北京郵電大學

实验报告

题目：Linux 环境和 GCC 工具链

班　　级：2021211301

学　　号：2021210967

姓　　名：王心雨

学　　院：计算机学院

2022 年 10 月 10 日

一、实验目的

1、 熟悉 linux 系统的常用命令；
2、 掌握 gcc 编译器的使用方法；
3、 掌握 gdb 的调试工具使用；
4、 掌握 objdump 反汇编工具使用；
5、 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

列举所使用的软件工具

1、 HUWAWEI Windows 11，终端：LAPTOP-V1T802I1；
2、 Visual Interface；
3、 GCC 编译器；
4、 GCD 编译器；
5、 Objdump 反汇编工具。

三、实验内容

现有两个 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 和-fno-stack-protector 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，练习下列 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[97]+b[97]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. 打开 Windows PowerShell 窗口，输入指令 ssh2021210967@10.120.11.12 登录 linux 环境；
2. 配置好 linux 环境后，使用 vi 编辑器写文件 main.c 源程序，命令输入为 vi main.c，完成数组 a+b 的功能，数组长度为 100，函数名称为 madd()，其中 a[i]=i-50,b[i]=i+7，最终 main.c 代码如下：

```c
#include <stdio.h>

void madd()
{
        int a[100],b[100];
        for(int i=0;i<100;i++){
                a[i]=i-50;
                b[i]=i+7;
        }
        for(int i=0;i<100;i++)
                a[i]+=b[i];
}

int main()
{
        madd();
        return 0;
}
```

3. 使用:wq 指令退出 vi 编辑器下的 main.c 文件，利用 gcc 编译该程序，命令输入为 gcc -g -fno-

stack-protector main.c -o main；接着用 **obdjump 工具**将 main.c 文件反汇编，则我们能找到 madd 函数的汇编程序：

```
2021210967@bupt1:~$ vi main.c
2021210967@bupt1:~$ 2021210967@bupt1:~$ gcc -g -fno-stack-protector main.c -o main
2021210967@bupt1:~$ objdump -d main
```

可以得到 madd 函数的汇编程序如下：

```
000000000000114a <madd>:
    114a:       55                      push   %rbp
    114b:       48 89 e5                mov    %rsp,%rbp
    114e:       48 81 ec b8 02 00 00    sub    $0x2b8,%rsp
    1155:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
    115c:       eb 28                   jmp    1186 <madd+0x3c>
    115e:       8b 45 fc                mov    -0x4(%rbp),%eax
    1161:       8d 50 ce                lea    -0x32(%rax),%edx
    1164:       8b 45 fc                mov    -0x4(%rbp),%eax
    1167:       48 98                   cltq
    1169:       89 94 85 60 fe ff ff    mov    %edx,-0x1a0(%rbp,%rax,4)
    1170:       8b 45 fc                mov    -0x4(%rbp),%eax
    1173:       8d 50 07                lea    0x7(%rax),%edx
    1176:       8b 45 fc                mov    -0x4(%rbp),%eax
    1179:       48 98                   cltq
    117b:       89 94 85 d0 fc ff ff    mov    %edx,-0x330(%rbp,%rax,4)
    1182:       83 45 fc 01             addl   $0x1,-0x4(%rbp)
    1186:       83 7d fc 63             cmpl   $0x63,-0x4(%rbp)
    118a:       7e d2                   jle    115e <madd+0x14>
    118c:       c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
    1193:       eb 2a                   jmp    11bf <madd+0x75>
    1195:       8b 45 f8                mov    -0x8(%rbp),%eax
    1198:       48 98                   cltq
    119a:       8b 94 85 60 fe ff ff    mov    -0x1a0(%rbp,%rax,4),%edx
    11a1:       8b 45 f8                mov    -0x8(%rbp),%eax
    11a4:       48 98                   cltq
    11a6:       8b 84 85 d0 fc ff ff    mov    -0x330(%rbp,%rax,4),%eax
    11ad:       01 c2                   add    %eax,%edx
    11af:       8b 45 f8                mov    -0x8(%rbp),%eax
    11b2:       48 98                   cltq
    11b4:       89 94 85 60 fe ff ff    mov    %edx,-0x1a0(%rbp,%rax,4)
    11bb:       83 45 f8 01             addl   $0x1,-0x8(%rbp)
    11bf:       83 7d f8 63             cmpl   $0x63,-0x8(%rbp)
    11c3:       7e d0                   jle    1195 <madd+0x4b>
    11c5:       90                      nop
    11c6:       c9                      leaveq
    11c7:       c3                      retq
```

