

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 汇编语言编程基础**

**院 系 ：计算机科学与技术**

**专业班级 ： 计算机202201**

**学 号 ： \_U202215365**

**姓 名 ： 叶俊江**

**指导教师 ： 朱 虹**

**2024 年 4 月 2 日**

**一、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、调试工具的使用；
2. 熟悉分支、循环程序、子程序的结构，掌握分支、循环、子程序的调试方法；

(3) 加深对转移指令、子程序调用和返回指令及一些常用的汇编指令的理解。

**二、实验内容**

**任务2.1 习题三，第2题。**

要求：（1）分别记录执行到“mov $10, %ecx”和“mov $1, %eax”之前的EBX,EBP,ESI,EDI各是多少。

（2）记录程序执行到退出之前数据段开始40个字节的内容，指出程序运行结果是否与设想的一致。

(3)在标号lopa前加上一段程序，实现新的功能：先显示提示信息“Press any key to begin!”, 然后，在按了一个键之后继续执行lopa处的程序。

**任务2.2 习题三，第3题。**

要求：(1) 内存单元中数据的访问采用变址寻址方式。

(2) 记录程序执行到退出之前数据段开始40个字节的内容，检查程序运行结果是否与设想的一致。

(3) 观察并记录机器指令代码在内存中的存放形式，并与反汇编语句及自己编写的源程序语句进行对照，也与任务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形式）。

**任务2.3 设计实现一个数据处理的程序**

有一个计算机系统运行状态的监测系统会按照要求收集三个状态信息a，b，c（均为有符号双字整型数）。假设4组状态信息已保存在内存中。对每组的三个数据进行处理的模型是f=(5a+b-c+100)/128（最后结果只保留整数部分）。当f<100时，就将该组数据复制到LOWF存储区，当f=100时，就将该组数据复制到MIDF存储区，当f>100时，就将该组数据复制到HIGHF存储区。

每组数据的定义方法可以参考如下：

sdmid .fill 9, 1, 0 # 每组数据的流水号（可以从1开始编号）

sda .long 256809 # 状态信息a

sdb .long -1023 # 状态信息b

sdc .long 1265 # 状态信息c

sf .long 0 # 处理结果f

请编写完整的汇编语言程序，并按照要求设计子程序。

(1) 编写一子程序calculate，完成f的计算并保存，并且子程序的入口参数为esi（调用子程序前后esi的值不变），保存状态信息a的地址，若f=100，则(eax)=0, f>100, 则(eax)=1, 若f<100则(eax)=-1，返回eax；先按照不考虑溢出的要求编写程序，再按照考虑溢出的要求修改程序；

(2) 编写一子程序copy\_data，要求用堆栈传递参数，实现一组存储区状态信息的复制，要求每次拷贝4字节，多余的1个字节单独拷贝，参数为待复制的存储区的首地址和拷贝的目标地址，拷贝的字节长度。

(3) 查看运行到返回操作系统的指令之前，三个存储区域LOWF、MIDF、HIGHF内前50字节的值，只需要截图即可。

(4) 熟悉加减乘除等运算指令，内存拷贝方法。思考：如果三个状态信息是无符号数，程序需要做什么调整？

**三、实验记录及问题回答**

（1）任务 2.1 的运行结果等记录

①运行结果

执行到“mov $10, %ecx”之前查询EBX,EBP,ESI,EDI的值：

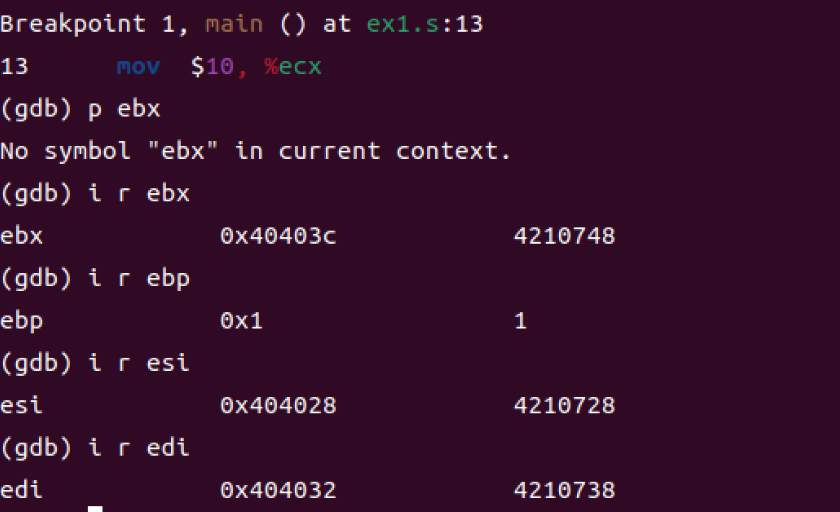


图 3.1-1查询EBX,EBP,ESI,EDI的值

执行到“mov $1, %eax”之前查询EBX,EBP,ESI,EDI的值：

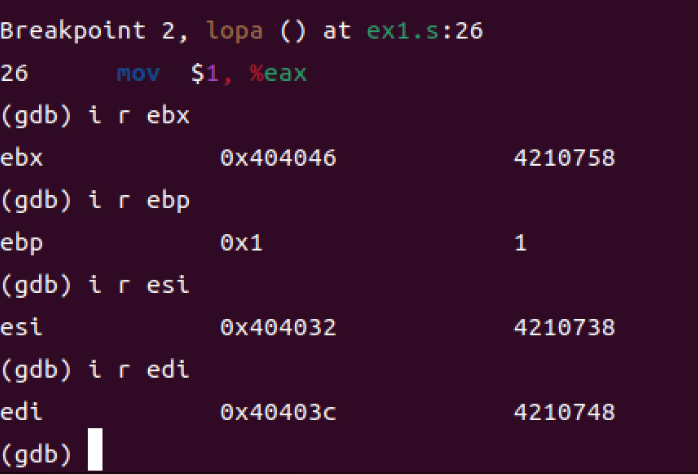


图 3.1-2查询EBX,EBP,ESI,EDI的值

②记录的运行结果并与预期对比

由程序代码可知，该程序是一个循环程序，有四个数据段buf1～4，数据段buf1值为0，1，2，3，4，5，6，7，8，9。buf2～4先由.fill复制10个字节大小且初始值为0的数据段，在程序中通过计数器CX=10来控制循环次数来给buf2～4分别赋值，buf2为0，1，2，3，4，5，6，7，8，9；buf3为1，2，3，4，5，6，7，8，9，A；buf4为4，5，6，7，8，9，A，B，C，D。即预期结果为：

