

**实验报告**



**题目： 键盘驱动程序的分析与修改**

**班 级： 2018211318**

**学 号： 2018210547**

**姓 名： 胡天翼**

**学 院： 计算机学院**

**2019年 12 月 13 日**

一、实验目的

1. 熟悉linux操作的基本操作；
2. 掌握gcc编译方法；
3. 掌握gdb的调试工具使用；
4. 掌握objdump反汇编工具使用；
5. 熟悉理解反汇编程序（对照源程序与objdump生成的汇编程序）。
6. 实验环境

列举你所使用的软件工具

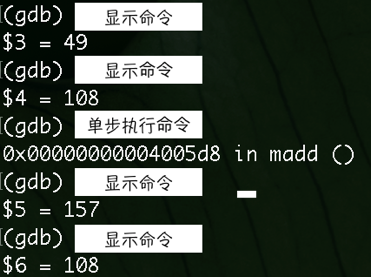
1. Windows10
2. Remote Terminal

三、实验内容

现有int型数组a[i]=i-50,b[i]=i+y，其中y取自于学生本人学号2018211x\*y的个位。登录bupt1服务器，在linux环境下使用vi编辑器编写C语言源程序，完成数组a+b的功能，规定数组长度为100，函数名为madd（），数组a，b均定义在函数内，采用gcc编译该程序（不使用优化选项），

1. 使用objdump工具生成汇编程序，找到madd函数的汇编程序，给出截图；
2. 用gdb进行调试，练习如下gdb命令，给出截图；gdb、file、kill、quit、break、delete、clear、info break、run、continue、nexti、stepi、disassemble、list、print、x、info reg、watch
3. 找到a[i]+b[i]对应的汇编指令，指出a[i]和b[i]位于哪个寄存器中，给出截图；
4. 使用单步指令及gdb相关命令，显示a[xy]+b[xy]对应的汇编指令执行前后操作数寄存器十进制和十六进制的值，其中x，y取自于学生本人学号2018211x\*y的百位和个位。

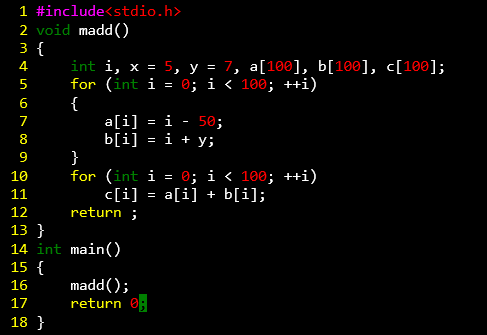
学号2018211999，a[99]+b[99]单步执行前后的参考截图如下（实际命令未显示出）：

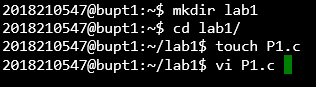


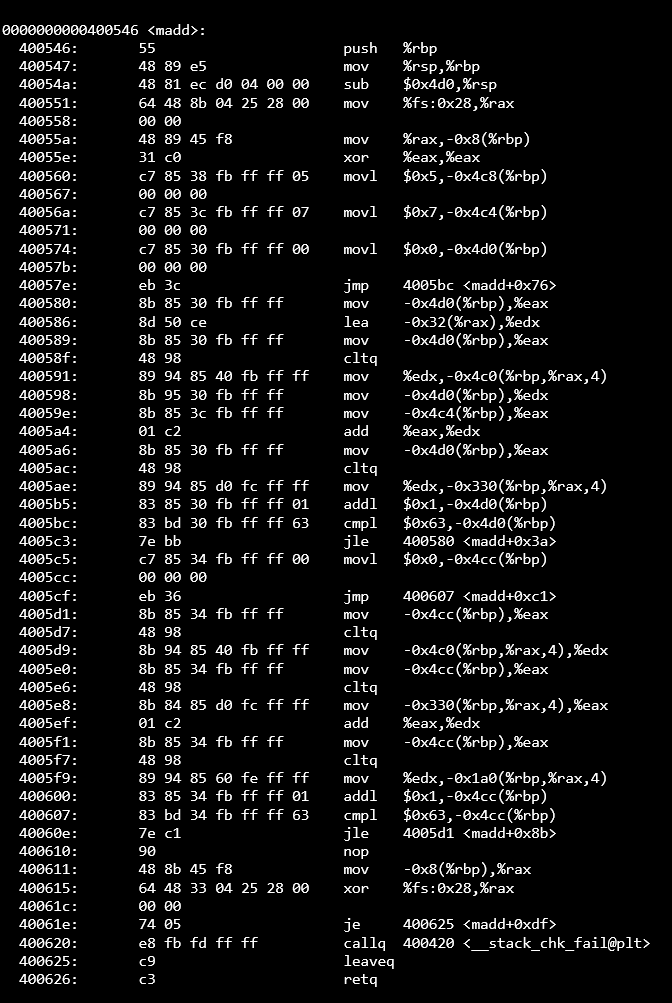
四、实验步骤及实验分析

需要给出操作步骤、运行截图、分析过程的内容

准备阶段：使用mkdir创建lab1目录，cd该目录，使用touch创建P1.c，使用vi编辑该文件，创建madd()函数, 使用gcc 编译该程序.





