

北京邮电大学

实验报告



题目: Linux 系统及其相关软件环境

班 级: _____

学 号: _____

姓 名: _____

学 院: _____

2023 年 10 月 21 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、远程登陆工具: MobaXterm (服务器: 10.120.11.12)
- 2、操作系统: Linux
- 3、编译器: Gcc
- 4、调试工具: GDB
- 5、反汇编工具: Objdump

三、实验概况

实验内容一

在 linux 环境下，编辑课件中源程序（注意程序的完整性）（包含源程序的开发环境截图），采用 gcc 编译该程序（要求分别采用 -o 和 -O 参数，并比较两者性能，编译指令截图），采用 gdb 进行调试，让程序运行到 for 函数语句（调试截图），运用 objdump 工具生成汇编程序（给出 main 函数的汇编程序截图）。

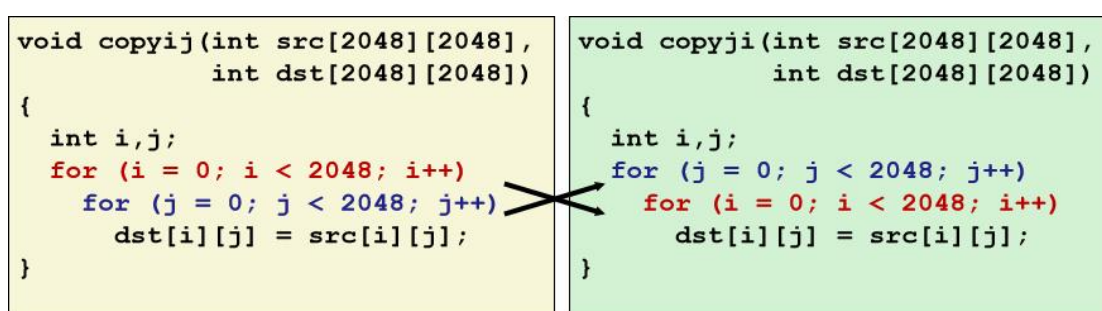
```
#include<stdio.h>

int main(void)
{
    double counter;
    double result;
    double temp;
    for(counter=0;counter<2000.0*2000.0*2000.0/20.0+2020;
    counter+=(5-1)/4){
```

```
temp=counter/1979;
result=counter;
}
printf(Result is%lf\\n,result);
return 0;
}
```

实验内容二

在 linux 环境下，分别打印输出如下算法所需时间：



分别设置不同优化参数，给出运行时间。

实验内容三

现有两个 int 型数组 $a[i]=i-50$ ， $b[i]=i+y$ ，其中 y 取自于学生本人学号 2022211x*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 -fno-pie -fno-stack-protector 选项），

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；

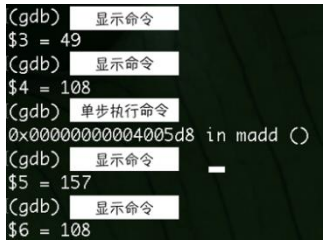
用 gdb 进行调试，练习下列 gdb 命令，给出截图；

gdb、file、kill、quit、break、delete、clear、info break、run、continue、nexti、stepi、disassemble、list、print、x、info reg、watch

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；

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 2022211x*y 的百位和个位。

学号 2022211999, 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) :



实验内容四

任选高复杂度算法 (具体算法自选, 类型分为高计算量类型和高内存需求类型 2 类算法) , 通过设置不同优化参数, 分析算法的运行效率。

四、实验步骤

实验内容一

1、编辑源程序

安装 MobaXterm, 输入 ip 地址与用户名, 登入服务器, 进入 linux 环境;
输入命令 vi test.c, 按下 i 进入 insert 模式, 编辑图 1 所示源程序;
结束编辑后, 按下 esc 进入 command 模式, 输入命令:wq!, 保存 test.c 文件。

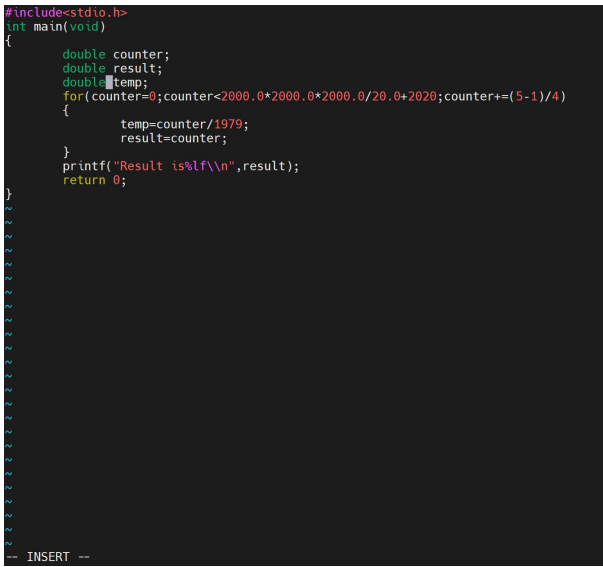


图 1: 源程序的开发环境

2、采用 gcc 编译, 比较不同优化参数下的程序性能

输入命令 gcc test.c -o test_for_o, 编译生成 test_for_o 可执行文件;

输入命令 `gcc -O0 test.c -o test_for_o0`，编译生成 O0 优化参数的 `test_for_o0` 可执行文件，`-O1`、`-O2`、`-O3`、`-Os` 参数同理。

```
2022211073@bupt1:~$ gcc test.c -o test_for_o
2022211073@bupt1:~$ gcc -O0 test.c -o test_for_o0
2022211073@bupt1:~$ gcc -O1 test.c -o test_for_o1
2022211073@bupt1:~$ gcc -O2 test.c -o test_for_o2
2022211073@bupt1:~$ gcc -O3 test.c -o test_for_o3
2022211073@bupt1:~$ gcc -Os test.c -o test_for_os
```

图 2: gcc 编译

输入命令 `time ./test_for_o`，查看运行时间，`-O0`、`-O1`、`-O2`、`-O3`、`-Os` 参数生成的文件同理。根据运行时间比较性能。

```
2022211073@bupt1:~$ time ./test_for_o0
Result is400002019.000000\n
real    0m1.417s
user    0m1.417s
sys     0m0.000s
2022211073@bupt1:~$ time ./test_for_o1
Result is400002019.000000\n
real    0m0.550s
user    0m0.550s
sys     0m0.000s
2022211073@bupt1:~$ time ./test_for_o2
Result is400002019.000000\n
real    0m0.554s
user    0m0.549s
sys     0m0.004s
2022211073@bupt1:~$ time ./test_for_o3
Result is400002019.000000\n
real    0m0.550s
user    0m0.550s
sys     0m0.000s
2022211073@bupt1:~$ time ./test_for_o4
-bash: ./test_for_o4: No such file or directory

real    0m0.001s
user    0m0.001s
sys     0m0.000s
2022211073@bupt1:~$ time ./test_for_os
Result is400002019.000000\n
real    0m0.554s
user    0m0.550s
sys     0m0.004s
```

