

北京邮电大学

实验报告



题目： Linux 环境和 GCC 工具链

班 级： 2019211309

学 号： 2019211397

姓 名： 毛子恒

学 院： 计算机学院

2020 年 10 月 22 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）。

二、实验环境

- 1、 macOS Catalina 10.15.6 终端：iTerm2 Build 3.3.12
- 2、 bupt1 服务器：Ubuntu 16.04.6 LTS
- 3、 Vim version 7.4.1689
- 4、 gcc version 5.4.0
- 5、 GNU gdb 7.11.1 GNU objdump 2.26.1

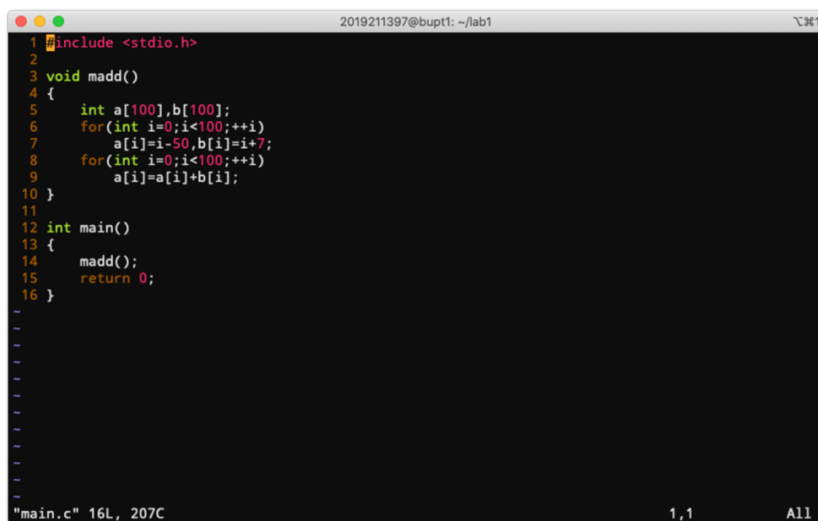
三、实验内容

现有 int 型数组 $a[i]=i-50$, $b[i]=i+7$ 。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（使用 $-g$ 选项，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 $a[37]+b[37]$ 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。

四、实验步骤及实验分析

1. 使用 SSH 连接到 bupt1 服务器，命令为：ssh 2019211397@10.120.11.12
2. 使用 Vi 编辑器编写 main.c，命令为 vi main.c，编写完成后内容如下：
(Vi 编辑器事先配置了行号和代码高亮)



```
1 #include <stdio.h>
2
3 void madd()
4 {
5     int a[100],b[100];
6     for(int i=0;i<100;++i)
7         a[i]=i-50,b[i]=i+7;
8     for(int i=0;i<100;++i)
9         a[i]=a[i]+b[i];
10 }
11
12 int main()
13 {
14     madd();
15     return 0;
16 }
```

3. 保存并退出后使用 gcc 编译该程序，并用 objdump 反汇编，将反汇编结果输出到 main.d 文件中：

```
2019211397@bupt1: ~/lab1
2019211397@bupt1:~/lab1$ vi main.c
2019211397@bupt1:~/lab1$ gcc main.c -o main -g
2019211397@bupt1:~/lab1$ objdump -S main >main.d
2019211397@bupt1:~/lab1$ ls
main main.c main.d
2019211397@bupt1:~/lab1$
```