buf1 = 00, 01, 02, 03, 04, 05, 06, 07, 08, 09

buf2 = 00, 01, 02, 03, 04, 05, 06, 07, 08, 09

buf3 = 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A

buf4 = 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D

调试前先读取数据段中buf1的地址：

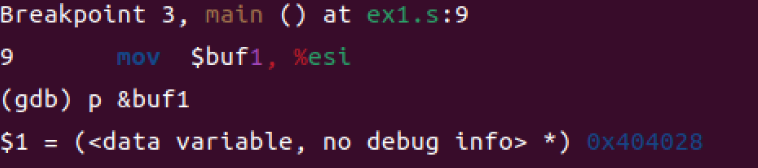


图 3.1-3查询buf1的地址

buf1的地址为0x404028。

在主程序进行之前设置断点，根据buf1的地址读取数据段前40字节的内容：

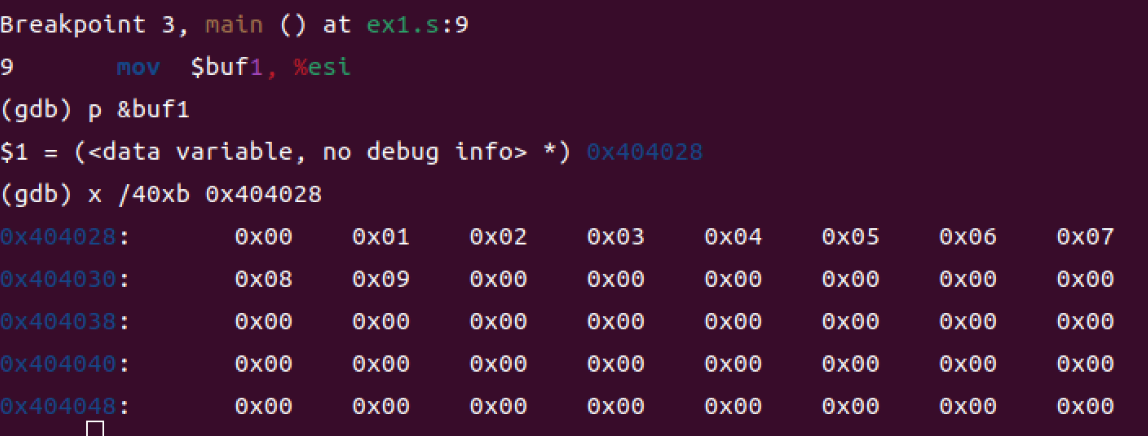


图 3.1-4数据段前40字节的内容

在主程序结束之前设置断点，再次读取内容：

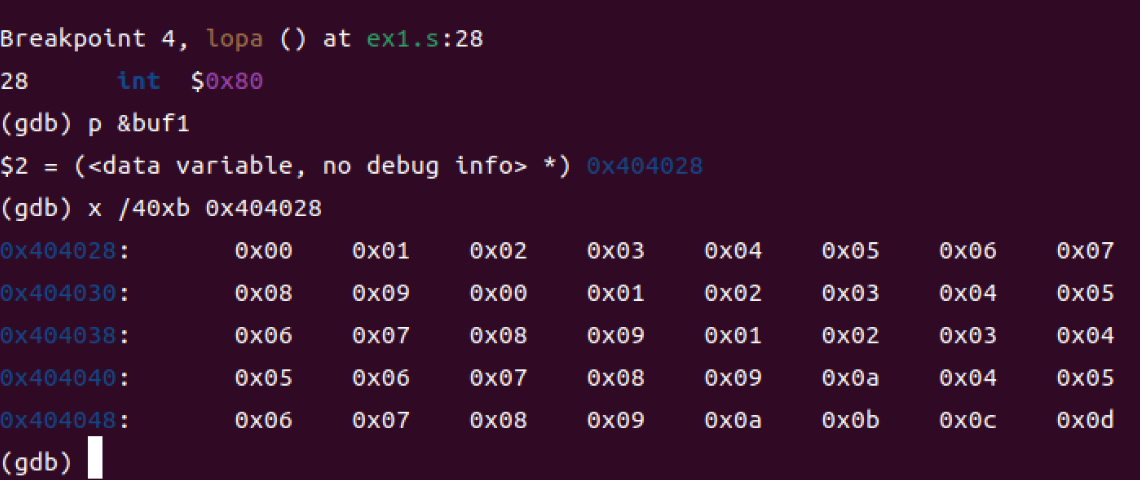


图 3.1-5数据段前40字节的内容

由此可知，运行结果与预期结果一致。

③修改的代码以及运行结果

添加字符串、字符变量：