图 3: 比较性能

3、采用 gdb 调试，让程序运行到 for 语句

输入命令 `gcc -g -o test_for_debug test.c`，编译生成可执行文件 `test_for_debug`;

输入命令 `gdb test_for_debug`，启动 gdb;

输入命令 `l`，查看代码;

输入命令 `b 7`，在第 7 行 for 语句处设置断点;

输入命令 `r`，程序执行至断点处。

```

2022211073@bupt1:~$ gcc -g -o test_for_debug test.c
2022211073@bupt1:~$ gdb test_for_debug
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" ...
Reading symbols from test_for_debug ...
(gdb) run
Starting program: /students/2022211073/test_for_debug
Result is 400002019.000000\n[Inferior 1 (process 1952045) exited normally]
(gdb) l
1      #include<stdio.h>
2      int main(void)
3      {
4          double counter;
5          double result;
6          double temp;
7          for(counter=0; counter<2000.0*2000.0*2000.0/20.0+2020; counter+=(5-1)/4)
8          {
9              temp=counter/1979;
10             result=counter;
(gdb) b 7
Breakpoint 1 at 0x555555555162: file test.c, line 7.
(gdb) r
Starting program: /students/2022211073/test_for_debug

Breakpoint 1, main () at test.c:7
7      for(counter=0; counter<2000.0*2000.0*2000.0/20.0+2020; counter+=(5-1)/4)

```

图 4: gdb 调试

4、采用 objdump 生成汇编程序

输入命令 `gcc -c -o test.o test.c`，编译生成 test.o;

输入命令 `objdump -s -d test.o > test.o.txt`，利用 objdump 将 test.o 的反汇编结果输出到文件 test.o.txt 中;

输入命令 `cat test.o.txt`，查看汇编程序。

```

2022211073@bupt1:~$ cat test.o.txt
test.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
0: 55                push    %rbp
1: 48 89 e5          mov     %rsp,%rbp
4: 48 83 ec 30       sub     $0x30,%rsp
8: 66 0f ef c0       pxor    %xmm0,%xmm0
c: f2 0f 11 45 e8    movsd   %xmm0,-0x18(%rbp)
11: eb 36            jmp     49 <main+0x49>
13: f2 0f 10 45 e8    movsd   -0x18(%rbp),%xmm0
18: f2 0f 10 0d 00 00 00 movsd   0x0(%rip),%xmm1      # 20 <main+0x20>
1f: 00
20: f2 0f 5e c1       divsd   %xmm1,%xmm0
24: f2 0f 11 45 f8    movsd   %xmm0,-0x8(%rbp)
29: f2 0f 10 45 e8    movsd   -0x18(%rbp),%xmm0
2e: f2 0f 11 45 f0    movsd   %xmm0,-0x10(%rbp)
33: f2 0f 10 4d e8    movsd   -0x18(%rbp),%xmm1
38: f2 0f 10 05 00 00 00 movsd   0x0(%rip),%xmm0      # 40 <main+0x40>
3f: 00
40: f2 0f 58 c1       addsd   %xmm1,%xmm0
44: f2 0f 11 45 e8    movsd   %xmm0,-0x18(%rbp)
49: f2 0f 10 05 00 00 00 movsd   0x0(%rip),%xmm0      # 51 <main+0x51>
50: 00
51: 66 0f 2e 45 e8    ucomisd -0x18(%rbp),%xmm0
56: 77 bb            ja      13 <main+0x13>
58: 48 8b 45 f0       mov     -0x10(%rbp),%rax
5c: 48 89 45 d8       mov     %rax,-0x28(%rbp)
60: f2 0f 10 45 d8    movsd   -0x28(%rbp),%xmm0
65: 48 8d 3d 00 00 00 00 lea     0x0(%rip),%rdi      # 6c <main+0x6c>
6c: b8 01 00 00 00    mov     $0x1,%eax
71: e8 00 00 00 00    callq   76 <main+0x76>
76: b8 00 00 00 00    mov     $0x0,%eax
7b: c9               leaveq  %eax
7c: c3               retq

```

图 5: main 函数的汇编程序

实验内容二

- 1、在 vi 编辑器下分别编辑 copy1.c, copy2.c 文件, 对应两种算法;
- 2、gcc 编译, 设置不同优化参数。

```
2022211073@bupt1:~$ gcc copy1.c -o copy1_for_o
2022211073@bupt1:~$ gcc -O0 copy1.c -o copy1_for_o0
2022211073@bupt1:~$ gcc -O1 copy1.c -o copy1_for_o1
2022211073@bupt1:~$ gcc -O2 copy1.c -o copy1_for_o2
2022211073@bupt1:~$ gcc -O3 copy1.c -o copy1_for_o3
2022211073@bupt1:~$ gcc -Os copy1.c -o copy1_for_os
2022211073@bupt1:~$ gcc copy2.c -o copy2_for_o
2022211073@bupt1:~$ gcc -O0 copy2.c -o copy2_for_o0
2022211073@bupt1:~$ gcc -O1 copy2.c -o copy2_for_o1
2022211073@bupt1:~$ gcc -O2 copy2.c -o copy2_for_o2
2022211073@bupt1:~$ gcc -O3 copy2.c -o copy2_for_o3
2022211073@bupt1:~$ gcc -Os copy2.c -o copy2_for_os
```

图 6: gcc 编译

- 3、查看不同优化参数下, 算法 1、2 的性能

```
2022211073@bupt1:~$ time ./copy1_for_o
Segmentation fault (core dumped)
real    0m0.187s
user    0m0.002s
sys     0m0.000s
2022211073@bupt1:~$ time ./copy1_for_o0
Segmentation fault (core dumped)
real    0m0.188s
user    0m0.002s
sys     0m0.000s
2022211073@bupt1:~$ time ./copy1_for_o1
Segmentation fault (core dumped)
real    0m0.188s
user    0m0.001s
sys     0m0.001s
2022211073@bupt1:~$ time ./copy1_for_o2
real    0m0.001s
user    0m0.000s
sys     0m0.002s
2022211073@bupt1:~$ time ./copy1_for_o3
real    0m0.001s
user    0m0.001s
sys     0m0.000s
2022211073@bupt1:~$ time ./copy1_for_os
Segmentation fault (core dumped)
real    0m0.189s
user    0m0.001s
sys     0m0.001s
```