4. 使用 vi 编辑器打开 main.d 文件，找到 madd 函数的汇编程序：

```
2019211397@bupt1: ~/lab1
125 000000000400546: <madd>:
126 #include <stdio.h>
127
128 void madd()
129 {
130     400546: 55                push    %rbp
131     400547: 48 89 e5          mov     %rsp,%rbp
132     40054a: 48 81 ec 40 03 00 00 sub     $0x340,%rsp
133     400551: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
134     400558: 00 00
135     40055a: 48 89 45 f8       mov     %rax,-0x8(%rbp)
136     40055e: 31 c0            xor     %eax,%eax
137     int a[100],b[100];
138     for(int i=0;i<100;++i)
139     400560: c7 85 c8 fc ff ff 00 movl    $0x0,-0x338(%rbp)
140     400567: 00 00 00
141     40056a: eb 37            jmp     4005a3 <madd+0x5d>
142     a[i]=i-50,b[i]=i+7;
143     40056c: 8b 85 c8 fc ff ff mov     -0x338(%rbp),%eax
144     400572: 8d 50 ce         lea     -0x32(%rax),%edx
145     400575: 8b 85 c8 fc ff ff mov     -0x338(%rbp),%eax
146     40057b: 48 98            cltq
147     40057d: 89 84 e5 d0 fc ff ff mov     %edx,-0x330(%rbp,%rax,4)
148     400584: 8b 85 c8 fc ff ff mov     -0x338(%rbp),%eax
149     40058a: 8d 50 07         lea     0x7(%rax),%edx
150     40058d: 8b 85 c8 fc ff ff mov     -0x338(%rbp),%eax
151     400593: 48 98            cltq
152     400595: 89 84 e5 d0 fc ff ff mov     %edx,-0x330(%rbp,%rax,4)
153 #include <stdio.h>
154
155 void madd()
156 {
157     int a[100],b[100];
158     for(int i=0;i<100;++i)
159     40059c: 83 85 c8 fc ff ff 01 addl    $0x1,-0x338(%rbp)
160     4005a3: 83 bd c8 fc ff ff 63 cmpl    $0x63,-0x338(%rbp)
161     4005aa: 7e c0            jle     40056c <madd+0x26>
162     a[i]=i-50,b[i]=i+7;
163     for(int i=0;i<100;++i)
164     4005ac: c7 85 cc fc ff ff 00 movl    $0x0,-0x334(%rbp)
165     4005b5: 00 00 00
166     4005b8: eb 36            jmp     4005ee <madd+0xa8>
167     a[i]=a[i]+b[i];
168     4005ba: 8b 85 cc fc ff ff mov     -0x334(%rbp),%eax
169     4005be: 48 98            cltq
170     4005c0: 8b 84 e5 d0 fc ff ff mov     -0x330(%rbp,%rax,4),%edx
171     4005c7: 8b 85 cc fc ff ff mov     -0x334(%rbp),%eax
172     4005cd: 48 98            cltq
173     4005cf: 8b 84 e5 d0 fc ff ff mov     -0x330(%rbp,%rax,4),%eax
174     4005d6: 01 c2            add     %eax,%edx
175     4005d8: 8b 85 cc fc ff ff mov     -0x334(%rbp),%eax
176     4005de: 48 98            cltq
177     4005e0: 89 84 e5 d0 fc ff ff mov     %edx,-0x330(%rbp,%rax,4)
178 void madd()
179
178,3 59%
```

5. 打开 gdb，使用 file 指令打开文件，使用 list 指令查看源程序的前 10 行，使用 break 指令在第 6 行和第 8 行循环分别设置断点，使用 run 指令运行程序，程序在第 1 个断点（第 6 行）暂停：

```
2019211397@bupt1:~/lab1$ gdb
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".
(gdb) file main
Reading symbols from main...done.
(gdb) list
1      #include <stdio.h>
2
3      void madd()
4      {
5          int a[100],b[100];
6          for(int i=0;i<100;++i)
7              a[i]=i-50,b[i]=i+7;
8          for(int i=0;i<100;++i)
9              a[i]=a[i]+b[i];
10     }
(gdb) break 6
Breakpoint 1 at 0x400560: file main.c, line 6.
(gdb) break 8
Breakpoint 2 at 0x4005ac: file main.c, line 8.
(gdb) run
Starting program: /home/students/2019211397/lab1/main
Breakpoint 1, madd () at main.c:6
6          for(int i=0;i<100;++i)
(gdb)
```

6. 使用 disassemble 指令反汇编 madd 函数，使用 stepi 命令执行 4 条指令，使用 print 命令打印变量 i 的值，使用 watch 命令监视 i 变量的值，之后通过 stepi 跳转到 0x40059c <+86> addl 指令

