

北京邮电大学

实验报告



题目： Linux 环境和 GCC 工具链

班 级： 2020211314

学 号： 2020211616

姓 名： 付容天

学 院： 计算机学院（国家示范性软件学院）

2021 年 10 月 24 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

列举你所使用的软件工具

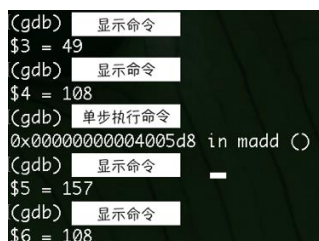
- 1、 Dev-C++ 5.15
- 2、 Microsoft Visual Studio 2019
- 3、 Typora 0.11.8
- 4、 PyCharm Community Edition 2021.2.2
- 5、 Microsoft Edge 版本 94.0.992.38
- 6、 Chrome 版本 94.0.4606.81

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 2019211x*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（使用 `-g` 选项, 但不使用优化选项）,

- 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 2019211x*y 的百位和个位。

学号 2019211999, $a[99]+b[99]$ 单步执行前后的参考截图如下（实际命令未显示出）:



```
(gdb) 显示命令
$3 = 49
(gdb) 显示命令
$4 = 108
(gdb) 单步执行命令
0x00000000004005d8 in madd ()
(gdb) 显示命令
$5 = 157
(gdb) 显示命令
$6 = 108
```

四、实验步骤及实验分析

首先启动了 powershell, 在正式的实验开始之前, 遇到的问题是登录异常, 经过网络搜索后, 使用命令 `ssh-keygen -R 10.120.11.12` 之后, 解决了这个问题。

首先的任务是使用 `vi` 进行程序编写, 在复习了课程相关内容、看了 PPT 上有关 `vi` 的内容之后, 很快写出了程序, 但是在保存文件并推出 `vi` 的时候, 有多种选择, 最终我使用了指令 `:wq` 和 `:w! [new file name]`,

将文件命名为 Lab1.c。之后用 gcc 指令编译运行了 Lab1.c。

接着使用 objdump 工具生成汇编程序，命名为 Lab1.s，通过 vi 程序进入 Lab1.s 文件，很快找到 madd 函数对应的汇编代码部分，截图如下：

```
madd:
.LFB0:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $832, %rsp
movq %fs:40, %rax
movq %rax, -8(%rbp)
xorl %eax, %eax
movl $0, -828(%rbp)
jmp .L2

.L3:
movl -828(%rbp), %eax
leal -50(%rax), %edx
movl -828(%rbp), %eax
cltq %edx, -816(%rbp, %rax, 4)
movl -828(%rbp), %eax
leal 6(%rax), %edx
movl -828(%rbp), %eax
cltq %edx, -416(%rbp, %rax, 4)
addl $1, -828(%rbp)

.L2:
cmpl $99, -828(%rbp)
jle .L3
movl $0, -824(%rbp)
jmp .L4

.L5:
movl -824(%rbp), %eax
cltq %edx, -816(%rbp, %rax, 4), %edx
movl -824(%rbp), %eax
cltq %edx, %eax
addl $1, -824(%rbp), %eax
-- INSERT --
```

(图 1-madd)

```
.L2:
cmpl $99, -828(%rbp)
jle .L3
movl $0, -824(%rbp)
jmp .L4

.L5:
movl -824(%rbp), %eax
cltq %edx, -816(%rbp, %rax, 4), %edx
movl -824(%rbp), %eax
cltq %edx, %eax
addl $1, -824(%rbp), %eax
-- INSERT --

.L4:
cmpl $99, -824(%rbp)
jle .L5
movl $0, -820(%rbp)
jmp .L6

.L8:
movl -820(%rbp), %eax
cltq %edx, -816(%rbp, %rax, 4), %eax
movl %eax, %esi
leaq .LC0(%rip), %rdi
movl $0, %eax
call printf@PLT
movl -820(%rbp), %eax
leal 1(%rax), %ecx
movslq %ecx, %rax
imulq $1717986919, %rax, %rax
shrq $32, %rax
movl %eax, %edx
sarl $2, %edx
movl %ecx, %eax
sarl $31, %eax
subl %eax, %edx
movl %edx, %eax
sall $4, %eax
addl %edx, %eax
-- INSERT --
```