步骤一：使用objdump工具生成汇编程序，找到madd函数的汇编程序

步骤二：用gdb进行调试，练习课堂中讲授的gdb命令，给出截图。

2018210547@bupt1:~/lab1$ gdb P1

GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1

Copyright (C) 2016 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law. Type "show copying"

and "show warranty" for details.

This GDB was configured as "x86\_64-linux-gnu".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<http://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other documentation resources online at:

<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from P1...done.

(gdb) list

2 void madd()

3 {

4 int i, x = 5, y = 7, a[100], b[100], c[100];

5 for (int i = 0; i < 100; ++i)

6 {

7 a[i] = i - 50;

8 b[i] = i + y;

9 }

10 for (int i = 0; i < 100; ++i)

11 c[i] = a[i] + b[i];

(gdb) break 11

Breakpoint 1 at 0x4005d1: file P1.c, line 11.

(gdb) info break

Num Type Disp Enb Address What

1 breakpoint keep y 0x00000000004005d1 in madd at P1.c:11

(gdb) r

Starting program: /home/students/2018210547/lab1/P1

Breakpoint 1, madd () at P1.c:11

11 c[i] = a[i] + b[i];

(gdb) delete 1

(gdb) clear

No breakpoint at this line.

(gdb) watch i

Hardware watchpoint 2: i

(gdb) continue

Continuing.

Hardware watchpoint 2: i

Old value = 0

New value = 1

0x0000000000400607 in madd () at P1.c:10

10 for (int i = 0; i < 100; ++i)

(gdb) nexti

0x000000000040060e 10 for (int i = 0; i < 100; ++i)

(gdb) nexti

11 c[i] = a[i] + b[i];

(gdb) nexti

0x00000000004005d7 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005d9 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005e0 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005e6 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005e8 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005ef 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005f1 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005f7 11 c[i] = a[i] + b[i];

(gdb) stepi

0x00000000004005f9 11 c[i] = a[i] + b[i];

(gdb) stepi

10 for (int i = 0; i < 100; ++i)

(gdb) print i

$1 = 1

(gdb) print a[i]

$2 = -49

(gdb) print b[i]

$3 = 8

(gdb) print c[i]

$4 = -41

(gdb) info reg

rax 0x1 1

rbx 0x0 0

rcx 0x0 0

rdx 0xffffffd7 4294967255

rsi 0x7fffffffe448 140737488348232

rdi 0x1 1

rbp 0x7fffffffe350 0x7fffffffe350

rsp 0x7fffffffde80 0x7fffffffde80

r8 0x4006b0 4196016

r9 0x7ffff7de7ac0 140737351940800

r10 0x846 2118

r11 0x7ffff7a2d740 140737348032320

r12 0x400450 4195408

r13 0x7fffffffe440 140737488348224

r14 0x0 0

r15 0x0 0

rip 0x400600 0x400600 <madd+186>

eflags 0x296 [ PF AF SF IF ]

cs 0x33 51

ss 0x2b 43

ds 0x0 0

es 0x0 0

fs 0x0 0

gs 0x0 0

(gdb) x/w $rsp

0x7fffffffde80: 100

(gdb) disas

Dump of assembler code for function madd:

0x0000000000400546 <+0>: push %rbp

0x0000000000400547 <+1>: mov %rsp,%rbp

0x000000000040054a <+4>: sub $0x4d0,%rsp

0x0000000000400551 <+11>: mov %fs:0x28,%rax

0x000000000040055a <+20>: mov %rax,-0x8(%rbp)

0x000000000040055e <+24>: xor %eax,%eax

0x0000000000400560 <+26>: movl $0x5,-0x4c8(%rbp)

0x000000000040056a <+36>: movl $0x7,-0x4c4(%rbp)

0x0000000000400574 <+46>: movl $0x0,-0x4d0(%rbp)

0x000000000040057e <+56>: jmp 0x4005bc <madd+118>

0x0000000000400580 <+58>: mov -0x4d0(%rbp),%eax

0x0000000000400586 <+64>: lea -0x32(%rax),%edx

0x0000000000400589 <+67>: mov -0x4d0(%rbp),%eax

0x000000000040058f <+73>: cltq

0x0000000000400591 <+75>: mov %edx,-0x4c0(%rbp,%rax,4)

0x0000000000400598 <+82>: mov -0x4d0(%rbp),%edx

0x000000000040059e <+88>: mov -0x4c4(%rbp),%eax

0x00000000004005a4 <+94>: add %eax,%edx

0x00000000004005a6 <+96>: mov -0x4d0(%rbp),%eax

0x00000000004005ac <+102>: cltq

0x00000000004005ae <+104>: mov %edx,-0x330(%rbp,%rax,4)