(该指令的作用为向内存中 `i` 的位置 `-0x334(%rbp)` 加 1, 即对应 `i++` 语句), 再使用 `nexti` 命令执行 1 条指令, 提示监视的变量 `i` 的值从 0 更新到 1:

```
(gdb) disassemble madd
Dump of assembler code for function madd:
0x0000000000400546 <+0>:    push    %rbp
0x0000000000400547 <+1>:    mov     %rsp,%rbp
0x000000000040054a <+4>:    sub     $0x340,%rsp
0x0000000000400551 <+15>:   mov     %fs:0x28,%rax
0x000000000040055a <+20>:   mov     %rax,-0x8(%rbp)
0x000000000040055e <+24>:   xor     %eax,%eax
=> 0x0000000000400560 <+26>:   movl    $0x0,-0x338(%rbp)
0x000000000040056a <+36>:   jmp     0x4005a3 <madd+93>
0x000000000040056c <+38>:   mov     -0x338(%rbp),%eax
0x0000000000400572 <+44>:   lea     -0x32(%rax),%edx
0x0000000000400575 <+47>:   mov     -0x338(%rbp),%eax
0x000000000040057b <+53>:   cltq
0x000000000040057d <+55>:   mov     %edx,-0x330(%rbp,%rax,4)
0x0000000000400584 <+62>:   mov     -0x338(%rbp),%eax
0x000000000040058a <+68>:   lea     0x7(%rax),%edx
0x000000000040058d <+71>:   mov     -0x338(%rbp),%eax
0x0000000000400593 <+77>:   cltq
0x0000000000400595 <+79>:   mov     %edx,-0x1a0(%rbp,%rax,4)
0x000000000040059c <+86>:   addl    $0x1,-0x338(%rbp)
0x00000000004005a3 <+93>:   cmpl    $0x63,-0x338(%rbp)
0x00000000004005aa <+100>:  jle     0x40056c <madd+38>
0x00000000004005ac <+102>:  movl    $0x0,-0x334(%rbp)
0x00000000004005b6 <+112>:  jmp     0x4005ee <madd+168>
0x00000000004005b9 <+114>:  mov     -0x334(%rbp),%eax
0x00000000004005be <+120>:  cltq
0x00000000004005c0 <+122>:  mov     -0x330(%rbp,%rax,4),%edx
0x00000000004005c7 <+129>:  mov     -0x334(%rbp),%eax
0x00000000004005cd <+135>:  cltq
0x00000000004005cf <+137>:  mov     -0x1a0(%rbp,%rax,4),%eax
0x00000000004005d6 <+144>:  add     %eax,%edx
0x00000000004005db <+146>:  mov     -0x334(%rbp),%eax
0x00000000004005de <+152>:  cltq
0x00000000004005e0 <+154>:  mov     %edx,-0x330(%rbp,%rax,4)
0x00000000004005e7 <+161>:  addl    $0x1,-0x334(%rbp)
0x00000000004005ee <+168>:  cmpl    $0x63,-0x334(%rbp)
0x00000000004005f5 <+175>:  jle     0x4005b8 <madd+114>
0x00000000004005f7 <+177>:  nop
0x00000000004005f8 <+178>:  mov     -0x8(%rbp),%rax
0x00000000004005fc <+182>:  xor     %fs:0x28,%rax
0x0000000000400605 <+191>:  je      0x40060c <madd+198>
0x0000000000400607 <+193>:  callq   0x400420 <__stack_chk_fail@plt>
0x000000000040060c <+198>:  leaveq
0x000000000040060d <+199>:  retq
End of assembler dump.
(gdb) stepi 4
7      a[i]=i-50,b[i]=i+7;
(gdb) print i
$1 = 0
(gdb) watch i
Hardware watchpoint 3: i
(gdb) stepi 10
6      for(int i=0;i<100;++i)
(gdb) nexti
Hardware watchpoint 3: i

Old value = 0
New value = 1
0x00000000004005a3 in madd () at main.c:6
6      for(int i=0;i<100;++i)
(gdb)
```

7. 使用 `info break` 指令显示断点信息, 使用 `delete` 指令删除第 3 个断点 (即 watchpoint `i`), 使用 `continue` 指令跳转到下一个断点, 此时是跳转到第 2 个断点 (第 8 行), 再次查看断点信息, 之后使用 `clear` 指令删除第 8 行的断点, 使用 `delete` 指令删除第 1 个断点, 查看断点信息, 显示没有断点:

```
(gdb) info break
Num   Type             Disp Enb Address            What
1     breakpoint       keep y  0x0000000000400560 in madd at main.c:6
      breakpoint already hit 1 time
2     breakpoint       keep y  0x00000000004005ac in madd at main.c:8
      hw watchpoint keep y
      breakpoint already hit 1 time
(gdb) delete 3
(gdb) continue
Continuing.

Breakpoint 2, madd () at main.c:8
8      for(int i=0;i<100;++i)
(gdb) info break
Num   Type             Disp Enb Address            What
1     breakpoint       keep y  0x0000000000400560 in madd at main.c:6
      breakpoint already hit 1 time
2     breakpoint       keep y  0x00000000004005ac in madd at main.c:8
      breakpoint already hit 1 time
(gdb) list
3      void madd()
4      {
5          int a[100],b[100];
6          for(int i=0;i<100;++i)
7              a[i]=i-50,b[i]=i+7;
8          for(int i=0;i<100;++i)
9              a[i]=a[i]+b[i];
10     }
11
12     int main()
(gdb) clear 8
(gdb) delete 2
Deleted breakpoint 2
(gdb) delete 1
Deleted breakpoint 1
(gdb) info break
No breakpoints or watchpoints.
(gdb)
```