图 6: 算法 1 不同优化参数下的性能

```
2022211073@bupt1:~$ time ./copy2_for_o
Segmentation fault (core dumped)
real    0m0.187s
user    0m0.001s
sys     0m0.001s
2022211073@bupt1:~$ time ./copy2_for_o0
Segmentation fault (core dumped)
real    0m0.186s
user    0m0.001s
sys     0m0.001s
2022211073@bupt1:~$ time ./copy2_for_o1
Segmentation fault (core dumped)
real    0m0.187s
user    0m0.001s
sys     0m0.001s
2022211073@bupt1:~$ time ./copy2_for_o2
real    0m0.001s
user    0m0.001s
sys     0m0.000s
2022211073@bupt1:~$ time ./copy2_for_o3
real    0m0.001s
user    0m0.001s
sys     0m0.000s
2022211073@bupt1:~$ time ./copy2_for_os
Segmentation fault (core dumped)
real    0m0.186s
user    0m0.001s
sys     0m0.000s
```

图 7: 算法 2 不同优化参数下的性能

实验内容三

1、使用 vi 编辑器编写程序 work3.c, 实现将两个 int 型数组 a[i]=i-50、b[i]=i+3 (学号个位为 3) 相加的功能。利用 gcc 编译器, 输入命令 -g -fno-pie -fno-stack-protector 编译程序, 使用 objdump 工具生成汇编程序。

```
#include <stdio.h>

void madd(int *res)
{
    int y = 3, a[100], b[100];
    for (int i = 0; i < 100; i++)
    {
        a[i] = i - 50;
        b[i] = i + 3;
        res[i] = a[i] + b[i];
    }
    return res;
}

int main(void)
{
    int res[100];
    madd(res);
    return 0;
}
```

图 8: 用 vi 编写源程序 work3.c

```
000000000000114a <add>:
114a: 55                push   %rbp
114b: 48 89 e5          mov     %r9p,%rbp
114c: 48 81 ec c0 02 00 00 sub     $0x2c0,%rsp
114d: 48 89 bd c8 fc ff ff mov     %rdi,-0x338(%rbp)
114e: c7 45 f8 03 00 00 00 movl    $0x3,-0x3(%rbp)
114f: c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp)
1150: eb 5b            jmp     11c7 <madd-0x7d-0x4(%rbp),%eax>
1151: 8d 50 ce         lea     -0x32(%rax),%edx
1152: 8b 45 fc         mov     -0x4(%rbp),%eax
1153: 89 94 85 60 fe ff ff cltq    %edx,-0x1a0(%rbp,%rax,4)
1154: 8b 45 fc         mov     -0x4(%rbp),%eax
1155: 8d 50 03         lea     0x3(%rax),%edx
1156: 8b 45 fc         mov     -0x4(%rbp),%eax
1157: 48 98            cltq
1158: 89 94 85 d0 fc ff ff mov     %edx,-0x338(%rbp,%rax,4)
1159: 8b 45 fc         mov     -0x4(%rbp),%eax
115a: 48 98            cltq
115b: 8b 8c 85 60 fe ff ff mov     -0x1a0(%rbp,%rax,4),%ecx
115c: 8b 45 fc         mov     -0x4(%rbp),%eax
115d: 48 98            cltq
115e: 8b 94 85 d0 fc ff ff mov     -0x338(%rbp,%rax,4),%edx
115f: 8b 45 fc         mov     -0x4(%rbp),%eax
1160: 48 98            cltq
1161: 48 d8 34 85 00 00 00 lea     0x0(%rax,4),%rsi
1162: 68              mov     %eax,%eax
1163: 48 8b 85 c8 fc ff ff mov     -0x338(%rbp),%rax
1164: 48 f0 f1         add     %rsi,%rax
1165: 01 ca           add     %ecx,%edx
1166: 89 10           mov     %edx,%rax
1167: 83 45 fc 01     addl    $0x1,-0x4(%rbp)
1168: 83 7d fc 03     cmpl    $0x63,-0x4(%rbp)
1169: 7e 9f           jle     116c <madd-0x2-0x338(%rbp),%rax>
116a: 48 8b 85 c8 fc ff ff mov     -0x338(%rbp),%rax
116b: c9              leaveq  %rax
```

图 9: madd 函数的汇编程序

2、用 gdb 进行调试, 练习 gdb 命令。

```
202211073@bupti:~$ 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.
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 work3_for_debug
Reading symbols from work3_for_debug...
(gdb) list
1  int* madd(int *res)
2  {
3      int y = 3, a[100], b[100];
4      for (int i = 0; i < 100; i++)
5      {
6          a[i] = i - 50;
7          b[i] = i + 3;
8          res[i] = a[i] + b[i];
9      }
10 }
11
```

图 10: gdb, file, list

```
(gdb) break 3
Breakpoint 1 at 0x116c: file work3.c, line 4.
(gdb) run
Starting program: /students/202211073/work3_for_debug
Breakpoint 1, madd (res=0x7fffffff920) at work3.c:4
4 {
(gdb) watch a[1]
Hardware watchpoint 2: a[1]
(gdb) continue
Continuing.

Hardware watchpoint 2: a[1]

Old value = 0
New value = -49
madd (res=0x7fffffff920) at work3.c:9
9      b[i] = i + 3;
(gdb) continue
Continuing.

Watchpoint 2 deleted because the program has left the block in
which its expression is valid.
main () at work3.c:19
19      return 0;
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 2063313) killed]
```

图 11: break, run, watch, continue, kill

```
(gdb) break 3
Breakpoint 1 also set at pc 0x116c.
Breakpoint 3 at 0x116c: file work3.c, line 4.
(gdb) break 5
Breakpoint 4 at 0x1170: file work3.c, line 5.
(gdb) info break
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000000000116c in madd at work3.c:4
2 breakpoint already hit 1 time
3 breakpoint keep y 0x000000000000116c in madd at work3.c:4
4 breakpoint keep y 0x0000000000001170 in madd at work3.c:5
(gdb) break 10
Breakpoint 5 at 0x11c1: file work3.c, line 10.
(gdb) delete 4
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000000000116c in madd at work3.c:4
2 breakpoint already hit 1 time
3 breakpoint keep y 0x000000000000116c in madd at work3.c:4
4 breakpoint keep y 0x00000000000011c1 in madd at work3.c:10
5 breakpoint keep y 0x0000000000001170 in madd at work3.c:5
(gdb) delete 3
(gdb) clear 4
(gdb) info break
Deleted breakpoint 1 Num Type Disp Enb Address What
5 breakpoint keep y 0x00000000000011c1 in madd at work3.c:10
(gdb) break 4
Breakpoint 6 at 0x116c: file work3.c, line 4.
(gdb) run
Starting program: /students/202211073/work3_for_debug
Breakpoint 6, madd (res=0x7fffffff920) at work3.c:4
4 {
(gdb) next
0x0000000000001170 4 {
```