(图 2-madd)

```
leaq .LC0(%rip), %rdi
movl $0, %eax
call printf@PLT
movl -820(%rbp), %eax
leal 1(%rax), %ecx
movslq %ecx, %rax
imulq $1717986919, %rax, %rax
shrq $32, %rax
movl %eax, %edx
sarl $2, %edx
movl %ecx, %eax
sarl $31, %eax
subl %eax, %edx
movl %edx, %eax
sall $4, %eax
addl %edx, %eax
addl %eax, %eax
subl %eax, %ecx
movl %ecx, %edx
testl %edx, %edx
jne .L7
movl $10, %edi
call putchar@PLT

.L7:
addl $1, -820(%rbp)

.L6:
cmpl $99, -820(%rbp)
jle .L8
movl $0, %eax
movq -8(%rbp), %rsi
xorq %fs:40, %rsi
je .L10
call __stack_chk_fail@PLT

.L10:
leave
.cfi_def_cfa 7, 0
ret
.cfi_endproc

.LFE0:
.size madd, .-madd
.globl main
.type main, @function

main:
.LFB1:
-- INSERT --
```

(图 3-madd)

然后的任务就是使用 gdb 进行调试并练习使用相关指令，调试过程截图如下：

```
2020211616@bupt1:~$ gdb
GNU gdb (Ubuntu 9.1-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gplv3>
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 Lab1
Reading symbols from Lab1...
(gdb) list
3      int a[100], b[100];
4      for(int i=0;i<100;i++){
5          a[i]=i-50;
6          b[i]=i+6; //2020211616
7      }
8      for(int i=0;i<100;i++){
9          a[i]=b[i];
10     }
11     for(int i=0;i<100;i++){
12         printf("\t%d",a[i]);
(gdb) l
13         if((i+1)%10==0) printf("\n");
14     }
15     return 0;
16 }
17 int main(){
18     madd();
19     return 0;
20 }
(gdb) l
Line number 21 out of range; Lab1.c has 20 lines.
(gdb) l
Line number 21 out of range; Lab1.c has 20 lines.
(gdb) break 9
Breakpoint 1 at 0x401000: file Lab1.c, line 9.
(gdb) run
Starting program: /students/2020211616/Lab1
Breakpoint 1, madd () at Lab1.c:9
9      a[i]=b[i];
(gdb) watch i
Hardware watchpoint 2: i
(gdb) watch a[i]
Hardware watchpoint 3: a[i]
(gdb) continue
Continuing.
Hardware watchpoint 3: a[i]
Old value = -49
New value = -42
madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) continue
Continuing.
Hardware watchpoint 2: i
Old value = 1
New value = 2
Hardware watchpoint 3: a[i]
Old value = -42
New value = -48
in madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) continue
Continuing.
Breakpoint 1, madd () at Lab1.c:9
9      a[i]=b[i];
(gdb) continue
Continuing.
Hardware watchpoint 3: a[i]
Old value = -48
New value = -40
```

(图 4-gdb, file, list)

```
(gdb) l
Line number 21 out of range; Lab1.c has 20 lines.
(gdb) l
Line number 21 out of range; Lab1.c has 20 lines.
(gdb) break 9
Breakpoint 1 at 0x401000: file Lab1.c, line 9.
(gdb) run
Starting program: /students/2020211616/Lab1
Breakpoint 1, madd () at Lab1.c:9
9      a[i]=b[i];
(gdb) watch i
Hardware watchpoint 2: i
(gdb) watch a[i]
Hardware watchpoint 3: a[i]
(gdb) continue
Continuing.
Hardware watchpoint 3: a[i]
Old value = -50
New value = -44
madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) continue
Continuing.
Undefined command: "continue". Try "help".
(gdb) continue
Continuing.
Hardware watchpoint 2: i
Old value = 0
New value = 1
Hardware watchpoint 3: a[i]
Old value = -44
New value = -49
in madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) continue
Continuing.
Breakpoint 1, madd () at Lab1.c:9
9      a[i]=b[i];
(gdb) continue
Continuing.
Hardware watchpoint 3: a[i]
Old value = -48
New value = -40
```