8. 执行 6 条指令, 此时在 `0x4005c0 <+122> mov` 指令 (该指令的作用是从内存中取 `a[i]` 的值并且存入 `%edx` 寄存器), 使用 `watch` 指令分别监视寄存器 `%eax` 和 `%edx` 的值, 执行 1 条指令, 提示 `%edx` 寄存器的值更新为 `a[0]` 的值 50:

```

(gdb) disassemble
Dump of assembler code for function madd:
0x0000000000400546 <+0>:      push    %rbp
0x0000000000400547 <+1>:      mov     %rsp,%rbp
0x000000000040054a <+4>:      sub     $0x340,%rsp
0x0000000000400551 <+11>:     mov     %fs:0x28,%eax
0x000000000040055a <+20>:     mov     %rax,-0x8(%rbp)
0x000000000040055e <+24>:     xor     %eax,%eax
0x0000000000400560 <+26>:     movl    $0x0,-0x338(%rbp)
0x000000000040056a <+36>:     jmp     0x4005a3 <madd+93>
0x000000000040056c <+38>:     mov     -0x338(%rbp),%eax
0x0000000000400572 <+44>:     lea     -0x32(%rax),%edx
0x0000000000400575 <+47>:     mov     -0x338(%rbp),%eax
0x000000000040057b <+53>:     cltq
0x000000000040057d <+55>:     mov     %edx,-0x330(%rbp,%rax,4)
0x0000000000400584 <+62>:     mov     -0x338(%rbp),%eax
0x000000000040058a <+68>:     lea     0x7(%rax),%edx
0x000000000040058d <+71>:     mov     -0x338(%rbp),%eax
0x0000000000400593 <+77>:     cltq
0x0000000000400595 <+79>:     mov     %edx,-0x1a0(%rbp,%rax,4)
0x000000000040059c <+86>:     addl    $0x1,-0x338(%rbp)
0x00000000004005a3 <+93>:     cmpl    $0x63,-0x338(%rbp)
0x00000000004005aa <+100>:    jle     0x40056c <madd+38>
=> 0x00000000004005ac <+102>:    movl    $0x0,-0x334(%rbp)
0x00000000004005b6 <+112>:    jmp     0x4005ee <madd+168>
0x00000000004005b8 <+114>:    mov     -0x334(%rbp),%eax
0x00000000004005be <+120>:    cltq
0x00000000004005c0 <+122>:    mov     -0x330(%rbp,%rax,4),%edx
0x00000000004005c7 <+129>:    mov     -0x334(%rbp),%eax
0x00000000004005cd <+135>:    cltq
0x00000000004005cf <+137>:    mov     -0x1a0(%rbp,%rax,4),%eax
0x00000000004005d6 <+144>:    add     %eax,%edx
0x00000000004005d8 <+146>:    mov     -0x334(%rbp),%eax
0x00000000004005de <+152>:    cltq
0x00000000004005e0 <+154>:    mov     %edx,-0x330(%rbp,%rax,4)
0x00000000004005e7 <+161>:    addl    $0x1,-0x334(%rbp)
0x00000000004005ee <+168>:    cmpl    $0x63,-0x334(%rbp)
0x00000000004005f5 <+175>:    jle     0x4005b8 <madd+114>
0x00000000004005f7 <+177>:    nop
0x00000000004005f8 <+178>:    mov     -0x8(%rbp),%rax
0x00000000004005fc <+182>:    xor     %fs:0x28,%rax
0x0000000000400605 <+191>:    je      0x40060c <madd+198>
0x0000000000400607 <+193>:    callq   0x400420 <__stack_chk_fail@plt>
0x000000000040060c <+198>:    leaveq
0x000000000040060d <+199>:    retq
End of assembler dump.
(gdb) stepi 6
0x00000000004005c0      9      a[i]=a[i]+b[i];
(gdb) watch $edx
Watchpoint 4: $edx
(gdb) watch $eax
Watchpoint 5: $eax
(gdb) stepi
Watchpoint 4: $edx
Old value = 106
New value = -50
0x00000000004005c7 in madd () at main.c:9
9      a[i]=a[i]+b[i];
(gdb)

