北京郵電大學

实验报告

题目：熟悉 Linux 系统及其相关软件环境

班　　级：2020211301

学　　号：2020211221

姓　　名：翁岳川

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（10 分）

简述使用的工具

1. SecureCRT 或其他远程登陆工具（服务器：10.120.11.12）
2. Linux
3. Gcc 编译器
4. GDB 调试工具
5. Objdump 命令反汇编

三、实验概况

简述实验内容和基本设想

实验内容一（15 分）

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（15 分）

在 linux 环境下，分别打印输出如下算法所需时间

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

分别设置不同优化参数，给出运行时间

实验内容三（30 分）

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（不使用优化选项），

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

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 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）：

实验内容四（加分项，20 分）

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（60-80 分）

操作步骤+运行截图

实验内容一（15 分）

1. 源程序

```c
#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;
                }
~
~
```

2. 采用-o 和 -O 进行编译比较运行时间

```
2020211221@bupt1:~/firsttest$ gcc a.c -o atest10
2020211221@bupt1:~/firsttest$ gcc -O a.c -o atest11
2020211221@bupt1:~/firsttest$ time ./atest10
Result is 400002019.000000

real    0m1.410s
user    0m1.406s
sys     0m0.004s
2020211221@bupt1:~/firsttest$ time ./atest11
Result is 400002019.000000

real    0m0.544s
user    0m0.544s
sys     0m0.000s
```

3. 采用 gdb 进行调试，将程序运行到 for 循环停止

```
2020211221@bupt1:~/firsttest$ gdb -q test00
Reading symbols from test00...
(gdb) list
1       #include<stdio.h>
2       int main(void){
3               double counter;
4               double result;
5               double temp;
6               for(counter=0;counter<2000.0*2000.0*2000.0/20.0+2020;counter+=(5-1)/4){
7                       temp=counter/1979;
8                       result=counter;
9               }
10              printf("Result is %lf\n",result);
(gdb)
11              return 0;
12      }
(gdb) b 6
Breakpoint 1 at 0x1155: file a.c, line 6.
(gdb) run
Starting program: /students/2020211221/firsttest/test00

Breakpoint 1, main () at a.c:6
6               for(counter=0;counter<2000.0*2000.0*2000.0/20.0+2020;counter+=(5-1)/4){
```

4. 使用 objdump 生成汇编程序（main 函数的汇编程序截图）

```
#include<stdio.h>
int main(void){
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       48 83 ec 20             sub    $0x20,%rsp
        double counter;
        double result;
        double temp;
        for(counter=0;counter<2000.0*2000.0*2000.0/20.0+2020;counter+=(5-1)/4){
    1155:       66 0f ef c0             pxor   %xmm0,%xmm0
    1159:       f2 0f 11 45 e8          movsd  %xmm0,-0x18(%rbp)
    115e:       eb 36                   jmp    1196 <main+0x4d>
                temp=counter/1979;
    1160:       f2 0f 10 45 e8          movsd  -0x18(%rbp),%xmm0
    1165:       f2 0f 10 0d ab 0e 00    movsd  0xeab(%rip),%xmm1        # 2018 <_IO_stdin_used+0x18>
    116c:       00
    116d:       f2 0f 5e c1             divsd  %xmm1,%xmm0
    1171:       f2 0f 11 45 f8          movsd  %xmm0,-0x8(%rbp)
                result=counter;
    1176:       f2 0f 10 45 e8          movsd  -0x18(%rbp),%xmm0
    117b:       f2 0f 11 45 f0          movsd  %xmm0,-0x10(%rbp)
        for(counter=0;counter<2000.0*2000.0*2000.0/20.0+2020;counter+=(5-1)/4){
    1180:       f2 0f 10 4d e8          movsd  -0x18(%rbp),%xmm1
    1185:       f2 0f 10 05 93 0e 00    movsd  0xe93(%rip),%xmm0        # 2020 <_IO_stdin_used+0x20>
    118c:       00
    118d:       f2 0f 58 c1             addsd  %xmm1,%xmm0
    1191:       f2 0f 11 45 e8          movsd  %xmm0,-0x18(%rbp)
    1196:       f2 0f 10 05 8a 0e 00    movsd  0xe8a(%rip),%xmm0        # 2028 <_IO_stdin_used+0x28>
    119d:       00
    119e:       66 0f 2f 45 e8          comisd -0x18(%rbp),%xmm0
    11a3:       77 bb                   ja     1160 <main+0x17>
        }
        printf("Result is %lf\n",result);
    11a5:       48 8b 45 f0             mov    -0x10(%rbp),%rax
    11a9:       66 48 0f 6e c0          movq   %rax,%xmm0
    11ae:       48 8d 3d 53 0e 00 00    lea    0xe53(%rip),%rdi        # 2008 <_IO_stdin_used+0x8>
    11b5:       b8 01 00 00 00          mov    $0x1,%eax
    11ba:       e8 91 fe ff ff          callq  1050 <printf@plt>
        return 0;
    11bf:       b8 00 00 00 00          mov    $0x0,%eax
}
```

