输出商品折扣
MOV DX, 16
CALL F2T10
MOV DL, '-'
MOV AH, 02H
INT 21H
MOV DL, '>'
MOV AH, 02H
INT 21H
LEA DX, IN_TMP
MOV AH, 0AH
INT 21H
CMP IN_TMP+1, 0
JE PCA
MOV SI, OFFSET IN_TMP + 2
MOV CX, 0
MOV CL, IN_TMP+1
```


汇编语言程序设计实验报告

```
CALL F10T2
CMP SI, -1
JE DISA
MOV 10[BX], AL

PCA:    LEA DX, CRLF
        MOV AH, 09H
        INT 21H
        LEA DX, PCH
        MOV AH, 09H
        INT 21H
        MOV AX, 11[BX]    ;输出进货价
        MOV DX, 16
        CALL F2T10
        MOV DL, '-'
        MOV AH, 02H
        INT 21H
        MOV DL, '>'
        MOV AH, 02H
        INT 21H
        LEA DX, IN_TMP
        MOV AH, 0AH
        INT 21H
        CMP IN_TMP+1, 0
        JE PA
        MOV SI, OFFSET IN_TMP + 2
        MOV CX, 0
        MOV CL, IN_TMP+1
        CALL F10T2
        CMP SI, -1
        JE PCA
        MOV 11[BX], AX

PA:     LEA DX, CRLF
        MOV AH, 09H
        INT 21H
        LEA DX, PH
        MOV AH, 09H
        INT 21H
        MOV AX, 13[BX]    ;输出销售价
        MOV DX, 16
        CALL F2T10
        MOV DL, '-'
        MOV AH, 02H
        INT 21H
        MOV DL, '>'
        MOV AH, 02H
        INT 21H
        LEA DX, IN_TMP
        MOV AH, 0AH
        INT 21H
        CMP IN_TMP+1, 0
        JE TSA
        MOV SI, OFFSET IN_TMP + 2
        MOV CX, 0
        MOV CL, IN_TMP+1
        CALL F10T2
```

汇编语言程序设计实验报告

```
CMP SI, -1
JE PA
MOV 13[BX], AX

TSA:    LEA DX, CRLF
        MOV AH, 09H
        INT 21H
        LEA DX, TSH
        MOV AH, 09H
        INT 21H
        MOV AX, 15[BX]    ;输出进货总数
        MOV DX, 16
        CALL F2T10
        MOV DL, '-'
        MOV AH, 02H
        INT 21H
        MOV DL, '>'
        MOV AH, 02H
        INT 21H
        LEA DX, IN_TMP
        MOV AH, 0AH
        INT 21H
        CMP IN_TMP+1, 0
        JE RECOVER
        MOV SI, OFFSET IN_TMP + 2
        MOV CX, 0
        MOV CL, IN_TMP+1
        CALL F10T2
        CMP SI, -1
        JE TSA
        MOV 15[BX], AX

RECOVER:LEA DX, CRLF
        MOV AH, 09H
        INT 21H
        POP SI
        POP DX
        POP CX
        POP AX
        RET

CHANGE  ENDP

CODE    ENDS
        END
```

3.3.4 实验步骤

1. 准备上机环境，编辑、汇编 NEW.ASM（新增模块）， F2T10.ASM, F10T2.ASM, 观察是否报错，若报错，则修改源文件直至不再报错。

2. 编辑 TEST.ASM，在 TEST.ASM 中直接将商品信息的首地址送到 BX 中（PRINTS 子程序的入口参数），然后调用 PRINTS 子程序以测试此子程序的功能，汇编 TEST.ASM，并与之间生成的.OBJ 文件连接，运行程序，观察是否能输出商品名称。

汇编语言程序设计实验报告

3. 编辑 TEST.ASM, 在 TEST.ASM 中直接将某商品信息的首地址送到 BX 中 (DISPLAY 子程序的入口参数), 然后调用 DISPLAY 子程序以测试此子程序的功能, 汇编 TEST.ASM, 并与之间生成的 .OBJ 文件连接, 运行程序, 观察能否正确在屏幕上显示商品的信息。

4. 编辑 TEST.ASM, 在 TEST.ASM 中直接将某商品信息的首地址送到 BX 中 (CHANGE 子程序的入口参数), 然后调用 CHANGE 子程序, 然后再用测试过的 DISPLAY 子程序输出修改后的商品信息, 以测试 CHANGE 子程序的功能, 汇编 TEST.ASM, 并与之间生成的 .OBJ 文件连接, 运行程序, 观察是否能完成修改商品信息的功能。

5. 编辑、汇编 TASK1.ASM (主模块), 并与 NEW.OBJ、F2T10.OBJ、F10T2.OBJ 连接, 生成 TASK1.EXE, 运行可执行程序, 测试系统的各个功能是否都能正确运行。

6. 将程序装载到 TD 调试工具中, 用 TD 调试工具单步跟踪到子程序内部, 特别关注子程序调用和返回时堆栈的变化。

7. 在 TD 调试工具中单步跟踪到子程序内部, 在 RET 指令执行之前修改栈顶的数值, 观察程序返回到什么位置。

8. 对一个 NEAR 类型的子程序强制使用 FAR 调用, 在 TD 调试工具观察现象; 然后对一个 FAR 类型的子程序强制使用 NEAR 调用, 在 TD 调试工具中观察现象。在这个过程中也要注意观察 FAR、NEAR 类型子程序的 RET 指令的机器码的区别。

9. 在 TD 调试工具中观察新增模块 NEW.ASM 中的数据段所定义的变量在指令语句中体现的偏移地址, 是否和它在 NEW.ASM 源文件中的定义的数据段的偏移地址相同, 若不同, 计算差值, 判断是否和主模块中数据段的大小相等。

10. 在 TD 调试工具中观察数据段中的内容, 观察四个模块中的数据段合并后的放置次序。

11. 改写部分程序, 改用堆栈法传递参数, 用 TD 调试工具观察将参数压入堆栈以及在子程序中弹出的过程。

12. 在 TD 调试工具中观察宏调用处的替换和扩展。

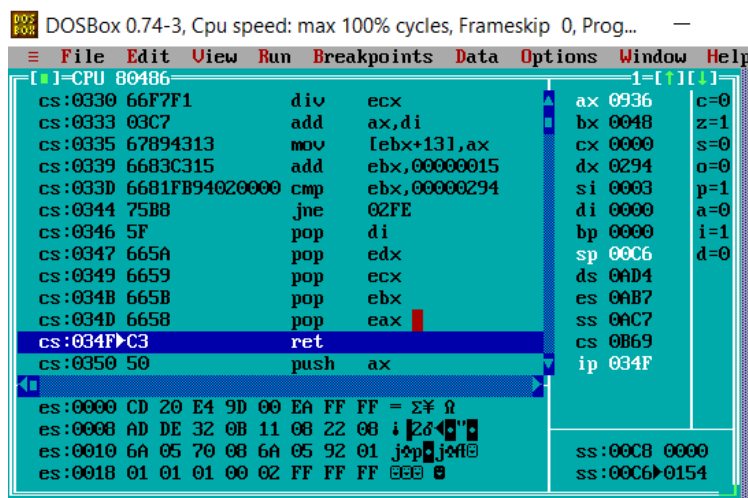
13. 用 TD 调试工具观察 EXTRN 说明语句放在 .386 之前和之后的区别。

3.3.5 实验记录与分析

1. 关于调用子程序时堆栈的变化以及 NEAR 调用与 FAR 调用

调用子程序时会下一条指令的地址压入堆栈中。如果是 NEAR 调用, 只将调用指令的下一条指令的偏移地址压入堆栈; 如果是 FAR 调用, 会先将调用指令的下一条指令所在段的段首址压入堆栈, 然后将偏移地址压入堆栈。返回时, NEAR 调用返回的反汇编语句为 ret, 机器码为 C3; FAR 调用返回的反汇编语句为 retf, 机器码为 CB。NEAR 调用返回只需将栈顶存储的偏移地址弹出送给 IP/EIP; FAR 调用返回, 先将栈顶存储的偏移地址弹出送给 IP/EIP, 再将栈顶存储的段首址弹出送给 CS 段寄存器。NEAR 调用返回时相关反汇编语句、机器码以及堆栈段见图 3.4; FAR 调用时返回相关反汇编语句、机器码以及堆栈段见图 3.5。

汇编语言程序设计实验报告

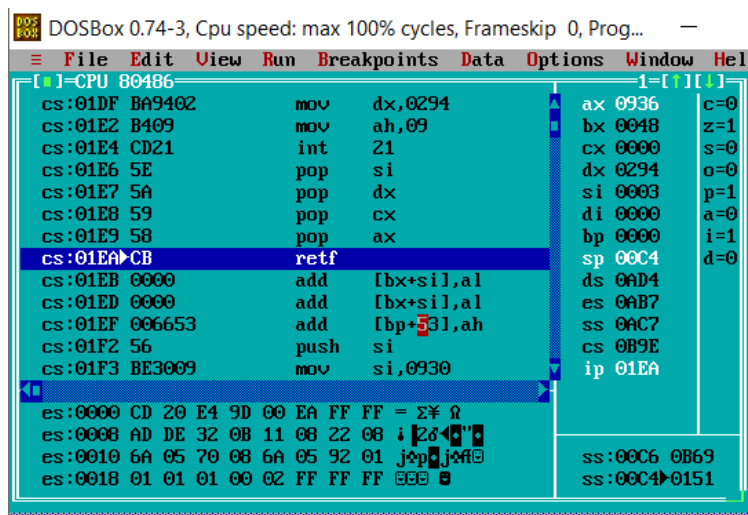


```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Prog...
File Edit View Run Breakpoints Data Options Window Help
[CPU 80486]
cs:0330 66F7F1 div ecx
cs:0333 03C7 add ax,di
cs:0335 67894313 mov [ebx+131],ax
cs:0339 6683C315 add ebx,00000015
cs:033D 6681FB94020000 cmp ebx,00000294
cs:0344 75B8 jne 02FE
cs:0346 5F pop di
cs:0347 665A pop edx
cs:0349 6659 pop ecx
cs:034B 665B pop ebx
cs:034D 6658 pop eax
cs:034F C3 ret
cs:0350 50 push ax

ax 0936 c=0
bx 0048 z=1
cx 0000 s=0
dx 0294 o=0
si 0003 p=1
di 0000 a=0
bp 0000 i=1
sp 00C6 d=0
ds 0AD4
es 0AB7
ss 0AC7
cs 0B69
ip 034F

es:0000 CD 20 E4 9D 00 EA FF FF = Σ¥ Ω
es:0008 AD DE 32 0B 11 08 22 08 i 2d-4"
es:0010 6A 05 70 08 6A 05 92 01 jsp,jyf
es:0018 01 01 01 00 02 FF FF FF 333 3
```

图 3.4 NEAR 调用返回情况



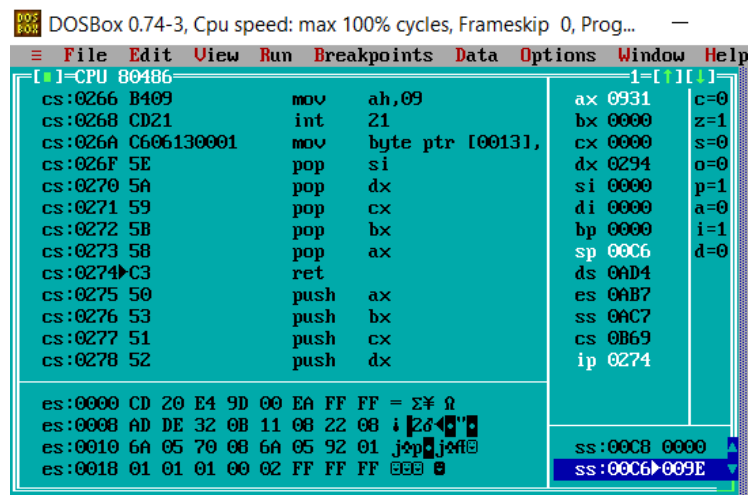
```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Prog...
File Edit View Run Breakpoints Data Options Window Hel
[CPU 80486]
cs:01DF BA9402 mov dx,0294
cs:01E2 B409 mov ah,09
cs:01E4 CD21 int 21
cs:01E6 5E pop si
cs:01E7 5A pop dx
cs:01E8 59 pop cx
cs:01E9 58 pop ax
cs:01EA CB retf
cs:01EB 0000 add [bx+sil,al]
cs:01ED 0000 add [bx+sil,al]
cs:01EF 006653 add [bp+531,ah]
cs:01F2 56 push si
cs:01F3 BE3009 mov si,0930

ax 0936 c=0
bx 0048 z=1
cx 0000 s=0
dx 0294 o=0
si 0003 p=1
di 0000 a=0
bp 0000 i=1
sp 00C4 d=0
ds 0AD4
es 0AB7
ss 0AC7
cs 0B9E
ip 01EA

es:0000 CD 20 E4 9D 00 EA FF FF = Σ¥ Ω
es:0008 AD DE 32 0B 11 08 22 08 i 2d-4"
es:0010 6A 05 70 08 6A 05 92 01 jsp,jyf
es:0018 01 01 01 00 02 FF FF FF 333 3
```

图 3.5 FAR 调用返回情况

如果在子程序调用返回前（RET 指令执行之前）修改栈顶的数值，程序会跳转到修改后的栈顶数值所指向的位置。图 3.6 是在 RET 指令执行之前将栈顶的 0095H 修改为 009EH；图 3.7 是 RET 指令执行之后返回的位置。



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Prog...
File Edit View Run Breakpoints Data Options Window Help
[CPU 80486]
cs:0266 B409 mov ah,09
cs:0268 CD21 int 21
cs:026A C606130001 mov byte ptr [00131],
cs:026F 5E pop si
cs:0270 5A pop dx
cs:0271 59 pop cx
cs:0272 5B pop bx
cs:0273 58 pop ax
cs:0274 C3 ret
cs:0275 50 push ax
cs:0276 53 push bx
cs:0277 51 push cx
cs:0278 52 push dx

ax 0931 c=0
bx 0000 z=1
cx 0000 s=0
dx 0294 o=0
si 0000 p=1
di 0000 a=0
bp 0000 i=1
sp 00C6 d=0
ds 0AD4
es 0AB7
ss 0AC7
cs 0B69
ip 0274

es:0000 CD 20 E4 9D 00 EA FF FF = Σ¥ Ω
es:0008 AD DE 32 0B 11 08 22 08 i 2d-4"
es:0010 6A 05 70 08 6A 05 92 01 jsp,jyf
es:0018 01 01 01 00 02 FF FF FF 333 3
```