```

- 使用 `info reg` 指令查看寄存器信息，使用 `print` 指令查看寄存器 `%eax` 和 `%edx` 中存储的值，向后跳转三条指令，提示 `%eax` 的值发生变化（此时发生从内存中取出 `b[0]` 的值 7 存入寄存器 `%eax` 中），再次使用 `print` 指令打印 `%eax` 的值：

```

(gdb) stepi
0x00000000004005cd      9      a[i]=a[i]+b[i];
(gdb) info reg
rax      0x0      0
rbx      0x0      0
rcx      0x0      0
rdx      0xffffffff 4294967246
rsi      0x7fffffff408 140737488348168
rdi      0x1      1
rbp      0x7fffffff310 0x7fffffff310
rsp      0x7fffffffdf0 0x7fffffffdf0
r8       0x4006a0 4196000
r9       0x7ffff7de7ac0 140737351940800
r10      0x846     2118
r11      0x7ffff7a2d740 140737348032320
r12      0x400450 4195408
r13      0x7fffffff400 140737488348160
r14      0x0      0
r15      0x0      0
rip      0x4005cd 0x4005cd <madd+135>
eflags   0x293     [ CF AF SF IF ]
cs       0x33     51
ss       0x2b     43
ds       0x0      0
es       0x0      0
fs       0x0      0
gs       0x0      0
(gdb) print $eax
$2 = 0
(gdb) print $edx
$3 = -50
(gdb) stepi 2
Watchpoint 5: $eax
Old value = 0
New value = 7
0x00000000004005d6 in madd () at main.c:9
9      a[i]=a[i]+b[i];
(gdb) print $eax
$4 = 7
(gdb)

```

- 删除 4、5 两个 watchpoint，使用 `x` 指令查看以十六进制表示的从 `madd` 函数开始的 200 个字节，以及特定的 `0x400586` 地址中的值，打印 `i` 的值，通过查看汇编得到 `i` 在内存中的地址为 `%rbp-0x334`，使用 `print` 打印该地址的值，向后执行数条语句，再次查看这两个值，发现这两个值保持一致；使用 `kill` 指令终止运行，使用 `quit` 指令退出 `gdb`：

```

(gdb) delete 4
(gdb) delete 5
(gdb) x/200xb madd
0x400546 <madd>: 0x55 0x48 0x89 0x45 0x48 0x81 0xec 0xd0
0x40054e <madd+8>: 0x03 0x00 0x00 0x64 0x48 0x8b 0x04 0x25
0x400556 <madd+16>: 0x28 0x00 0x00 0x00 0x48 0x89 0x45 0xf8
0x40055e <madd+24>: 0x31 0xc0 0xc7 0x85 0xc8 0xfc 0xff 0xff
0x400566 <madd+32>: 0x00 0x00 0x00 0x00 0xeb 0x37 0x8b 0x85
0x40056e <madd+40>: 0xc8 0xfc 0xff 0xff 0x8d 0x50 0xce 0x8b
0x400576 <madd+48>: 0x85 0xc8 0xfc 0xff 0xff 0x48 0x98 0x89
0x40057e <madd+56>: 0x94 0x85 0xd0 0xfc 0xff 0xff 0xb5 0x85
0x400586 <madd+64>: 0xc8 0xfc 0xff 0xff 0x8d 0x50 0x07 0x8b
0x40058e <madd+72>: 0x85 0xc8 0xfc 0xff 0xff 0x48 0x98 0x89
0x400596 <madd+80>: 0x94 0x85 0x60 0xfe 0xff 0xff 0x83 0x85
0x40059e <madd+88>: 0xc8 0xfc 0xff 0xff 0x01 0x83 0xbd 0xc8
0x4005a6 <madd+96>: 0xfc 0xff 0xff 0x63 0x7e 0xc0 0xc7 0x85
0x4005ae <madd+104>: 0xcc 0xfc 0xff 0xff 0x00 0x00 0x00 0x00
0x4005b6 <madd+112>: 0xeb 0x36 0xb5 0x85 0xcc 0xfc 0xff 0xff
0x4005be <madd+120>: 0x48 0x98 0x8b 0x94 0x85 0xd0 0xfc 0xff
0x4005c6 <madd+128>: 0xff 0x8b 0x85 0xcc 0xfc 0xff 0xff 0x48
0x4005ce <madd+136>: 0x98 0x8b 0x84 0x85 0x60 0xfe 0xff 0xff
0x4005d6 <madd+144>: 0x01 0xc2 0xb5 0x85 0xcc 0xfc 0xff 0xff
0x4005de <madd+152>: 0x48 0x98 0x89 0x94 0x85 0xd0 0xfc 0xff
0x4005e6 <madd+160>: 0xff 0x83 0x85 0xcc 0xfc 0xff 0xff 0x01
0x4005ee <madd+168>: 0x83 0xbd 0xcc 0xfc 0xff 0xff 0x63 0x7e
0x4005f6 <madd+176>: 0xc1 0x90 0x48 0x8b 0x45 0xf8 0x64 0x48
0x4005fe <madd+184>: 0x33 0x04 0x25 0x28 0x00 0x00 0x00 0x74
0x400606 <madd+192>: 0x05 0xe8 0x14 0xfe 0xff 0xff 0xc9 0xc3
(gdb) x/b 0x400586
0x400586 <madd+64>: 0xc8
(gdb) print i
$4 = 0
(gdb) print *(int *)($rbp-0x334)
$5 = 0
(gdb) step
8      for(int i=0;i<100;++i)
(gdb) step
9      a[i]=a[i]+b[i];
(gdb) step
8      for(int i=0;i<100;++i)
(gdb) step
9      a[i]=a[i]+b[i];
(gdb) step
8      for(int i=0;i<100;++i)
(gdb) step
9      a[i]=a[i]+b[i];
(gdb) print i
$6 = 3
(gdb) print *(int *)($rbp-0x334)
$7 = 3
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb) quit
2019211397@bupt1:~/lab1$