(图 5,6-break, run, continue, watch)

```

Breakpoint 1, madd () at Lab1.c:9
9      a[i]+=b[i];
(gdb) info break
Num      Type      Disp Enb Address      What
1        breakpoint keep y      in madd at Lab1.c:9
2        hw watchpoint keep y      i
3        hw watchpoint keep y      a[i]
(gdb) disassemble $pc
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
<+5>:      mov     %rsp,%rbp
<+8>:      sub     $0x340,%rsp
<+15>:     mov     %fs:0x28,%rax
<+24>:     mov     %rax,-0x8(%rbp)
<+28>:     xor     %eax,%eax
<+30>:     movl    $0x0,-0x33c(%rbp)
<+40>:     jmp     <madd+97>
<+42>:     mov     -0x33c(%rbp),%eax
<+48>:     lea     -0x32(%rax),%edx
<+51>:     mov     -0x33c(%rbp),%eax
<+57>:     cltq
<+59>:     mov     %edx,-0x330(%rbp,%rax,4)
<+66>:     mov     -0x33c(%rbp),%eax
<+72>:     lea     0x6(%rax),%edx
<+75>:     mov     -0x33c(%rbp),%eax
<+81>:     cltq
<+83>:     mov     %edx,-0x1a0(%rbp,%rax,4)
<+90>:     addl    $0x1,-0x33c(%rbp)
<+97>:     cmpl    $0x63,-0x33c(%rbp)
<+104>:    jle     <madd+42>
<+106>:    movl    $0x0,-0x338(%rbp)
<+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
=>

```

(图 7-info break, disassemble)

```

(gdb) info break
Num      Type      Disp Enb Address      What
1        breakpoint keep y      in madd at Lab1.c:9
2        hw watchpoint keep y      i
3        hw watchpoint keep y      a[i]
4        breakpoint keep y      in madd at Lab1.c:12
(gdb) delete 4
(gdb) info break
Num      Type      Disp Enb Address      What
1        breakpoint keep y      in madd at Lab1.c:9
2        hw watchpoint keep y      i
3        hw watchpoint keep y      a[i]
(gdb) nexti
9      a[i]+=b[i];
(gdb) nexti
9      a[i]+=b[i];
(gdb) nexti
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) continue
Continuing.

Hardware watchpoint 3: a[i]

Old value = -43
New value = -30
madd () at Lab1.c:8

```

(图 8-delete, nexti, stepi)

```

(gdb) continue
Continuing.

Breakpoint 1, madd () at Lab1.c:9
9      a[i]+=b[i];
(gdb) nexti
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
Hardware watchpoint 3: a[i]

Old value = -42
New value = -28
madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) ni

```

(图 9-stepi, nexti)

```

Breakpoint 1, madd () at Lab1.c:9
9      a[i]+=b[i];
(gdb) si
9      a[i]+=b[i];
(gdb) si
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
(gdb) stepi
9      a[i]+=b[i];
Hardware watchpoint 3: a[i]

Old value = -40
New value = -24
madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) stepi

Hardware watchpoint 2: i

Old value = 10
New value = 11

```

(图 10-stepi)


```

(gdb) continue
Continuing.

Hardware watchpoint 3: a[i]

Old value = -36
New value = -16
madd () at Lab1.c:8
8      for(int i=0;i<100;i++){
(gdb) print i
$1 = 14
(gdb) print a[i]
$2 = -16
(gdb) print b[i]
$3 = 20
(gdb) info reg
rax            0xe            14
rbx            0x555555552f0    93824992236272
rcx            0x5555555552f0    93824992236272
rdx            0xfffffffff0     4294967280
rsi            0x7fffffffefe8    140737488350184
rdi            0x1             1
rbp            0x7fffffffcae0    0x7fffffffcae0
rsp            0x7fffffff7a0     0x7fffffff7a0
r8             0x0             0
r9             0x7ffff7fe0d50    140737354009936
r10            0xf             15
r11            0x2             2
r12            0x555555550a0     93824992235680
r13            0x7fffffffefe0    140737488350176
r14            0x0             0
r15            0x0             0
rip            0x5555555522e     0x5555555522e <madd+165>
eflags         0x296            [ PF AF SF IF ]
cs             0x33            51
ss             0x2b            43
ds             0x0             0
es             0x0             0
fs             0x0             0
gs             0x0             0

```

(图 11-info reg, print)

```

: 0x0000002a
(gdb) x
: 0x0000002b
(gdb) x/3un
: Undefined output format "n".
(gdb) x/3uh
: 45      0      46
(gdb) x/4uh
: 0       47      0      48
(gdb) x/10uh
: 0       49      0      50      0      51      0      52
: 0       53
(gdb) x/8uh
: 0       54      0      55      0      56      0      57
(gdb) x/3dh
: 0       58      0
(gdb) continue
Continuing.

Hardware watchpoint 2: i

Old value = 14
New value = 15

Hardware watchpoint 3: a[i]

Old value = -16
New value = -35

      in madd () at Lab1.c:8
      for(int i=0;i<100;i++){
(gdb) continue
Continuing.

Breakpoint 1, madd () at Lab1.c:9
9      a[i]+=b[i];
(gdb) continue
Continuing.

Hardware watchpoint 3: a[i]

Old value = -35
New value = -14
madd () at Lab1.c:8
8      for(int i=0;i<100;i++){