图 3.6 RET 指令执行前修改栈顶的数值

汇编语言程序设计实验报告

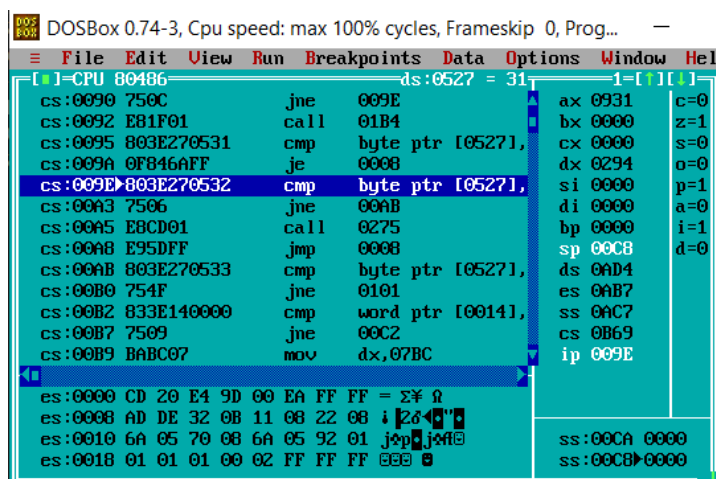


图 3.7 RET 指令执行后返回的位置

可以看出，本来程序应该返回到 CS:[0095]所指向的指令，但由于将栈顶由 0095 修改为了 009E，程序跳转到了 CS:[009E]所指向的指令。

如果对 NEAR 类型的子程序强制使用 FAR 调用，可以通过汇编和连接，程序也可以正常运行，但区别在于调用时不仅保存了返回地址的偏移地址，也保存了返回地址的段首址，这个段首址也就是调用处所在段的段首址，相当于“保存了多余的信息”。如果对 FAR 类型的子程序强制使用 NEAR 调用，不能通过连接，原因在于连接时检查到这是个 FAR 类型，但是强制使用 NEAR 调用时缺失了返回处的段首址信息，因而无法连接。对 NEAR 类型的子程序强制使用 FAR 调用后堆栈区情况见图 3.8；对 FAR 类型的子程序强制使用 NEAR 调用导致连接时报错见图 3.9。

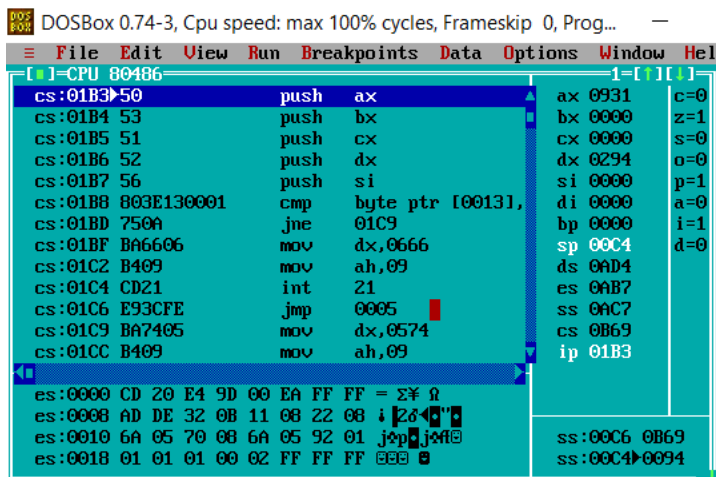


图 3.8 对 NEAR 类型子程序强制使用 FAR 调用

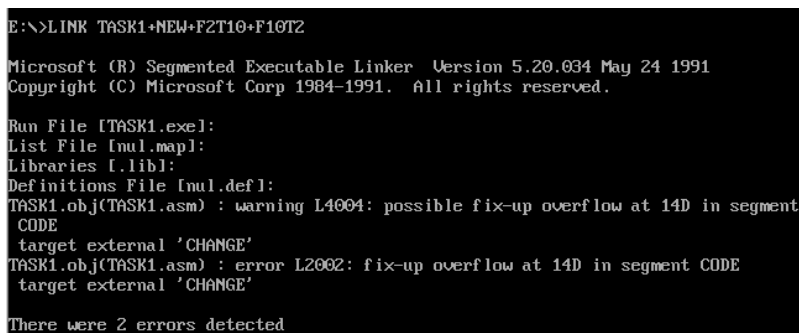


图 3.9 对 FAR 类型子程序强制使用 NEAR 调用后连接时报错

汇编语言程序设计实验报告

根据以上记录和分析可以知道，子程序的调用和返回是用堆栈存储返回指令的地址来实现的，NEAR 调用仅保存了指令的偏移地址，FAR 调用既保存了返回指令的偏移地址，也保存了段首址。

2. 关于数据段的合并

在主模块 TASK1 数据段尾增加一个测试变量 TEST 然后再代码段使用这个变量就可以知道主模块数据段最后一个变量的偏移地址（见图 3.10），然后在新增模块 NEW 中使用定义的第一个变量就可以看到新增模块数据段第一个变量的偏移地址（见图 3.11）。

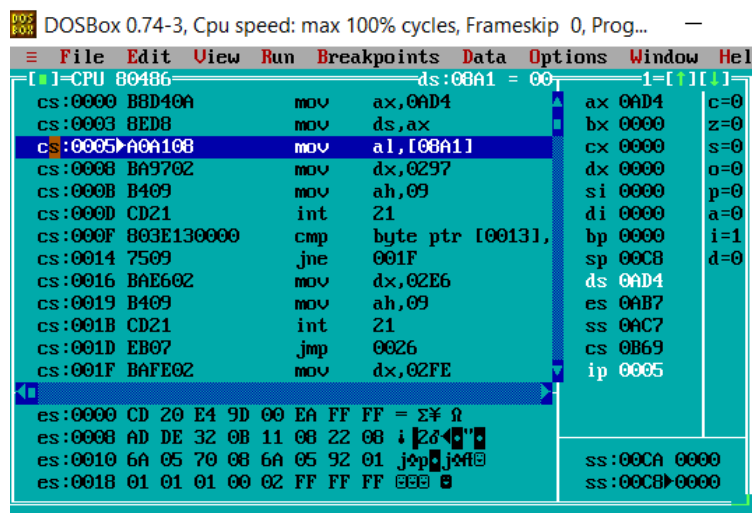


图 3.10 主模块中数据段最后一个字节的偏移地址

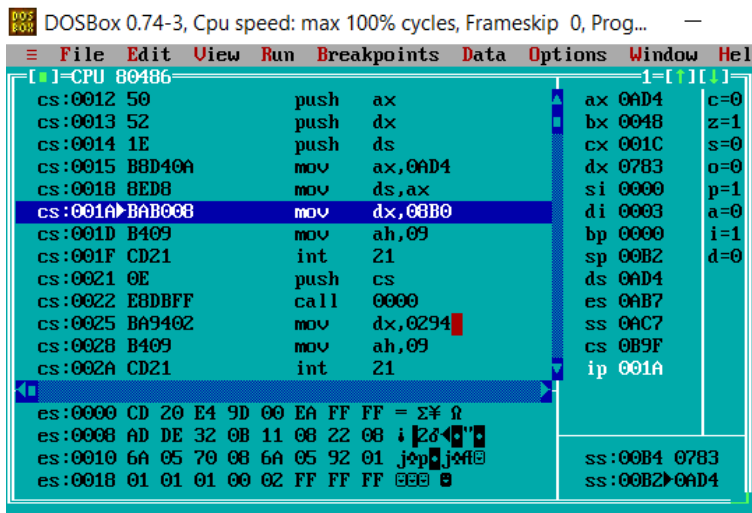


图 3.11 新增模块中数据段第一个字节的偏移地址

由图中可以看出，主模块中最后一个字节的偏移地址为 08A1H，新增模块中第一个字节的偏移地址为 08B0H，二者相差 15 个字节，没有紧密排列的原因是该数据段使用的定位方式为‘PARA’，该数据段的起始地址必须为 16 的倍数。查看数据段情况（见图 3.12）后可以看出，四个数据段的放置顺序为 TASK1+NEW+F2T10+F10T12，这个顺序也是当时连接时的顺序。

汇编语言程序设计实验报告

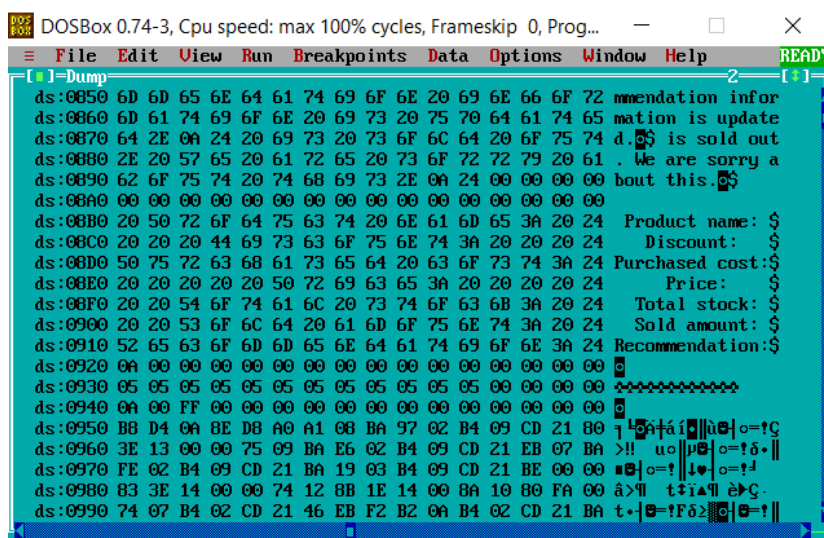


图 3.12 数据段情况

经过各种尝试后可以知道，几个模块相同类别的数据段合并后的放置顺序和连接时的顺序有关系，但不一定是完全相同。不管连接时顺序如何，主模块的数据段一定在最前面；当连接时主模块放在最前面时，数据段的放置顺序就会和连接时的顺序完全一致。

3. 关于宏的替换和扩展以及其与子程序的区别

实验时，在 TD 调试工具中已经完全看不到宏的影子，因为宏在汇编时就已经被替换和展开。从这一点上可以看到宏和子程序的本质区别，宏只是做简单的替换和展开，不涉及调用和返回，因此也不涉及堆栈变化和 IP/EIP 的跳转。

4. 关于 EXTERN 伪指令在 .386 语句前后的区别

在我们现在的 16 位段编程过程中，我们是将 EXTERN 伪指令放在 .386 语句之前，这样不会出现问题。若将 EXTERN 伪指令放在 .386 之后，在 TD 调试工具中可以看出，EXTERN 后声明的子程序将在机器码和反汇编语句中以 32 位寻址方式呈现（见图 3.12），但我们使用的段是 16 位段，从而导致程序出错，跳转到不正确的地方（见图 3.13）。

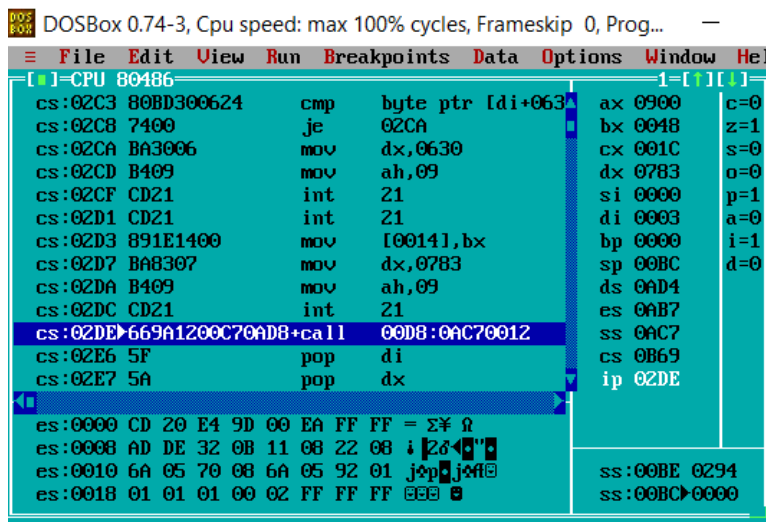


图 3.12 EXTERN 伪指令放在 .386 语句之后相关子程序变成 32 位寻址方式

汇编语言程序设计实验报告

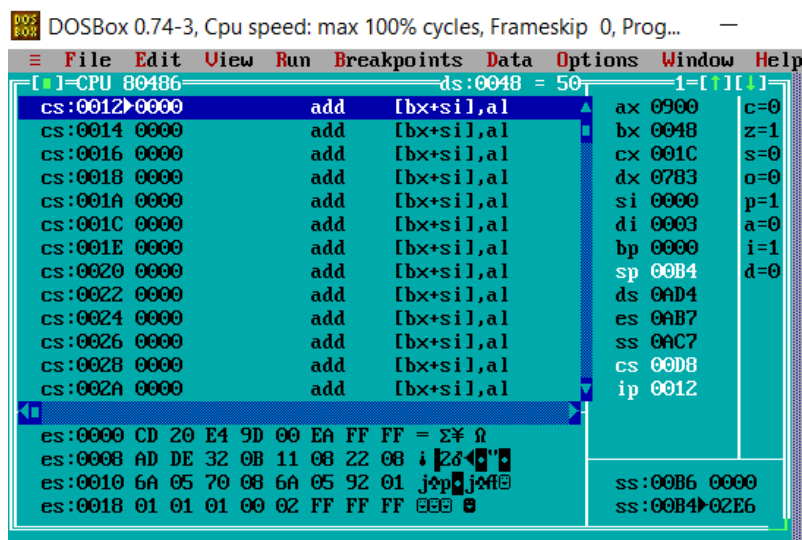


图 3.13 以 32 位寻址方式调用 16 位段中的子程序导致程序跳转到不正确的地方

3.4 任务 3.2 实验过程

3.4.1 实验方法说明

1. 开发环境：Visual Studio 2019 下 32 位程序，控制台输出。

2. C 语言与汇编语言混合编程形式：C 语言的模块与汇编语言的模块组合到一个工程里，编译连接成一个程序。在这次实验中，C 语言的模块主要处理输入输出以及登录时的用户名和密码字符串的比较，汇编语言的模块实现了搜索商品、下订单、显示商品信息以及修改商品信息的功能。对于除了登录以外的四个功能，均为 C 语言的模块调用汇编语言的子程序，同时，这四个功能中涉及到输入输出的地方，均是通过调用 C 语言中定义的相关函数来解决的。

3. 函数/子程序原型以及模块间的调用关系

(1) C 语言程序中定义的函数原型

C 语言程序中定义的供汇编语言程序调用的函数原型如下。

```
void printString(char *str);
void printChar(char ch);
void printShort(short sh);
void nextline(void);
int getInt(void);
```

C 语言程序中定义的不被汇编语言程序调用的函数原型如下。

```
void printMenu(void);
int login_check(char*, char*);
int login(void);
```

(2) 汇编语言程序中定义的函数原型

```
RECOM PROTO C
printInfo PROTO C
CHANGE PROTO C
SEARCH PROTO C
ORDER PROTO C
```

汇编语言程序中定义的所有子程序均被 C 语言程序调用。

4. 实验步骤：

汇编语言程序设计实验报告

(1) 通过在编译器配置中的设置在编译的同时生成汇编语言源程序，通过观察 C 程序对应的汇编源程序了解 C 语言编译器对各种符号的命名规则、主程序和子程序之间参数传递的机制以及通过堆栈传递参数后堆栈空间回收的方法。

(2) 用 VS 源码级调试工具观察由 C 语言形成的程序代码与由汇编语言形成的程序代码之间的关系，包括偏移的值、汇编指令访问 C 的变量时的翻译方法等。

(3) 观察 C 语言编译器的优化策略对代码的影响，并比较本实验中汇编语言程序和 C 语言程序的执行效率。

(4) 尝试在 C 语言源程序中不合理地嵌入汇编语言的指令语句，以达到破坏 C 语言程序的正确性的目的，体会在 C 语言程序中不考虑上下语句编译成的机器码而随意嵌入汇编语言指令语句时存在的风险。

