

**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202203**

**学 号 ： U202215643**

**姓 名 ： 王国豪**

**指导教师 ： 张 勇**

**2024 年 4 月 5 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。

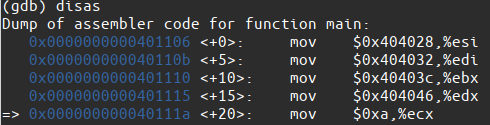
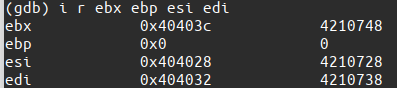
(4) 熟悉加减乘除等运算指令，内存拷贝方法。思考：如果三个状态信息是无符号数，程序需要做什么调整？

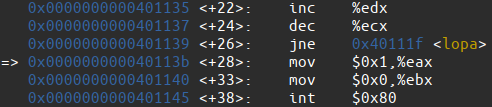
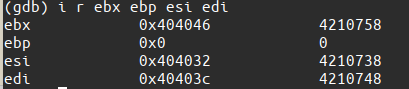
**三、实验记录及问题回答**

（1）任务 2.1 的运行结果等记录

①运行结果

执行到“mov $10, %ecx”前:

  
EBX,EBP,ESI,EDI各是多少  


执行到“mov $1, %eax”之前：  
  
我们再次查看EBX,EBP,ESI,EDI是多少  


②记录的运行结果并与预期对比

该运算程序是一个循环程序，有4个数据段，每个数据段长度为10。第一个数据段BUF1值为0，1，2，3，4，5，6，7，8，9；第二个数据段为BUF2，第三个数据段为BUF3，第四个数据段为BUF4，初始值均为0；通过设置计数器CX=10,来控制循环次数，依次将第二个数据段设为0，1，2，3，4，5，6，7，8，9;将第三个数据段设为1，2，3，4，5，6，7，8，9，A;将第四个数据段设为4，5，6，7，8，9，A,B,C,D;