图 12: info break, delete, clear, next

```
(gdb) next
0x0000000000001179 4 {
(gdb) step
1 int y = 3, a[100], b[100];
(gdb) step
for (int i = 0; i < 100; i++)
(gdb) disassemble
Dump of assembler code for function madd:
0x0000000000001170 <+>: push   %rbp
0x0000000000001171 <+>: mov     %r9p,%rbp
0x0000000000001172 <+>: sub     $0x2c0,%rsp
0x0000000000001173 <+>: mov     %rdi,-0x338(%rbp)
0x0000000000001174 <+>: mov     %fsi,-0x3(%rbp)
0x0000000000001175 <+>: mov     %fsi,-0x4(%rbp)
0x0000000000001176 <+>: jmp     0x00000000000011c7 <madd-0x7d-0x4(%rbp),%eax>
0x0000000000001177 <+>: lea     -0x32(%rax),%edx
0x0000000000001178 <+>: mov     -0x4(%rbp),%eax
0x0000000000001179 <+>: cltq    %edx,-0x1a0(%rbp,%rax,4)
0x000000000000117a <+>: mov     -0x4(%rbp),%eax
0x000000000000117b <+>: lea     0x3(%rax),%edx
0x000000000000117c <+>: mov     -0x4(%rbp),%eax
0x000000000000117d <+>: cltq
0x000000000000117e <+>: mov     %edx,-0x338(%rbp,%rax,4)
0x000000000000117f <+>: mov     -0x4(%rbp),%eax
0x0000000000001180 <+>: cltq
0x0000000000001181 <+>: mov     -0x1a0(%rbp,%rax,4),%ecx
0x0000000000001182 <+>: mov     -0x4(%rbp),%eax
0x0000000000001183 <+>: cltq
0x0000000000001184 <+>: mov     -0x338(%rbp,%rax,4),%edx
0x0000000000001185 <+>: mov     -0x4(%rbp),%eax
0x0000000000001186 <+>: cltq
0x0000000000001187 <+>: lea     0x0(%rax,4),%rsi
0x0000000000001188 <+>: mov     -0x338(%rbp),%rax
0x0000000000001189 <+>: add     %rsi,%rax
0x000000000000118a <+>: add     %ecx,%edx
```

图 13: stepi, disassemble

```
0x000000000000118b <+>: cmpl    $0x63,-0x4(%rbp)
0x000000000000118c <+>: jle     0x000000000000118f <madd-0x2-0x338(%rbp),%rax>
0x000000000000118d <+>: mov     -0x338(%rbp),%rax
0x000000000000118e <+>: mov     -0x338(%rbp),%rsi
0x000000000000118f <+>: leaveq  %rax
(gdb) print a[1]
$1 = 0
(gdb) x/x 0x0000000000001185 <madd-0x43>
0x0000000000001185: 0x00000000
(gdb) info reg
rax 0x0
rbx 0x0
rcx 0x0
rdx 0x0
rsi 0x0
rdi 0x0
rbp 0x0
r8 0x0
r9 0x0
r10 0x0
r11 0x0
r12 0x0
r13 0x0
r14 0x0
r15 0x0
rip 0x0
eflags 0x0
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 2064666) killed]
(gdb) quit
```