3.4.2 实验记录与分析

1. 观察 C 程序对应的汇编源程序

(1) 可以观察到 C 语言编译器对各种符号的命名规则。

定义的函数以及声明的外部函数/子程序命名都带有下划线（见图 3.14）。

PUBLIC	__local_stdio_printf_options	EXTRN	__report_rangecheckfailure:PROC
PUBLIC	__local_stdio_scanf_options	EXTRN	__imp_acrt_iob_func:PROC
PUBLIC	__vfprintf_l	EXTRN	__imp_getchar:PROC
PUBLIC	_printf	EXTRN	__imp_stdio_common_vfprintf:PROC
PUBLIC	_vfscanf_s_l	EXTRN	__imp_stdio_common_vfscanf:PROC
PUBLIC	_scanf_s	EXTRN	__imp_strcpy_s:PROC
PUBLIC	_printMenu	EXTRN	_RECOM:PROC
PUBLIC	_login_check	EXTRN	_printInfo:PROC
PUBLIC	_login	EXTRN	_CHANGE:PROC
PUBLIC	_printString	EXTRN	_SEARCH:PROC
PUBLIC	_printChar	EXTRN	_ORDER:PROC
PUBLIC	_printShort		
PUBLIC	_nextline		
PUBLIC	_getInt		
PUBLIC	_printv		
PUBLIC	_main		

图 3.14 C 语言编译器对函数名的命名规则

分支结构、循环结构中的跳转指令用到的标号命名方式：\$+字母组合+数字+@所在函数/子过程名称（见图 3.15）。

```
; 131 :    for (i = 0; i < 16; i++) {  
  
    mov DWORD PTR _i$[ebp], 0  
    jmp SHORT $LN7@login  
$LN5@login:  
    mov eax, DWORD PTR _i$[ebp]  
    add eax, 1  
    mov DWORD PTR _i$[ebp], eax  
$LN7@login:  
    cmp DWORD PTR _i$[ebp], 16          ; 00000010H  
    jge SHORT $LN6@login
```

图 3.15 C 语言编译器对标号的命名规则

printf 函数输出的字符串常量命名用到了字符串的内容，还有下划线和@符号（见图 3.16）。

汇编语言程序设计实验报告

<pre>if(in_name[i]==username[i] && in_name[i]!=0) 00221D01 mov eax,dword ptr [in_name] 00221D04 add eax,dword ptr [i] 00221D07 movsx ecx,byte ptr [eax] 00221D0A mov edx,dword ptr [i] 00221D0D movsx eax,byte ptr username[edx] 00221D12 cmp ecx,eax 00221D14 jne login_check+0B7h (0221D27h) 00221D16 mov eax,dword ptr [in_name] 00221D19 add eax,dword ptr [i] 00221D1C movsx ecx,byte ptr [eax] 00221D1F test ecx,ecx 00221D21 jne login_check+0B7h (0221D27h) break; 00221D23 jmp login_check+0D5h (0221D45h) 00221D25 jmp login_check+0D3h (0221D43h) else if(in_name[i]!=username[i]) 00221D27 mov eax,dword ptr [in_name] 00221D2A add eax,dword ptr [i] 00221D2D movsx ecx,byte ptr [eax] 00221D30 mov edx,dword ptr [i] 00221D33 movsx eax,byte ptr username[edx] 00221D38 cmp ecx,eax 00221D3A je login_check+0D3h (0221D43h) flag=0; 00221D3C mov dword ptr [flag],0 } 00221D43 jmp login_check+7Ch (0221CECh)</pre>		<pre>300 CMPNAME:;验证用户输入的姓名是否正确 301 ;不正确则跳转到WRONGN 302 ;正确则跳转到CMPWD(继续比较密码) 303 CMP IN_NAME+1,NLENGTH 304 JNE WRONGN 305 MOV CX,NLENGTH 306 CNL: MOV SI,CX 307 MOV AL,BNAME-1[SI] 308 CMP AL,IN_NAME+1[SI] 309 JNE WRONGN 310 DEC CX 311 JNZ CNL 312 JMP CMPPWD 313 314 WRONGN: WRITE WNH ;提示用户输入的姓名错误 315 JMP F1</pre>	
--	--	---	--

图 3.19 比较 C 语言和汇编语言源程序的执行效率

同时，根据图 3.19 可以看出，C 编译器将 C 语言的语句编译成汇编指令时基本是以 C 语句为单位翻译的，很难太多地考虑到“上下文”可以带来的优化，因而如果直接用汇编语言来编写相同功能的程序，效率会相对更高。

4. 在不合理的地方修改了 DS 寄存器之后，调试就会出错（见图 3.20）。说明在 C 语言程序中，若不考虑上下文语句翻译成怎样的机器码而随意嵌入汇编指令语句时，有可能存在出错的风险。

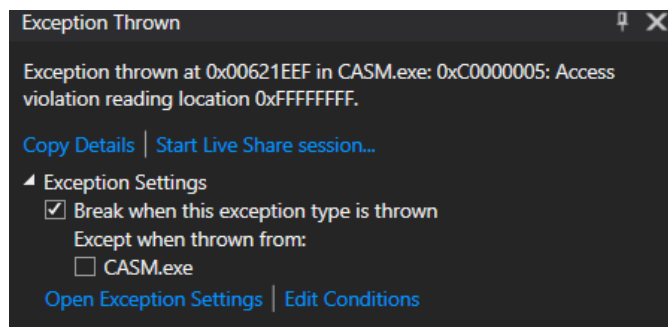


图 3.20 不合理地嵌入汇编语句

3.5 小结

通过这一次实验的任务一，我深刻理解了子程序的调用和返回原理，掌握了子程序几种不同的传递参数的方法，也体会到了宏和子程序的本质区别，也掌握了模块化程序的原理和设计方法。

子程序的调用和返回的原理就是把将来要返回到的指令的地址保存在堆栈中。对于段内调用（NEAR 类型）我们只需要保存偏移地址即可，因为不涉及到段寄存器值的变化；对于段间调用（FAR 调用），由于还涉及到不同代码段之间的转移，我们不仅需要返回处指令的偏移地址保存在堆栈中，还需要将返回处的段首址保存在堆栈中，以保证程序能够正确返回。在这一点上，一定要注意保护现场和恢复现场的对称性，不然会导致程序不能够正确返回。

汇编语言程序设计实验报告

子程序的参数传递有多种方法——寄存器法、堆栈法和指定地址法。个人用的比较多的是寄存器法。但在实验过程中也尝试并实现了其他两种方法，从实现的角度而言，堆栈法稍有复杂，需要自己去计算参数距离栈顶有多远，这一步搞错了就会导致程序出问题。归纳起来传递参数就是让调用程序和被调用程序相互做一个约定，把子程序需要的东西放到统一的地方，以供其使用。

关于宏的扩展与替换，这是其与子程序根本上的不同。宏只是在汇编时做一个简单的“复制粘贴”，但子程序的实现还涉及到堆栈的变化和 IP/EIP 的跳转，这是二者根本上的不同。

实验前对于模块化编程不太理解，也不太会实现，但试验后对模块化编程有了基本的概念和掌握。模块化编程一个最基本的问题就是不同模块之间的“通信协议”，就是我要用到哪些其他模块的东西，我有哪些东西是可以让其他模块来用的，并且要把这些东西的类别搞清楚，不然就会出错。其次就是不同模块中段的关系问题，相同类别的段可以合并，合并后偏移地址相比于合并前会发生变化。代码段合不合并直接决定了子程序调用的类型，数据段合不合并直接决定了在不同模块的数据段中切换需不需要改变 DS 寄存器中值，这些都是设计模块化程序必须要考虑的问题。

通过这次实验的任务二，了解和掌握了汇编语言程序与 C 语言程序混合编程的方法，也了解了 C 语言编译器的命名方法以及主、子程序之间参数传递的机制。

关于 C 语言编译器的命名方法，函数/子过程都在前面加上了下划线，字符串常量的名称中包含了字符串的内容外加其他符号，标号是按照“\$+字母组合+数字+@所在函数/子过程名称”的方式命名的。

C 语言中主、子程序之间的参数传递是通过堆栈法实现的，右边的参数先入栈，左边的参数后入栈，且通过堆栈法传递参数后堆栈空间的回收是通过直接修改 ESP 寄存器的值实现的。

关于汇编语言程序与 C 语言程序混合编程的方法。在实验前我是不清楚怎么把用 C 语言写的程序和用汇编语言写的程序结合起来的，也不知道如何相互调用子程序/函数以及如何相互使用定义的变量，但学习了老师给的示例程序后我知道了怎么在 C 中调用汇编语言程序中定义的子程序，也知道了怎么在汇编语言程序中调用 C 中定义的函数。将汇编语言程序中定义的子程序在 C 中声明一下就可以调用了，声明的时候形参的类型和顺序与汇编语言程序中的定义保持一致；同样，将 C 中定义的函数在汇编语言程序声明一下，然后用 invoke 就可以调用了，参数的类型和顺序也要和 C 中的定义保持一致。对于使用外部变量，直接 extern 声明即可。

由于 C 语言程序最终还是要要在 CPU 上运行，因而不可避免地要使用到寄存器、堆栈等，从而会影响到 C 寄存器、堆栈等的状态，如果我们不考虑上下语句，随意嵌入不合理的汇编语句，可能会导致寄存器、堆栈等状态发生异常改变，从而导致程序出错。因而在我们嵌入汇编语句时必须考略上下语句将翻译成怎样的汇编指令，这样才能避免程序出错。

除此之外，通过观察反汇编语句，还比较了 C 语言程序和汇编语言程序的效率。由于 C 语言程序在编译的过程中很难太多地考虑到“上下文”可以带来的优化，因而效率会相对更低，但是 C 语言编程比汇编语言编程更加简洁，可读性也更高，因而也相对更容易维护。与之对应的，汇编语言虽略显繁琐，但执行效率更高，因而我们可以将 C 语言和汇编语言结合起来，各取所需，使得程序在简洁性和可读性有保证的前提下执行效率也相对较高，从而获得更大的收益。

4 中断与反跟踪

4.1 实验目的与要求

- (1) 熟悉 I/O 访问, BIOS 功能调用方法;
- (2) 掌握中断矢量表的概念;
- (3) 掌握实方式下中断处理程序的编制与调试方法;
- (4) 进一步熟悉内存的一些基本操纵技术;
- (5) 熟悉跟踪与反跟踪的技术以及相关的反汇编工具;
- (6) 提升对计算机系统的理解与分析能力。

4.2 实验内容

任务 4.1: 实现“7. 迁移商店运行环境”的功能。

在操作系统和虚拟机中,经常要进行内存的调度迁移。这里的迁移运行环境的含义是指将“网店商品信息管理系统”当前的数据段、堆栈段、代码段切换到另外一套数据段、堆栈段和代码段中去,并保证切换前后程序的状态一致(比如,切换前正在浏览某个商品的信息,切换后也应保留该浏览状态)。本次实验只要求切换任务 3.1 程序的堆栈段。切换的操作是在指定时间下,由中断服务程序完成。

另,为便于观察,需要调整“8. 显示当前代码段首址”的功能为:“8. 显示当前段寄存器 SS 的内容”(即按照 16 进制方式显示这个段寄存器的内容)。

任务 4.2: 数据加密与反跟踪

在任务 4.1 的网店商品信息管理程序的基础上,老板的密码采用密文的方式存放在数据段中,各种商品的进货价也以密文方式存放在数据段中。加密方法自选(但不应选择复杂的加密算法)。

可以采用计时、中断矢量表检查、堆栈检查、间接寻址等反跟踪方法中的几种方法组合起来进行反跟踪(建议采用不少于两种反跟踪方法,重点是深入理解和运用好所选择的反跟踪方法)。

为简化录入和处理的工作量,只需要定义三种商品的信息即可。

任务 4.3: 跟踪与数据解密

解密同组同学的加密程序,获取各个商品的进货价。

建议尽量使用到以下的技术:

- 1) 利用静态反汇编工具(如 SOFTICE, OLLYDBG 等)将执行程序反汇编成源程序,观察源程序的特点。
- 2) 利用二进制文件编辑工具,直接观察和修改执行文件中的信息(如老板名字信息等)。
- 3) 动态跟踪调试,注意观察和跳过反跟踪的代码。
- 4) 有余力的学生可以设计实现:(a)一个暴力猜解密码的程序;(b)接管键盘的中断服务程序,驻留该程序之后再运行网店商品信息管理程序,截取用户输入用户名之后的字符串信息,保存在指定内存中;退出网店商品信息管理程序之后,用 TD 去观察中断服务程序记录的字符串信息。

汇编语言程序设计实验报告

4.3 任务 4.1 实验过程

4.3.1 设计思想及存储单元分配

本次实验的中断服务程序以及安装和还原的子程序均定义在一个新模块 TRAN. ASM 中，同时为了将输入的分钟和秒的信息转化为 BCD 码，新增一个 F10TBCD. ASM 模块。

1. 接管 8 号时钟中断的中断服务程序

(1) 存储单元分配

模块 TRAN. ASM 中定义了一个大小和主模块中一样的数据段，用于‘备份’的堆栈段。同时在代码段中定义了子类型的变量 TMP_SS 和 OLD_INT，前者用于保存主模块中定义的原有的堆栈段的段首址，后者用于保存原有 8 号始终中断的程序入口地址。除此之外，代码段中还定义了字节类型的变量 DIRECT、SEC、MIN 和 COUNT。DIRECT 用来标记下一次堆栈段切换的方向，SEC 和 MIN 分别记录了堆栈段切换时刻的‘秒’和‘分’信息，COUNT 用于计数，保证在堆栈段切换的那一秒内只进行一次切换操作。

(2) 设计思想

新中断服务程序首先通过已保存的原有中断程序的入口地址和 PUSHF、CALL 指令来模拟原有时钟中断，然后开始计数判断这是这一秒内第几次中断（相当于模 18 计数器），只有在这一秒内最后一次中断才进行接下来的操作，即通过比较来判断时间信息是否吻合，若吻合，则开始进行切换堆栈段操作，通过设定的变量 DIRECT 来判断堆栈段切换的方向，按字节的方式从 SP 指向的位置开始一直到栈底，将原有堆栈段所有内容复制到新的堆栈段，最后修改 SS 段寄存器的值即可。

2. 安装和还原的子程序

(1) 存储单元分配

AL 寄存器和 BL 寄存器作为安装程序的入口参数，传递切换堆栈段时刻的‘秒’和‘分’信息，并将数值送到代码段中定义的变量 SEC、MIN 中。除此之外，这两个子程序无其他入口和出口参数。

(2) 设计思想

安装程序一开始除了要保护现场之外，还要讲 DS 段寄存器的值（此时为主模块数据段首址）压入堆栈，然后将主模块传递进来的时间信息保存在代码段中定义的变量中，接着在将 DS 段寄存器置零后获得中断矢量表中 8 号时钟中断程序的入口地址，并保存在代码段的变量中，然后修改中断矢量表中 8 号时钟中断的偏移地址和段首址即可。最后弹出主模块中数据段首址到 DS 段寄存器中并恢复现场即可。还原的子程序除了不需要设定时刻信息以外基本和以上步骤相同，只不过是原有中断的程序入口地址送回到中断矢量表中。