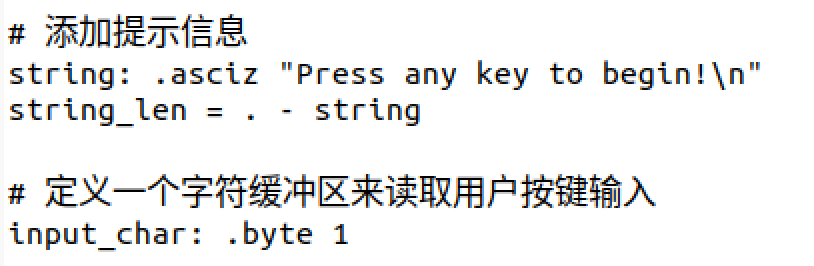


图 3.1-6修改代码段

在lopa标识之前调用write和read程序：

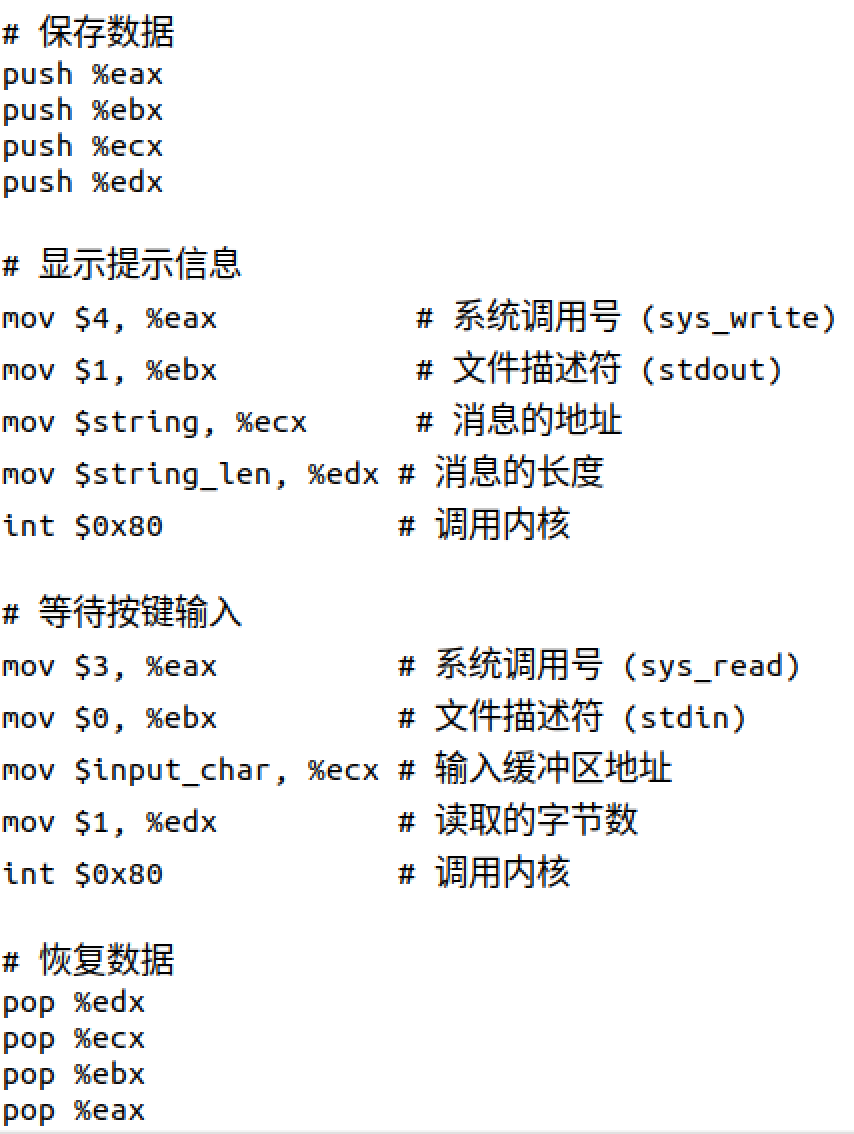


图 3.1-6修改代码段

运行结果：

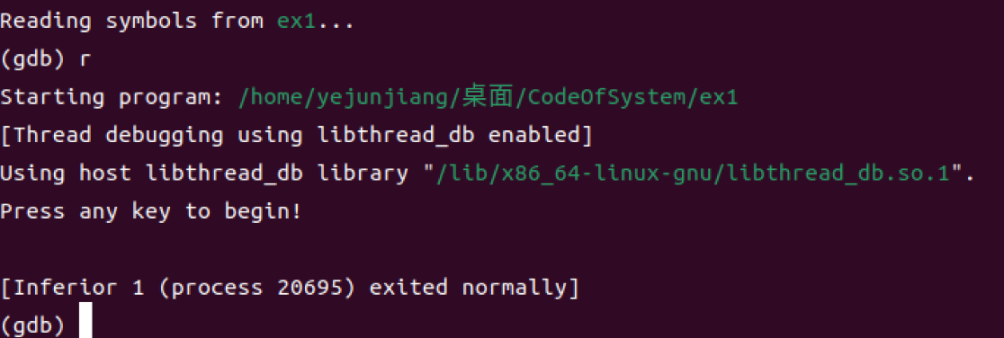


图 3.1-7运行结果截图

（2）任务 2.2 的算法思想、运行结果等记录

①修改的代码

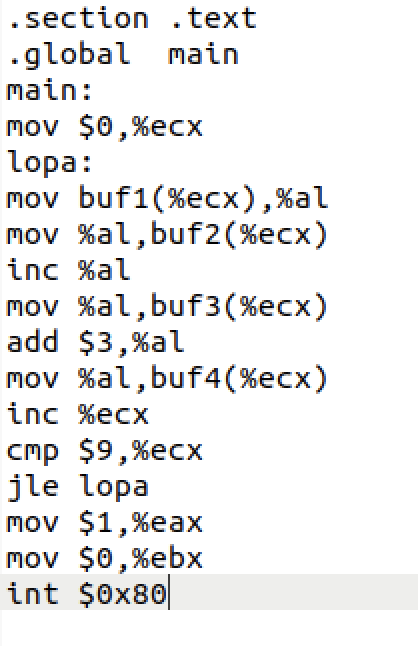


图 3.2-1修改代码段

②运行结果与预期对比

与上文任务一类似地：

主程序执行前：

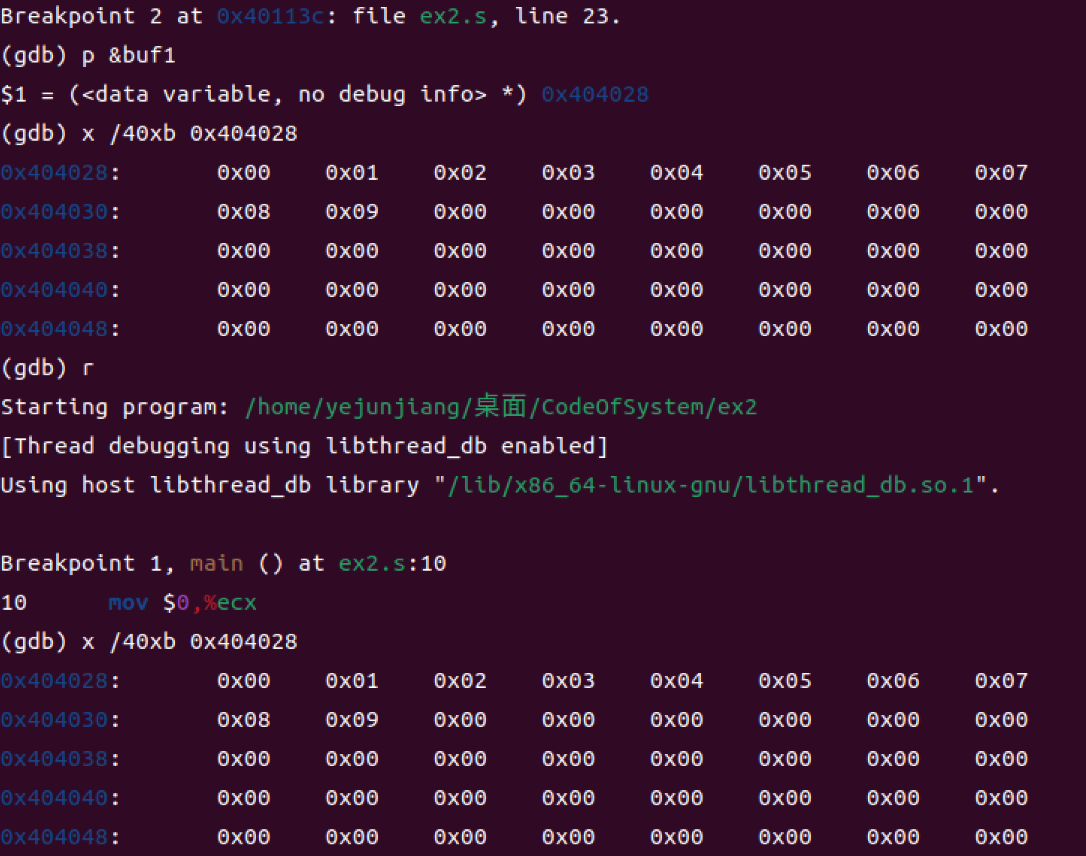


图 3.2-2数据段前40字节的内容

主程序结束前：

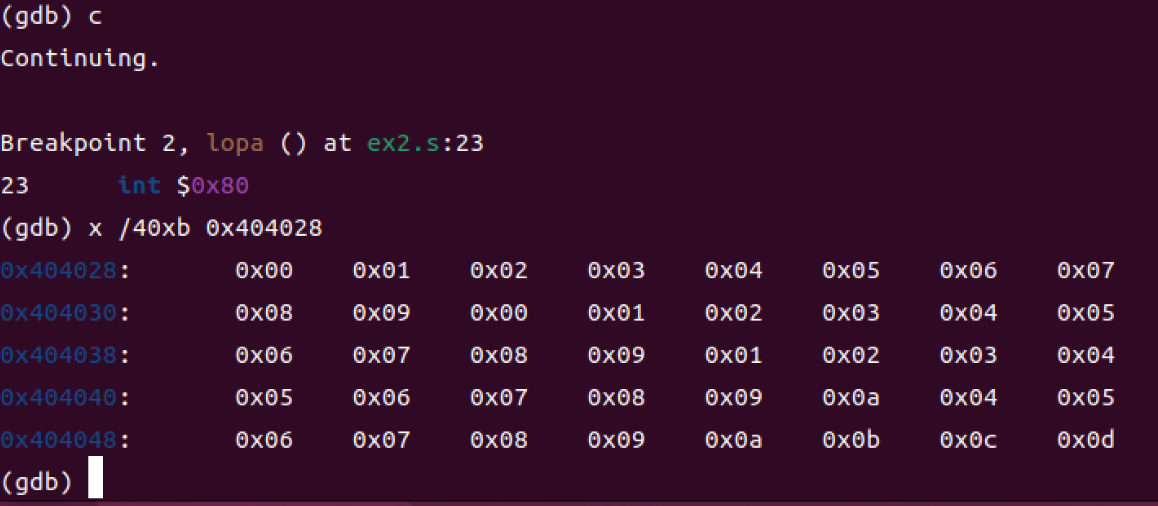


图 3.2-3数据段前40字节的内容

与预期结果对比，与预期一致。

③观察记录

表 3.2-1机器指令、反汇编语句与源代码语句对比

| **机器指令** | **反汇编语句** | **源代码语句** |
| --- | --- | --- |
| cd 21 | int 21h | syscall |
| 8b 45 08 | mov 0x8(%ebp),%eax | mov %eax, var1 |
| 89 45 fc | mov %eax,0xfffffffc(%ebp) | mov var2,%eax |
| 83 ec 0c | sub $0xc,%esp | sub $12,%esp |
| 83 f8 01 | cmp $0x1,%eax | cmp $1,%eax |
| 75 06 | jne 0x6 | jne 0x6 |

分析，有：

·机器指令以二进制形式存在，需要通过反汇编才能转换成人类可读的汇编语句。

·反汇编语句是机器指令的人类可读形式，通常以汇编语言表示。

·源代码语句是高级语言代码的表示，经过编译后生成相应的机器指令。

通过对机器指令、反汇编语句和源代码语句的对比分析，可以较为深入了解计算机底层运行的机制。

1. 机器指令的组成

机器指令是计算机硬件直接执行的操作命令，由操作码（Opcode）和地址码（Address）两部分组成。操作码指示计算机要执行的操作类型，而地址码提供操作数和结果存放的内存地址，如mov $1, %eax。