图 14: print, x, info reg, quit

3、启动 gdb，调试 work3_for_debug。在 a[i]+b[i]语句处设置断点，开始运行；在断点处停止时输入 disassemble 命令查看汇编指令，箭头所指<+103>处为 a[i]+b[i]语句的汇编指令起点；

```

(gdb) file work3_for_debug
Load new symbol table from 'work3_for_debug'? (y or n) y
Reading symbols from work3_for_debug...
(gdb) info break
Num Type Disp Enb Address What
3 breakpoint keep y 0x00000000000001c1 in madd at work3.c:10
Breakpoint already hit 1 time
(gdb) l
1 int* madd(int *res)
2 {
3     int y = 3, a[100], b[100];
4     for (int i = 0; i < 100; i++)
5     {
6         a[i] = i - 50;
7         b[i] = i + 3;
8         res[i] = a[i] + b[i];
9     }
10 }
(gdb) run
Starting program: /students/202211073/work3_for_debug
Breakpoint 3, madd (res=0x7fffffffe920) at work3.c:10
10     res[i] = a[i] + b[i];
(gdb) disassemble
Dump of assembler code for function madd:
0x00005555555515b <+1>: push %rbp
0x00005555555515c <+2>: mov %rsp,%rbp
0x00005555555515d <+3>: sub $0x20,%rsp
0x00005555555515e <+4>: mov %rdi,-0x348(%rbp)
0x00005555555515f <+5>: mov %fs:0x28,%rax
0x000055555555160 <+6>: mov %rax,-0x8(%rbp)
0x000055555555161 <+7>: xor %eax,%eax
0x000055555555162 <+8>: movl $0x3,-0x334(%rbp)
0x000055555555163 <+9>: movl $0x0,-0x338(%rbp)
0x000055555555164 <+10>: jmp 0x55555555204 <madd+170>
0x000055555555165 <+11>: mov -0x338(%rbp),%eax
0x000055555555166 <+12>: lea -0x32(%rax),%edx
0x000055555555167 <+13>: mov -0x338(%rbp),%eax
0x000055555555168 <+14>: cltq
0x000055555555169 <+15>: mov %edx,-0x330(%rbp,%rax,4)
0x00005555555516a <+16>: mov -0x338(%rbp),%eax
0x00005555555516b <+17>: lea 0x3(%rax),%edx
0x00005555555516c <+18>: mov -0x338(%rbp),%eax
0x00005555555516d <+19>: cltq
0x00005555555516e <+20>: mov -0x1a0(%rbp,%rax,4),%edx
0x00005555555516f <+21>: mov -0x338(%rbp),%eax
0x000055555555170 <+22>: cltq
0x000055555555171 <+23>: lea 0x0(,%rax,4),%rsi
0x000055555555172 <+24>: mov -0x348(%rbp),%rax
0x000055555555173 <+25>: add %rsi,%rax
0x000055555555174 <+26>: add %ecx,%edx
0x000055555555175 <+27>: mov %edx,(%rax)
0x000055555555176 <+28>: addl $0x1,-0x338(%rbp)
0x000055555555177 <+29>: cmpl $0x63,-0x338(%rbp)
0x000055555555178 <+30>: jle 0x55555555191 <madd+55>
0x000055555555179 <+31>: mov -0x348(%rbp),%rax
0x00005555555517a <+32>: mov -0x8(%rbp),%rdi
0x00005555555517b <+33>: mov %edx,-0x330(%rbp,%rax,4)
0x00005555555517c <+34>: mov -0x338(%rbp),%eax
0x00005555555517d <+35>: cltq
0x00005555555517e <+36>: lea 0x0(,%rax,4),%rsi
0x00005555555517f <+37>: mov -0x348(%rbp),%rax
0x000055555555180 <+38>: add %rsi,%rax
0x000055555555181 <+39>: add %ecx,%edx
0x000055555555182 <+40>: mov %edx,(%rax)
0x000055555555183 <+41>: addl $0x1,-0x338(%rbp)
0x000055555555184 <+42>: cmpl $0x63,-0x338(%rbp)
0x000055555555185 <+43>: jle 0x55555555191 <madd+55>
0x000055555555186 <+44>: mov -0x348(%rbp),%rax
0x000055555555187 <+45>: mov -0x8(%rbp),%rdi
0x000055555555188 <+46>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555189 <+47>: mov -0x338(%rbp),%eax
0x00005555555518a <+48>: cltq
0x00005555555518b <+49>: lea 0x0(,%rax,4),%rsi
0x00005555555518c <+50>: mov -0x348(%rbp),%rax
0x00005555555518d <+51>: add %rsi,%rax
0x00005555555518e <+52>: add %ecx,%edx
0x00005555555518f <+53>: mov %edx,(%rax)
0x000055555555190 <+54>: addl $0x1,-0x338(%rbp)
0x000055555555191 <+55>: cmpl $0x63,-0x338(%rbp)
0x000055555555192 <+56>: jle 0x55555555191 <madd+55>
0x000055555555193 <+57>: mov -0x348(%rbp),%rax
0x000055555555194 <+58>: mov -0x8(%rbp),%rdi
0x000055555555195 <+59>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555196 <+60>: mov -0x338(%rbp),%eax
0x000055555555197 <+61>: cltq
0x000055555555198 <+62>: lea 0x0(,%rax,4),%rsi
0x000055555555199 <+63>: mov -0x348(%rbp),%rax
0x00005555555519a <+64>: add %rsi,%rax
0x00005555555519b <+65>: add %ecx,%edx
0x00005555555519c <+66>: mov %edx,(%rax)
0x00005555555519d <+67>: addl $0x1,-0x338(%rbp)
0x00005555555519e <+68>: cmpl $0x63,-0x338(%rbp)
0x00005555555519f <+69>: jle 0x55555555191 <madd+55>
0x0000555555551a0 <+70>: mov -0x348(%rbp),%rax
0x0000555555551a1 <+71>: mov -0x8(%rbp),%rdi
0x0000555555551a2 <+72>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551a3 <+73>: mov -0x338(%rbp),%eax
0x0000555555551a4 <+74>: cltq
0x0000555555551a5 <+75>: lea 0x0(,%rax,4),%rsi
0x0000555555551a6 <+76>: mov -0x348(%rbp),%rax
0x0000555555551a7 <+77>: add %rsi,%rax
0x0000555555551a8 <+78>: add %ecx,%edx
0x0000555555551a9 <+79>: mov %edx,(%rax)
0x0000555555551aa <+80>: addl $0x1,-0x338(%rbp)
0x0000555555551ab <+81>: cmpl $0x63,-0x338(%rbp)
0x0000555555551ac <+82>: jle 0x55555555191 <madd+55>
0x0000555555551ad <+83>: mov -0x348(%rbp),%rax
0x0000555555551ae <+84>: mov -0x8(%rbp),%rdi
0x0000555555551af <+85>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551b0 <+86>: mov -0x338(%rbp),%eax
0x0000555555551b1 <+87>: cltq
0x0000555555551b2 <+88>: lea 0x0(,%rax,4),%rsi
0x0000555555551b3 <+89>: mov -0x348(%rbp),%rax
0x0000555555551b4 <+90>: add %rsi,%rax
0x0000555555551b5 <+91>: add %ecx,%edx
0x0000555555551b6 <+92>: mov %edx,(%rax)
0x0000555555551b7 <+93>: addl $0x1,-0x338(%rbp)
0x0000555555551b8 <+94>: cmpl $0x63,-0x338(%rbp)
0x0000555555551b9 <+95>: jle 0x55555555191 <madd+55>
0x0000555555551ba <+96>: mov -0x348(%rbp),%rax
0x0000555555551bb <+97>: mov -0x8(%rbp),%rdi
0x0000555555551bc <+98>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551bd <+99>: mov -0x338(%rbp),%eax
0x0000555555551be <+100>: cltq
0x0000555555551bf <+101>: lea 0x0(,%rax,4),%rsi
0x0000555555551c0 <+102>: mov -0x348(%rbp),%rax
0x0000555555551c1 <+103>: add %rsi,%rax
0x0000555555551c2 <+104>: add %ecx,%edx
0x0000555555551c3 <+105>: mov %edx,(%rax)
0x0000555555551c4 <+106>: addl $0x1,-0x338(%rbp)
0x0000555555551c5 <+107>: cmpl $0x63,-0x338(%rbp)
0x0000555555551c6 <+108>: jle 0x55555555191 <madd+55>
0x0000555555551c7 <+109>: mov -0x348(%rbp),%rax
0x0000555555551c8 <+110>: mov -0x8(%rbp),%rdi
0x0000555555551c9 <+111>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551ca <+112>: mov -0x338(%rbp),%eax
0x0000555555551cb <+113>: cltq
0x0000555555551cc <+114>: lea 0x0(,%rax,4),%rsi
0x0000555555551cd <+115>: mov -0x348(%rbp),%rax
0x0000555555551ce <+116>: add %rsi,%rax
0x0000555555551cf <+117>: add %ecx,%edx
0x0000555555551d0 <+118>: mov %edx,(%rax)
0x0000555555551d1 <+119>: addl $0x1,-0x338(%rbp)
0x0000555555551d2 <+120>: cmpl $0x63,-0x338(%rbp)
0x0000555555551d3 <+121>: jle 0x55555555191 <madd+55>
0x0000555555551d4 <+122>: mov -0x348(%rbp),%rax
0x0000555555551d5 <+123>: mov -0x8(%rbp),%rdi
0x0000555555551d6 <+124>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551d7 <+125>: mov -0x338(%rbp),%eax
0x0000555555551d8 <+126>: cltq
0x0000555555551d9 <+127>: lea 0x0(,%rax,4),%rsi
0x0000555555551da <+128>: mov -0x348(%rbp),%rax
0x0000555555551db <+129>: add %rsi,%rax
0x0000555555551dc <+130>: add %ecx,%edx
0x0000555555551dd <+131>: mov %edx,(%rax)
0x0000555555551de <+132>: addl $0x1,-0x338(%rbp)
0x0000555555551df <+133>: cmpl $0x63,-0x338(%rbp)
0x0000555555551e0 <+134>: jle 0x55555555191 <madd+55>
0x0000555555551e1 <+135>: mov -0x348(%rbp),%rax
0x0000555555551e2 <+136>: mov -0x8(%rbp),%rdi
0x0000555555551e3 <+137>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551e4 <+138>: mov -0x338(%rbp),%eax
0x0000555555551e5 <+139>: cltq
0x0000555555551e6 <+140>: lea 0x0(,%rax,4),%rsi
0x0000555555551e7 <+141>: mov -0x348(%rbp),%rax
0x0000555555551e8 <+142>: add %rsi,%rax