结果预测：

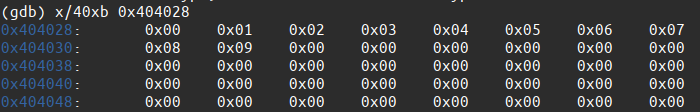
BUF1:00,01,02,03,04,05,06,07,08,09

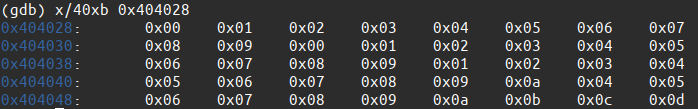
BUF2:00,01,02,03,04,05,06,07,08,09

BUF3:01,02,03,04,05,06,07,08,09,0A

BUF4:04,05,06,07,08,09,0A,0B,0C,0D

数据段以buf1开始，我们先查看buf1的地址  

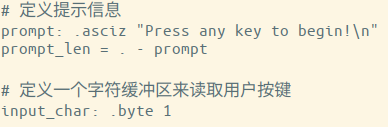

我们输出开始的时候前四十个字节的内容  


我们再执行完程序，查看前四十个字节  


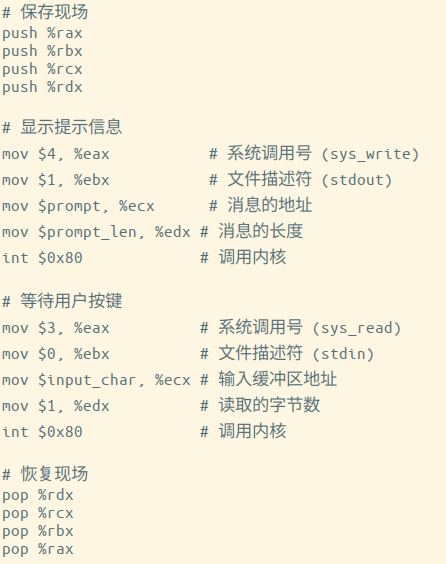
发现和我们的预期一致。

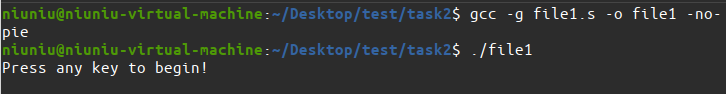
③修改的代码（不要所有代码贴上来）以及运行结果

我们需要定义变量

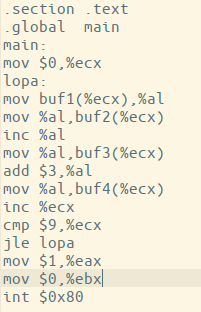


接着直接在lopa之前调用wirte和read



运行截图  


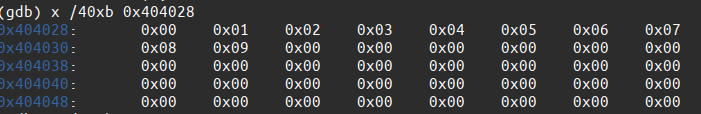
（2）任务 2.2 的算法思想、运行结果等记录

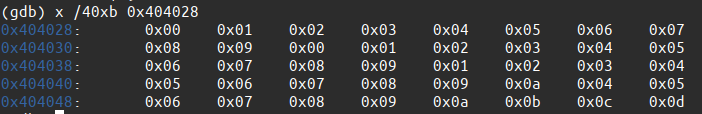
①修改的代码如下：   


②运行结果与预期对比

查看开始数据段开始地址



代码刚开始执行时数据段开始40个字节的内容：  
  
执行完毕数据段开始40个字节的内容：



与我们预期进行比对，发现一致，完美切合。

③观察记录

机器指令，反汇编语句和源代码语句记录表

|  |  |  |
| --- | --- | --- |
| 机器指令 | 反汇编语句 | 源代码语句 |
| b9 00 00 00 00 | mov $0x0,%ecx | mov $0,%ecx |
| 67 8a 81 28 40 40 00 | mov 0x404028(%ecx),%al | mov buf1(%ecx),%al |
| fe c0 | inc %al | inc %al |
| 67 88 81 46 40 40 00 | mov %al,0x404046(%ecx) | mov %al,buf3(%ecx) |
| 83 f9 09 | cmp $0x9,%ecx | cmp $9,%ecx |
| 7e d9 | jle 40110b <lopa> | jle lopa |

通过对比分析我们得出一下结论：

一、机器指令的格式

机器指令是计算机硬件能够直接识别和执行的操作命令，它们以二进制代码的形式保存在内存中。每一条机器指令通常包括两部分：

操作码（Opcode）：它指定了计算机要执行的操作类型，如加法、减法、跳转等。

地址码（Address）：它提供了操作数（即参与运算的数据）和运算结果存放的内存地址。

二、反汇编过程中的转换规则

常数表示：反汇编语句中的常数通常以16进制形式表示，以便更清晰地展示二进制数据的结构。例如，cmp $9,%ecx 在反汇编时可能会变为 cmp $0x9,%ecx。

地址表示：源代码中的地址和标签在反汇编过程中会被转换为它们在内存中的实际偏移地址。

三、不同寻址方式的特点

立即数寻址：在这种寻址方式中，操作数直接以立即数的形式包含在指令中。这些立即数在反汇编时通常会被转换为16进制表示。

直接寻址：直接寻址是指令中直接给出操作数的内存地址。在反汇编过程中，源代码中的地址会被转换为对应的16进制地址。

寄存器寻址：在这种寻址方式中，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，因为寄存器名在汇编语言和机器指令中都是固定的。