```

(图 12-x)

第三个实验任务是找到 `a[i]+b[i]` 指令对应的汇编指令，我采取的办法是：在语句 `a[i]+=b[i]` 所在行设置断点，然后执行，此时程序会在断点行停下，等待命令。这时，断点行尚未执行，采用 `disassemble` 指令查看此时的汇编指令，然后 `nexti` 逐步执行，直到 `a[i]+=b[i]` 完成，此时再次使用 `disassemble` 查看汇编指令。上述两次汇编指令查看结果所构成的区间便是 `a[i]+=b[i]` 对应的汇编代码（区间），结果如下：

```

(gdb) disassemble
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
<+5>:      mov     %rsp,%rbp
<+8>:      sub     $0x340,%rsp
<+15>:     mov     %fs:0x28,%rax
<+24>:     mov     %rax,-0x8(%rbp)
<+28>:     xor     %eax,%eax
<+38>:     movl    $0x0,-0x33c(%rbp)
<+40>:     jmp     <madd+97>
<+42>:     mov     -0x33c(%rbp),%eax
<+48>:     lea     -0x32(%rax),%edx
<+51>:     mov     -0x33c(%rbp),%eax
<+57>:     cltq
<+59>:     mov     %edx,-0x330(%rbp,%rax,4)
<+66>:     mov     -0x33c(%rbp),%eax
<+72>:     lea     0xe(%rax),%edx
<+75>:     mov     -0x33c(%rbp),%eax
<+81>:     cltq
<+83>:     mov     %edx,-0x1a0(%rbp,%rax,4)
<+90>:     addl    $0x1,-0x33c(%rbp)
<+97>:     cmpl    $0x63,-0x33c(%rbp)
<+104>:    jle     <madd+42>
<+106>:    movl    $0x0,-0x338(%rbp)
<+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
<+139>:    cltq
<+141>:    mov     -0x1a0(%rbp,%rax,4),%eax
<+148>:    add     %eax,%edx
<+150>:    mov     -0x338(%rbp),%eax
<+156>:    cltq
<+158>:    mov     %edx,-0x330(%rbp,%rax,4)
<+165>:    addl    $0x1,-0x338(%rbp)
<+172>:    cmpl    $0x63,-0x338(%rbp)
<+179>:    jle     <madd+118>
--Type <RET> for more, q to quit, c to continue without paging.-c
<+181>:    movl    $0x0,-0x334(%rbp)

```

(图 13)

```

--Type <RET> for more, q to quit, c to continue without paging.-q
Quit
(gdb) nexti
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) ni
9      a[i]+=b[i];
(gdb) disassemble
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
<+5>:      mov     %rsp,%rbp
<+8>:      sub     $0x340,%rsp
<+15>:     mov     %fs:0x28,%rax
<+24>:     mov     %rax,-0x8(%rbp)
<+28>:     xor     %eax,%eax
<+38>:     movl    $0x0,-0x33c(%rbp)
<+40>:     jmp     <madd+97>

```

(图 14)

```

(gdb) ni
8      for(int i=0;i<100;i++){
(gdb) disassemble
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
<+5>:      mov     %rsp,%rbp
<+8>:      sub     $0x340,%rsp
<+15>:     mov     %fs:0x28,%rax
<+24>:     mov     %rax,-0x8(%rbp)
<+28>:     xor     %eax,%eax
<+38>:     movl    $0x0,-0x33c(%rbp)
<+40>:     jmp     <madd+97>
<+42>:     mov     -0x33c(%rbp),%eax
<+48>:     lea     -0x32(%rax),%edx
<+51>:     mov     -0x33c(%rbp),%eax
<+57>:     cltq
<+59>:     mov     %edx,-0x330(%rbp,%rax,4)
<+66>:     mov     -0x33c(%rbp),%eax
<+72>:     lea     0x6(%rax),%edx
<+75>:     mov     -0x33c(%rbp),%eax
<+81>:     cltq
<+83>:     mov     %edx,-0x1a0(%rbp,%rax,4)
<+90>:     addl    $0x1,-0x33c(%rbp)
<+97>:     cmpl    $0x63,-0x33c(%rbp)
<+104>:    jle     <madd+42>
<+106>:    movl    $0x0,-0x338(%rbp)
<+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
<+139>:    cltq
<+141>:    mov     -0x1a0(%rbp,%rax,4),%eax
<+148>:    add     %eax,%edx
<+150>:    mov     -0x338(%rbp),%eax
<+156>:    cltq
<+158>:    mov     %edx,-0x330(%rbp,%rax,4)
<+165>:    addl    $0x1,-0x338(%rbp)
<+172>:    cmpl    $0x63,-0x338(%rbp)
<+179>:    jle     <madd+118>
<+181>:    movl    $0x0,-0x334(%rbp)
<+193>:    mov     -0x334(%rbp),%eax