4. 接下来进行第二个任务，输入命令**“gdb”【打开 gdb 并用 gdb 进行调试】**，用 **file 指令【打开指定文件】**打开 main.c 文件，接着用 **list 指令【查看源程序】**查看 main.c 文件的源程序，这时候我们会看到程序每一行会标注行标，接下来使用 **break 指令【打断点调试】**在第 6 行和第 8 行循环分别设置断点，再使用 **run 指令【运行程序】**，程序在第 6 行（第一个断点）暂停[语句“breakpoint 1,madd()at main.c 6”]：

```
2021210967@bupt1:~$ gdb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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...
(gdb) list
2
3       void madd()
4       {
5               int a[100],b[100];
6               for(int i=0;i<100;i++){
7                       a[i]=i-50;
8                       b[i]=i+7;
9               }
10              for(int i=0;i<100;i++)
11                      a[i]+=b[i];
(gdb) b 6
Breakpoint 1 at 0x1155: file main.c, line 6.
(gdb) b 10
Breakpoint 2 at 0x118c: file main.c, line 10.
(gdb) run
Starting program: /students/2021210967/main

Breakpoint 1, madd () at main.c:6
6               for(int i=0;i<100;i++){
(gdb)
```

5. 使用 **disas/disassemble【求反汇编函数】**指令求 madd 的反汇编程序，如下截图显示：

```
(gdb) disas
Dump of assembler code for function madd:
   0x000055555555514a <+0>:     push   %rbp
   0x000055555555514b <+1>:     mov    %rsp,%rbp
   0x000055555555514e <+4>:     sub    $0x2b8,%rsp
=> 0x0000555555555155 <+11>:    movl   $0x0,-0x4(%rbp)
   0x000055555555515c <+18>:    jmp    0x555555555186 <madd+60>
   0x000055555555515e <+20>:    mov    -0x4(%rbp),%eax
   0x0000555555555161 <+23>:    lea    -0x32(%rax),%edx
   0x0000555555555164 <+26>:    mov    -0x4(%rbp),%eax
   0x0000555555555167 <+29>:    cltq
   0x0000555555555169 <+31>:    mov    %edx,-0x1a0(%rbp,%rax,4)
   0x0000555555555170 <+38>:    mov    -0x4(%rbp),%eax
   0x0000555555555173 <+41>:    lea    0x7(%rax),%edx
   0x0000555555555176 <+44>:    mov    -0x4(%rbp),%eax
   0x0000555555555179 <+47>:    cltq
   0x000055555555517b <+49>:    mov    %edx,-0x330(%rbp,%rax,4)
   0x0000555555555182 <+56>:    addl   $0x1,-0x4(%rbp)
   0x0000555555555186 <+60>:    cmpl   $0x63,-0x4(%rbp)
   0x000055555555518a <+64>:    jle    0x555555555515e <madd+20>
   0x000055555555518c <+66>:    movl   $0x0,-0x8(%rbp)
   0x0000555555555193 <+73>:    jmp    0x5555555551bf <madd+117>
   0x0000555555555195 <+75>:    mov    -0x8(%rbp),%eax
   0x0000555555555198 <+78>:    cltq
   0x000055555555519a <+80>:    mov    -0x1a0(%rbp,%rax,4),%edx
   0x00005555555551a1 <+87>:    mov    -0x8(%rbp),%eax
   0x00005555555551a4 <+90>:    cltq
   0x00005555555551a6 <+92>:    mov    -0x330(%rbp,%rax,4),%eax
   0x00005555555551ad <+99>:    add    %eax,%edx
   0x00005555555551af <+101>:   mov    -0x8(%rbp),%eax
   0x00005555555551b2 <+104>:   cltq
   0x00005555555551b4 <+106>:   mov    %edx,-0x1a0(%rbp,%rax,4)
   0x00005555555551bb <+113>:   addl   $0x1,-0x8(%rbp)
   0x00005555555551bf <+117>:   cmpl   $0x63,-0x8(%rbp)
   0x00005555555551c3 <+121>:   jle    0x555555555195 <madd+75>
   0x00005555555551c5 <+123>:   nop
   0x00005555555551c6 <+124>:   leaveq
   0x00005555555551c7 <+125>:   retq
End of assembler dump.
(gdb)
```