实验内容二（15 分）

1. 源代码

```
#include<stdio.h>
#include<time.h>
int src[2048][2048];
int dst[2048][2048];
int main(void){
          int p=0,q=0;
              for(int i=0;i<2048;i++)//将两者赋值相同
                              for(int j=0;j<2048;j++){
                                              src[i][j]=1;
                                                              dst[i][j]=1;
                                                                      }
                              for(p=0;p<2048;p++)//算法一
                                              for(q=0;q<2048;q++) {
                                                                      dst[p][q]=src[p][q];
                                                                              }
                                                              return 0;
}
~
```

```c
#include<stdio.h>
#include<string.h>
int src[2048][2048];
int dst[2048][2048];

int main(void){
        int p=0,q=0;
        for(int i=0;i<2048;i++){
                for(int j=0;j<2048;j++){
                        src[i][j]=1;
                        dst[i][j]=1;
                }
                for(p=0;p<2048;p++){
                        for(q=0;q<2048;q++){
                        dst[q][p]=src[q][p];
                        }
                }
        }
        return 0;
}
```

2. 算法运行时间比较（采用-o 进行操作）

算法 1:

```
2020211221@bupt1:~/firsttest$ gcc b.c -o btest10
2020211221@bupt1:~/firsttest$ time ./btest10

real    0m0.051s
user    0m0.030s
sys     0m0.021s
```

算法 2:

```
2020211221@bupt1:~/firsttest$ time ./etest01

real    0m0.204s
user    0m0.192s
sys     0m0.012s
```

3. 采用不同参数进行编译

算法 1：

①采用-o 进行编译

```
2020211221@bupt1:~/firsttest$ gcc b.c -o btest01
2020211221@bupt1:~/firsttest$ time ./btest01

real    0m0.050s
user    0m0.042s
sys     0m0.008s
```

②采用-O 进行编译

```
2020211221@bupt1:~/firsttest$ gcc  -O b.c -o btest02
2020211221@bupt1:~/firsttest$ time ./btest02

real    0m0.033s
user    0m0.021s
sys     0m0.012s
```

③采用-O2 进行编译

```
2020211221@bupt1:~/firsttest$ gcc  -O2 b.c -o btest03
2020211221@bupt1:~/firsttest$ time ./btest03

real    0m0.032s
user    0m0.008s
sys     0m0.024s
```

④采用-O3 进行编译

```
2020211221@bupt1:~/firsttest$ gcc  -O3 b.c -o btest04
2020211221@bupt1:~/firsttest$ time ./btest04

real    0m0.028s
user    0m0.008s
sys     0m0.020s
```

算法 2:

①采用-o 进行编译

```
2020211221@bupt1:~/firsttest$ gcc e.c -o btest01
2020211221@bupt1:~/firsttest$ time ./btest01

real    0m0.373s
user    0m0.340s
sys     0m0.032s
```

②采用-O 进行编译

```
2020211221@bupt1:~/firsttest$ gcc -O e.c -o btest02
2020211221@bupt1:~/firsttest$ time ./btest02

real    0m0.126s
user    0m0.106s
sys     0m0.020s
```

③采用-O2 进行编译

```
2020211221@bupt1:~/firsttest$ gcc -O2 e.c -o btest03
2020211221@bupt1:~/firsttest$ time ./btest03

real    0m0.123s
user    0m0.111s
sys     0m0.012s
```