变址寻址：变址寻址涉及到一个基地址和一个偏移量。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。

（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子程序实现：

这个子程序分计算(5a + b - c + 100) / 128 ，以及根据运算结果对eax寄存器进行赋值两部分。

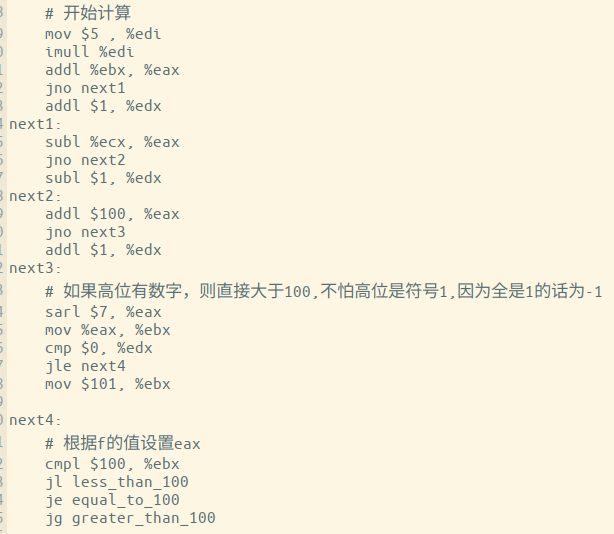
计算的时候我们需要先保存现场，将要用到的寄存器都先push，然后由于esi中保存的是a的地址，我们查看数据data定义可知，a,b,c三个变量的存储地址紧连着，我们就可以采用变址寻址的方式movl 4(%esi), %ecx分别把a,b,c的值存储到寄存器中。5\*a可以采取有符号的双操作数实现，接着的+b和-c都可以用常规的sub和add实现，最后一步的/128可以用算术右移完美实现。根据运算结果对eax寄存器进行赋值这个模块的时候，可以定义三个标签，我们再根据cmpl $100, %ebx的结果进行跳转即可，每个标签运行完以后都会跳转到end\_calculate:模块。

copy\_data子程序实现：

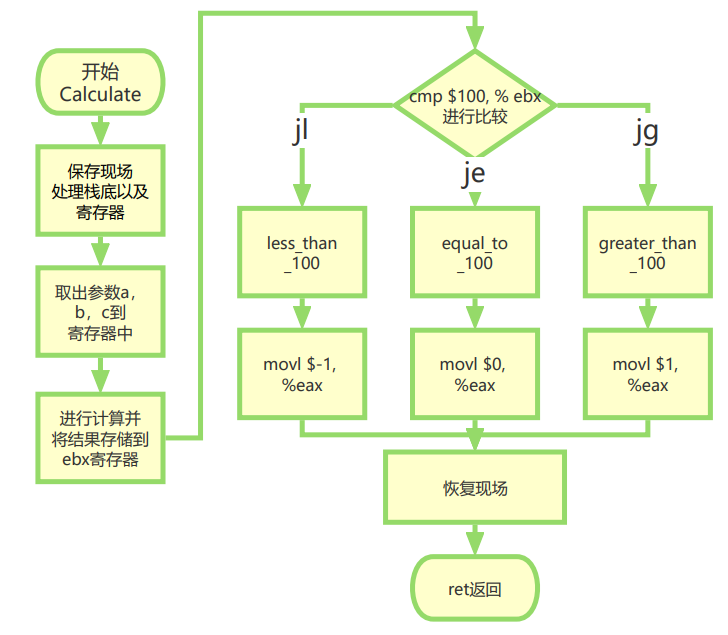
要求用堆栈传递参数，且参数为待复制的存储区的首地址和拷贝的目标地址，拷贝的字节长度。子程序的开始我们就要先push %ebp，mov %esp, %ebp，这是基本的操作，接着我们又开始保存要使用到的寄存器，然后再使用变址寻址的方式分别将字节长度，源地址以及目标地址分别取出来，由于我们需要拷贝的长度为25，我们需要考虑4个字节和1个字节的拷贝，我们先进行4个字节的拷贝，每次比较前cmpl $4, %ecx比较剩余要拷贝的长度和4的大小，如果大于4字节，则接着进行4个字节拷贝流程，否则就进入1个字节拷贝流程，拷贝完成以后我们还需要恢复现场，以及把mov %ebp, %esp然后pop %ebp。