6. ①使用 **stepi 命令【执行指令】**执行 4 条指令；②使用 **print【显示指定寄存器内容】**打印变量 i 的值，再使用 **watch 命令【监视 i 变量的值】**，然后通过 stepi 跳转到<+86> addl 指令（该指令的作用为向内存中 i 的位置%rbp 加 1，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。由此我们可知，nexti 与 stepi 命令在执行指令时功能相同，不同的是 nexti 可以用于函数调用：

```
(gdb) stepi 4
7                       a[i]=i-50;
(gdb) stepi
0x0000555555555161      7                       a[i]=i-50;
(gdb) nexti
0x0000555555555164      7                       a[i]=i-50;
(gdb) print i
$1 = 0
(gdb) watch i
Hardware watchpoint 3: i
(gdb) stepi 10

Hardware watchpoint 3: i

Old value = 0
New value = 1
0x0000555555555186 in madd () at main.c:6
6               for(int i=0;i<100;i++){
(gdb) nexti
0x000055555555518a      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【删除断点】**删除第 3 个断点（即 watchpoint i），使用 **continue 指令【从当前断点开始执行到下一个断点】**，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 10 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。可得到，clear 与 delete 对断点的操作，效果相同，但是 clear 后接断点行数，而 delete 接断点的个数：

```
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555155 in madd at main.c:6
        breakpoint already hit 1 time
2       breakpoint     keep y   0x000055555555518c in madd at main.c:10
3       hw watchpoint  keep y                      i
        breakpoint already hit 1 time
(gdb) delete 3
(gdb) continue
Continuing.

Breakpoint 2, madd () at main.c:10
10              for(int i=0;i<100;i++)
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555155 in madd at main.c:6
        breakpoint already hit 1 time
2       breakpoint     keep y   0x000055555555518c in madd at main.c:10
        breakpoint already hit 1 time
(gdb) list
5               int a[100],b[100];
6               for(int i=0;i<100;i++){
7                       a[i]=i-50;
8                       b[i]=i+7;
9               }
10              for(int i=0;i<100;i++)
11                      a[i]+=b[i];
12      }
13
14      int main()
(gdb)
```

```
(gdb) clear 10

(gdb) delete 1
(gdb) info break
Deleted breakpoint 2 No breakpoints or watchpoints.
(gdb)
```

8. 使用 **stepi** 执行 6 条指令，使用 **watch** 指令分别监视寄存器%eax 和%edx 的值，执行 1 条指令，我们可以看到%edx 寄存器的值从值为 106 更新为-50。通过读汇编语言，我们深入了解了这条指令在计算机组件内实现的过程以及结果的展示：