1. 反汇编过程中的转换规则

在进行反汇编时，我们遵循以下转换规则：

常数表示：常数通常以十六进制形式表示，以便清晰地展示二进制数据的结构。例如，十进制数9可能在反汇编中表示为0x9。

地址表示：源代码中的地址和标签会被转换为它们在内存中的实际偏移地址，以确保指令的正确执行。

1. 不同寻址方式的特点

针对不同的寻址方式，我们观察到以下特点：

直接寻址：指令中直接给出操作数的内存地址。在反汇编时，源代码中的地址会被转换为对应的十六进制地址，以便指示指令在内存中的位置。

立即数寻址：操作数直接以立即数的形式包含在指令中。在反汇编过程中，常常将这些立即数转换为十六进制表示，以便更好地理解。

寄存器寻址：操作数存储在寄存器中。寄存器寻址在反汇编前后的表示通常是一致的，因为寄存器名称在汇编语言和机器指令中都是固定的。

变址寻址：涉及到基地址和偏移量。在反汇编过程中，变量或标签会被转换为它们相对于基地址的偏移量，以便正确计算地址。

通过理解这些转换规则和特点，我们可以更好地理解计算机底层的工作原理，并能够更有效地理解反汇编代码。

（3）任务 2.3 的算法思想、流程图、遇到的问题及其解决方法（选择3-4个）、运行结果等记录

①算法实现

calculate子程序的算法思路：

calculate子程序首先保存了现场的寄存器内容，以防止被调用时破坏。这包括保存了 ebp, ebx, ecx, edx, 和 edi 寄存器的内容。接下来，子程序从 esi 寄存器所指向的内存位置读取了三个整数值，分别是 a, b, 和 c。然后，子程序开始执行一系列数学计算，这些计算涉及乘法、加法和减法，这些操作是为了根据特定的逻辑判断条件来修改结果。

首先，将 a 与 5 相乘，结果保存在 eax 中，然后与 b 相加。如果这一步出现溢出，则会将 edx寄存器中的溢出标志设置为 1。接着，将结果减去 c，同样如果出现溢出，则更新溢出标志。接下来，将结果加上 100。如果出现溢出，则更新溢出标志。然后，通过右移eax寄存器 7 位，将 eax 中的高位提取出来，并将其存储在 ebx 中。接着，比较溢出标志 edx 是否小于等于 0，如果是，则跳到标签 next4。如果高位有数字，则将 ebx 设置为 101。最后，根据 f` 的值设置 eax寄存器的返回值。如果 f 的值小于 100，则将 eax 设置为 -1。如果 f 的值等于 100，则将 eax 设置为 0。如果 f 的值大于 100，则将 eax 设置为 1。最后，子程序恢复现场寄存器的内容，并返回。

copy\_data子程序的算法思路：

copy\_data子程序同样也保存了现场的寄存器内容。子程序从堆栈中读取了三个参数：源地址、目标地址和字节长度。然后，子程序通过移动数据的方式将源地址处的数据复制到目标地址处，直到完成了字节长度指定的拷贝长度。首先，检查要拷贝的字节长度是否小于 4，如果是，则跳到标签 copy\_last\_byte。否则，从源地址读取 4 个字节的数据，将其拷贝到目标地址，并将指针向后移动 4 个字节。然后，将字节长度减去 4。重复上述步骤直至字节长度小于 4。如果字节长度不足 4 个字节，剩余的不足 4 个字节的数据将单独进行拷贝。最后，子程序恢复现场寄存器的内容，并返回。

②流程图

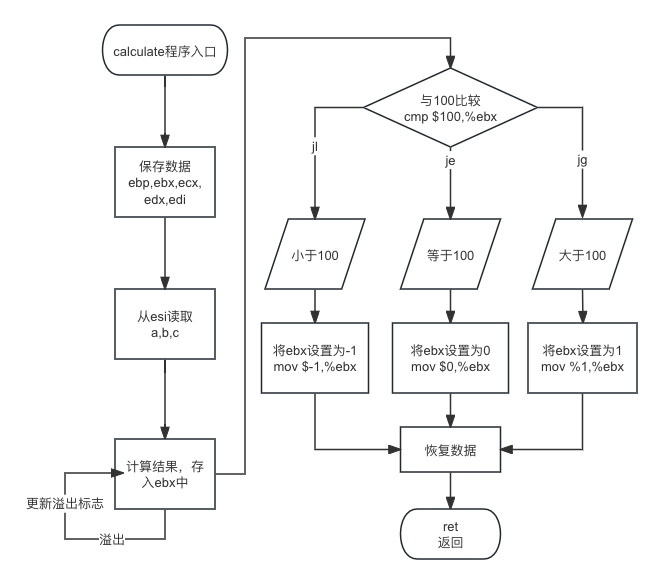


图 3.3-1 calculate子程序流程图

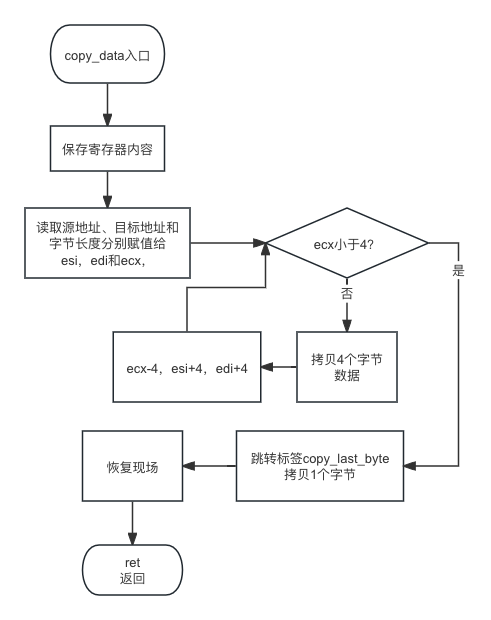


图 3.3-2 copy\_data子程序流程图

③遇到的问题及其解决方法

1. 调试问题。一开始没有注意到代码入口为\_start，在使用终端命令编译文件到时候还是按照入口为 main的指令进行编译，导致卡了一会。后面在同学提醒下发现问题，查阅资料解决了。
2. 溢出和标志位处理。在进行数学运算时，可能会出现溢出的情况，此时需要正确处理溢出，并根 据标志位进行逻辑判断，当时编写程序时未考虑到。后来是对比了同学已经实现到程序代码才发现这 个问题。
3. 数据拷贝。在 copy\_data 子程序中，需要实现将数据从源地址复制到目标地址的功能，这需要考 虑到字节长度、循环拷贝等问题。后来查阅到循环处理的方法，便将方法用于程序中了。