考虑溢出问题：

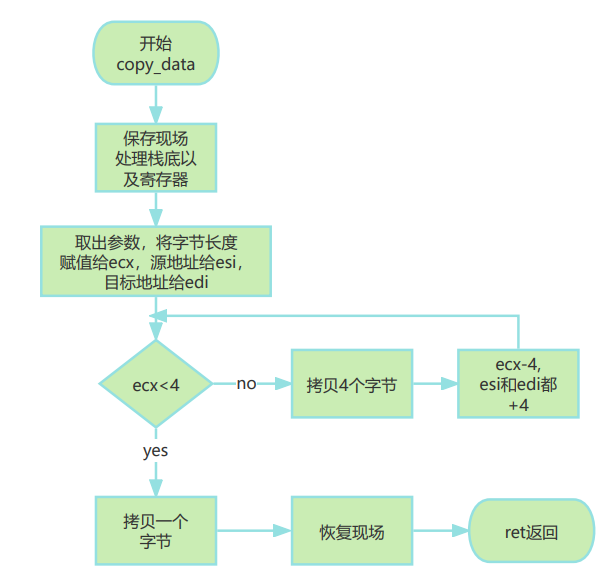
这样的话我们需要将32位扩展为64位，我们可以利用到单操作数乘法，这样我们数据的高位就在edx，有效解决了我们的溢出问题，但是后续进行计算的时候如果eax有进位或者借位的时候相对应的edx就需要+1或者-1处理，且最后一步除以128的时候，如果高位edx有有效数据位，那么除以128后也是大于100的，如果edx无有效数据位，那么直接对eax右移7位再进行判断即可。主要代码实现如下：



②流程图



图表 1 calculate程序流程图



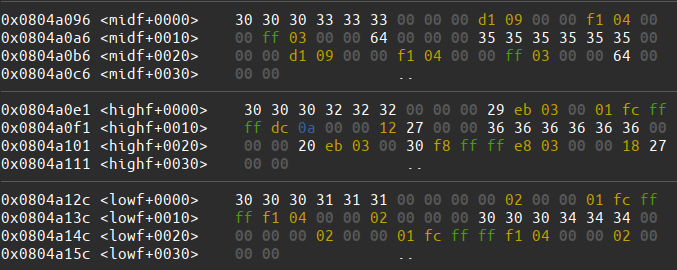
图表 2 copy\_data 子程序流程图

③遇到的问题及其解决方法

我遇到了挺多的问题，下面我们一一讲解：

1. 堆栈传参问题。我一开始写子程序的时候，没有写push %ebp,mov %esp,%ebp,这也导致我的程序读取参数的时候经常出错而找不到问题所在，后来写上这两句话的时候就很容易通过变址寻址的方式读取了参数。
2. 测试问题。写好两个子程序以后，但是没有main函数来测试，导致不知道对错，如果只是仅仅通过gdb调试单个函数的话可能结果正常，但是如果多次调用子程序的话，如果恢复现场等工作没有做到位的话，会导致eip出错，程序报错。专门写了一个main函数模拟题目要求，实现了循环测试所有数据的流程。
3. 解决溢出问题。一开始我没怎么理解溢出问题，以为需要用到rax和edx，但是显然32位编译的话是不能用的，后来在老师的指点下，用了单操作乘法，这样我的高位就在edx，后期我还需要考虑eax向edx进位和借位的问题。

④运行结果(展示LOWF,HIGHF,MIDF三个存储区的数据)



⑤思考（不一定要写成程序，但是一定要有解决思路）

在我的代码中，有符号和无符号运算最大的区别就在与跳转指令的写法，如果改成无符号数字的话，判断大于的jg要改成ja，jno要改成jnc。其他的逻辑不需要改变。

**四、体会**

实践出真知，这次实验让我对堆栈传参，处理溢出问题以及无符号和有符号跳转指令等方面了解更加深入，感谢我的老师能够耐心回答我的问题，让我收获颇丰。这种实践和学习的过程是非常宝贵的。

**五、源码**

实验任务 2.1代码

|  |
| --- |
| .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  # 定义提示信息  prompt: .asciz "Press any key to begin!\n"  prompt\_len = . - prompt    # 定义一个字符缓冲区来读取用户按键  input\_char: .byte 1  .section .text  .global main  main:  mov $buf1, %esi  mov $buf2, %edi  mov $buf3, %ebx  mov $buf4, %edx  mov $10, %ecx  # 保存现场  push %rax  push %rbx  push %rcx  push %rdx  # 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 $prompt, %ecx # 消息的地址  mov $prompt\_len, %edx # 消息的长度  int $0x80 # 调用内核    # 等待用户按键  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  # 恢复现场  pop %rdx  pop %rcx  pop %rbx  pop %rax  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代码

|  |
| --- |
| .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代码

|  |
| --- |
| .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  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:  # 恢复现场  pop %eax  pop %edi  pop %esi  pop %ecx    mov %ebp, %esp  pop %ebp  ret |