```
(gdb) disas
Dump of assembler code for function madd:
   0x000055555555514a <+0>:     push   %rbp
   0x000055555555514b <+1>:     mov    %rsp,%rbp
   0x000055555555514e <+4>:     sub    $0x2b8,%rsp
   0x0000555555555155 <+11>:    movl   $0x0,-0x4(%rbp)
   0x000055555555515c <+18>:    jmp    0x555555555186 <madd+60>
   0x000055555555515e <+20>:    mov    -0x4(%rbp),%eax
   0x0000555555555161 <+23>:    lea    -0x32(%rax),%edx
   0x0000555555555164 <+26>:    mov    -0x4(%rbp),%eax
   0x0000555555555167 <+29>:    cltq
   0x0000555555555169 <+31>:    mov    %edx,-0x1a0(%rbp,%rax,4)
   0x0000555555555170 <+38>:    mov    -0x4(%rbp),%eax
   0x0000555555555173 <+41>:    lea    0x7(%rax),%edx
   0x0000555555555176 <+44>:    mov    -0x4(%rbp),%eax
   0x0000555555555179 <+47>:    cltq
   0x000055555555517b <+49>:    mov    %edx,-0x330(%rbp,%rax,4)
   0x0000555555555182 <+56>:    addl   $0x1,-0x4(%rbp)
   0x0000555555555186 <+60>:    cmpl   $0x63,-0x4(%rbp)
   0x000055555555518a <+64>:    jle    0x55555555515e <madd+20>
=> 0x000055555555518c <+66>:    movl   $0x0,-0x8(%rbp)
   0x0000555555555193 <+73>:    jmp    0x5555555551bf <madd+117>
   0x0000555555555195 <+75>:    mov    -0x8(%rbp),%eax
   0x0000555555555198 <+78>:    cltq
   0x000055555555519a <+80>:    mov    -0x1a0(%rbp,%rax,4),%edx
   0x00005555555551a1 <+87>:    mov    -0x8(%rbp),%eax
   0x00005555555551a4 <+90>:    cltq
   0x00005555555551a6 <+92>:    mov    -0x330(%rbp,%rax,4),%eax
   0x00005555555551ad <+99>:    add    %eax,%edx
   0x00005555555551af <+101>:   mov    -0x8(%rbp),%eax
   0x00005555555551b2 <+104>:   cltq
   0x00005555555551b4 <+106>:   mov    %edx,-0x1a0(%rbp,%rax,4)
   0x00005555555551bb <+113>:   addl   $0x1,-0x8(%rbp)
   0x00005555555551bf <+117>:   cmpl   $0x63,-0x8(%rbp)
   0x00005555555551c3 <+121>:   jle    0x555555555195 <madd+75>
   0x00005555555551c5 <+123>:   nop
   0x00005555555551c6 <+124>:   leaveq
   0x00005555555551c7 <+125>:   retq
```

```
(gdb) stepi 6
0x000055555555519a      11                      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
0x00005555555551a1 in madd () at main.c:11
11                      a[i]+=b[i];
(gdb)
```

9. 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**，向后跳转两条、一条指令，并再次打印%eax 的值，可看出随着指令顺序发生，%eax 中的值发生变化：

```
(gdb) info reg
rax            0x0                 0
rbx            0x5555555551e0      93824992236000
rcx            0x5555555551e0      93824992236000
rdx            0xfffffffce         4294967246
rsi            0x7fffffffebf8      140737488350200
rdi            0x1                 1
rbp            0x7fffffffeaf0      0x7fffffffeaf0
rsp            0x7fffffffe838      0x7fffffffe838
r8             0x0                 0
r9             0x7ffff7fe0d60      140737354009952
r10            0xf                 15
r11            0x2                 2
r12            0x555555555040      93824992235584
r13            0x7fffffffebf0      140737488350192
r14            0x0                 0
r15            0x0                 0
rip            0x5555555551a1      0x5555555551a1 <madd+87>
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)
```

```
(gdb) print $eax
$2 = 0
(gdb) print $edx
$3 = -50
(gdb) stepi 2
0x00005555555551a6      11                      a[i]+=b[i];
(gdb) print $eax
$4 = 0
(gdb) stepi

Watchpoint 5: $eax

Old value = 0
New value = 7
0x00005555555551ad in madd () at main.c:11
11                      a[i]+=b[i];
(gdb) print $eax
$5 = 7
(gdb)
```

10. 使用 delete 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 150 个字节，以及特定的 0x55 地址中的值，再次 print 打印 i；使用 **kill 指令【终止程序】**，使用 **quit 指令【退出 gdb 调试】**：