```

11. 使用 gdb disassemble 指令反汇编得到的代码截图如下，红框部分对应 `a[i]+b[i]` 语句，`a[i]` 在寄存器 `%edx` 中，`b[i]` 在寄存器 `%eax` 中：

```

(gdb) disassemble madd
Dump of assembler code for function madd:
0x0000000000400546 <+0>:      push    %rbp
0x0000000000400547 <+1>:      mov     %rsp,%rbp
0x000000000040054a <+4>:      sub     $0x340,%rsp
0x0000000000400551 <+11>:     mov     %fs:0x28,%rax
0x000000000040055a <+20>:     mov     %rax,-0x8(%rbp)
0x000000000040055e <+24>:     xor     %eax,%eax
0x0000000000400560 <+26>:     movl    $0x0,-0x338(%rbp)
0x000000000040056a <+36>:     jmp     0x4005a3 <madd+93>
0x000000000040056c <+38>:     mov     -0x338(%rbp),%eax
0x0000000000400572 <+44>:     lea     -0x32(%rax),%edx
0x0000000000400575 <+47>:     mov     -0x338(%rbp),%eax
0x000000000040057b <+53>:     cltq
0x000000000040057d <+55>:     mov     %edx,-0x330(%rbp,%rax,4)
0x0000000000400584 <+62>:     mov     -0x338(%rbp),%eax
0x000000000040058a <+68>:     lea     0x7(%rax),%edx
0x000000000040058d <+71>:     mov     -0x338(%rbp),%eax
0x0000000000400593 <+77>:     cltq
0x0000000000400595 <+79>:     mov     %edx,-0x1a0(%rbp,%rax,4)
0x000000000040059c <+86>:     addl    $0x1,-0x338(%rbp)
0x00000000004005a3 <+93>:     cmpl    $0x63,-0x338(%rbp)
0x00000000004005aa <+100>:    jle     0x40056c <madd+38>
0x00000000004005ac <+102>:    movl    $0x0,-0x334(%rbp)
0x00000000004005b6 <+112>:    jmp     0x4005ee <madd+168>
0x00000000004005b8 <+114>:    mov     -0x334(%rbp),%eax
0x00000000004005be <+120>:    cltq
0x00000000004005c0 <+122>:    mov     -0x330(%rbp,%rax,4),%edx
0x00000000004005c7 <+129>:    mov     -0x334(%rbp),%eax
0x00000000004005cd <+135>:    cltq
0x00000000004005cf <+137>:    mov     -0x1a0(%rbp,%rax,4),%eax
0x00000000004005d6 <+144>:    add     %eax,%edx
0x00000000004005d8 <+146>:    mov     -0x334(%rbp),%eax
0x00000000004005de <+152>:    cltq
0x00000000004005e0 <+154>:    mov     %edx,-0x330(%rbp,%rax,4)
0x00000000004005e7 <+161>:    addl    $0x1,-0x334(%rbp)
0x00000000004005ee <+168>:    cmpl    $0x63,-0x334(%rbp)
0x00000000004005f5 <+175>:    jle     0x4005b8 <madd+114>
0x00000000004005f7 <+177>:    nop
0x00000000004005f8 <+178>:    mov     -0x8(%rbp),%rax
0x00000000004005fc <+182>:    xor     %fs:0x28,%rax
0x0000000000400605 <+191>:    je      0x40060c <madd+198>
0x0000000000400607 <+193>:    callq   0x400420 <__stack_chk_fail@plt>
0x000000000040060c <+198>:    leaveq
0x000000000040060d <+199>:    retq
End of assembler dump.
(gdb)