汇编语言程序设计实验报告

4.3.2 流程图

1. 中断服务程序流程图

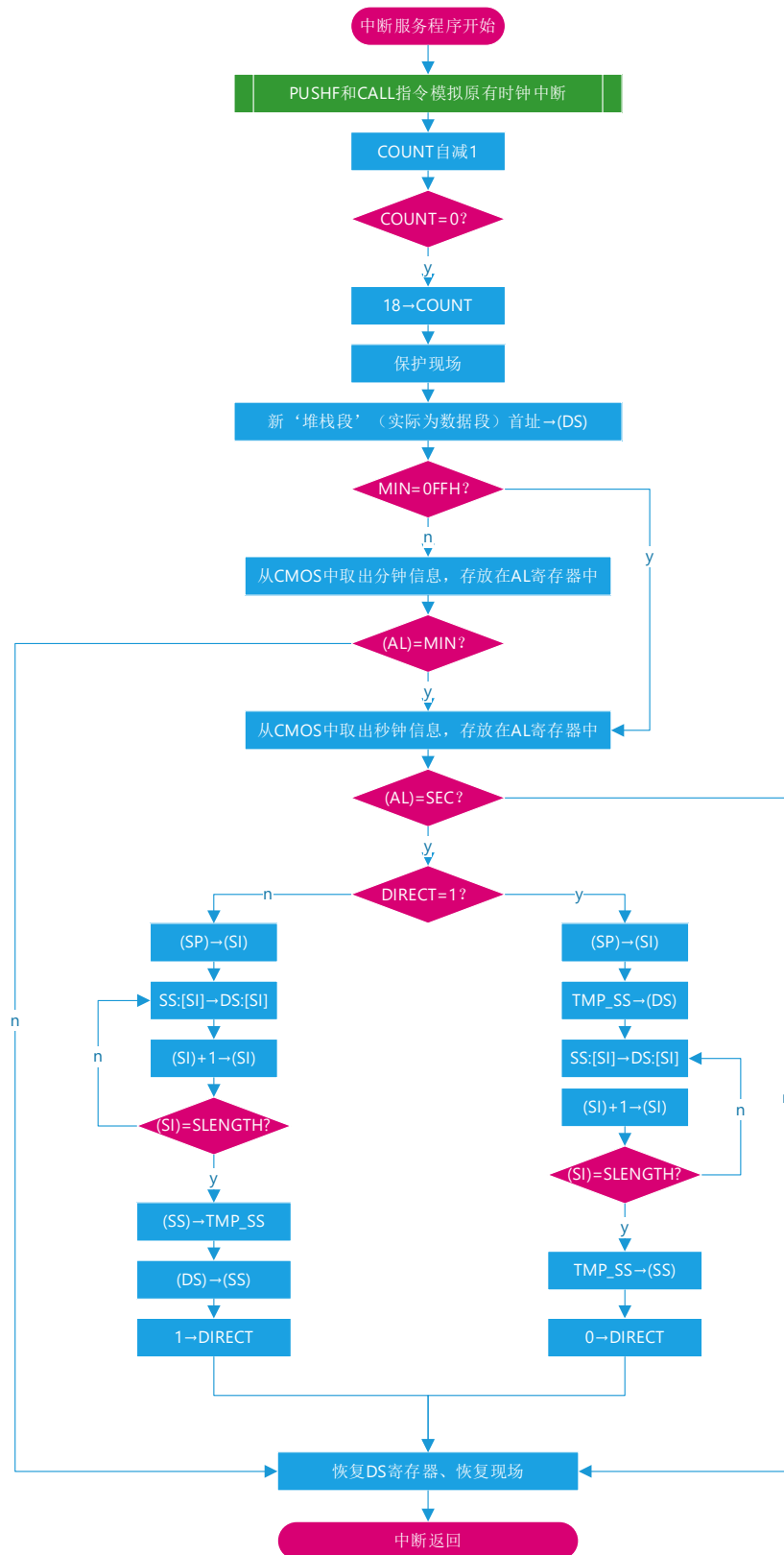


图 4.1 中断服务程序流程图

汇编语言程序设计实验报告

2. 安装子程序流程图

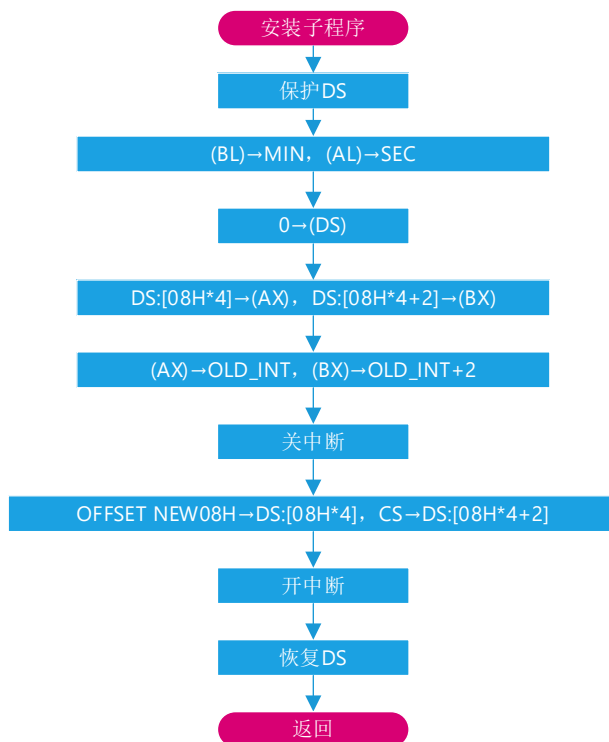


图 4.2 安装子程序流程图

3. 恢复子程序流程图

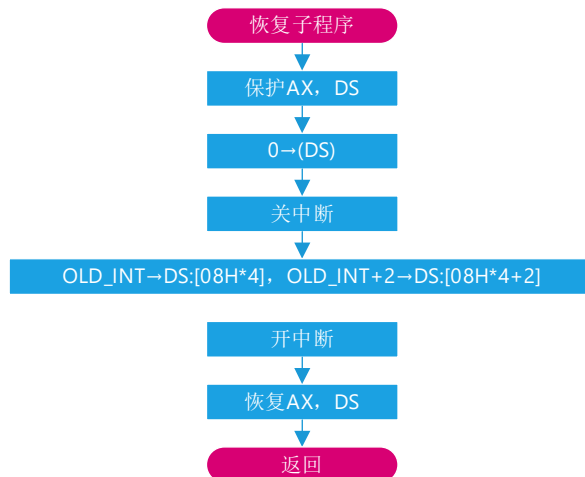


图 4.3 恢复子程序流程图

4.3.3 源程序

;本模块完成功能：编写新 8 号时钟中断程序以及安装和还原的子程序

;本模块编写者：ACM1801 班李永康

PUBLIC INSTALL, RECOVER

.386

;子程序 INSTALL

;功能：安装新 8 号时钟中断，并设定切换堆栈段的时刻

;入口参数：‘分’存放在 BL 中，‘秒’存放在 AL 中

;出口参数：无

汇编语言程序设计实验报告

```
;子程序 RECOVER
;功能：还原原有的 8 号时钟中断
;入口参数：无
;出口参数：无
DATA    SEGMENT USE16 PUBLIC PARA 'ST'
STACKBAK DB 200 DUP(0)
HINT     DB 0AH,'NOTE: THE STACK HAS BEEN SWITCHED JUST NOW! ',0AH,'$'
DATA     ENDS
CODE     SEGMENT USE16 PUBLIC PARA 'INT'
        ASSUME CS:CODE, DS:DATA
SLENGTH EQU 200
TMP_SS   DW ?
OLD_INT  DW ?, ?
DIRECT   DB 0
SEC      DB ?
MIN      DB ?
COUNT   DB 18

NEW08H: PUSHF
        CALL DWORD PTR CS:OLD_INT

        DEC CS:COUNT
        JZ  DO
        IRET

DO:     MOV CS:COUNT, 18
        PUSH AX
        PUSH SI
        PUSH BP
        PUSH DS

        MOV AX, DATA
        MOV DS, AX
        CMP CS:MIN, 0FFH
        JE CSEC
        MOV AL, 2      ;2 是“分钟”信息的偏移地址
        OUT 70H, AL    ;设定将要访问的单元是偏移值为 2 的“分钟”信息（完成切换）
        JMP $+2        ;延时，保证端口操作的可靠性（因端口电路的响应速度较慢）
        IN  AL, 71H    ;读取“分钟”信息
        CMP AL, CS:MIN
        JNE EXIT
CSEC:   MOV AL, 0      ;0 是“秒”信息的偏移地址
        OUT 70H, AL    ;设定将要访问的单元是偏移值为 0 的“秒”信息（完成切换）
        JMP $+2        ;延时，保证端口操作的可靠性（因端口电路的响应速度较慢）
        IN  AL, 71H    ;读取“秒”信息
        CMP AL, CS:SEC
        JNE EXIT
CHANGE: CMP CS:DIRECT, 1
        JE  RETURN

        MOV BP, SP
LOAD1:  MOV AL, SS:[BP]
        MOV DS:[BP], AL
        INC BP
        CMP BP, SLENGTH
        JNE LOAD1
```

汇编语言程序设计实验报告

```
ELOAD1: MOV CS:TMP_SS, SS
        MOV BP, DS
        MOV SS, BP
        MOV CS:DIRECT, 1
        LEA DX, HINT
        MOV AH, 09H
        INT 21H
        JMP EXIT

RETURN:  MOV BP, SP
        MOV AX, CS:TMP_SS
        MOV DS, AX
LOAD2:  MOV AL, SS:[BP]
        MOV DS:[BP], AL
        INC BP
        CMP BP, SLENGTH
        JNE LOAD2
ELOAD2: MOV SS, CS:TMP_SS
        MOV CS:DIRECT, 0
        MOV AX, DATA
        MOV DS, AX
        LEA DX, HINT
        MOV AH, 09H
        INT 21H

EXIT:   POP DS
        POP BP
        POP SI
        POP AX
        IRET

INSTALL PROC FAR
        PUSH DS

        MOV CS:MIN, BL
        MOV CS:SEC, AL

        XOR AX, AX
        MOV DS, AX
        MOV AX, DS:[08H*4]
        MOV BX, DS:[08H*4+2]
        MOV CS:OLD_INT, AX
        MOV CS:OLD_INT+2, BX

        CLI
        MOV WORD PTR DS:[08H*4], OFFSET NEW08H
        MOV DS:[08H*4+2], CS
        STI

        POP DS
ENDI:   RET
INSTALL ENDP

RECOVER PROC FAR
        PUSH AX
        PUSH DS
```

汇编语言程序设计实验报告

```
XOR AX, AX
MOV DS, AX

CLI
MOV AX, OLD_INT
MOV DS:[08H*4], AX
MOV AX, OLD_INT+2
MOV DS:[08H*4+2], AX
STI

POP DS
POP AX
RET
RECOVER ENDP

CODE    ENDS
END
```

4.3.4 实验步骤

1. 准备上机环境，编辑、汇编、文件 TRAN.ASM 和 F10TBCD.ASM。
2. 若在上一步中出现报错，修改源文件直至通过汇编。
3. 编辑、汇编、连接源文件 TEST.ASM，用于测试新的 8 号时钟中断，具体测试方法为：在编写的新的 8 号中断程序中注释掉分钟和秒钟判断步骤，即在执行完原有中断程序后直接开始切换堆栈段，然后在 TEST.ASM 中用 PUSHF 和 CALL 模拟中断，在 TD 调试工具中观察程序执行过程，判断是否能正确完成切换堆栈段的功能。然后增加判断时间的步骤，先在内存中设定一个时间，然后在这个时间前后分别显示堆栈段首址，观察是否在指定时刻完成切换堆栈段的操作。
4. 然后连接 TASK1+NEW+F2T10+F10T2+F10TBCD+TRAN，若有报错，修改源文件直至通过连接并生成可执行文件 TASK.EXE。
5. 执行程序 TASK.EXE，由于 TD 调试工具中不能观察到中断矢量表的变化，故在安装新 8 号中断程序前后分别输出中断矢量表中 8 号中断程序的入口地址，以此来判断新 8 号中断程序是否安装成功，并用同样的方法来测试恢复原有中断程序的子程序的功能。
6. 在安装中断程序时程序会提示用户输入切换的时刻（分和秒），输入一个时刻后，在这个时刻前后输出堆栈段首址，以此来测试新 8 号时钟中断程序的正确性。
7. 用 TD 调试工具调试 TASK1.ASM，进入后先记下 DS 段寄存器的值，然后将其修改为 0000H，然后观察数据段中的数据（中断矢量表），观察整个中断矢量表中的入口地址有何特点。然后直接在数据段中把中断矢量表中的中断矢量（如 21H 等）的值随意修改成其他值（或其他中断的中断矢量），然后在 TD 中执行 21H 中断，观察程序执行情况。
8. 查阅课本上附录中的关于 DOS 系统功能调用的部分，尝试用系统功能调用获取中断入口地址，以及尝试用系统功能调用方法安装中断程序。
9. 选择几个常见的中断服务程序的入口地址，观察其对应的中断服务程序的代码。
10. 尝试直接在 TD 下使用 IN/OUT 指令获取 CMOS 数据，并观察 CMOS 里的时间信息的格式（压缩 BCD 码）。
11. 尝试用另一种方法——JMP 指令模拟中断。
12. 尝试修改 TRAN.ASM 使得新的堆栈段的大小变为原来的两倍。

汇编语言程序设计实验报告

13. 尝试内存操纵, 进行动态调整程序中各个部分的大小, 以及将连续的存储空间定义成合适的堆栈段、数据段和代码段。

4.3.5 实验记录与分析

1. 程序调试

源程序汇编无报错, 连接后生成可执行文件, 但不能正常运行, 用 TD 调试后发现以下问题。

(1) 在程序中 SS 指向离栈底最远的存储空间, SP 指向栈顶, 但忘记了 SS 是段首址, 错误地将其当成一个和 SP 一样的指针, 因而在复制之后给 SS 和 SP 赋值的过程中, 误以为 SS 和 SP 的差值, 从而使得 SP 指向了一个错误的栈顶, 在 IRET 过程中程序返回到了错误的地方, 从而无法正常运行。正确的理解应该是: SP 是一个相对栈底的概念, 在转移堆栈段的过程中不需要改变, 只需要改变 SS 段寄存器的值即可。

(2) 一开始测试新 8 号时钟中断程序时, 直接用了 PUSHF+跳转指令 JMP, 程序无法正常运行, 经 TD 调试后发现, JMP 之前未将下一条指令的段首址和偏移地址压入堆栈, 从而在 IRET 时出错, 于是在 JMP 之前增加了压栈的操作 (此处也可以用 CALL 指令), 于是程序得以正常运行。在这一步中我们可以看到 IRET 近似等于 RETF+POPF。