0x0000555555551e9 <+143>: add %ecx,%edx
0x0000555555551ea <+144>: mov %edx,(%rax)
0x0000555555551eb <+145>: addl $0x1,-0x338(%rbp)
0x0000555555551ec <+146>: cmpl $0x63,-0x338(%rbp)
0x0000555555551ed <+147>: jle 0x55555555191 <madd+55>
0x0000555555551ee <+148>: mov -0x348(%rbp),%rax
0x0000555555551ef <+149>: mov -0x8(%rbp),%rdi
0x0000555555551f0 <+150>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551f1 <+151>: mov -0x338(%rbp),%eax
0x0000555555551f2 <+152>: cltq
0x0000555555551f3 <+153>: lea 0x0(,%rax,4),%rsi
0x0000555555551f4 <+154>: mov -0x348(%rbp),%rax
0x0000555555551f5 <+155>: add %rsi,%rax
0x0000555555551f6 <+156>: add %ecx,%edx
0x0000555555551f7 <+157>: mov %edx,(%rax)
0x0000555555551f8 <+158>: addl $0x1,-0x338(%rbp)
0x0000555555551f9 <+159>: cmpl $0x63,-0x338(%rbp)
0x0000555555551fa <+160>: jle 0x55555555191 <madd+55>
0x0000555555551fb <+161>: mov -0x348(%rbp),%rax
0x0000555555551fc <+162>: mov -0x8(%rbp),%rdi
0x0000555555551fd <+163>: mov %edx,-0x330(%rbp,%rax,4)
0x0000555555551fe <+164>: mov -0x338(%rbp),%eax
0x0000555555551ff <+165>: cltq
0x000055555555200 <+166>: lea 0x0(,%rax,4),%rsi
0x000055555555201 <+167>: mov -0x348(%rbp),%rax
0x000055555555202 <+168>: add %rsi,%rax
0x000055555555203 <+169>: add %ecx,%edx
0x000055555555204 <+170>: mov %edx,(%rax)
0x000055555555205 <+171>: addl $0x1,-0x338(%rbp)
0x000055555555206 <+172>: cmpl $0x63,-0x338(%rbp)
0x000055555555207 <+173>: jle 0x55555555191 <madd+55>
0x000055555555208 <+174>: mov -0x348(%rbp),%rax
0x000055555555209 <+175>: mov -0x8(%rbp),%rdi
0x00005555555520a <+176>: mov %edx,-0x330(%rbp,%rax,4)
0x00005555555520b <+177>: mov -0x338(%rbp),%eax
0x00005555555520c <+178>: cltq
0x00005555555520d <+179>: lea 0x0(,%rax,4),%rsi
0x00005555555520e <+180>: mov -0x348(%rbp),%rax
0x00005555555520f <+181>: add %rsi,%rax
0x000055555555210 <+182>: add %ecx,%edx
0x000055555555211 <+183>: mov %edx,(%rax)
0x000055555555212 <+184>: addl $0x1,-0x338(%rbp)
0x000055555555213 <+185>: cmpl $0x63,-0x338(%rbp)
0x000055555555214 <+186>: jle 0x55555555191 <madd+55>
0x000055555555215 <+187>: mov -0x348(%rbp),%rax
0x000055555555216 <+188>: mov -0x8(%rbp),%rdi
0x000055555555217 <+189>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555218 <+190>: mov -0x338(%rbp),%eax
0x000055555555219 <+191>: cltq
0x00005555555521a <+192>: lea 0x0(,%rax,4),%rsi
0x00005555555521b <+193>: mov -0x348(%rbp),%rax
0x00005555555521c <+194>: add %rsi,%rax
0x00005555555521d <+195>: add %ecx,%edx
0x00005555555521e <+196>: mov %edx,(%rax)
0x00005555555521f <+197>: addl $0x1,-0x338(%rbp)
0x000055555555220 <+198>: cmpl $0x63,-0x338(%rbp)
0x000055555555221 <+199>: jle 0x55555555191 <madd+55>
0x000055555555222 <+200>: mov -0x348(%rbp),%rax
0x000055555555223 <+201>: mov -0x8(%rbp),%rdi
0x000055555555224 <+202>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555225 <+203>: mov -0x338(%rbp),%eax
0x000055555555226 <+204>: cltq
0x000055555555227 <+205>: lea 0x0(,%rax,4),%rsi
0x000055555555228 <+206>: mov -0x348(%rbp),%rax
0x000055555555229 <+207>: add %rsi,%rax
0x00005555555522a <+208>: add %ecx,%edx
0x00005555555522b <+209>: mov %edx,(%rax)
0x00005555555522c <+210>: addl $0x1,-0x338(%rbp)
0x00005555555522d <+211>: cmpl $0x63,-0x338(%rbp)
0x00005555555522e <+212>: jle 0x55555555191 <madd+55>
0x00005555555522f <+213>: mov -0x348(%rbp),%rax
0x000055555555230 <+214>: mov -0x8(%rbp),%rdi
0x000055555555231 <+215>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555232 <+216>: mov -0x338(%rbp),%eax
0x000055555555233 <+217>: cltq
0x000055555555234 <+218>: lea 0x0(,%rax,4),%rsi
0x000055555555235 <+219>: mov -0x348(%rbp),%rax
0x000055555555236 <+220>: add %rsi,%rax
0x000055555555237 <+221>: add %ecx,%edx
0x000055555555238 <+222>: mov %edx,(%rax)
0x000055555555239 <+223>: addl $0x1,-0x338(%rbp)
0x00005555555523a <+224>: cmpl $0x63,-0x338(%rbp)
0x00005555555523b <+225>: jle 0x55555555191 <madd+55>
0x00005555555523c <+226>: mov -0x348(%rbp),%rax
0x00005555555523d <+227>: mov -0x8(%rbp),%rdi
0x00005555555523e <+228>: mov %edx,-0x330(%rbp,%rax,4)
0x00005555555523f <+229>: mov -0x338(%rbp),%eax
0x000055555555240 <+230>: cltq
0x000055555555241 <+231>: lea 0x0(,%rax,4),%rsi
0x000055555555242 <+232>: mov -0x348(%rbp),%rax
0x000055555555243 <+233>: add %rsi,%rax
0x000055555555244 <+234>: add %ecx,%edx
0x000055555555245 <+235>: mov %edx,(%rax)
0x000055555555246 <+236>: addl $0x1,-0x338(%rbp)
0x000055555555247 <+237>: cmpl $0x63,-0x338(%rbp)
0x000055555555248 <+238>: jle 0x55555555191 <madd+55>
0x000055555555249 <+239>: mov -0x348(%rbp),%rax
0x00005555555524a <+240>: mov -0x8(%rbp),%rdi
0x00005555555524b <+241>: mov %edx,-0x330(%rbp,%rax,4)
0x00005555555524c <+242>: mov -0x338(%rbp),%eax
0x00005555555524d <+243>: cltq
0x00005555555524e <+244>: lea 0x0(,%rax,4),%rsi
0x00005555555524f <+245>: mov -0x348(%rbp),%rax
0x000055555555250 <+246>: add %rsi,%rax
0x000055555555251 <+247>: add %ecx,%edx
0x000055555555252 <+248>: mov %edx,(%rax)
0x000055555555253 <+249>: addl $0x1,-0x338(%rbp)
0x000055555555254 <+250>: cmpl $0x63,-0x338(%rbp)
0x000055555555255 <+251>: jle 0x55555555191 <madd+55>
0x000055555555256 <+252>: mov -0x348(%rbp),%rax
0x000055555555257 <+253>: mov -0x8(%rbp),%rdi
0x000055555555258 <+254>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555259 <+255>: mov -0x338(%rbp),%eax
0x00005555555525a <+256>: cltq
0x00005555555525b <+257>: lea 0x0(,%rax,4),%rsi
0x00005555555525c <+258>: mov -0x348(%rbp),%rax
0x00005555555525d <+259>: add %rsi,%rax
0x00005555555525e <+260>: add %ecx,%edx
0x00005555555525f <+261>: mov %edx,(%rax)
0x000055555555260 <+262>: addl $0x1,-0x338(%rbp)
0x000055555555261 <+263>: cmpl $0x63,-0x338(%rbp)
0x000055555555262 <+264>: jle 0x55555555191 <madd+55>
0x000055555555263 <+265>: mov -0x348(%rbp),%rax
0x000055555555264 <+266>: mov -0x8(%rbp),%rdi
0x000055555555265 <+267>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555266 <+268>: mov -0x338(%rbp),%eax
0x000055555555267 <+269>: cltq
0x000055555555268 <+270>: lea 0x0(,%rax,4),%rsi
0x000055555555269 <+271>: mov -0x348(%rbp),%rax
0x00005555555526a <+272>: add %rsi,%rax
0x00005555555526b <+273>: add %ecx,%edx
0x00005555555526c <+274>: mov %edx,(%rax)
0x00005555555526d <+275>: addl $0x1,-0x338(%rbp)
0x00005555555526e <+276>: cmpl $0x63,-0x338(%rbp)
0x00005555555526f <+277>: jle 0x55555555191 <madd+55>
0x000055555555270 <+278>: mov -0x348(%rbp),%rax
0x000055555555271 <+279>: mov -0x8(%rbp),%rdi
0x000055555555272 <+280>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555273 <+281>: mov -0x338(%rbp),%eax
0x000055555555274 <+282>: cltq
0x000055555555275 <+283>: lea 0x0(,%rax,4),%rsi
0x000055555555276 <+284>: mov -0x348(%rbp),%rax
0x000055555555277 <+285>: add %rsi,%rax
0x000055555555278 <+286>: add %ecx,%edx
0x000055555555279 <+287>: mov %edx,(%rax)
0x00005555555527a <+288>: addl $0x1,-0x338(%rbp)
0x00005555555527b <+289>: cmpl $0x63,-0x338(%rbp)
0x00005555555527c <+290>: jle 0x55555555191 <madd+55>
0x00005555555527d <+291>: mov -0x348(%rbp),%rax
0x00005555555527e <+292>: mov -0x8(%rbp),%rdi
0x00005555555527f <+293>: mov %edx,-0x330(%rbp,%rax,4)
0x000055555555280 <+294>: mov -0x338(%rbp),%eax
0x000055555555281 <+295>: cltq
0x000055555555282 <+296>: lea 0x0(,%rax,4),%rsi
0x000055555555283 <+297>: mov -0x348(%rbp),%rax
0x000055555555284 <+298>: add %rsi,%rax
0x000055555555285 <+299>: add %ecx,%edx
0x000055555555286 <+300
```