④采用-O3 进行编译

```
2020211221@bupt1:~/firsttest$ gcc -O3 e.c -o btest04
2020211221@bupt1:~/firsttest$ time ./btest04

real    0m0.029s
user    0m0.008s
sys     0m0.021s
```

分析：随着编译优化参数的提高，算法的时间复杂度降低，其运行时间减少，但总体而言算法 1 明显优于算法 2，在最大优化程序（O3）下，两算法所用时间趋近于相同。

实验内容三（30 分）

1. 源代码

```c
#include<stdio.h>
void madd();
int main(void){
        madd();
        return 0;
}

void madd(){
        int a[100]={0},b[100]={0};
        int c[100]={0};
        int y=1;
        for(int i=0;i<100;i++){
                a[i]=i-50;
                b[i]=i+y;
                printf("%d ",a[i]+b[i]);
        }
        printf("\n");
}
```

2. gcc 进行编译并生成汇编程序

```
2020211221@bupt1:~/firsttest$ gcc -g -o objtest c.c
2020211221@bupt1:~/firsttest$ objdump -S objtest |more
```

3. madd 函数的汇编程序

```
00000000000011a2 <madd>:

void madd(){
    11a2:       f3 0f 1e fa             endbr64
    11a6:       55                      push   %rbp
    11a7:       48 89 e5                mov    %rsp,%rbp
    11aa:       48 81 ec d0 04 00 00    sub    $0x4d0,%rsp
    11b1:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    11b8:       00 00
    11ba:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    11be:       31 c0                   xor    %eax,%eax
        int a[100]={0},b[100]={0};
    11c0:       48 8d 95 40 fb ff ff    lea    -0x4c0(%rbp),%rdx
    11c7:       b8 00 00 00 00          mov    $0x0,%eax
    11cc:       b9 32 00 00 00          mov    $0x32,%ecx
    11d1:       48 89 d7                mov    %rdx,%rdi
    11d4:       f3 48 ab                rep stos %rax,%es:(%rdi)
    11d7:       48 8d 95 d0 fc ff ff    lea    -0x330(%rbp),%rdx
    11de:       b8 00 00 00 00          mov    $0x0,%eax
    11e3:       b9 32 00 00 00          mov    $0x32,%ecx
    11e8:       48 89 d7                mov    %rdx,%rdi
    11eb:       f3 48 ab                rep stos %rax,%es:(%rdi)
        int c[100]={0};
    11ee:       48 8d 95 60 fe ff ff    lea    -0x1a0(%rbp),%rdx
    11f5:       b8 00 00 00 00          mov    $0x0,%eax
    11fa:       b9 32 00 00 00          mov    $0x32,%ecx
    11ff:       48 89 d7                mov    %rdx,%rdi
    1202:       f3 48 ab                rep stos %rax,%es:(%rdi)
        int y=1;
    1205:       c7 85 3c fb ff ff 01    movl   $0x1,-0x4c4(%rbp)
    120c:       00 00 00
        for(int i=0;i<100;i++){
    120f:       c7 85 38 fb ff ff 00    movl   $0x0,-0x4c8(%rbp)
    1216:       00 00 00
    1219:       eb 6f                   jmp    128a <madd+0xe8>
                a[i]=i-50;
    121b:       8b 85 38 fb ff ff       mov    -0x4c8(%rbp),%eax
    1221:       8d 50 ce                lea    -0x32(%rax),%edx
    1224:       8b 85 38 fb ff ff       mov    -0x4c8(%rbp),%eax
    122a:       48 98                   cltq
    122c:       89 94 85 40 fb ff ff    mov    %edx,-0x4c0(%rbp,%rax
                b[i]=i+y;
    1233:       8b 95 38 fb ff ff       mov    -0x4c8(%rbp),%edx
    1239:       8b 85 3c fb ff ff       mov    -0x4c4(%rbp),%eax
    123f:       01 c2                   add    %eax,%edx
    1241:       8b 85 38 fb ff ff       mov    -0x4c8(%rbp),%eax
    1247:       48 98                   cltq
    1249:       89 94 85 d0 fc ff ff    mov    %edx,-0x330(%rbp,%rax
                printf("%d ",a[i]+b[i]);
    1250:       8b 85 38 fb ff ff       mov    -0x4c8(%rbp),%eax
    1256:       48 98                   cltq
    1258:       8b 94 85 40 fb ff ff    mov    -0x4c0(%rbp,%rax,4),%
    125f:       8b 85 38 fb ff ff       mov    -0x4c8(%rbp),%eax
    1265:       48 98                   cltq
    1267:       8b 84 85 d0 fc ff ff    mov    -0x330(%rbp,%rax,4),%
    126e:       01 d0                   add    %edx,%eax
    1270:       89 c6                   mov    %eax,%esi
    1272:       48 8d 3d 8b 0d 00 00    lea    0xd8b(%rip),%rdi
    1279:       b8 00 00 00 00          mov    $0x0,%eax
    127e:       e8 0d fe ff ff          callq  1090 <printf@plt>
        for(int i=0;i<100;i++){
    1283:       83 85 38 fb ff ff 01    addl   $0x1,-0x4c8(%rbp)
    128a:       83 bd 38 fb ff ff 63    cmpl   $0x63,-0x4c8(%rbp)
    1291:       7e 88                   jle    121b <madd+0x79>
        }
        printf("\n");
    1293:       bf 0a 00 00 00          mov    $0xa,%edi
    1298:       e8 d3 fd ff ff          callq  1070 <putchar@plt>
}
    129d:       90                      nop
    129e:       48 8b 45 f8             mov    -0x8(%rbp),%rax
    12a2:       64 48 33 04 25 28 00    xor    %fs:0x28,%rax
    12a9:       00 00
    12ab:       74 05                   je     12b2 <madd+0x110>
    12ad:       e8 ce fd ff ff          callq  1080 <__stack_chk_fai
    12b2:       c9                      leaveq
    12b3:       c3                      retq
    12b4:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
    12bb:       00 00 00
    12be:       66 90                   xchg   %ax,%ax
```