2. 中断矢量表

(1) 进入 TD 后, 直接给 DS 赋值为 0000, 再打开数据段, 即可切换到中断矢量表所在内存区域 (见图 4.4)。

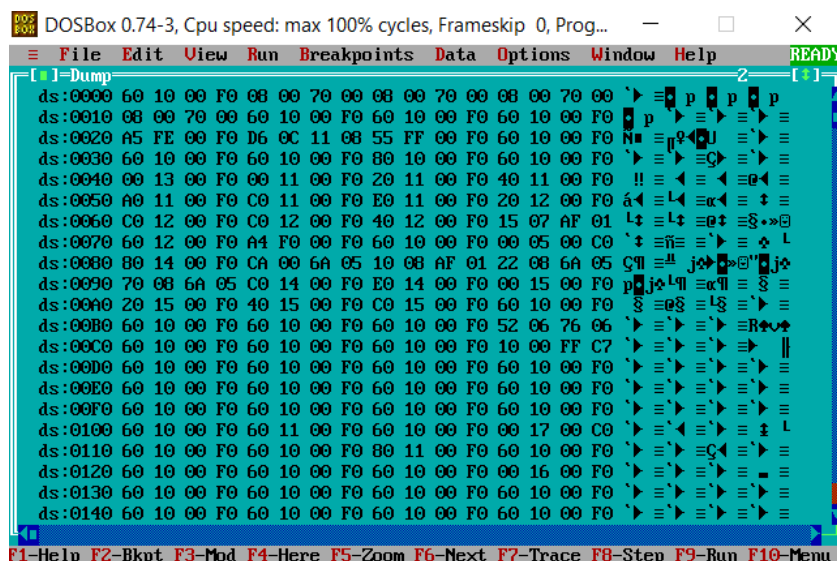


图 4.4 中断矢量表所在内存区域

(2) 源程序中获得 8 号中断的程序入口地址是通过直接访问内存的方法, 对于 n 号中断, 其访问方法是在 DS 为 0000H 的情况下访问 DS:[n*4] (偏移地址) 和 DS:[n*4+2] (段首址)。除此之外, 也可以通过系统调用的方式获得以及修改某个中断矢量, 分别用到 35H 号系统功能调用和 25 号系统功能调用。用系统调用的方法修改中断矢量表的程序见图 4.5, 经过测试, 修改后的源代码可正确运行。

汇编语言程序设计实验报告

```
MOV AL, 08H
MOV AH, 35H
INT 21H
MOV CS:OLD_INT, BX
MOV CS:OLD_INT+2, ES
MOV AX, CS
MOV DS, AX
MOV DX, OFFSET NEW08H
MOV AL, 08H
MOV AH, 25H
INT 21H
```

图 4.5 用系统功能调用实现修改中断矢量

在以上两步实验中，我们能体会到子程序调用和中断服务程序的一点点区别，子程序的入口地址是显式的，在 TD 调试工具中可直接看到子程序的入口地址，反汇编的话甚至可以直接找到子程序；但中断服务程序的入口地址是隐式的，是通过查表的实现的，不用特殊方法（改 DS、系统功能调用）的话是看不到程序的入口地址的，这样在一定程度上保护了中断矢量表和中断服务程序。

3. 中断服务程序

在源程序中通过 PUSHF 和 CALL 指令模拟中断服务程序，除此之外我们也可以通过 PUSH 和 JMP 的组合来模拟中断服务程序，该部分程序见图 4.6，经过测试该程序可正确运行。

```
NEW08H: PUSHF
        PUSH CS
        PUSH FLAG
        JMP DWORD PTR CS:OLD_INT
FLAG:   ---
```

图 4.6 用 PUSH 和 JMP 的组合模拟中断服务程序

在这一步实验中我们能深刻体会到中断服务程序和子程序的区别。对于 NEAR 类型的子程序，调用时只需将下一条指令的偏移地址压入堆栈；对于 FAR 类型的子程序，调用时先将 CS 段寄存器压入堆栈，再将下一条指令的偏移地址压入堆栈。而中断服务程序在中断时，在将 CS 和 IP 压入堆栈前还需要将标志寄存器压入堆栈，这是子程序和中断服务程序的较大的区别。

4. 内存操纵

（1）迁移数据段和迁移堆栈段的操作大同小异，不过大多数情况下我们并不清楚一个数据段占用了多少字节，数据段的大小一般不会像堆栈段那样在定义时直接给出，因为一个程序需要用到的变量不止一个，解决方法大致有三个：把数据段中所有变量占据空间加起来、TD 中查看最后一个变量的偏移、直接将新的数据段的大小尽量设置大一点。除此之外，DS 段寄存器的值可以直接通过传输指令改变。

（2）迁移代码段就会出现新的问题——CS 段寄存器值的修改无法通过传输指令完成。经过 TD 调试工具中尝试，直接用传输指令给 CS 赋值后，IP 的值会变，不能正确的指向新的代码段中的下一条指令。因而 CS 段寄存器值的修改就需要通过转移指令来完成，一个可行的思路先将新的代码段首址压入堆栈，再将下一条指令的偏移地址压入堆栈，然后 RETF 即可完成 CS 段寄存器值的修改。迁移代码段的程序见图 4.7，经过测试该程序可正确迁移代码段。

汇编语言程序设计实验报告

```
MOV SI, FINAL
MOV AX, NEWCODE
MOV ES, AX
MOVE: MOV AL, CS:[SI]
      MOV ES:[SI], AL
      DEC SI
      JNE MOVE
      MOV AL, CS:[0]
      MOV ES:[0], AL
      PUSH NEWCODE
      PUSH AFTER
      RETF

AFTER: JMP MAIN
```

图 4.7 迁移代码段

4.4 任务 4.2 实验过程

4.4.1 实验方法说明

1. 加密方法说明

(1) 用户名和密码的加密：用户名的密文为用户名和密码异或；密码的密文为用户名和密码相加。

(2) 进货价的加密方法：用户名和密码的密文的第一位拼接起来得到一个字类型的常量，然后乘以 3，再和进货价异或就得到了进货价的密文。

(3) AUTH 变量的加密方法：为了阻止因为修改 AUTH 而产生的欺骗，对 AUTH 进行加密，加密密钥为输入缓冲区 IN_NAME 和 IN_PWD 的用户名和密码的首位，加密方法为 AUTH 分别与两个首位异或再相加。如果希望通过修改 AUTH 变量来欺骗程序，至少需要知道用户名和密码的首位。如果检测到 AUTH 值被异常修改，则通过间接转移指令退出程序。相关处理见图 4.8。

```
MOV AL, IN_NAME+2
XOR AL, AUTH
MOV BL, IN_PWD+2
XOR BL, AUTH
ADD BL, AL
CMP BL, 94H      ;判断用户是否已经登陆过
JE UPDATE
CMP AUTH, 0
JNE indi4
```

图 4.8 阻止 AUTH 的修改引起的欺骗

2. 抵制静态反汇编方法说明

(1) 间接转移

将 9 个功能的入口地址存放在数据段的前 18 个字节，得到用户的选择（一个 31H 到 39H 之间的字符，减去 31H，再乘以 2，就是其入口地址在数据段中的偏移地址。除此之外，在很多跳转指令的位置也用了地址表的方式来抵制反汇编。

除此之外，在很多反跟踪代码之后的跳转使用寄存器跳转，可以达到抵制反汇编的目的。

(2) 定义冗余信息

在一些反跟踪代码中间和解密进货价以及比较用户名和密码字符串中间定义很多冗余信息，不让破解者识别这是一段反跟踪代码以及避免破解这看透加密或者解密方法，从而达到扰乱视线的目的。相关处理见图 4.9。

汇编语言程序设计实验报告

```
mov cx, indi1
jmp goon3
db 'sdfkljsdkfldskfjdsjfldjfkssoirewnjfdiufdsfsdhewr'
indi1: MOV AL, IN_NAME+2[SI]
mov cx, indi2
jmp goon3
db 'erwlkjnkj343nm,djfsi98324k45nxzn,mziu984234,mjk432'
indi2: XOR AL, IN_PWD+2[DI]
mov cx, indi3
jmp goon3
db 'trjnfdsmiubbmiewyvcnbqk39874ndnmjfidf37'
indi3: CMP AL, BNAME[SI]
```

图 4.9 通过间接转移和定义冗余信息避免用户掌握加密方法

3. 反跟踪方法说明

(1) 接管中断矢量表反跟踪

在程序的入口修改中断矢量表，1 号中断由登录子程序接管，3 号中断由修改商品信息子程序接管，如果跳过这一段代码，便不能正常进入登录和修改商品信息的子程序。

在关键处（比较用户名和密码字符串、解密进货价）检查中断矢量表（，如果不是被修改后的值，则说明程序被跟踪，不继续进行后面的加密/解密操作，避免破解者知道加密/解密方法。

(2) 计时法反跟踪

在计算推荐度解密进货价处、修改商品信息解密进货价处各加入一次计时代码，计时代码中间插入计算密钥的代码，如果跳过计时反跟踪便不能得到正确的密钥，也不会知道加密方法。

(3) 堆栈检查反跟踪

在计算推荐度、修改商品信息中解密进货价时，利用堆栈取代寄存器来保存密钥，并且在这个过程中多次检查堆栈，如果被跟踪，则不能得到正确的密钥，也就不能解密进货价。

4.4.2 实验记录与分析

1. 在程序入口处有一次堆栈法反跟踪，如果被检查到，程序将异常退出，如果通过这次检查，将进行间接转移，在一定程度上可以抵制静态反汇编（见图 4.10）。

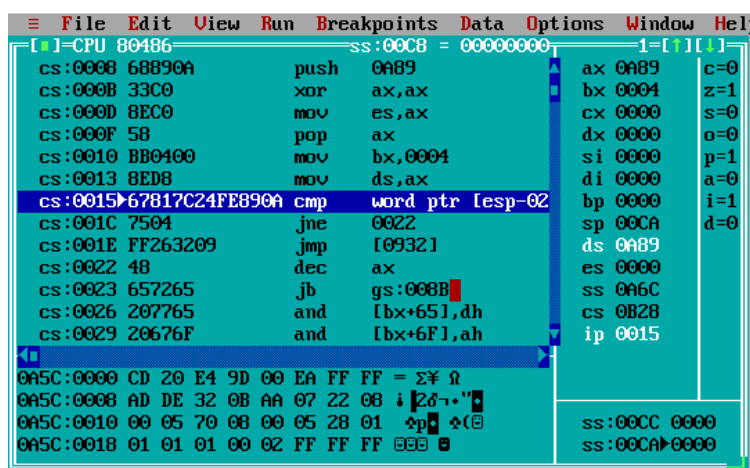


图 4.10 堆栈法反跟踪以及间接转移抵制反汇编

2. 间接转移之后进入到修改中断矢量表之后，由于新中断内嵌登录子程序和修改商品信息子程序，如果跳过这一部分，将不能登录或者修改商品信息；如果不跳过，将导致程序异常，不能继续跟踪，也不能继续运行，且无法退出 td 调试工具（见图 4.11）。

汇编语言程序设计实验报告

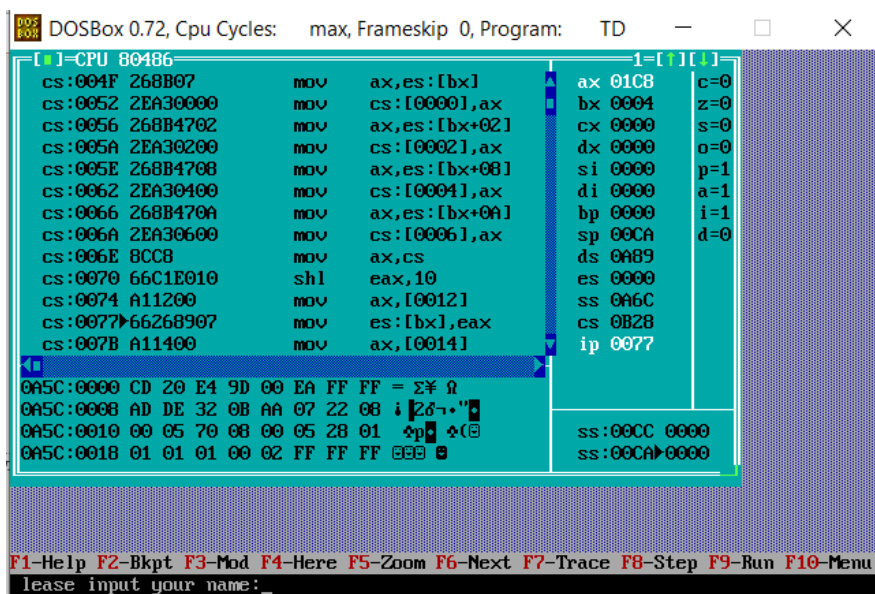


图 4.11 修改中断矢量表反跟踪

3. 在修改中断矢量表之前通过修改 IP 进入新中断程序（登录子程序），输入用户名之后进行第二次堆栈法反跟踪（见图 4.12），此时不再使用[esp-2]，而是[bp]进行伪装，通过之后仍然是一次间接转移（01C6H 处为 CX 寄存器跳转）以抵制静态反跟踪，而如果不能通过这次堆栈检查，后面是定义的很多冗余信息，指令无内在关联以混淆视线。

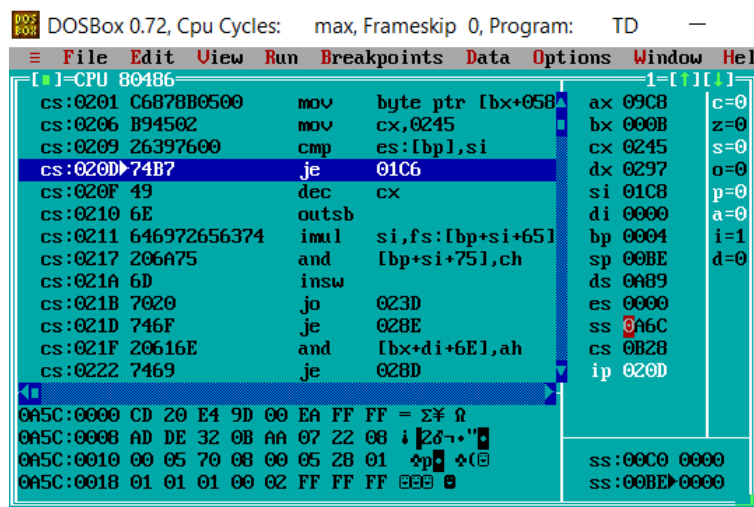


图 4.12 堆栈法反跟踪以及寄存器间接转移抵制静态反汇编

4. 用第 2 步中同样方法进入修改商品信息子程序，在解密进货价部分看到计时指令，两次计时中间安插了密钥计算（见图 4.13）。如果不能通过这次反跟踪，后面都是定义的冗余信息，以达到扰乱视线的目的。

汇编语言程序设计实验报告

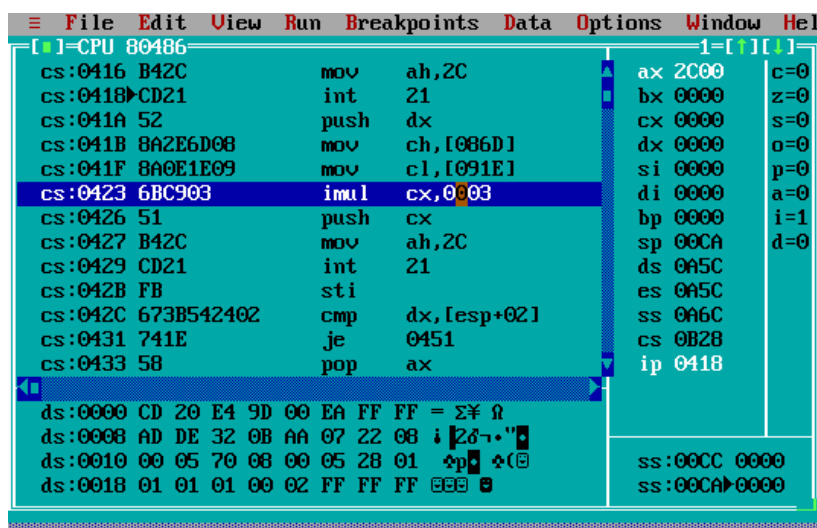


图 4.13 计时法反跟踪

5. 其他反跟踪部分的方法与以上所述几种方法类似。如果进入 TD 调试工具后手动修改 AUTH 变量的值，再点击 F9，选择功能 6 之后，程序就会终止（见图 4.14）。说明对 AUTH 的加密（密钥为 IN_NAME 和 IN_PWD 输入缓冲区的首位）可以在一定程度上阻止修改 AUTH 值带来的欺骗。