4、重新启动 gdb，调试该程序。输入命令 `break 10 if i==3`（学号百位、个位 xy 为 03），使得程序在进行 `i=3` 的循环时停下；

交替使用 `ni`、`info reg` 命令，显示 `edx`、`ecx` 在 `res[i] = a[i]+b[i]` 执行前后储存的值。

```
(gdb) r ttle work3_for_debug
Reading symbols from work3_for_debug...
(gdb) l
2
3     int* madd(int *res)
4     {
5         int y = 3, a[100], b[100];
6         for (int i = 0; i < 100; i++)
7         {
8             a[i] = i - 50;
9             b[i] = i + 3;
10            res[i] = a[i] + b[i];
11        }
(gdb) info break
No breakpoints or watchpoints.
(gdb) break 10 if i==3
Breakpoint 1 at 0x11c1: file work3.c, line 10.
(gdb) run
Starting program: /students/2022211073/work3_for_debug
Breakpoint 1, madd (res=0x7fffffffe920) at work3.c:10
10    res[i] = a[i] + b[i];
```

图 18：设置断点

```
(gdb) i r edx
edx      0x6      6
(gdb) i r ecx
ecx      0xffffffff0      -48
(gdb) ni
0x000055555555551c7      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551c9      10      res[i] = a[i] + b[i];
(gdb) print a[i]
$1 = -47
(gdb) i r edx
edx      0x6      6
(gdb) i r ecx
ecx      0xffffffff0      -48
(gdb) ni
0x000055555555551d0      10      res[i] = a[i] + b[i];
(gdb) i r ecx
ecx      0xffffffff1      -47
(gdb) ni
0x000055555555551d6      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551d8      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551df      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551e5      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551e7      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551ef      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551f6      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551f9      10      res[i] = a[i] + b[i];
(gdb) ni
0x000055555555551fb      10      res[i] = a[i] + b[i];
(gdb) ni
9      for (int i = 0; i < 100; i++)
(gdb) print res[i]
$2 = -41
(gdb) i r edx
edx      0xffffffff7      -41
```

图 19、20：edx、ecx 寄存器情况

实验内容四

1、在 vi 编辑器中分别编辑 `time.c` 和 `space.c` 两个文件，内容分别为高计算量类型的冒泡排序算法，高内存需求类型的归并排序算法；

```
#include <stdio.h>
#include <time.h>

void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    const int SIZE = 10000; // 设置数组的大小为10000
    int arr[SIZE];

    // 初始化待排序数组，这里使用逆序的方式生成
    for (int i = 0; i < SIZE; i++) {
        arr[i] = SIZE - i;
    }

    clock_t start_time = clock();
    bubbleSort(arr, SIZE);
    clock_t end_time = clock();
    double execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Sorted array of size %d in %f seconds.\n", SIZE, execution_time);
    return 0;
}
```