```

其前后的语句分析解释如下：

`i` 在内存中的地址为 `-0x334(%rbp)`


```

<+102> movl 即 i=0
<+112> jmp 跳转到<+168>
<+168> cmpl 即 i<100
<+175> jle 如果满足条件, 跳转到<+114>
<+114> mov <+120> cltq 把 i 的值赋给%eax, 并且位扩展
<+122> mov 从内存中取出 a[i], 赋给%edx
<+129> mov <+135> cltq 把 i 的值赋给%eax, 并且位扩展
<+137> mov 从内存中取出 b[i], 赋给%eax
<+144> add %edx=%edx+%eax, 即计算 a[i]+b[i]
<+146> mov <+152> cltq 把 i 的值赋给%eax, 并且位扩展
<+154> mov 把%edx 的值赋给内存中 a[i]的位置
<+161> addl i=i+1

```

12. 使用 gdb 调试 main, 设置第 8 行断点, 运行, 在第 9 行设置 $i==37$ 的条件断点, 跳转, 打印 i 的值确认执行到 $i==37$ 的循环的 $a[i]=a[i]+b[i]$ 语句:

```

2019211397@bupt:~/lab1$ gdb main
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 main...done.
(gdb) b 8
Breakpoint 1 at 0x4005ac: file main.c, line 8.
(gdb) r
Starting program: /home/students/2019211397/lab1/main

Breakpoint 1, madd () at main.c:8
8   for(int i=0;i<100;++i)
(gdb) break 9 if i==37
Breakpoint 2 at 0x4005b8: file main.c, line 9.
(gdb) c
Continuing.

Breakpoint 2, madd () at main.c:9
9   a[i]=a[i]+b[i];
(gdb) print i
$1 = 37
(gdb)