④运行结果(展示LOWF,HIGHF,MIDF三个存储区的数据)

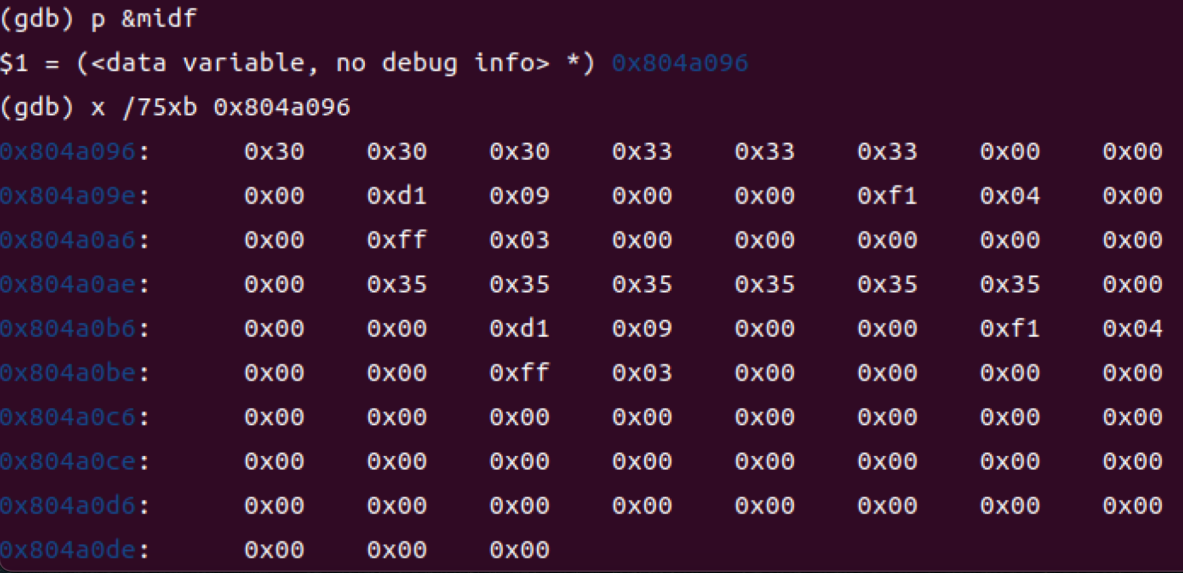


图 3.3-3 midf存储区数据

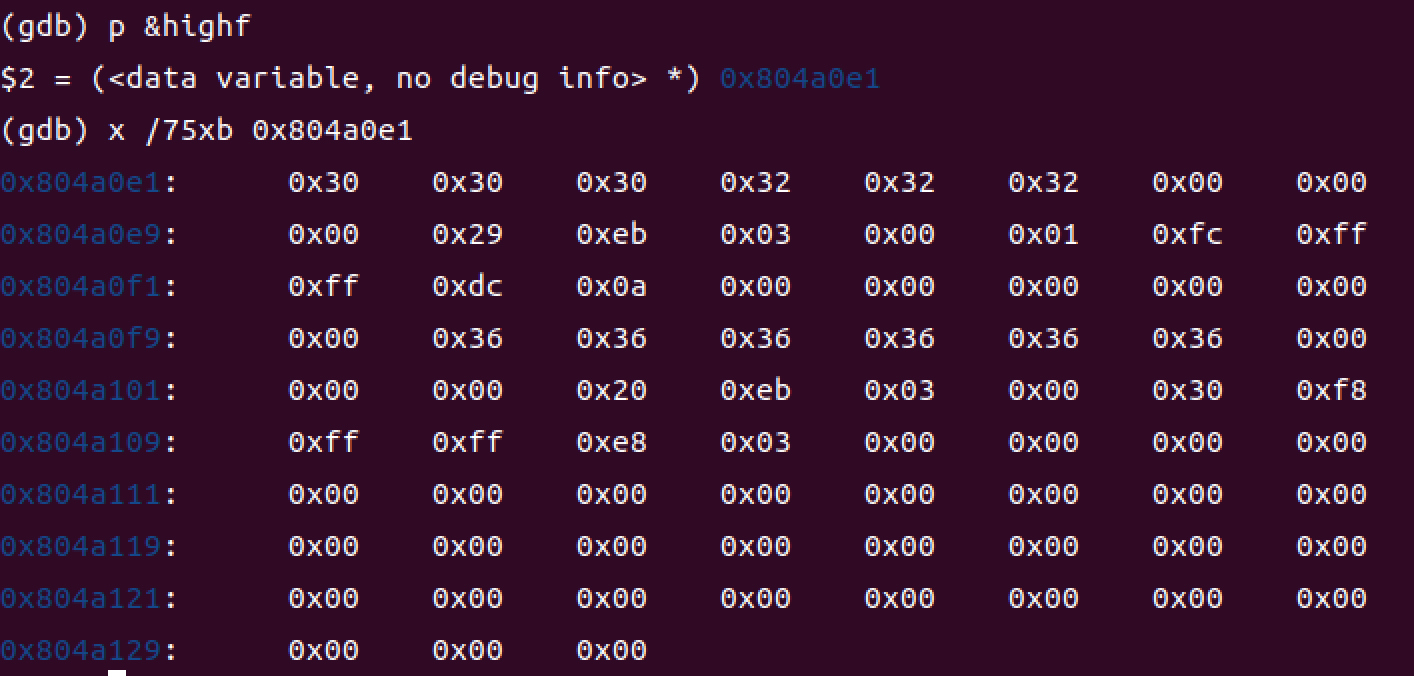


图 3.3-4 highf存储区数据

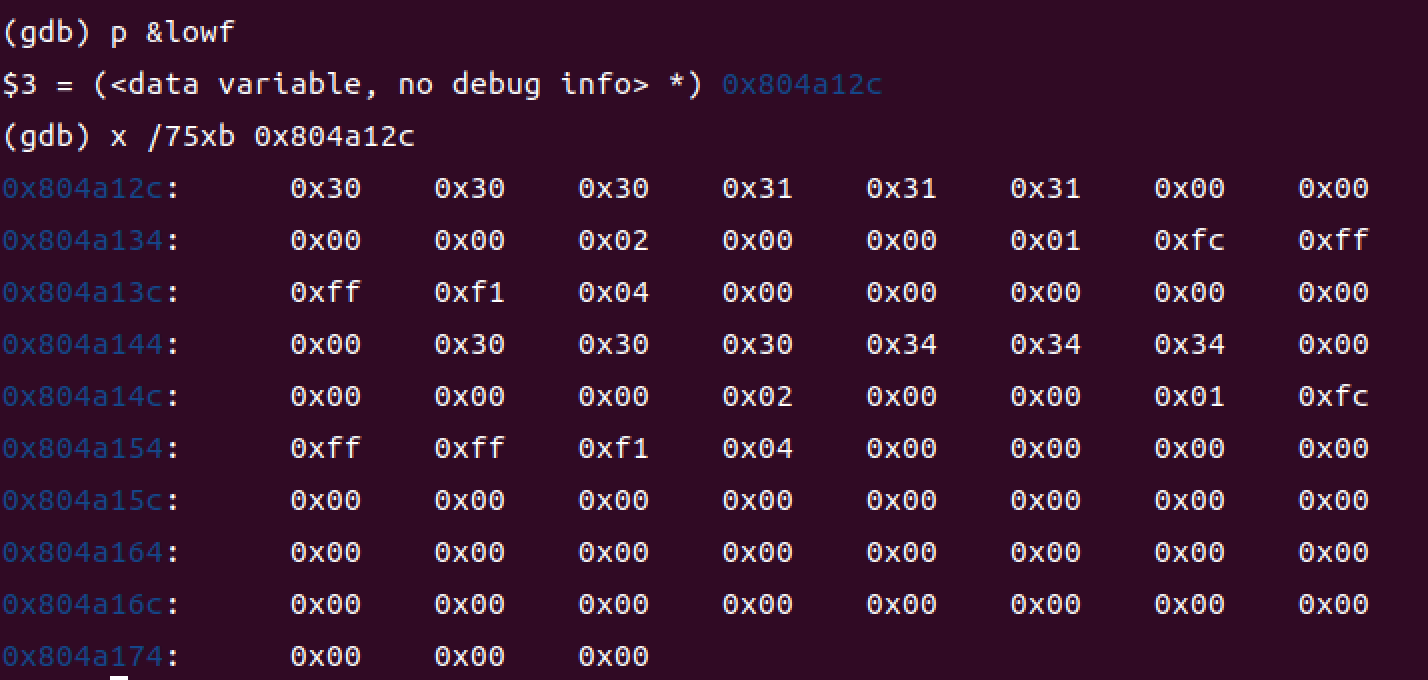


图 3.3-5 lowf存储区数据

⑤思考（不一定要写成程序，但是一定要有解决思路）

在既有程序中，有无符号数的判断区别在于跳转指令不同，即将jg改成ja，jno改成jnc即可满足。

**四、体会**

通过这次实验任务，我深入了解了汇编语言的基本结构和一些常用指令的使用。在任务中，我掌握了gdb的基本使用方法，并熟悉了分支、循环程序、子程序的结构以及它们的调试方法。此外，我加深了对转移指令、子程序调用和返回指令以及一些常用的汇编指令的理解。

在实验过程中，任务2.1和2.2我完成的比较顺利。我通过分析程序运行结果与预期对比，以及观察机器指令、反汇编语句和源代码语句的对应关系，加深了对计算机底层运行机制的理解，学会了如何在汇编语言中实现一些基本的算术运算、条件判断和循环结构，以及如何通过堆栈传递参数来实现数据的复制操作。

在任务2.3中，我设计并实现了一个数据处理的程序，通过子程序实现了对数据的处理和复制，并且根据特定条件将数据存储到不同的存储区域。在实现过程中，我遇到了一些问题，如调试问题、溢出处理和数据拷贝等，但通过查阅资料和与同学讨论，我逐步解决了这些问题，并且完善了程序的功能。

经历本次实验，我不仅学会了如何编写和调试汇编程序，还深入了解了计算机底层的工作原理，加强了对计算机体系结构的理解。

**五、源码**

实验任务2.1

.section .data

buf1: .byte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

buf2: .fill 10, 1, 0

buf3: .fill 10, 1, 0

buf4: .fill 10, 1, 0

# 添加提示信息

string: .asciz "Press any key to begin!\n"

string\_len = . - string

# 定义一个字符缓冲区来读取用户按键输入

input\_char: .byte 1

.section .text

.global main

main:

mov $buf1, %esi

mov $buf2, %edi

mov $buf3, %ebx

mov $buf4, %edx

mov $10, %ecx

# 保存数据

push %eax

push %ebx

push %ecx

push %edx

# 显示提示信息

mov $4, %eax # 系统调用号 (sys\_write)

mov $1, %ebx # 文件描述符 (stdout)

mov $string, %ecx # 消息的地址

mov $string\_len, %edx # 消息的长度

int $0x80 # 调用内核

# 等待按键输入

mov $3, %eax # 系统调用号 (sys\_read)

mov $0, %ebx # 文件描述符 (stdin)