图 21：实现冒泡排序的 `time.c`

```

#include <stdio.h>
#include <time.h>

void merge(int arr[], int left, int middle, int right)
{
    int i, j, k, leftArr[n1], rightArr[n2];
    int n1 = middle - left + 1, n2 = right - middle;

    for (i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        rightArr[j] = arr[middle + 1 + j];

    i = 0, j = 0, k = left;
    while (i < n1 && j < n2)
    {
        if (leftArr[i] <= rightArr[j])
            arr[k] = leftArr[i], i++;
        else
            arr[k] = rightArr[j], j++;
        k++;
    }
    while (i < n1)
        arr[k] = leftArr[i], i++, k++;
    while (j < n2)
        arr[k] = rightArr[j], j++, k++;
    return;
}

void mergeSort(int arr[], int left, int right)
{
    if (left < right)
    {
        int middle = left + (right - left) / 2;
        mergeSort(arr, left, middle);
        mergeSort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
}

int main()
{
    const int SIZE = 10000; // 数组大小
    int arr[SIZE];

    for (int i = 0; i < SIZE; i++)
        arr[i] = SIZE - i;

    clock_t start_time = clock();
    mergeSort(arr, 0, SIZE - 1);
    clock_t end_time = clock();
    double execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Sorted array of size %d in %f seconds.\n", SIZE, execution_time);
    return 0;
}

```

图 22、23: 实现归并排序的 space.c

2、gcc 编译，设置不同优化参数：

```

2022211073@bupt1:~$ gcc time.c -o time_for_o
2022211073@bupt1:~$ gcc -O0 time.c -o time_for_o0
2022211073@bupt1:~$ gcc -O1 time.c -o time_for_o1
2022211073@bupt1:~$ gcc -O2 time.c -o time_for_o2
2022211073@bupt1:~$ gcc -O3 time.c -o time_for_o3
2022211073@bupt1:~$ gcc -Os time.c -o time_for_os
2022211073@bupt1:~$ gcc space.c -o space_for_o
2022211073@bupt1:~$ gcc -O0 space.c -o space_for_o0
2022211073@bupt1:~$ gcc -O1 space.c -o space_for_o1
2022211073@bupt1:~$ gcc -O2 space.c -o space_for_o2
2022211073@bupt1:~$ gcc -O3 space.c -o space_for_o3
2022211073@bupt1:~$ gcc -Os space.c -o space_for_os

```

图 24: gcc 编译

3、分析两种算法的运行效率

```

2022211073@bupt1:~$ time ./space_for_o
Sorted array of size 100000 in 0.019422 seconds.

real    0m0.022s
user    0m0.022s
sys     0m0.000s
2022211073@bupt1:~$ time ./space_for_o0
Sorted array of size 100000 in 0.020742 seconds.

real    0m0.023s
user    0m0.023s
sys     0m0.000s
2022211073@bupt1:~$ time ./space_for_o1
Sorted array of size 100000 in 0.008582 seconds.

real    0m0.010s
user    0m0.010s
sys     0m0.000s
2022211073@bupt1:~$ time ./space_for_o2
Sorted array of size 100000 in 0.007890 seconds.

real    0m0.010s
user    0m0.006s
sys     0m0.003s
2022211073@bupt1:~$ time ./space_for_o3
Sorted array of size 100000 in 0.007113 seconds.

real    0m0.009s
user    0m0.009s
sys     0m0.000s
2022211073@bupt1:~$ time ./space_for_os
Sorted array of size 100000 in 0.009824 seconds.

real    0m0.012s
user    0m0.008s
sys     0m0.004s

```

图 25: 冒泡排序算法性能

```

2022211073@bupt1:~$ time ./time_for_o
Sorted array of size 10000 in 0.234826 seconds.

real    0m0.236s
user    0m0.236s
sys     0m0.000s
2022211073@bupt1:~$ time ./time_for_o0
Sorted array of size 10000 in 0.236356 seconds.

real    0m0.238s
user    0m0.238s
sys     0m0.000s
2022211073@bupt1:~$ time ./time_for_o1
Sorted array of size 10000 in 0.090488 seconds.

real    0m0.092s
user    0m0.088s
sys     0m0.004s
2022211073@bupt1:~$ time ./time_for_o2
Sorted array of size 10000 in 0.092556 seconds.

real    0m0.094s
user    0m0.094s
sys     0m0.000s
2022211073@bupt1:~$ time ./time_for_o3
Sorted array of size 10000 in 0.090446 seconds.

real    0m0.092s
user    0m0.088s
sys     0m0.004s
2022211073@bupt1:~$ time ./time_for_os
Sorted array of size 10000 in 0.092592 seconds.

real    0m0.094s
user    0m0.094s
sys     0m0.000s

```

图 26: 归并排序算法性能

五、实验分析

1、实验内容一

gcc 编译时采取不同的优化参数，我猜想这会影响到程序的性能，通过运行时间的长短体现。对比时间后，我发现未优化和优化参数为 O0 编译生成的文件性能基本相同，时间基本为 1s 左右；而 O1、O2、O3、Os 编译生成的文件明显性能更佳，时间基本为 0.5s，变为原来的 50%。

2、实验内容二

实验二基本沿用了实验一的方法，将两种算法分别采用不同的优化参数进行编译，再比较其时间，发现两种算法在相同优化参数下的运行时间基本相同，性能基本相同；两种算法在遇到-O2、-O3 参数之后性能都明显提高，时间大幅缩短。

3、实验内容三

为找到 $a[i]+b[i]$ 语句的汇编指令，我在语句处设置断点，输入 run 命令后在该处停下，并查看汇编指令。ni 一直执行到下一行 c 语言代码时，重新查看汇编指令，两次查看中间的汇编指令即为 $a[i]+b[i]$ 语句所对应的指令。

为找到 $a[i]$ 、 $b[i]$ 相应的寄存器，我需要分别 print 它们的值，并通过 ir 指令查看汇编代码前后各个寄存器的值。当寄存器的值改变为 $a[i]$ 、 $b[i]$ 相应的值时，说明该寄存器是用于储存这个变量的。

显示寄存器在 $res[i] = a[i]+b[i]$ 执行前后储存的值，我的实验思路与上文提到的思路基本相似。

4、实验内容四

实验四查看性能基本沿用了实验二的方法，将两种算法分别采用不同的优化参数进行编译，再比较其时间。唯一不同的点在于我在程序中指定位置使用了 clock_t，来更精确地计算程序中特定语句的运行时间，同时方便后期比对。

对于高计算量类型的冒泡排序算法，正常编译与优化系数为-O0 编译的两种情况，运行效率相差不大，都为 0.020s 左右。当优化系数为-O2、-O3、-Os 时，运行时间骤减，运行效率增加，在 0.007s 上下浮动，说明这三种优化参数大大提高了运行效率。

对于高内存需求类型的归并排序算法，和冒泡排序的数据不同，但是规律是基本相同的。

综上，gcc 编译时更换不同的优化参数，对于高计算量类型算法和高内存需求算法均有不同程度的运行效率的提高。

六、实验总结

七、诚信声明

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

此外，我还参考了以下资料：

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照 0 分计。

(签名)