```
(gdb) delete 4
(gdb) delete 5
(gdb) x madd
0x55555555514a <madd>:  0xe5894855
(gdb) x/150xb madd
0x55555555514a <madd>:  0x55    0x48    0x89    0xe5    0x48    0x81    0xec    0xb8
0x555555555152 <madd+8>:        0x02    0x00    0x00    0xc7    0x45    0xfc    0x00    0x00
0x55555555515a <madd+16>:       0x00    0x00    0xeb    0x28    0x8b    0x45    0xfc    0x8d
0x555555555162 <madd+24>:       0x50    0xce    0x8b    0x45    0xfc    0x48    0x98    0x89
0x55555555516a <madd+32>:       0x94    0x85    0x60    0xfe    0xff    0xff    0x8b    0x45
0x555555555172 <madd+40>:       0xfc    0x8d    0x50    0x07    0x8b    0x45    0xfc    0x48
0x55555555517a <madd+48>:       0x98    0x89    0x94    0x85    0xd0    0xfc    0xff    0xff
0x555555555182 <madd+56>:       0x83    0x45    0xfc    0x01    0x83    0x7d    0xfc    0x63
0x55555555518a <madd+64>:       0x7e    0xd2    0xc7    0x45    0xf8    0x00    0x00    0x00
0x555555555192 <madd+72>:       0x00    0xeb    0x2a    0x8b    0x45    0xf8    0x48    0x98
0x55555555519a <madd+80>:       0x8b    0x94    0x85    0x60    0xfe    0xff    0xff    0x8b
0x5555555551a2 <madd+88>:       0x45    0xf8    0x48    0x98    0x8b    0x84    0x85    0xd0
0x5555555551aa <madd+96>:       0xfc    0xff    0xff    0x01    0xc2    0x8b    0x45    0xf8
0x5555555551b2 <madd+104>:      0x48    0x98    0x89    0x94    0x85    0x60    0xfe    0xff
0x5555555551ba <madd+112>:      0xff    0x83    0x45    0xf8    0x01    0x83    0x7d    0xf8
0x5555555551c2 <madd+120>:      0x63    0x7e    0xd0    0x90    0xc9    0xc3    0x55    0x48
0x5555555551ca <main+2>:        0x89    0xe5    0xb8    0x00    0x00    0x00    0x00    0xe8
0x5555555551d2 <main+10>:       0x74    0xff    0xff    0xff    0xb8    0x00    0x00    0x00
0x5555555551da <main+18>:       0x00    0x5d    0xc3    0x0f    0x1f    0x00
```

```
(gdb) x/xb madd
0x55555555514a <madd>:  0x55
(gdb)
```

```
(gdb) print i
$6 = 0
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 2474218) killed]
(gdb) quit
```

11. 再次打开 gdb 调试器，使用 **disas 指令**反汇编得到的代码截图如下，得知汇编指令【<+99>:add %eax，%edx】代表目标 **a[i]+=b[i]语句**，a[i]在寄存器%edx 中，b[i]在寄存器%eax 中：