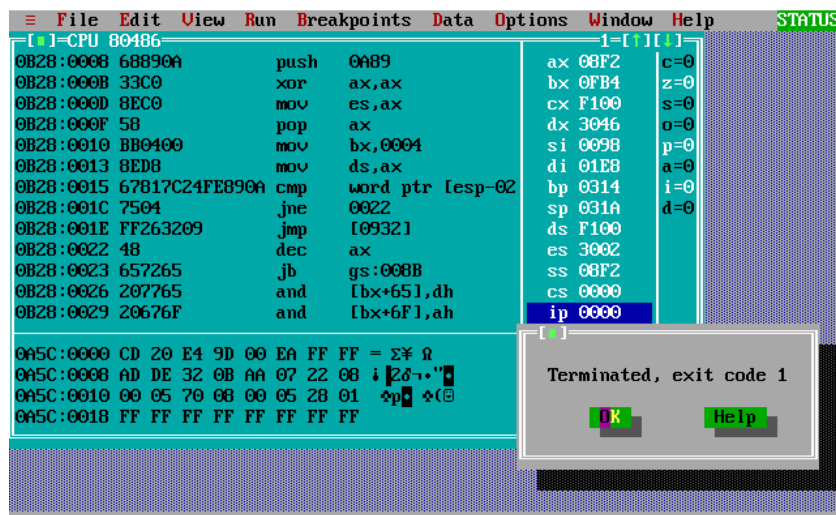


图 4.14 手动修改 AUTH 的值导致程序终止

6. 反汇编结果中可以看到，9 个功能的子程序入口全部被改成了地址表，且其偏移地址可以用功能号计算出来（见图 4.15），在很大程度上抵制了反汇编。

```
:0001.0156 BB0000      mov bx, 0000
:0001.0159 8A1E2405      mov bl, [0524]
:0001.015D 83EB30      sub bx, 0030
:0001.0160 8BF3      mov si, bx

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0001.00EC(C), :0001.00EE(C)
|
:0001.0162 03F6      add si, si
:0001.0164 BB0000      mov bx, 0000
:0001.0167 8B40FE      mov ax, [bx+si-02]
:0001.016A FFD0      call ax
:0001.016C E914FF      jmp 0083
```

图 4.15 地址表抵制静态反汇编

汇编语言程序设计实验报告

4.5 任务 4.3 实验过程

4.5.1 实验方法说明

1. 用二进制编辑工具打开可执行文件并观察

(1) 观察数据段中商品信息那部分数据，就可以直接得到商品的各种信息和进货价的密文。

(2) 观察代码部分中有意义的语言或者字母出现很频繁的位置，很可能是冗余信息，比如图 4.16 中代码部分出现了一句完整的英文，便很可能是定义的冗余信息。

图 4.16 冗余信息在二进制编辑工具中的显示

2. 静态反汇编

(1) 通过静态反汇编得到源程序，观察一下程序中是否涉及到修改中断矢量相关手段（cli 和 sti 指令、35 号和 25 号系统功能调用、将某段寄存器设置为 0），如果存在相关代码，需要先检查一下相关接管中断的程序是否是一段有用的代码，如果有，先判断一下应该是什么功能，如果涉及到登录子程序，则记下这个程序的入口，方便后面动态调试。

(2) 观察源程序是否存在 cli 和 sti 指令，是否涉及到计时法反跟踪，如果涉及，观察计时中间是否是一段有用的代码，若无用，则可以直接掠过，也方便后面动态调试，如果有，需要记下这个位置，也可以利于后面动态调试。

(3) 观察源程序中涉及到 add、sub、xor 等数学运算指令比较密集的地方，这些地方很可能是涉及到加密/解密的位置，先大致判断一下是用户名/密码的部分还是进货价的部分，观察加密/解密方法，看看是否能找到突破口。如果判断出来是计算推荐度的相关代码，观察源程序中的计算方法，可以反推进货价。

3. 动态调试

(1) 尽量跳过修改中断矢量反跟踪的代码，如果新中断中涉及到重要功能（登录、修改商品信息、计算推荐度），则记下新中断的入口地址，不要修改中断矢量表，直接修改 ip，就可以跳转到相关程序。

(2) 尽量跳过计时反跟踪，最好只是跳过计时代码，不要遗漏中间的有用代码。

(3) 堆栈检查反跟踪，观察到除了保护现场以外的压栈指令，需要警觉，如果后面看到 sp 或者 esp 或者 bp 的间接寻址，并且是比较指令，需要跳过这一部分比较，注意观察需要跳转到的正确位置。

(4) 如果能正确避免被反跟踪，则进入登录子程序（选择功能 1 或者知道程序入口之后修改 ip），在输入用户名和密码之后注意观察相关代码，重点寻找密文在数据段的偏移地址，观察比较字符串的循环出口，看是否能获得用户名或者密码的串长，从而获得密文。仔细观察程序，看看是否能得到用户名和密码的加密方法，如果找到了，并且已知密文，看看能否解出明文，如果可以，则可以直接用正确的用户名和密码看到进货价。

汇编语言程序设计实验报告

(5) 在第(4)步中, 注意观察密码字符串比较出口的传输指令, 看看是否能找到 AUTH 变量在数据段中的偏移地址, 如果可以, 则直接在数据段中修改这个变量, 然后选择功能6, 看看是否可以看到进货价。

(6) 如果可以正确避免被反跟踪而且进入计算推荐度/修改商品信息程序, 尽量找到进货价的密文在数据段中的偏移地址以及加密方法, 或者推荐度的计算公式, 也可以帮助我们得到商品的进货价。

4. 暴力破解

实现的具体思路: 如果在之前的步骤中能够得到用户名或者密码的串长, 那么在后面暴力尝试的时候就可以只生成这个固定长度的用户名或密码。如果不知道串长但是也可以在程序中人工尝试从而知道最大串长, 那么在后面暴力尝试的时候从生成小于这个串长的所有字符串来暴力求解。在求解时, 把暴力求解程序和待破解程序的输出互相作为对方的输入。每一次尝试时, 暴力求解程序输出1(代表选择登录功能), 然后依次输出当前生成的用户名和密码, 接着在下次选择功能时选择6(修改商品信息), 从而判断是否登录成功, 当识别到登录成功时, 把当前尝试的用户名和密码输出, 就完成了暴力破解。

4.5.2 实验记录与分析

任务 4.3 同组同学: 杨彪。

1. 用二进制编辑工具打开可执行文件

(1) 打开后可以找到存放商品信息的地方, 见图 4.17。

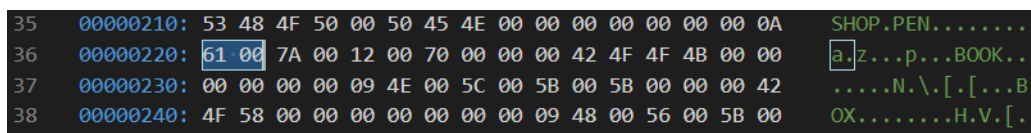


图 4.17 商品信息存放处

从而可以根据经验推测 PEN、BOOK 和 BOX 的进货价的密文反别为 0061H、004EH 和 0048H。

(2) 在二级制下浏览该文件, 没有发现字母密集或者出现具有意义的单词, 初步判定没有太多的冗余信息。且实验后询问同组同学, 确实没有定义冗余信息。

2. 静态反汇编

(1) 从程序入口处可以直接看到该程序存在修改中断矢量表反汇编(见图 4.18), 且接管的是 1 号中断和 3 号中断。

```
//***** Program Entry Point *****
:0001.0007 B80000      mov ax, 0000
:0001.000A 8ED8        mov ds, ax
:0001.000C 33C0        xor ax, ax
:0001.000E 8EC0        mov es, ax
:0001.0010 6726A104000000 mov ax, word ptr es:[00000004]
:0001.0017 A38601      mov word ptr [0186], ax
:0001.001A 6726A106000000 mov ax, word ptr es:[00000006]
:0001.0021 A38801      mov word ptr [0188], ax
:0001.0024 6726A10C000000 mov ax, word ptr es:[0000000C]
:0001.002B A38A01      mov word ptr [018A], ax
:0001.002E 6726A10E000000 mov ax, word ptr es:[0000000E]
:0001.0035 A38C01      mov word ptr [018C], ax
:0001.0038 FA          cli
:0001.0039 B88004      mov ax, 0480
:0001.003C 6726A304000000 mov word ptr es:[00000004], ax
:0001.0043 67268C0D06000000 mov es:[00000006], cs
:0001.004B 6726A30C000000 mov word ptr es:[0000000C], ax
:0001.0052 67268C0D0E000000 mov es:[0000000E], cs
:0001.005A FB          sti
```

图 4.18 程序入口处存在修改中断矢量表反汇编

汇编语言程序设计实验报告

从图 4.18 中可以看到，新中断程序的入口地址都为 CS:[0480]，找到这个程序入口（见图 4.19），发现新中断程序中没有有用的代码，因而在后续的动态调试中可以直接跳过这个修改中断的部分。

```
:0001.0480 CF                                iret
```

图 4.19 新中断程序入口

（2）通过在经过反汇编得到的源程序中寻找 cli 指令，可以发现在 CS:0171H 这里安插了计时反跟踪（见图 4.20），因而在后续动态跟踪时需要注意规避。

```
:0001.0171 FA                                cli
:0001.0172 B42C                               mov ah, 2C
:0001.0174 CD21                               int 21
:0001.0176 52                                push dx
:0001.0177 BE0200                             mov si, 0002
:0001.017A BBFFFF                             mov bx, FFFF
:0001.017D B42C                               mov ah, 2C
:0001.017F CD21                               int 21
:0001.0181 FB                                sti
:0001.0182 673B1424                             cmp dx, word ptr [esp]
```

图 4.20 反汇编观察到的计时法反跟踪

在源程序中没有找到更多计时（2C 号系统功能调用），因而初步判定只有这一次计时反跟踪。

（3）同样是在寻找 cli 指令时，在 CS:0211H 处安插了一次堆栈法反跟踪（见图 4.21），将栈顶（内容为 0249H）弹出到 AX 后，又将 SS:-2[ESP]赋给了 BX，再进行间接转移，因而在后续动态跟踪执行 CS:022FH 处的指令时注意将 BX 寄存器的值修改为 0249H，从而规避这次反跟踪。

```
:0001.0211 FA                                cli
:0001.0212 684902                             push 0249
:0001.0215 BAC000                             mov dx, 00C0
:0001.0218 B409                               mov ah, 09
:0001.021A CD21                               int 21
:0001.021C B20A                               mov dl, 0A
:0001.021E B402                               mov ah, 02
:0001.0220 CD21                               int 21
:0001.0222 B20D                               mov dl, 0D
:0001.0224 B402                               mov ah, 02
:0001.0226 CD21                               int 21
:0001.0228 58                                pop ax
:0001.0229 678B5C24FE                         mov bx, word ptr [esp-02]
:0001.022E FB                                sti
:0001.022F FFE3                               jmp bx
```

图 4.21 反汇编观察到的堆栈法反跟踪

在源程序中没有找到更多的 esp 或 sp 出现的地方，因而初步判定只有这一次堆栈法反跟踪。

（4）通过在源程序中寻找 xor 指令，找到这样一个突破口（见图 4.22）。在这条指令前面不远处，出现了[bx+2]基址寻址，而且在这条指令前面不远处调用了 0AH 号系统功能调用，通过自己正常运行该程序可以知道，整个程序输入字符串的地方只有四处：用户名、密码、商品名称和修改商品信息的时候，由于商品名称没有解密的必要，且修改商品信息的时候会连续出现多次 0AH 号系统功能调用，此处不符合。因而初步判定这是检查用户名或密码字符串的相关代码。在这条 xor 指令之后不远处再次出现 xor 指令（见图 4.23），因而基本判定这是在匹配用户名和密码的字符串，可以初步推测，用户名的密钥为 59H，密码的密钥为 58H，加密方法为异或运算。

汇编语言程序设计实验报告

```

:0001.0156 B40A      mov ah, 0A
:0001.0158 CD21      int 21
:0001.015A B20A      mov dl, 0A
:0001.015C B402      mov ah, 02
:0001.015E CD21      int 21
:0001.0160 B20D      mov dl, 0D
:0001.0162 B402      mov ah, 02
:0001.0164 CD21      int 21
:0001.0166 BB5500     mov bx, 0055
:0001.0169 807F020D   cmp byte ptr [bx+02], 0D
:0001.016D 0F84D800   je 0249
:0001.0171 FA        cli
:0001.0172 B42C      mov ah, 2C
:0001.0174 CD21      int 21
:0001.0176 52        push dx
:0001.0177 BE0200     mov si, 0002
:0001.017A BBFFFF     mov bx, FFFF
:0001.017D B42C      mov ah, 2C
:0001.017F CD21      int 21
:0001.0181 FB        sti
:0001.0182 673B1424   cmp dx, word ptr [esp]
:0001.0186 5A        pop dx
:0001.0187 7567      jne 01F0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0001.019A(C)
|
:0001.0189 43        inc bx
:0001.018A 83FB10     cmp bx, 0010
:0001.018D 7420      je 01AF
:0001.018F 8A905500   mov dl, [bx+si+0055]
:0001.0193 80F259     xor dl, 59
:0001.0196 3A970000   cmp dl, [bx+0000]
:0001.019A 74ED      je 0189
:0001.019C 80FA54     cmp dl, 54
:0001.019F 7404      je 01A5
:0001.01A1 0F858C00   jne 0231

```

图 4.22 用户名的密钥

```

:0001.01DB 43        inc bx
:0001.01DC 83FB06     cmp bx, 0006
:0001.01DF 7416      je 01F7
:0001.01E1 8A906300   mov dl, [bx+si+0063]
:0001.01E5 80F258     xor dl, 58
:0001.01E8 3A970A00   cmp dl, [bx+000A]
:0001.01EC 7543      jne 0231
:0001.01EE 74EB      je 01DB

```

图 4.23 密码的密钥

通过以上代码，还可以初步判定密码的密文在数据段中的偏移地址为 000AH，且串长为 6；用户名的密文在数据段中的偏移地址为 0000H，虽然在程序中看到串长应该为 10H，但用户名的数据区不可能覆盖掉密码的数据区，因而用户名的串长不超过 10。

(5) 通过继续观察 xor 指令，找到一处使用很多数学运算指令的地方（见图 4.24）。多次出现 idiv 指令，因而判定这是在计算商品推荐度。观察代码还可以知道商品信息的加密方法也是异或运算，且密钥为 42H，而且不止是进货价，商品所有除了折扣以外的数字信息都被加密了。