mov $input\_char, %ecx # 输入缓冲区地址

mov $1, %edx # 读取的字节数

int $0x80 # 调用内核

# 恢复数据

pop %edx

pop %ecx

pop %ebx

pop %eax

lopa: mov (%esi), %al

mov %al, (%edi)

inc %al

mov %al, (%ebx)

add $3, %al

mov %al, (%edx)

inc %esi

inc %edi

inc %ebx

inc %edx

dec %ecx

jnz lopa

mov $1, %eax

movl $0, %ebx

int $0x80

实验任务2.2

.section .data

buf1: .byte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

buf2: .fill 10, 1, 0

buf3: .fill 10, 1, 0

buf4: .fill 10, 1, 0

.section .text

.global main

main:

mov $0,%ecx

lopa:

mov buf1(%ecx),%al

mov %al,buf2(%ecx)

inc %al

mov %al,buf3(%ecx)

add $3,%al

mov %al,buf4(%ecx)

inc %ecx

cmp $9,%ecx

jle lopa

mov $1,%eax

mov $0,%ebx

int $0x80

实验任务2.3

.section .data

sdmid: .ascii "000111", "\0\0\0" # 每组数据的流水号（可以从1开始编号）

sda: .long 512 # 状态信息a

sdb: .long -1023 # 状态信息b

sdc: .long 1265 # 状态信息c

sf: .long 0 # 处理结果f

.ascii "000222","\0\0\0"

.long 256809 # 状态信息a

.long -1023 # 状态信息b

.long 2780 # 状态信息c

.long 0 # 处理结果f

.ascii "000333","\0\0\0"

.long 2513# 状态信息a

.long 1265 # 状态信息b

.long 1023 # 状态信息c

.long 0 # 处理结果f

.ascii "000444","\0\0\0"

.long 512 # 状态信息a

.long -1023 # 状态信息b

.long 1265 # 状态信息c

.long 0 # 处理结果f

.ascii "555555","\0\0\0"

.long 2513

.long 1265

.long 1023

.long 0

.ascii "666666","\0\0\0"

.long 256800

.long -2000

.long 1000

.long 0

num = 6

midf: .fill 9, 1, 0

.long 0, 0, 0, 0

.fill 9, 1, 0

.long 0,0,0,0

.fill 9, 1,0

.long 0,0,0,0

highf: .fill 9, 1, 0

.long 0, 0, 0, 0

.fill 9, 1, 0

.long 0,0,0,0

.fill 9,1,0

.long 0,0,0,0

lowf: .fill 9, 1, 0

.long 0, 0, 0, 0

.fill 9, 1, 0

.long 0,0,0,0

.fill 9,1,0

.long 0,0,0,0

len=25

.section .text

.global \_start

\_start:

# 这里要写

pushl %ebp

movl %esp, %ebp

# -4(读id) -16(midf p) -12(highf p) -8(lowf p)

sub $16, %esp

movl $0, -4(%ebp)

movl $lowf, -8(%ebp)

movl $highf, -12(%ebp)

movl $midf, -16(%ebp)

mov $sdmid, %ebx

L1:

lea 9(%ebx), %esi

call calculate

push $len

cmp $0, %eax

jne L2

pushl -16(%ebp)

addl $len, -16(%ebp)

jmp L4

L2:

jg L3

pushl -8(%ebp)

addl $len, -8(%ebp)

jmp L4

L3:

pushl -12(%ebp)

addl $len, -12(%ebp)

L4:

mov -4(%ebp), %ecx

push %ebx

call copy\_data

L5:

add $len, %ebx

incl -4(%ebp)

cmpl $num, -4(%ebp)

jl L1

movl %ebp, %esp

popl %ebp

mov $1, %eax

mov $0, %ebx

int $0x80

.type calculate @function

calculate:

push %ebp

mov %esp, %ebp

pushl %ebx #保存数据

pushl %ecx

pushl %edx

pushl %edi

movl (%esi), %eax # eax = a

movl 4(%esi), %ebx # ebx = b

movl 8(%esi), %ecx # ecx = c

# 开始计算

mov $5 , %edi

imull %edi

addl %ebx, %eax

jno next1

addl $1, %edx

next1:

subl %ecx, %eax

jno next2

subl $1, %edx

next2:

addl $100, %eax

jno next3

addl $1, %edx

next3:

# 如果高位有数字，则直接大于100,不怕高位是符号1,因为全是1的话为-1

sarl $7, %eax

mov %eax, %ebx

cmp $0, %edx

jle next4

mov $101, %ebx

next4:

# 根据f的值设置eax

cmpl $100, %ebx

jl less\_than\_100

je equal\_to\_100

jg greater\_than\_100

less\_than\_100:

movl $-1, %eax

jmp end\_calculate

equal\_to\_100:

movl $0, %eax

jmp end\_calculate

greater\_than\_100:

movl $1, %eax

end\_calculate:

popl %edi

popl %edx

popl %ecx

popl %ebx

mov %ebp, %esp

pop %ebp

ret

.type copy\_data @function

copy\_data:

# 使用堆栈传递参数: 源地址, 目标地址, 字节长度

push %ebp

mov %esp, %ebp

push %ecx

push %esi

push %edi

push %eax

movl 16(%ebp), %ecx # ecx = 字节长度

movl 12(%ebp), %esi # esi = 源地址

movl 8(%ebp), %edi # edi = 目标地址

# 拷贝数据，每次4字节，剩余1字节单独拷贝

cpy\_loop:

cmpl $4, %ecx

jl copy\_last\_byte

mov (%edi), %eax

mov %eax, (%esi) # 拷贝4字节

subl $4, %ecx

addl $4, %esi

addl $4, %edi

jmp cpy\_loop

copy\_last\_byte:

je end\_copy

movb (%esi), %al

movb %al, (%edi) # 拷贝1字节

end\_copy:

# 恢复数据

pop %eax

pop %edi

pop %esi

pop %ecx

mov %ebp, %esp

pop %ebp

ret