4. gdb 进行调试，gdb 指令的使用

①gdb

```
2020211221@bupt1:~/firsttest$ gcc -g c.c -o ctest01
2020211221@bupt1:~/firsttest$ gdb ctest01
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 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 ctest01...
```

②file

```
(gdb) file
Deleted breakpoint 1 A program is being debugged already.
Are you sure you want to change the file? (y or n) y
No executable file now.
Discard symbol table from `/students/2020211221/firsttest/ctest01'? (y or n) y
Error in re-setting breakpoint 3: No source file named /students/2020211221/firsttest/c.c.
Watchpoint 4 deleted because the program has left the block
in which its expression is valid.
Watchpoint 5 deleted because the program has left the block
in which its expression is valid.
Watchpoint 6 deleted because the program has left the block
in which its expression is valid.
No symbol file now.
```

③kill

```
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 418193) killed]
```

④quit

```
(gdb) quit
2020211221@bupt1:~/firsttest$
```

⑤break

```
(gdb) break 4
Breakpoint 1 at 0x1191: file c.c, line 4.
```

⑥delete

```
(gdb) delete
Delete all breakpoints? (y or n) y
```

⑦clear

```
(gdb) clear 4
```

⑧info break

```
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001191 in main at c.c:4
```

⑨run

```
(gdb) run
Starting program: /students/2020211221/firsttest/ctest01

Breakpoint 1, main () at c.c:4
4               madd();
```

⑩continue

```
Breakpoint 1, main () at c.c:4
4               madd();
(gdb) continue
Continuing.

Breakpoint 3, madd () at c.c:12
12              for(int i=0;i<100;i++){
```

⑪nexti

```
(gdb) nexti
0x0000555555555266      12              for(int i=0;i<100;i++){
```

⑫stepi

```
(gdb) stepi
0x000055555555526d      12              for(int i=0;i<100;i++){
```