```

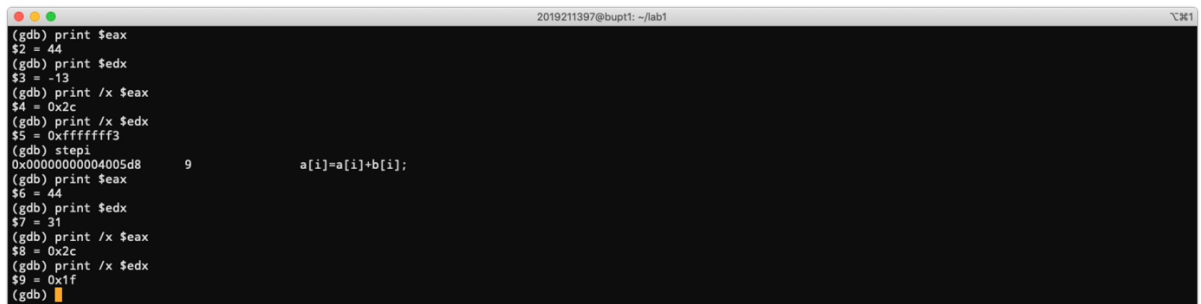
13. 反汇编, 跳转到 0x4005d6 <+144> add %eax,%edx 指令:

```

(gdb) disassemble
Dump of assembler code for function madd:
0x0000000000400546 <+0>: push %rbp
0x0000000000400547 <+1>: mov %rsp,%rbp
0x000000000040054a <+4>: sub $0x340,%rsp
0x0000000000400551 <+11>: mov %fs:0x28,%rax
0x000000000040055a <+20>: mov %rax,-0x8(%rbp)
0x000000000040055e <+24>: xor %eax,%eax
0x0000000000400560 <+26>: movl $0x0,-0x338(%rbp)
0x000000000040056a <+36>: jmp 0x4005a3 <madd+93>
0x000000000040056c <+38>: mov -0x338(%rbp),%eax
0x0000000000400572 <+44>: lea -0x32(%rax),%edx
0x0000000000400575 <+47>: mov -0x338(%rbp),%eax
0x000000000040057b <+53>: cltq
0x000000000040057d <+55>: mov %edx,-0x330(%rbp,%rax,4)
0x0000000000400584 <+62>: mov -0x338(%rbp),%eax
0x000000000040058a <+68>: lea 0x7(%rax),%edx
0x000000000040058d <+71>: mov -0x338(%rbp),%eax
0x0000000000400593 <+77>: cltq
0x0000000000400595 <+79>: mov %edx,-0x1a0(%rbp,%rax,4)
0x000000000040059c <+86>: addl $0x1,-0x338(%rbp)
0x00000000004005a3 <+93>: cmpl $0x63,-0x338(%rbp)
0x00000000004005aa <+100>: jle 0x40056c <madd+38>
0x00000000004005ac <+102>: movl $0x0,-0x334(%rbp)
0x00000000004005b6 <+112>: jmp 0x4005ee <madd+168>
=> 0x00000000004005b8 <+114>: mov -0x334(%rbp),%eax
0x00000000004005be <+120>: cltq
0x00000000004005c0 <+122>: mov -0x330(%rbp,%rax,4),%edx
0x00000000004005c7 <+129>: mov -0x334(%rbp),%eax
0x00000000004005cd <+135>: cltq
0x00000000004005cf <+137>: mov -0x1a0(%rbp,%rax,4),%eax
0x00000000004005d6 <+144>: add %eax,%edx
0x00000000004005d8 <+146>: mov -0x334(%rbp),%eax
0x00000000004005de <+152>: cltq
0x00000000004005e0 <+154>: mov %edx,-0x330(%rbp,%rax,4)
0x00000000004005e7 <+161>: addl $0x1,-0x334(%rbp)
0x00000000004005ee <+168>: cmpl $0x63,-0x334(%rbp)
0x00000000004005f5 <+175>: jle 0x4005b8 <madd+114>
0x00000000004005f7 <+177>: nop
0x00000000004005f8 <+178>: mov -0x8(%rbp),%rax
0x00000000004005fc <+182>: xor %fs:0x28,%rax
0x0000000000400605 <+191>: je 0x40060c <madd+198>
0x0000000000400607 <+193>: callq 0x400420 <__stack_chk_fail@plt>
0x000000000040060c <+198>: leaveq
0x000000000040060d <+199>: retq
End of assembler dump.
(gdb) stepi 6
0x00000000004005d6 9 a[i]=a[i]+b[i];
(gdb)

```

14. 分别以十进制和十六进制打印%eax 和%edx 寄存器的值，执行该指令前%eax 中存储 b[37]的值 44，%edx 中存储 a[37]的值-13，执行一条指令，再打印相同的内容，发现执行该指令后%eax 中存储 b[37]的值 44，%edx 中存储 a[37]+b[37]的值 31：



```
(gdb) print $eax
$2 = 44
(gdb) print $edx
$3 = -13
(gdb) print /x $eax
$4 = 0x2c
(gdb) print /x $edx
$5 = 0xffffffff3
(gdb) stepi
0x00000000004005d8    9      a[i]=a[i]+b[i];
(gdb) print $eax
$6 = 44
(gdb) print $edx
$7 = 31
(gdb) print /x $eax
$8 = 0x2c
(gdb) print /x $edx
$9 = 0x1f
(gdb)
```

五、总结体会

本次实验使我对 Vi 编辑器、gcc、gdb、objdump 等工具的使用方法有了初步的认识，实验中我编写了一个简单的程序，编译并反汇编后将其汇编码与源码比对，使我阅读汇编码的能力有了进一步提高，并且对 for 循环的汇编表示有了更深的印象。由于我之前有过使用 Linux、Vim、gcc 的经验，在这些方面没有遇到太大的困难。在使用 gdb 工具时，我仔细阅读使用文档和示例，认真理解对各个指令的使用方法，相信能在之后熟练的利用该工具进行调试。在使用 objdump 工具时，加上-S 选项可以使阅读汇编码的难度大大降低。

意见和建议：实验课上讲授 Linux 基本命令、Vi 编辑器的使用时有些过于泛泛，难以使同学们抓住重点，在实操时可能会遇到非常大的困难，可以在上课时只讲授重点的操作方法，课下给出更多的参考资料以供研究。另外对于 gdb 和 objdump 这两个重要工具的使用，介绍有些简略，课下需要大量阅读参考资料以掌握使用技巧。