```
(gdb) disas madd
Dump of assembler code for function madd:
   0x000000000000114a <+0>:     push   %rbp
   0x000000000000114b <+1>:     mov    %rsp,%rbp
   0x000000000000114e <+4>:     sub    $0x2b8,%rsp
   0x0000000000001155 <+11>:    movl   $0x0,-0x4(%rbp)
   0x000000000000115c <+18>:    jmp    0x1186 <madd+60>
   0x000000000000115e <+20>:    mov    -0x4(%rbp),%eax
   0x0000000000001161 <+23>:    lea    -0x32(%rax),%edx
   0x0000000000001164 <+26>:    mov    -0x4(%rbp),%eax
   0x0000000000001167 <+29>:    cltq
   0x0000000000001169 <+31>:    mov    %edx,-0x1a0(%rbp,%rax,4)
   0x0000000000001170 <+38>:    mov    -0x4(%rbp),%eax
   0x0000000000001173 <+41>:    lea    0x7(%rax),%edx
   0x0000000000001176 <+44>:    mov    -0x4(%rbp),%eax
   0x0000000000001179 <+47>:    cltq
   0x000000000000117b <+49>:    mov    %edx,-0x330(%rbp,%rax,4)
   0x0000000000001182 <+56>:    addl   $0x1,-0x4(%rbp)
   0x0000000000001186 <+60>:    cmpl   $0x63,-0x4(%rbp)
   0x000000000000118a <+64>:    jle    0x115e <madd+20>
   0x000000000000118c <+66>:    movl   $0x0,-0x8(%rbp)
   0x0000000000001193 <+73>:    jmp    0x11bf <madd+117>
   0x0000000000001195 <+75>:    mov    -0x8(%rbp),%eax
   0x0000000000001198 <+78>:    cltq
   0x000000000000119a <+80>:    mov    -0x1a0(%rbp,%rax,4),%edx
   0x00000000000011a1 <+87>:    mov    -0x8(%rbp),%eax
   0x00000000000011a4 <+90>:    cltq
   0x00000000000011a6 <+92>:    mov    -0x330(%rbp,%rax,4),%eax
   0x00000000000011ad <+99>:    add    %eax,%edx
   0x00000000000011af <+101>:   mov    -0x8(%rbp),%eax
   0x00000000000011b2 <+104>:   cltq
   0x00000000000011b4 <+106>:   mov    %edx,-0x1a0(%rbp,%rax,4)
   0x00000000000011bb <+113>:   addl   $0x1,-0x8(%rbp)
   0x00000000000011bf <+117>:   cmpl   $0x63,-0x8(%rbp)
   0x00000000000011c3 <+121>:   jle    0x1195 <madd+75>
   0x00000000000011c5 <+123>:   nop
   0x00000000000011c6 <+124>:   leaveq
   0x00000000000011c7 <+125>:   retq
End of assembler dump.
```

12. 由于刚刚使用 clear 10 指令，我们再次设置第 10 行断点并输入 **run 指令【运行程序】**，在第 **11 行设置 i==97 的条件断点**；再输入 c 指令继续执行，确认执行到 i==97 的循环的 a[i]+=b[i]语句时，**打印 i 的值“$1=97”**：

```
(gdb) b 10
Breakpoint 1 at 0x118c: file main.c, line 10.
(gdb) r
Starting program: /students/2021210967/main

Breakpoint 1, madd () at main.c:10
10              for(int i=0;i<100;i++)
(gdb) break 11 if i==97
Breakpoint 2 at 0x555555555195: file main.c, line 11.
(gdb) c
Continuing.

Breakpoint 2, madd () at main.c:11
11                      a[i]+=b[i];
(gdb) print i
$1 = 97
(gdb)
```

13. 反汇编，利用 **stepi 指令**经过 6 次再次跳转到<+99> add %eax,%edx 指令：