```

(图 15)

停在断点处的时候，第一次进入查看汇编代码，指在了<+118>处，然后执行 `nexti` 直到 `a[i]+=b[i]` 完成，然后第二次查看汇编代码，指在了<+165>处，因此，语句 `a[i]+=b[i]` 所对应的汇编代码（区间）就是<+118>

到<+158>,也就是图 15 中的高亮部分。接下来想要确定 $a[i] += b[i]$ 的结果所在的寄存器,采取的办法依然是基于逐步执行的分析方法,第一次遇到断点的时候查看汇编代码并打印此时的 $a[i]$ 的值,然后 nexti 逐步执行,并在每次执行的时候打印此时的 $a[i]$ 的值,一旦法发现 $a[i]$ 的值发生了变化,再次查看汇编代码,在此时指示的行的上下附近行寻找可疑的指令(比如 mov 或者 add 指令),观察其中涉及到的寄存器,分析哪一个寄存器可能是存储 $a[i]$ 的寄存器,并使用指令 info register <register name>来查看其中的值,过程截图如下:

```
Breakpoint 1, madd () at Lab1.c:9
9      a[i] += b[i];
(gdb) disassemble $pc
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
<+5>:      mov     %rsp,%rbp
<+8>:      sub     $0x340,%rsp
<+15>:     mov     %fs:0x28,%rax
<+24>:     mov     %rax,-0x8(%rbp)
<+28>:     xor     %eax,%eax
<+30>:     movl    $0x0,-0x33c(%rbp)
<+40>:     jmp     <madd+97>
<+42>:     mov     -0x33c(%rbp),%eax
<+48>:     lea     -0x32(%rax),%edx
<+51>:     mov     -0x33c(%rbp),%eax
<+57>:     cltq
<+59>:     mov     %edx,-0x330(%rbp,%rax,4)
<+66>:     mov     -0x33c(%rbp),%eax
<+72>:     lea     0x6(%rax),%edx
<+75>:     mov     -0x33c(%rbp),%eax
<+81>:     cltq
<+83>:     mov     %edx,-0x1a0(%rbp,%rax,4)
<+90>:     addl    $0x1,-0x33c(%rbp)
<+97>:     cmpl    $0x63,-0x33c(%rbp)
<+104>:    jle     <madd+42>
<+106>:    movl    $0x0,-0x338(%rbp)
=> <+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
<+139>:    cltq
<+141>:    mov     -0x1a0(%rbp,%rax,4),%eax
<+148>:    add     %eax,%edx
<+150>:    mov     -0x338(%rbp),%eax
<+156>:    cltq
<+158>:    mov     %edx,-0x330(%rbp,%rax,4)
<+165>:    addl    $0x1,-0x338(%rbp)
<+172>:    cmpl    $0x63,-0x338(%rbp)
<+179>:    jle     <madd+118>
<+181>:    movl    $0x0,-0x334(%rbp)
```