⑬disassemble

```
(gdb) disassemble
Dump of assembler code for function main:
   0x0000555555555189 <+0>:     endbr64
   0x000055555555518d <+4>:     push   %rbp
   0x000055555555518e <+5>:     mov    %rsp,%rbp
=> 0x0000555555555191 <+8>:     mov    $0x0,%eax
   0x0000555555555196 <+13>:    callq  0x5555555551a2 <madd>
   0x000055555555519b <+18>:    mov    $0x0,%eax
   0x00005555555551a0 <+23>:    pop    %rbp
   0x00005555555551a1 <+24>:    retq
End of assembler dump.
```

⑭list

```
(gdb) list
1       #include<stdio.h>
2       void madd();
3       int main(void){
4               madd();
5               return 0;
6       }
7
8       void madd(){
9               int a[100],b[100];
10              int c[100];
(gdb)
11              int y=1;
12              for(int i=0;i<100;i++){
13                      a[i]=i-50;
14                      b[i]=i+y;
15                      c[i]=a[i]+b[i];
16                      printf("%d ",c[i]);
17              }
18              printf("\n");
19      }
(gdb)
Line number 20 out of range; c.c has 19 lines.
```

⑮print

```
(gdb) print a[i]+b[i]
$1 = -47
```

⑯x

```
(gdb) print&c
$3 = (int (*)[100]) 0x7fffffffe8e0
(gdb) print &c
$4 = (int (*)[100]) 0x7fffffffe8e0
(gdb) x 0x7fffffffe8e0
0x7fffffffe8e0: 0xffffffcf
```

⑰info reg

```
(gdb) info reg
rax            0x29                41
rbx            0x5555555552a0      93824992236192
rcx            0x0                 0
rdx            0xffffffff          4294967287
rsi            0x555555556006      93824992239622
rdi            0x7ffff7fbd4c0      140737353864384
rbp            0x7fffffffea80      0x7fffffffea80
rsp            0x7fffffffe5b0      0x7fffffffe5b0
r8             0x0                 0
r9             0x3                 3
r10            0x555555556006      93824992239622
r11            0x7ffffffe486       140737488348294
r12            0x5555555550a0      93824992235680
r13            0x7ffffffeb80       140737488350080
r14            0x0                 0
r15            0x0                 0
rip            0x5555555551f1      0x5555555551f1 <madd+79>
eflags         0x287               [ CF PF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

⑱watch

```
(gdb) watch c
Watchpoint 4: c
(gdb) n
15                  c[i]=a[i]+b[i];
(gdb) n

Watchpoint 4: c

Old value =
  {-49, -47, -45, -43, -41, -39, -37, -35, -33, -31, -29, -27, -25, -23, -21, -19, -17, -15, -13, -11, 0 <repeats 50 times>, -4164, 32767, 0 <repeats 12 times>, 1431650368, 21845, 1577523
1, 0, 194, 0, -5513, 32767, -5514, 32767, 1431655149, 21845, -134479928, 32767, 1431655072, 21845}
New value =
  {-49, -47, -45, -43, -41, -39, -37, -35, -33, -31, -29, -27, -25, -23, -21, -19, -17, -15, -13, -11, -9, 0 <repeats 49 times>, -4164, 32767, 0 <repeats 12 times>, 1431650368, 21845, 157
75231, 0, 194, 0, -5513, 32767, -5514, 32767, 1431655149, 21845, -134479928, 32767, 1431655072, 21845}
```

5. 找到 a[i]+b[i]对应的汇编指令，并说明 a[i]+b[i]存在的寄存器

①对应的汇编指令

```
            c[i]=a[i]+b[i];
  1219:     8b 85 c4 fc ff ff       mov    -0x33c(%rbp),%eax
  121f:     48 98                   cltq
  1221:     8b 94 85 d0 fc ff ff    mov    -0x330(%rbp,%rax,4),%edx
  1228:     8b 85 c4 fc ff ff       mov    -0x33c(%rbp),%eax
  122e:     48 98                   cltq
  1230:     8b 84 85 60 fe ff ff    mov    -0x1a0(%rbp,%rax,4),%eax
  1237:     01 d0                   add    %edx,%eax
  1239:     89 85 cc fc ff ff       mov    %eax,-0x334(%rbp)
```

执行 a[i]+b[i]操作的指令

```
  1237:     01 d0                   add    %edx,%eax
```

②a[i]+b[i]存在的寄存器说明

将 a[i]存到 edx 将 b[i]存到 eax 再将两个值相加存到 eax

6.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;