汇编语言程序设计实验报告

```

:0001.029B 83C615      add si, 0015
:0001.029E 4F          dec di
:0001.029F 743E        je 02DF
:0001.02A1 0FB6440A     movzx ax, [si+0A]
:0001.02A5 8B5C0D       mov bx, [si+0D]
:0001.02A8 83F342       xor bx, 0042
:0001.02AB 0FAF        imul ax, bx
:0001.02AD C3          ret

:0001.02AE 99          cwd
:0001.02AF BB0A00       mov bx, 000A
:0001.02B2 F7FB        idiv bx
:0001.02B4 8BD8        mov bx, ax
:0001.02B6 8B440B       mov ax, [si+0B]
:0001.02B9 83F042       xor ax, 0042
:0001.02BC C1E007       shl ax, 07
:0001.02BF F7FB        idiv bx
:0001.02C1 8BC8        mov cx, ax
:0001.02C3 8B4411       mov ax, [si+11]
:0001.02C6 83F042       xor ax, 0042
:0001.02C9 C1E007       shl ax, 07
:0001.02CC 99          cwd
:0001.02CD 8B5C0F       mov bx, [si+0F]
:0001.02D0 83F342       xor bx, 0042
:0001.02D3 F7FB        idiv bx
:0001.02D5 03C1        add ax, cx
:0001.02D7 83F042       xor ax, 0042
:0001.02DA 894413       mov [si+13], ax
:0001.02DD EBBC        jmp 029B

```

图 4.24 反汇编找到计算商品推荐度位置和进货价解密密钥

通过第 1 步中在二进制下观察该可执行文件，通过经验推测三个商品的进货价的密文分别为 0061H、0048H、004EH，知道密钥之后我们就可以解出三个商品的进货价分别为 23H、0AH、0CH。

3. 动态调试

(1) 由于在反汇编时就已经得到用户名和密码的密文的偏移地址（0000H 和 000AH），进入 td 之后，给 ds 段寄存器赋初值之后直接观察数据段（如图 4.25）。

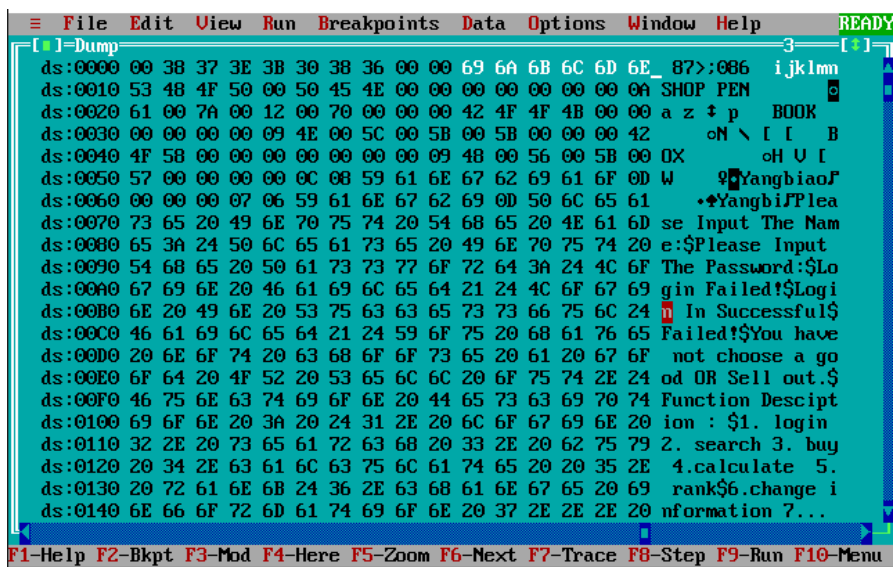


图 4.25 数据段中的用户名和密码的密文

经过反汇编中得到的密钥（59H 和 58H），破解得到用户名为 Yangbiao，密码为 123456。

汇编语言程序设计实验报告

退出 td 调试工具后正常运行该程序，选择功能 1，输入破解得到的用户名和密码，提示登录成功（见图 4.26），说明破解有效。

```
Function Description :
1. login 2. search 3. buy 4.calculate 5. rank
6.change information 7... 8. show ds,9.exit
1
Please Input The Name:
Yangbiao
Please Input The Password:
123456
Login In Successful

Function Description :
1. login 2. search 3. buy 4.calculate 5. rank
6.change information 7... 8. show ds,9.exit
```

图 4.26 破解成功效果图

（2）开始尝试通过动态调试的方法破解该程序。进入 TD 调试工具之后，执行给 DS 段寄存器赋值的工具之后直接通过修改 IP 寄存器从而跳过修改中断矢量表的相关代码（在反汇编时已经判断过新中断只有一条简单的 iret 指令，因而可以直接跳过）。然后在选择功能之前应该没有反跟踪（由前面反汇编可知），然后一直按 F8 直至提示选择功能，选择功能 1（登录），然后进入登录子程序，顺利输入用户名之后看到了一条 cli 指令，且在 0171H 位置附近，前面反汇编判断出来应该是计时反跟踪，于是小心地跳过了计时指令，没有漏掉两次计时中间的指令，从而规避了这次反跟踪，然后就看到了用户名的相关信息（见图 4.27），用户名的偏移地址为 0000H，加密方法是异或运算，密钥为 59H，从而可以破解出用户名。然后用同样的方法破解出密码的偏移地址为 000AH，加密方法也是异或运算，加密密钥为 58H。从而实现了动态调试的破解。

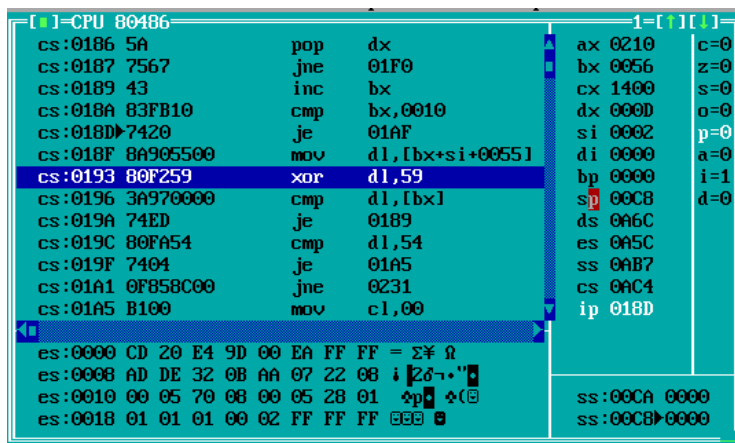


图 4.27 动态调试规避反跟踪得到用户名信息

（3）调试过程中通过观察比较密码字符串后的代码可以猜测 AUTH 变量的偏移地址为 0054H（见图 4.28），然后在退出 TD 后，重新将程序加载进 TD，然后在给 ds 段寄存器赋初值之后直接修改 0054H 单元的值为 01H，然后按 F9 运行程序，然后选择功能 6，成功欺骗了程序（见图 4.29）。

汇编语言程序设计实验报告

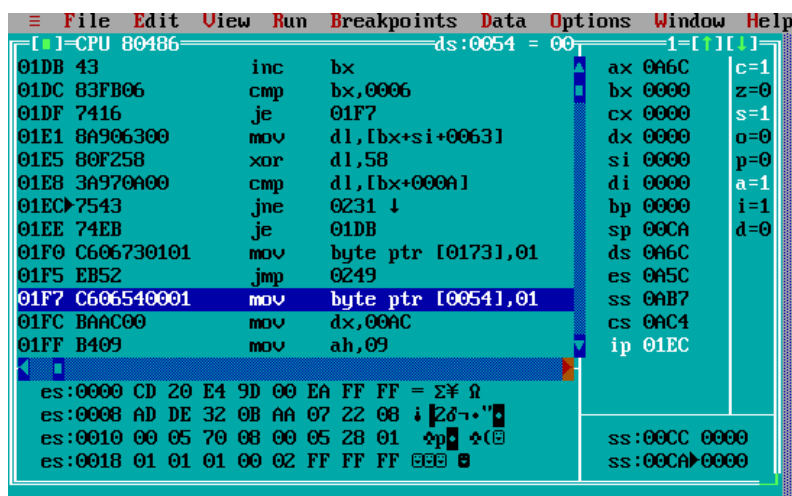


图 4.28 动态调试规避反跟踪得到 AUTH 变量地址

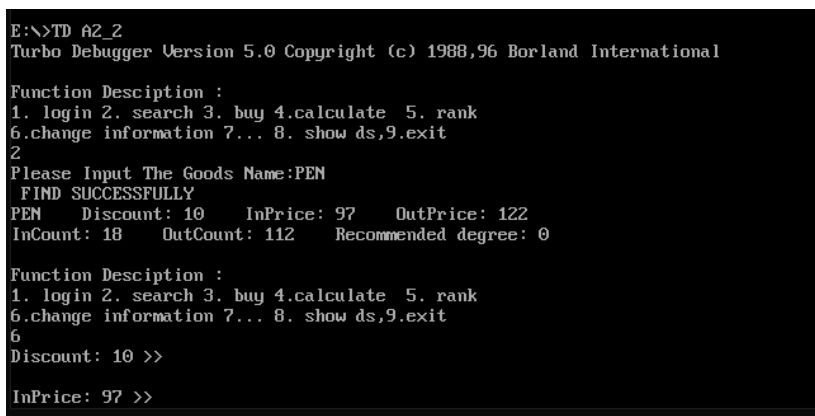


图 4.29 修改 AUTH 变量的值达到欺骗的目的

注意：该程序的商品信息除了折扣以外均为密文，因为同组同学在输出时没有对其进行解密。

4.6 小结

任务 4.1 的调试过程给我最大的感受就是堆栈段对于程序的正确运行有至关重要的作用，其中最明显的一点就是程序调用过程中指令地址的压栈出栈操作，如果在这个过程中 SS:[SP] 指向的不是正确的栈顶，很可能导致程序跳转到错误的地方，从而使得程序进入异常状态。因而我们在对堆栈段进行操作时，必须要非常细心谨慎，因为在堆栈段的处理过程失之毫厘谬以千里。

此外，在任务 4.1 中，我们还深刻理解了中断服务程序的实现机制及其与子程序的区别和中断矢量表，以及关于一段内存空间的使用解释。

关于中断服务程序与子程序的区别，一方面，中断服务程序的跳转过程要先将标志寄存器压栈然后在将 CS 和 IP 压栈；另一方面，中断服务程序的入口地址是通过查表的方式实现的，而子程序的入口地址是显式给出的，这在一定程度上保护了中断服务程序和中断矢量表。

而在内存操纵的实验部分，我们能体会到一段内存空间是堆栈段还是数据段还是代码段的决定因素——段寄存器的值。在汇编语言的框架内，我们能比较容易的定义堆栈段、数据段和代码段，但在程序运行的过程中，有时候代码段也是在自动切换的（子程序的 FAR 调用、中断服务程序），在多模块编程中有时还需要在程序中切换数据段才能访问不同模块的数据段定义的变量，

汇编语言程序设计实验报告

还包括本次实验的数据段当堆栈段或者代码段等等。在这些现象中我们能明显体会到一片内存地址的灵活性，它可以是用于存放数据的数据段，也可以是进行入栈出栈操作的堆栈段，也可以是保存指令的代码段，它最终的解释是由段寄存器决定的。也就是说，CS 寄存器指向哪哪就是代码段，SS 寄存器指向哪哪就是堆栈段，DS 寄存器指向哪哪就是数据段。知道这一点后，我们就可以在一定程度上突破现有框架，进行段与段之间的迁移使用，但在这个过程中我们必须保证 CS、DS、SS 必须指向正确的内存空间（包括 IP 和 SP 指向正确的位置），而且那个空间必须存放正确的内容，这样才不会出错。

第二次上机时完成了数据加密与反跟踪以及跟踪和数据解密的工作，在这些工作中对于跟踪与反跟踪、加密与解密有了十分深刻的认识。

先谈谈这次实验中汲取到的破解程序的经验。首先，我们在破解别人的程序时一定是有目的的，那么只要能达到这个目的任何合理的方法都可以。比如这次实验，目的是为了获得进货价，那么就存在两个思路：一个是进入登录状态，另一个就是破解进货价的加密方法。如果密码是用明文存放在数据段的，那么直接找到密码的偏移地址和串长，然后即可以在 TD 调试工具中看也可以在二进制编辑工具中看，进货价如果是明文，那么也可以通过这个方式。但如果数据都是密文存储的，我们就需要更多地依赖反汇编和动态跟踪了，由于加密过程涉及到很多数学运算，因而从静态反汇编的结果我们中要重点关注密集出现数学运算指令的地方（xor、div、mul 等），这些地方很可能就是我们解密的突破口，如果我们能找到加密方法并且这个加密方法是可逆的，那么接着只要寻找到偏移地址就能成功破解程序了，我们只要知道了用户名和密码、进货价这两者的任意一个就能得到进货价了。

如果无法破解用户名、密码或者进货价，我们还可以尝试其他方法看到进货价，比如找到记录登录状态的变量的偏移地址并去修改这个变量（本任务中是 AUTH）的值，从而尝试进行欺骗。或者在反汇编中找到能够输出进货价的相关代码位置，通过修改 IP（有时候也需要修改 CS）跳过判断是否处于登录状态的代码，从而看到进货价，但是这个过程需要敏锐地规避各种反跟踪，需要很多的经验。

由于在上机之前就和同组同学互相交换了程序进行破解，破解后也意识到了自己的程序有很多的不足，并且补上了发现的一些漏洞，也收获了很多跟踪和反跟踪的经验。首先一点就是要通过最大化利用静态反汇编来抵制反跟踪，找到 cli、[esp-2]、2CH 号、35 号、25 号系统功能调用等非常可疑的代码，还包括一些 ds、es 段寄存器置 0（修改中断矢量表）等高度可疑代码，可以记下来这些指令的偏移地址，方便后面动态调试规避这些反跟踪。其次就是根据静态反汇编的结果在动态跟踪中敏锐地抵制反跟踪，这也需要非常很多的经验和敏锐的洞察力。其中很重要的一点就是在单步执行转移指令之前多考虑考虑，多看看前后的指令关系，将要跳转的地方应该是要干什么，其次就是各种可疑指令更是要留心，万一进了陷阱也不用着急，还可以通过修改 IP 来跳出陷阱。修改 IP 的值是跟踪过程中另一个非常重要的一个方法，这个手段可以帮助我们跳转到我们想要跳转到的任何位置。

积累的跟踪经验同样可以帮助我们反跟踪。比如尽量少使用 cli、sti、[esp-2]、35H 号调用、25H 号调用、2CH 号等显眼的代码，最好是通过一些间接的方法完成这些反跟踪程序的编写。比如修改中断矢量表时只用 eax 一个 32 位通用寄存器一次性修改中断矢量，就不需要用到 cli 和 sti 指令了。另外，在 21 号中断之前如果想用 35H、25H、2CH 调用，可以提前将调用号保存在某一个寄存器中，然后跳转到执行中断处，从而在一定程度上达到反跟踪以及抵制静态反汇编的效果。