(图 16-第一次遇到断点)

```
<+201>:    mov     -0x330(%rbp,%rax,4),%eax
--Type <RET> for more, q to quit, c to continue without paging--ni
<+208>:    mov     %eax,%esi
<+210>:    lea     0xda2(%rip),%rdi      #
<+217>:    mov     $0x0,%eax
<+222>:    callq   <printf@plt>
<+227>:    mov     -0x334(%rbp),%eax
<+233>:    lea     0x1(%rax),%ecx
<+236>:    movslq  %ecx,%rax
<+239>:    imul    $0x66666667,%rax,%rax
<+246>:    shr     $0x20,%rax
<+250>:    mov     %eax,%edx
<+252>:    sar     $0x2,%edx
<+255>:    mov     %ecx,%eax
<+257>:    sar     $0x1f,%eax
<+260>:    sub     %eax,%edx
<+262>:    mov     %edx,%eax
<+264>:    shl     $0x2,%eax
<+267>:    add     %edx,%eax
<+269>:    add     %eax,%eax
<+271>:    sub     %eax,%ecx
<+273>:    mov     %ecx,%edx
<+275>:    test    %edx,%edx
<+277>:    jne     <madd+289>
<+279>:    mov     $0xa,%edi
<+284>:    callq   <putchar@plt>
<+289>:    addl    $0x1,-0x334(%rbp)
<+296>:    cmpl    $0x63,-0x334(%rbp)
<+303>:    jle     <madd+193>
<+305>:    mov     $0x0,%eax
<+310>:    mov     -0x8(%rbp),%rsi
<+314>:    xor     %fs:0x28,%rsi
<+323>:    je      <madd+330>
<+325>:    callq   <__stack_chk_fail@plt>
<+330>:    leaveq
<+331>:    retq
End of assembler dump.
(gdb) print a[i]
$19 = -42
(gdb) ni
9      a[i] += b[i];
```

(图 17-第一次遇到断点时的 $a[i]$ 的值为-42)

```
<+104>:    jle     <madd+42>
<+106>:    movl    $0x0,-0x338(%rbp)
=> <+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
<+139>:    cltq
<+141>:    mov     -0x1a0(%rbp,%rax,4),%eax
<+148>:    add     %eax,%edx
<+150>:    mov     -0x338(%rbp),%eax
<+156>:    cltq
<+158>:    mov     %edx,-0x330(%rbp,%rax,4)
<+165>:    addl    $0x1,-0x338(%rbp)
<+172>:    cmpl    $0x63,-0x338(%rbp)
<+179>:    jle     <madd+118>
<+181>:    movl    $0x0,-0x334(%rbp)
<+191>:    jmp     <madd+296>
<+193>:    mov     -0x334(%rbp),%eax
<+199>:    cltq
<+201>:    mov     -0x330(%rbp,%rax,4),%eax
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) ni
9      a[i] += b[i];
(gdb) print a[i]
$16 = -42
(gdb) ni
9      a[i] += b[i];
(gdb) print a[i]
$17 = -42
(gdb) ni
9      a[i] += b[i];
(gdb) print a[i]
$18 = -42
(gdb) ni
8      for(int i=0;i<100;i++){
(gdb) print a[i]
$19 = -28
(gdb) disassemble
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
```

(图 18- $a[i]$ 的值恰好更新)

```
<+97>:    cmpl    $0x63,-0x33c(%rbp)
<+104>:    jle     <madd+42>
<+106>:    movl    $0x0,-0x338(%rbp)
<+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
<+139>:    cltq
<+141>:    mov     -0x1a0(%rbp,%rax,4),%eax
<+148>:    add     %eax,%edx
<+150>:    mov     -0x338(%rbp),%eax
<+156>:    cltq
<+158>:    mov     %edx,-0x330(%rbp,%rax,4)
=> <+165>:    addl    $0x1,-0x338(%rbp)
<+172>:    cmpl    $0x63,-0x338(%rbp)
<+179>:    jle     <madd+118>
<+181>:    movl    $0x0,-0x334(%rbp)
<+191>:    jmp     <madd+296>
<+193>:    mov     -0x334(%rbp),%eax
<+199>:    cltq
<+201>:    mov     -0x330(%rbp,%rax,4),%eax
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) info register eax
eax             0x8                8
(gdb) info register edx
edx             0xffffffffe4       -28
(gdb) ni
8      for(int i=0;i<100;i++){
(gdb) info register eax
eax             0x8                8
(gdb) info register edx
edx             0xffffffffe4       -28
(gdb) ni
8      for(int i=0;i<100;i++){
(gdb) info register eax
eax             0x8                8
(gdb) info register edx
edx             0xffffffffe4       -28
(gdb) ni
```

(图 19-猜测 $a[i]$ 所在的寄存器并验证)