```
(gdb) print $eax
$14 = 22
(gdb) print /x $eax
$15 = 0x16
(gdb) print $edx
$16 = -29
(gdb) print /x $edx
$17 = 0xffffffe3
(gdb) stepi
0x0000555555555239      15                      c[i]=a[i]+b[i];
(gdb) print $eax
$18 = -7
(gdb) print /x $eax
$19 = 0xfffffff9
(gdb) print $edx
$20 = -29
(gdb) print /x $edx
$21 = 0xffffffe3
```

实验内容四（20 分）

1. 选取算法：n-皇后问题算法
2. 算法思路：利用哈希值和递归进行，由皇后的规则可知，皇后一定不在同一行也不在同一列，则可能的情况为 1-n 的全排列，此时可形成 n！种情况，其中要除去有皇后在同一对角线上的情况。
3. 源代码（当 n==11 时由于递归对内存的需求，可将此算法视为高复杂度算法）

```
(gdb) list
1       #include<stdio.h>
2       #include<stdlib.h>
3       #include<string.h>
4       #include<math.h>
5       #define maxn 1000
6       int count;
7       int hashTable[maxn]={0};
8       int P[maxn]={0};
9       int n;
10
(gdb)
11      void queenset(int front);
12
13      int main(void){
14              printf("Please let in the number of queens you want to operate:");
15              scanf("%d",&n);
16              queenset(1);
17              printf("%d\n",count);
18              return 0;
19      }
20      int k=1;
(gdb)
21      void queenset(int front){
22              if(front==n+1){
23                      int flag=1;
24                      for(int i=1;i<=n;i++){
25                              for(int j=i+1;j<=n;j++){
26                                      if(abs(i-j)==abs(P[i]-P[j])){
27                                                      flag=0;
28                                                      }
29
30                                      }
(gdb)
31
32                      }
33                      if(flag){
34                              count++;
35                      }
36                      return;
37              }
38              for(int x=1;x<=n;x++){
39                      if(hashTable[x]==0){
40                              P[front]=x;
(gdb)
41                              hashTable[x]=1;
42                              queenset(front+1);
43                              hashTable[x]=0;
44                      }
45              }
46      }
47
(gdb)
Line number 48 out of range; d.c has 47 lines.
```

4. 采用不同的参数运行

①采用-o 进行编译