```
(gdb) disas
Dump of assembler code for function madd:
   0x000055555555514a <+0>:     push   %rbp
   0x000055555555514b <+1>:     mov    %rsp,%rbp
   0x000055555555514e <+4>:     sub    $0x2b8,%rsp
   0x0000555555555155 <+11>:    movl   $0x0,-0x4(%rbp)
   0x000055555555515c <+18>:    jmp    0x555555555186 <madd+60>
   0x000055555555515e <+20>:    mov    -0x4(%rbp),%eax
   0x0000555555555161 <+23>:    lea    -0x32(%rax),%edx
   0x0000555555555164 <+26>:    mov    -0x4(%rbp),%eax
   0x0000555555555167 <+29>:    cltq
   0x0000555555555169 <+31>:    mov    %edx,-0x1a0(%rbp,%rax,4)
   0x0000555555555170 <+38>:    mov    -0x4(%rbp),%eax
   0x0000555555555173 <+41>:    lea    0x7(%rax),%edx
   0x0000555555555176 <+44>:    mov    -0x4(%rbp),%eax
   0x0000555555555179 <+47>:    cltq
   0x000055555555517b <+49>:    mov    %edx,-0x330(%rbp,%rax,4)
   0x0000555555555182 <+56>:    addl   $0x1,-0x4(%rbp)
   0x0000555555555186 <+60>:    cmpl   $0x63,-0x4(%rbp)
   0x000055555555518a <+64>:    jle    0x55555555515e <madd+20>
   0x000055555555518c <+66>:    movl   $0x0,-0x8(%rbp)
   0x0000555555555193 <+73>:    jmp    0x5555555551bf <madd+117>
=> 0x0000555555555195 <+75>:    mov    -0x8(%rbp),%eax
   0x0000555555555198 <+78>:    cltq
   0x000055555555519a <+80>:    mov    -0x1a0(%rbp,%rax,4),%edx
   0x00005555555551a1 <+87>:    mov    -0x8(%rbp),%eax
   0x00005555555551a4 <+90>:    cltq
   0x00005555555551a6 <+92>:    mov    -0x330(%rbp,%rax,4),%eax
   0x00005555555551ad <+99>:    add    %eax,%edx
   0x00005555555551af <+101>:   mov    -0x8(%rbp),%eax
   0x00005555555551b2 <+104>:   cltq
   0x00005555555551b4 <+106>:   mov    %edx,-0x1a0(%rbp,%rax,4)
   0x00005555555551bb <+113>:   addl   $0x1,-0x8(%rbp)
   0x00005555555551bf <+117>:   cmpl   $0x63,-0x8(%rbp)
   0x00005555555551c3 <+121>:   jle    0x555555555195 <madd+75>
   0x00005555555551c5 <+123>:   nop
   0x00005555555551c6 <+124>:   leaveq
   0x00005555555551c7 <+125>:   retq
End of assembler dump.
(gdb) stepi 6
0x00005555555551ad      11                      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[97]的值 104，%edx 中存储 a[97]的值 **47**，输入 stepi 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[97]的值 **104**，%edx 中存储 **a[97]+b[97]的值 151**，下面是显示的十六进制的数值。由此我们可以得到，汇编指令如何实现 a[i]+=b[i]加法运算的：

```
(gdb) print $eax
$2 = 104
(gdb) print $edx
$3 = 47
(gdb) print /x $eax
$4 = 0x68
(gdb) print /x $edx
$5 = 0x2f
(gdb) stepi
0x00005555555551af      11                      a[i]+=b[i];
(gdb) print $eax
$6 = 104
(gdb) print $edx
$7 = 151
(gdb) print /x $eax
$8 = 0x68
(gdb) print /x $edx
$9 = 0x97
(gdb)
```

五、总结体会

遇到的问题：

①总体来说，如果思路清晰，gcc 的实验并不难理解。在初期做实验的过程中，我尝试自己设定调试步骤，但发现最后难以归到数组相加的指令，于是我通过查找资料，反复对照，形成了一个较为完整的调试闭环，得到了最后的结果，完成了实验；

②刚开始做实验的时候，对整个 gcc、vi 编辑器、objdump 反汇编工具的应用仍然很模糊，因为不太了解所以上课也没能跟上老师的思路。幸好 ppt 有具体的指令解释，为我理解实验过程提供了重要的参考资料；相信通过后面的几次实验，我能够对汇编语言的理解更加深入，读写更加得心应手；也很期待在接下来的学习中能够了解更多的汇编语言的指令使用方法。

总结心得：

本次实验向我们介绍了 linux 环境下代码为编译和汇编程序以及介绍了 gdb、gcc、vi 等新的编译器的使用方法和对指令的具体操作方法，同时还利用了 obdjump 反汇编工具增进了对汇编语言的进一步认识和探索。在这个过程中，我们接触到了汇编语言源码和汇编码的比照，逐渐理解了汇编语言的运行法则与执行过程，更加巩固了《深入理解计算机系统》的关于汇编语言的课程内容。