在图 19 中，a[i] 的值已经更新（从 -42 到 -28， $-28 = -42 + 14$ ），观察临近的可疑汇编语句，发现有两个寄存器可能储存了 a[i] 或者 b[i] 的值：eax 和 edx，分别打印出其中的值，发现 edx 是存储 a[i] 值的寄存器。下面再分析存储 b[i] 的寄存器，从图 19 可以看出，两个可疑的寄存器都没有存储 b[i] 的值，所以猜测 b[i] 所在寄存器的值发生了更新，覆盖了原先在其中的 b[i]，为了验证这个猜想，我继续 nexti 逐步执行，并观察 eax 的值的变化（因为已经确定了 edx 存储了 a[i]，故不用考虑），过程截图如下：

```
Breakpoint 1, madd () at Lab1.c:9
9      a[i]+=b[i];
(gdb) info register eax
eax      0x8      8
(gdb) info register edx
edx      0xffffffffe4      -28
(gdb) ni
(gdb) info register eax
eax      0x9      9
(gdb) info register edx
edx      0xffffffffe4      -28
(gdb) ni
(gdb) info register eax
eax      0x9      9
(gdb) info register edx
edx      0xffffffffe4      -28
(gdb) ni
(gdb) info register eax
eax      0x9      9
(gdb) info register edx
edx      0xffffffffd7      -41
(gdb) ni
(gdb) info register eax
eax      0x9      9
(gdb) info register edx
edx      0xffffffffd7      -41
(gdb) disassemble
Dump of assembler code for function madd:
<+0>:      endbr64
<+4>:      push    %rbp
<+5>:      mov     %rsp,%rbp
<+8>:      sub     $0x340,%rsp
<+15>:     mov     %fs:0x28,%rax
<+24>:     mov     %rax,-0x8(%rbp)
<+28>:     xor     %eax,%eax
<+30>:     movl    $0x0,-0x33c(%rbp)
<+40>:     jmp     <madd+97>
<+42>:     mov     -0x33c(%rbp),%eax
<+48>:     lea     -0x32(%rax),%edx
```

（图 20-逐步执行并观察寄存器 eax 的值）

在执行的过程中，发现 eax 寄存器在执行过程中存储过 b[i] 的值（即图 21 中的“15”），并且也存储过 i 的值（即图 20 和图 21 中的“8”和“9”），因此就找到了执行过程中 b[i] 所在的寄存器是 eax，并且这个寄存器还具有存储循环变量 i 的作用。

最后一个任务是使用单步指令及 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 2019211x*y 的百位和个位，我的学号是 2020211616，因此 x=6, y=6，需显示 a[66]+=b[66] 前后对应寄存器的值，过程截图如下：

```
<+106>:    movl    $0x0,-0x338(%rbp)
<+116>:    jmp     <madd+172>
<+118>:    mov     -0x338(%rbp),%eax
<+124>:    cltq
<+126>:    mov     -0x330(%rbp,%rax,4),%edx
<+133>:    mov     -0x338(%rbp),%eax
<+139>:    cltq
<+141>:    mov     -0x1a0(%rbp,%rax,4),%eax
<+148>:    add     %eax,%edx
<+150>:    mov     -0x338(%rbp),%eax
<+156>:    cltq
<+158>:    mov     %edx,-0x330(%rbp,%rax,4)
<+165>:    addl    $0x1,-0x338(%rbp)
<+172>:    cmpl    $0x63,-0x338(%rbp)
<+179>:    jle     <madd+118>
<+181>:    movl    $0x0,-0x334(%rbp)
<+191>:    jmp     <madd+296>
<+193>:    mov     -0x334(%rbp),%eax
<+199>:    cltq
<+201>:    mov     -0x330(%rbp,%rax,4),%eax
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) ni
(gdb) info register eax
eax      0x9      9
(gdb) info register edx
edx      0xffffffffd7      -41
(gdb) ni
(gdb) info register eax
eax      0xf      15
(gdb) info register edx
edx      0xffffffffd7      -41
(gdb) ni
(gdb) info register eax
eax      0xf      15
(gdb) info register edx
edx      0xffffffffe6      -26
(gdb)
```

（图 21-确定 b[i] 所在的寄存器）

（附图见下页）

```

line number 21 out of range; Lab1.c has 20 lines.
(gdb) break 9 if i==66
Breakpoint 1 at : file Lab1.c, line 9.
(gdb) run
Starting program: /students/2020211616/Lab1

Breakpoint 1, madd () at Lab1.c:9
9          a[i]+=b[i];
(gdb) print i
$1 = 66
(gdb) print a[i]
$2 = 16
(gdb) info register edx
edx      0x56      86
(gdb) info register eax
eax      0x41      65
(gdb) ni
(gdb) info register edx
edx      0x56      86
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x56      86
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16