0x00000000004005b5 <+111>: addl $0x1,-0x4d0(%rbp)

0x00000000004005bc <+118>: cmpl $0x63,-0x4d0(%rbp)

0x00000000004005c3 <+125>: jle 0x400580 <madd+58>

0x00000000004005c5 <+127>: movl $0x0,-0x4cc(%rbp)

0x00000000004005cf <+137>: jmp 0x400607 <madd+193>

0x00000000004005d1 <+139>: mov -0x4cc(%rbp),%eax

0x00000000004005d7 <+145>: cltq

0x00000000004005d9 <+147>: mov -0x4c0(%rbp,%rax,4),%edx

0x00000000004005e0 <+154>: mov -0x4cc(%rbp),%eax

0x00000000004005e6 <+160>: cltq

0x00000000004005e8 <+162>: mov -0x330(%rbp,%rax,4),%eax

0x00000000004005ef <+169>: add %eax,%edx

0x00000000004005f1 <+171>: mov -0x4cc(%rbp),%eax

0x00000000004005f7 <+177>: cltq

0x00000000004005f9 <+179>: mov %edx,-0x1a0(%rbp,%rax,4)

=> 0x0000000000400600 <+186>: addl $0x1,-0x4cc(%rbp)

0x0000000000400607 <+193>: cmpl $0x63,-0x4cc(%rbp)

0x000000000040060e <+200>: jle 0x4005d1 <madd+139>

0x0000000000400610 <+202>: nop

0x0000000000400611 <+203>: mov -0x8(%rbp),%rax

0x0000000000400615 <+207>: xor %fs:0x28,%rax

---Type <return> to continue, or q <return> to quit---

0x000000000040061e <+216>: je 0x400625 <madd+223>

0x0000000000400620 <+218>: callq 0x400420 <\_\_stack\_chk\_fail@plt>

0x0000000000400625 <+223>: leaveq

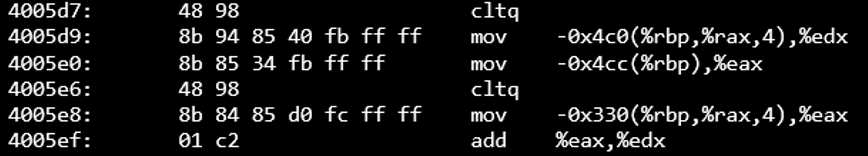
0x0000000000400626 <+224>: retq

End of assembler dump.

(gdb) kill

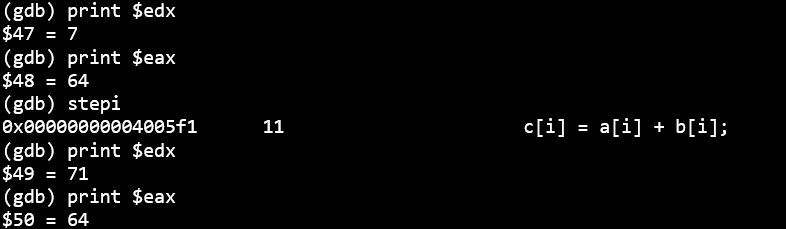
Kill the program being debugged? (y or n) y

(gdb) quit

步骤三：找到a[i]+b[i]对应的汇编指令，指出a[i]和b[i]位于哪个寄存器中，给出截图；

a[i]位于%edx中，b[i]位于%eax中

步骤四：使用单步指令及gdb相关命令，显示a[xy]+b[xy]对应的汇编指令执行前后操作数寄存器十进制和十六进制的值，其中x，y取自于学生本人学号2018211x\*y的百位和个位。

学号2018210547，a[57]+b[57]单步执行前后的截图如下：

五、总结体会

在本次“Linux环境与GCC工具链”实验中，我结合了理论课学习的汇编语言基础知识与实践课学习到的SSH与Linux的操作方法，以及GDB调试的操作技巧，成功地完成了一次对C语言与汇编代码的实践探究。实验过程中，我将理论知识应用于实践，得到了进一步的掌握与巩固。

通过这次实验练习，我对Linux环境下使用命令行编程有了初步的体验。编写代码，调试代码，查找寄存器的过程中遇到了不少问题。一开始，我很不适应没有图形化界面的Linux系统。通过对Linux基础命令的学习以及反复的上机实践后，我才熟悉了Linux系统的操作。在编写代码的过程中，我也不熟悉Vi的操作，常常按错快捷键，导致代码编写混乱。同样，通过熟记Vi的基本命令与多加练习，我最终成功地掌握了Vi的基本使用，并且领略到了它的高效与快捷。而在查找a[i]与b[i]的寄存器实验中，长而冗杂的汇编代码令我头疼。我静下心来，一步一步地分析每条语句地含义，并随时查看相关寄存器的值，以便于理解该语句的作用。最终，我成功地找到了题目所要求的语句。

这次实验我总共花了将近一周的时间，虽然过程是艰辛的，但是结果确是让人回味的。最后，衷心地感谢所有给予我帮助的老师和同学们！