```
Please let in the number of queens you want to operate:11
2680

real    0m16.092s
user    0m13.890s
sys     0m0.000s
```

②采用-O 进行编译

```
Please let in the number of queens you want to operate:11
2680

real    0m6.447s
user    0m5.240s
sys     0m0.000s
```

③采用-O2 进行编译

```
Please let in the number of queens you want to operate:11
2680

real    0m5.988s
user    0m4.551s
sys     0m0.000s
```

④采用-O3 进行编译

```
Please let in the number of queens you want to operate:11
2680

real    0m6.099s
user    0m5.423s
sys     0m0.000s
```

五、实验分析（20 分）

实验时的工作思路、设想、效果等综合分析

实验内容一

1. 工作思路：采用 vi 编辑器对代码进行编辑，用 gcc 不同的的编译参数进行编译，通过 gdb 在 for 循环出插入断点，在 gdb 调试模式下 run 程序，将停止在 for 循环处，退出 gdb 后，再利用 objdump 生成源程序的汇编程序。

2. 设想：利用不同的参数进行编译，用-0 的运行时间要明显小于用-o 编译的运行时间

3. 效果分析：由 time ./file 指令运行可知，利用-o 编译的程序 1.41s 和用-0 编译的程序时间为 0.544s，由此可知，在-0 的情况下，算法的运行时间缩短了约 61.4%，可见-0 极大的优化了算法的复杂度，对算法运行的效率有足够的提升

实验内容二

1. 工作思路：采用 vi 编辑器对代码进行编辑，将代码敲入并加上 main 函数，用 gcc 不同参数进行编译，利用 time ./file 对代码运行的时间进行计算，打印出运行时间，并对代码进行分析。

2. 设想：随着优化参数的增大，两个代码的运行时间都减少，但算法 1 仍优于算法 2。

3. 效果分析：由运行结果可知：

①对于算法 1，经分析可知-0 编译的结果运行时间相对于-o 约减少了 34%，算法得到了极大的优化，而采用-02 编译的结果相对于-0 的结果几乎没有差别，而采用 03 进行优化也有了极大的提升。

②对于算法 2,经分析可知-0 编译的结果运行时间相对于 o 约减少了 66.2%，算法得到了极大的优化而 02 相对于 0 没有明显提升,03 优化也有了极大的提升。

③观察可知算法 1 总体优于算法 2，也可见，当算法复杂度更高时，采用不同参数进行优化，算法的运行效率提升更加明显，但也可知，当采用 O3 进行编译时算法 1 和算法 2 的最优形式所用时间近似相等，可视为再 O3 优化下两程序几乎相同。

实验内容三

1. 工作思路：按照规则进行编写代码，并补充 main 函数，以保证程序的正常运行，利用 objdump 工具生成汇编程序，寻找到 madd（）的汇编程序；利用 gcc 生成程序的可调试执行程序，再利用 objdump 工具将程序打开，形成可视的汇编指令，找到其中 madd 函数的位置；利用 gdb 进行调试，并尝试不同的指令；在 gdb 环境下利用指令 break line if i==21 对循环进行控制，在 a[xy]+b[xy]停下，对此进行 stepi 对汇编指令逐步进行，最终找到 add 的位置，并输出数据寄存器 eax,edx 的值，观察前后的不同。

2. 工作预期：a[xy]存入 edx，b[xy]存入 eax，最终结果存入 eax。

3. 效果分析：对逐步指令进行分析，对程序进行逐步调试，寻找到 a[xy]+b[xy]的位置，并明确寄存器之间的关系。

实验内容四

1. 工作思路：选取高复杂度算法，选取 n 皇后问题的算法，对于 n 皇后问题，利用哈希值和递归进行，由皇后的规则可知，皇后一定不在同一行也不在同一列，则可能的情况为 1-n 的全排列，此时可形成 n！种情况，其中要除去有皇后在同一对角线上的情况，由于 n 皇后问题采用递归在内存上有高需求，当 n>=11 时可视为高复杂度算法。

2. 工作预期：在优化参数逐渐变大时，算法运行的时间缩短。

3. 效果分析：在 0 的情况下，算法的运行时间明显小于 o 的情况，而 02 相对于 0 仍有少量提升，采用 03 相对于 02 没有明显提升，采用 0 的情况比 o 的情况运行时间约缩短 59.3%，算法得到了极大的优化，而采用 02 的情况相对于采用 o 的情况，采用 03，02，0 的差别没有很大。

六、实验总结（10 分）

总结心得（包括遇到的困难，自己一些不成功的设计和设想）

1. 对算法进行采用的指令不够明确，而且对 Linux 编译环境不熟悉，常因为在 windows 环境下采用的快捷键而在 Linux 下无法使用而出现错误，程序反复重做花费了很多时间，在下次实验中，先通过老师的 ppt 等资料，对编程环境，题目要求进行初步了解，并且熟悉各种操作，并再实验的过程中谨慎小心，过程中也不断学习，不断尝试。

2. 对汇编语言的不熟悉，在实验内容三中，要求对汇编语言有一定了解，靠自身的理解对程序汇编语言的进行有些吃力，在不断尝试中逐步找到了程序运行的机制，并熟练利用 stepi 对汇编指令一条条进行分析。

3.对题干要求不太理解，例如在实验内容三中对寄存器的知识不够明确，打印的结果错误很多遍后进行修改，最终才完成任务。

4.最困难的一点是对程序运行机制的理解不够透彻，通过不断地尝试才能逐步理解其中的机制，最终可以完成调试任务。

七、诚信声明（不签扣 10 分）

需要填写如下声明，并在底部给出手写签名的电子版。

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：

1、简单描述交流内容，例如：来自**熊浩然**的建议，采用 time ./file 方式**对程序运行时间进行分析**。

此外，我还参考了以下资料：

1、https://www.bilibili.com/video/BV13U4y1p7kB?spm_id_from=333.999.0.0

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）