```

(图 22)

```

eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x10      16
(gdb) info register eax
eax      0x48      72
(gdb) ni
(gdb) info register edx
edx      0x58      88
(gdb) info register eax
eax      0x48      72
(gdb) ni
(gdb) info register edx
edx      0x58      88
(gdb) info register eax
eax      0x42      66
(gdb) ni
(gdb) info register edx
edx      0x58      88
(gdb) info register eax
eax      0x42      66
(gdb)

```

(图 23)

设置条件断点，使得 $i=66$ 的时候程序暂停，然后使用 `nexti` 逐步操作，并跟踪 `a[i]`、寄存器 `edx` 和寄存器 `eax` 的值，一开始寄存器 `edx` 的值是 86，这是因为此时的寄存器 `edx` 存储的仍然是上一次循环的结果；一开始的寄存器 `eax` 的值是 65，这也是因为此时的寄存器 `eax` 存储的仍然是上一次循环变量 `i`（前面已经分析知道寄存器 `eax` 既存储 `b[i]` 又存储循环变量 `i`）。之后，寄存器 `eax` 的值更新成了 66（即 $i+1$ ），然后寄存器 `edx` 的值更新成了 16（即 `a[66]`），之后再经过几步操作使得寄存器 `eax` 的值更新成了 72（即 `b[66]`），然后寄存器 `edx` 的值更新成了 88（即 $a[66]+b[66]=16+72=88$ ）。

五、总结体会

本次实验是我做的第一个计算机系统相关的实验，经过本次实验，我初步掌握了 `gdb` 工具以及 `gdb` 调试方法，为后面的实验做好了准备基础。这次实验中我遇到了很多问题，并且也一一解决了这些问题，下面我想谈谈两个让我印象深刻的问题以及我的解决办法。

第一，工具使用的不熟练。其一是 `vi` 编辑器使用的不熟练。我通常使用集成开发环境来编写程序，所以在使用 `vi` 的时候非常不适应，而且如果有程序错误，返回去修改也十分麻烦。唯一的解决办法就是多练习使用 `vi` 来编写程序，并对其中涉及到的指令进行背诵记忆。其二是 `gdb` 调试的不熟练。使用 `gdb` 进行调试要比使用集成开发环境更加接近底层，这就要求我们足够了解底层的程序运行过程、并且熟记各项相关的调试指令才能很好地进行调试。虽然一开始我觉得使用 `gdb` 进行调试并不方便，但是随着使用次数的增加，我逐渐发现了 `gdb` 调试其实是十分强大的，它可以根据调试者的想法来查看任何一处的值、并且可以逐

个汇编代码进行调试，这十分有助于我们分析程序真正的运行过程，而不仅仅是表面上的值的变化。

第二，知识储备的缺失。在实验内容 3 和 4 中，我能够理解它所表示的意义，但是我并不知道使用什么方法来处理这个问题，我只是能够感觉到需要观察汇编代码并逐步调试，但是如何具体操作我并不知道。实验中，我进行了多次尝试并思考，最终发现，想要确定一条语句对应的汇编代码（区间），需要知道这条语句的开始和结束对应汇编代码的哪条语句。所以，我设置断点在相应的语句上，并且在遇到断点的时候查看汇编代码，这就确定了这条语句在汇编代码里面的开始位置，然后使用 nexti 逐步执行，直到该语句执行完毕，此时再次查看汇编代码，便可以知道这条语句在汇编代码里面的结束位置。

在本次实验中，一开始我尝试着做了一次预实验，并且认为自己理解了所有内容，但是在书写实验报告的时候，我正式地做这个实验的时候，发现要合理地解释每一个细节，需要长时间的思考和查证，这就启示我：详细的实验报告是实验不可缺少的一部分，它能够帮助我们回顾整个实验，并且促使我们思考实验中的每一处细节，在这些细节上的思考，是真正有意义、真正能够提高我们计算机思考水平的有效途径。

给本实验的建议：每个同学在做本实验的时候都会遇到各种各样的问题，但是这些问题可能非常细节以至于很难在互联网上找到答案，经过同学们的思考，这些问题大部分都能够解决，那么能否在整个实验结束之后，组织同学们把这些非常细节的问题收集起来呢？我认为大部分同学对于实验的大方向的理解都没有太大偏差，那么解决这些细节的问题就成了实验中一个很有意义的部分，在实验结束之后将其收集起来，大家共同参考，这样有助于我们在实验结束之后回顾实验，并且反思自己是否仍然有没处理好的地方。