汇编语言程序设计实验报告

除此之外，多用间接转移指令，比如将地址保存在地址表或者寄存器中，这都是非常有效的抵制静态反汇编的手段。

通过这次实验，我们能感受到汇编语言的一大特色——贴近底层、操纵灵活，以及这一特色带来的强大功能——逆向工程以及抵制逆向。

对于逆向工程而言，由于我们唯一可以利用的就是可执行文件，因而想要逆向这个程序，只能借助汇编语言的知识来完成这个任务，如果对汇编语言不熟悉或者不了解，那么完成这项任务就会举步维艰。

而对于抵制逆向而言，由于高级语言（如 C 语言）具有一定的封装性，使用起来像一个“黑盒子”，基本上不能够在高级语言这个层面把这个“黑盒子”拆开，也不能够在高级语言这个层面直接操纵寄存器、堆栈、中断矢量表（虽然 C 语言中也有一些方法可以访问寄存器和堆栈，但操作起来异常麻烦，更不用说去检查堆栈、获得程序入口地址、修改中断矢量表等底层操作）。很明显，高级语言的抽象性让其基本丧失了底层操纵性，这是汇编语言得天独厚的优势，虽然其贴近硬件的特点给编写汇编程序带来了巨大的挑战，但我们也可以利用这个特点做一些高级语言难以胜任的工作，比如反跟踪工作和抵制静态反汇编。

5 WIN32 程序设计

5.1 实验目的与要求

- (1) 熟悉 WIN32/64 程序的设计和调试方法；
- (2) 了解不同操作系统环境下开发工具的特点；
- (3) 熟悉宏汇编语言中 INVOKE、结构变量、简化段定义等功能；
- (4) 进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系；了解 16 位段程序移植到 32/64 位段程序时需要注意的问题。

5.2 实验内容

任务 5.1 编写一个基于窗口的 WIN32/64 程序，实现网店商品信息管理系统的一部分功能。也即以任务 3.1 为基础，将其部分功能移植过来，具体要求如下描述。

1. 编写一个基于窗口的 WIN32/64 程序的菜单框架，具有以下的下拉菜单项：

```
File   Action           Help
Exit   Recommendation    About
      List Sort
```

点菜单 File 下的 Exit 选项时结束程序；点菜单 Help 下的选项 About，弹出一个消息框，显示本人信息，类似图 5.1 所示。点菜单 Action 下的选项 Recommendation、List Sort 将分别实现计算推荐度或显示排序后的 SHOP 所有商品信息的功能（详见要求“2”的描述）。

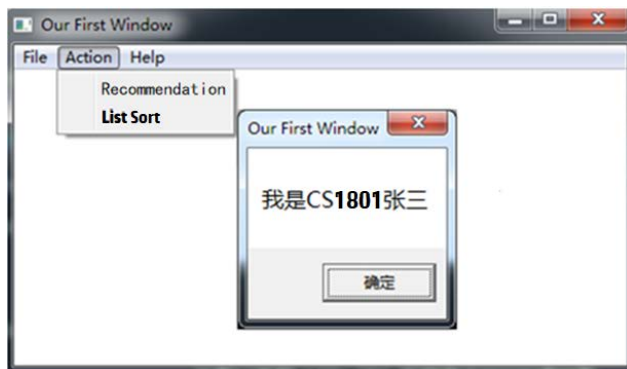


图 5.1 菜单示例

2. 要求采用结构变量存放商品的相关信息。商品数定义 5 种左右。

- (1) 点菜单项 Recommendation 时，按照任务 3.1 的方法计算所有商品的推荐度。
- (2) 点菜单项 List Sort 时，先对所有商品按照推荐度从高到低排序，然后按照排序结果在窗口中列出 SHOP 的所有商品的信息。具体显示格式自行定义，可以参照图 5.2 的样式（不要求用中文）。

汇编语言程序设计实验报告



商品名称	折扣	进货价	销售价	进货总数	已售数量	推荐度
PEN	4	5	8	50	40	21
NOTE	8	1	2	100	50	8
电风扇	5	30	50	30	2	5

图 5.2 商品信息显示示意图

5.3 任务 5.1 实验过程

5.3.1 实验方法说明

1. List Sort 的流程描述（基于选择排序的思想）

- (1) $5 \rightarrow \text{num}$
- (2) $5 \rightarrow \text{ecx}$
- (3) $0 \rightarrow \text{recommend}$
- (4) $-1 \rightarrow \text{maxarg}$
- (5) ecx 是否等于 0，若是则跳转到 (10)，否则跳转到 (6)
- (6) 第 ecx 个商品是否被选择过，若被选择过则跳转到 (7)，否则跳转到 (8)
- (7) $\text{ecx}-1 \rightarrow \text{ecx}$ ，回到 (5)
- (8) 第 ecx 个商品的推荐度是否比 recommend 大，若是，跳转到 (9)，否则跳转到 (7)
- (9) 第 ecx 个商品的推荐度 $\rightarrow \text{recommend}$ ， $\text{ecx} \rightarrow \text{maxarg}$ ，回到 next
- (10) maxarg 记录为当前所有未被选择的商品中推荐度最大的商品，并将其标记为已被选择
- (11) $\text{num}-1 \rightarrow \text{num}$ ， num 是否为 0，若为 0 则结束，否则返回 (2)

2. 实验步骤

- (1) 准备上机环境，编辑、汇编、连接以生成可执行文件，测试程序是否完成既定任务。
- (2) 在 32 位调试工具下调试程序，体会 32 位下调试工具与 16 位调试工具的区别。
- (3) 在调试工具中观察 INVOKE 指令的机器码，记录参数压栈顺序的规律。
- (4) 在调试工具中单步跟踪到调用系统 API 函数的位置，观察相关代码的特点。
- (5) 在调试工具中观察结构变量中相关字段的偏移，体会结构变量的优点。
- (6) 在调试工具中观察简化段的效果。
- (7) 尝试使用汇编语言程序的源码级调试工具，体会源码级调试和非源码级调试的异同。
- (8) 观察 WIN32 执行程序代码的特点和执行流程，体会基于窗口的应用程序所包含的四个部分之间的衔接关系。
- (9) 通过修改偏移地址来观察与被调试程序无关的区间，避免对应内存的数据因为被系统保护而读不出来。

汇编语言程序设计实验报告

5.3.2 实验记录与分析

1. 测试此基于窗口的应用程序。计算推荐度之前商品列表见图 5.3，计算推荐度完成之后提示信息见图 5.4，根据推荐度排名后的商品列表见图 5.5。



itemname	discount	purchased cost	price	stock	sold	recommendation
Pencil	8	1	5	100	65	0
Pen	8	6	10	80	40	0
Ruler	7	2	10	50	29	0
Book	9	50	60	65	30	0
Bag	6	80	120	20	10	0

图 5.3 计算商品推荐度之前的商品列表

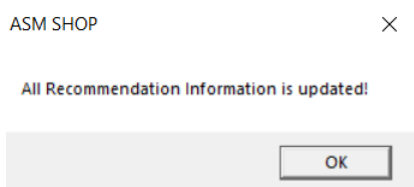


图 5.4 商品推荐度计算完成后的提示信息



itemname	discount	purchased cost	price	stock	sold	recommendation
Bag	6	80	120	20	10	174
Book	9	50	60	65	30	147
Pen	8	6	10	80	40	128
Pencil	8	1	5	100	65	73
Ruler	7	2	10	50	29	73

图 5.5 商品推荐计算完成后排名结果

2. 在 TD32 中调试可执行程序 TASK1.EXE（见图 5.6）。

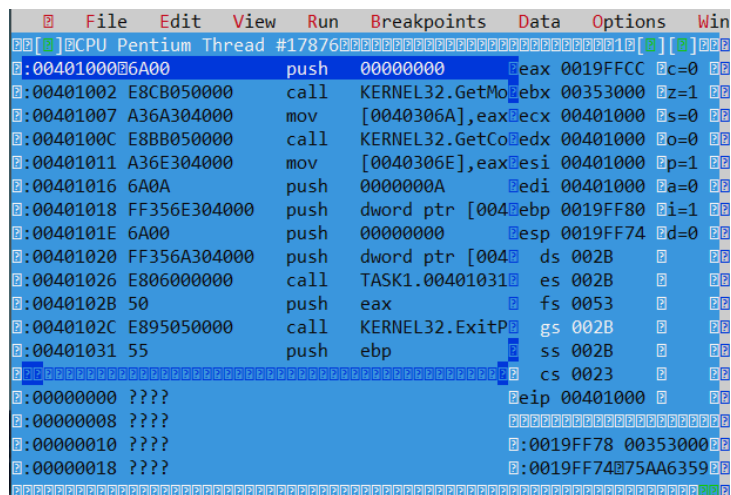


图 5.6 TD32 界面

可以看到 32 位调试工具大体和 16 位调试工具类似，但寄存器是直接显示的 32 位通用寄存器，堆栈段也是按 32 位显示的，指令的偏移地址、EIP 的值也都是 32 位的。

汇编语言程序设计实验报告

3. 在 TD 调试工具中可以看到（见图 5.7），INVOKE 指令在汇编后被分解为压栈指令 PUSH 和转移指令 CALL，而且可以观察到参数入栈的顺序，即右边的参数先入栈，左边的参数最后入栈（STDCALL）。

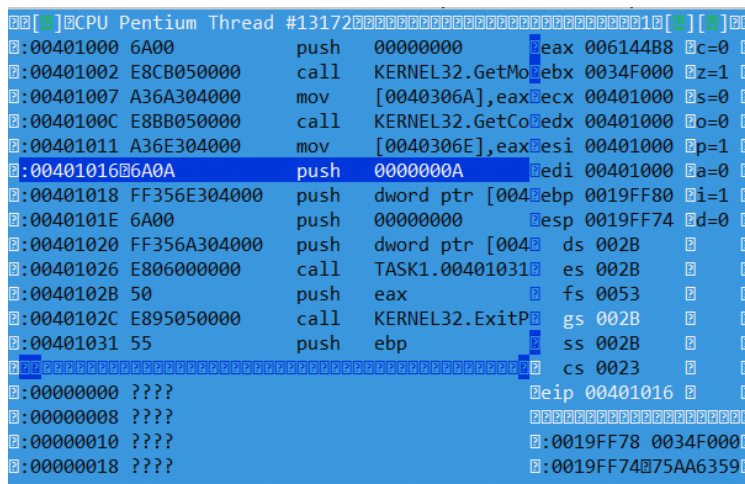


图 5.7 INVOKE 指令的机器码与参数入栈顺序

4. 在图 5.7 中，我们发现，从 16 位程序到 32 位程序的过程中，通用寄存器、偏移地址等等都从 16 位变成了 32 位，但段寄存器仍然为 16 位。因而在 32 位程序中段寄存器指向的不可能是段首址，应该是指向一个段描述符。这样，一方面可以保护与此程序无关的内存空间（保护方式），另一方面可以做到向下兼容。

5. 在调试工具中观察数据段（见图 5.8），全部都是‘？’，说明此时 ds 寄存器指向的内存空间与被调试程序无关。原因是该程序用了段的简化定义，而且模式是 FLAT，因而将代码和数据放到了同一个段中。



图 5.8 DS 段寄存器指向的内存空间

5.4 小结

这一次实验的内容在风格上和前面的实验截然不同，16 位段下程序和 WIN32 程序的编写有很大的不同。通过这一次实验，熟悉了 WIN32 程序的设计和调试方法，也了解了 WIN32 编程的各种开发工具及其特点，也用 32 位调试工具 TD32 观察了 WIN32 程序的一些特点，还理解了 16 位段和 32 位段的寻址方式的区别以及实方式和保护方式之间的一些关系。

一开始拿到这个任务的时候是非常没有头绪的，WIN32 编程没有任何的经验可言，好在老师给了一个 demo，这在很大程度上提高了可行性。通过观察、学习、模仿老师的程序，慢慢地完成给定的任务，这个过程也非常的坎坷，因为一开始不知道 WIN32 程序如何调试，如何使用 TD32，如何进行源码级调试，好在老师都准备了非常完备的学习资料，通过学习这些文档，一步一步地把程序给调通了。

汇编语言程序设计实验报告

我觉得这次实验收获最大的是更深刻地了解了 32 位段的寻址方式以及实方式和保护方式的一些关系。在实验之前，我以为 32 位段的段寄存器也是 32 位，但进入调试工具之后有些震惊，有些疑惑，于是开始翻阅上课的笔记和课本，这才理解了这个段寄存器指向的不是段首址，而是一个段描述符。而这个地方也是实方式和保护方式很大的一个不同。在实方式下，所有变量以及指令都是通过显式的段首址+偏移来得到物理地址的，但在保护方式下，虽然本质上也是通过段首址+偏移地址来寻址的，但这个段首址是在段描述符中，并不是显式给出的，再加上权限的管理，这在很大程度上能保护其他程序的内存空间。

在实验过程中也了解了 WIN32 一些常用的方法。比如 INVOKE 指令，其本质就是模仿 CALL 和保护现场。还有结构变量这个概念，有了结构变量这个方法之后，不需要自己再去计算偏移地址，可以直接通过访问成员的方式达到目的。从这两个角度来看，C 语言其实很贴近 WIN32 编程。

这次实验虽然非常坎坷，但还是学到了非常多的东西的，尤其是对于寻址方式的理解，起到了非常重要的作用。

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.《80 X 86 汇编语言程序设计》. 武汉. 华中科技大学出版社. 2005 年. 1 页-348 页
- [2] 张勇. TD.EXE 简要使用说明. 华工汇编在线答疑交流/群文件/虚拟机与 DOS 下编程环境. 2016 年 3 月 22 日上传
- [3] 曹忠升. timer.asm. ACM1801 汇编/群文件/实验. 2020 年 3 月 28 日上传
- [4] 曹忠升. F2T10.asm. ACM1801 汇编/群文件/实验. 2020 年 4 月 4 日上传
- [5] 曹忠升. F10T2.asm. ACM1801 汇编/群文件/实验. 2020 年 4 月 4 日上传
- [6] 周英飏. VS_CASM.zip. 华工汇编在线答疑交流/群文件/C 与汇编混合编程. 2018 年 4 月 26 日上传
- [7] 金良海. WIN32 汇编程序源码级调试_VS2019.rar. 华工汇编在线答疑交流/群文件. 2019 年 5 月 4 日上传

附录 实验程序文件列表

1. 实验一

- (1) 任务 1.2
TASK2.ASM
TASK2.EXE
- (2) 任务 1.3
TASK3.ASM
TASK3.EXE
- (3) 任务 1.4
TASK4.ASM
TASK4.EXE

2. 实验二

- (1) 任务 2.1
TASK1.ASM
TASK1.EXE
- (2) 任务 2.2
TASK2.ASM
TASK2.EXE

3. 实验三

- (1) 任务 3.1
F2T10.ASM
F10T2.ASM
NEW.ASM
TASK1.ASM
TASK1.EXE
- (2) 任务 3.2
c_asm.asm
MAIN.C
CASM.exe

4. 实验四

- (1) 任务 4.1
F2T10.ASM
F10T2.ASM
NEW.ASM
TASK1.ASM
TASK1.EXE
TRAN.ASM
- (2) 任务 4.2
F2T10.ASM
F10T2.ASM
NEW.ASM
TASK1.ASM
TASK1.EXE
TRAN.ASM
- (3) 任务 4.3
A2_2.EXE

5. 实验五

汇 编 语 言 程 序 设 计 实 验 报 告

- (1) 任务 5.1
 - menu.rc
 - menu.RES
 - menuID.inc
 - TASK1.asm
 - TASK1.exe