北京郵電大學

实验报告

题目: Linux 环境和 GCC 工具链

班级: 2022211320

学号: 2022212408

姓名: 胡宇杭

学院: 计算机学院 (国家示范性软件学院)

2023 年 4 月 23 日

1 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）。

2 实验环境

- 系统：***Linux Ubuntu* 20.04.5**
- 软件工具：***macOS Terminal*、*gcc* 7.5.0、*GNU gdb* 9.2、*GNU objdump* 2.34**

3 实验内容

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*** 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[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。

4 实验步骤及实验分析

4.1 实验内容一

1. 进入服务器，使用 **vi main.c** 创建 **C** 语言源程序

```
Last login: Tue Oct 10 10:28:15 on ttys000
hhh@EinMacBook-Pro ~ % ssh 2022212408@10.120.11.12
2022212408@10.120.11.12's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-156-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue 10 Oct 2023 10:47:32 AM CST

  System load:           0.12
  Usage of /home:        5.6% of 1.23TB
  Memory usage:          12%
  Swap usage:            0%
  Temperature:           32.0 C
  Processes:             686
  Users logged in:       56
  IPv4 address for eno1: 10.120.11.12
  IPv6 address for eno1: 2001:da8:215:81f7:2204:fff:fee7:2b00

  => There is 1 zombie process.

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

70 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Tue Oct 10 10:28:23 2023 from 10.29.45.91
Your email is: hyuhang@bupt.edu.cn
2022212408@bupt1:~$ vi main.c
```

2. 编写完成数组 **a + b** 的功能

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

const int N = 100;

void init(int *a, int *b)
{
    for (int i = 0; i < N; i ++ )
    {
        a[i] = i - 50;
        b[i] = i + 8;
    }

    return;
}

void madd()
{
    int a[N], b[N];
    init(a, b);

    for (int i = 0; i < N; i ++ )
    {
        a[i]  = a[i] + b[i];
    }

    return;
}

int main()
{
    madd();

    return 0;
}
~
```

3. 采用 **gcc**，使用 **-g -no-pie -fno-stack-protector** 选项编译该程序

```
2022212408@bupt1:~$ gcc -g -no-pie -fno-stack-protector -o work1 main.c
2022212408@bupt1:~$ time ./work1

real    0m0.001s
user    0m0.001s
sys     0m0.000s
2022212408@bupt1:~$
```

4. 使用 **objdump** 生成汇编程序

```
[2022212408@bupt1:~$ objdump -d work1 > work1.txt
[2022212408@bupt1:~$ cat work1.txt
```

5. **madd()** 函数的汇编程序如下图所示

```
00000000000011a8 <madd>:
    11a8:       55                      push   %rbp
    11a9:       48 89 e5                mov    %rsp,%rbp
    11ac:       53                      push   %rbx
    11ad:       48 83 ec 38             sub    $0x38,%rsp
    11b1:       48 89 e0                mov    %rsp,%rax
    11b4:       48 89 c3                mov    %rax,%rbx
    11b7:       b8 64 00 00 00          mov    $0x64,%eax
    11bc:       48 98                   cltq
    11be:       48 83 e8 01             sub    $0x1,%rax
    11c2:       48 89 45 e8             mov    %rax,-0x18(%rbp)
    11c6:       b8 64 00 00 00          mov    $0x64,%eax
    11cb:       48 98                   cltq
    11cd:       49 89 c2                mov    %rax,%r10
    11d0:       41 bb 00 00 00 00       mov    $0x0,%r11d
    11d6:       b8 64 00 00 00          mov    $0x64,%eax
    11db:       48 98                   cltq
    11dd:       48 89 c2                mov    %rax,%rdx
    11e0:       b9 00 00 00 00          mov    $0x0,%ecx
    11e5:       b8 64 00 00 00          mov    $0x64,%eax
    11ea:       48 98                   cltq
    11ec:       48 c1 e0 02             shl    $0x2,%rax
    11f0:       48 8d 50 03             lea    0x3(%rax),%rdx
    11f4:       b8 10 00 00 00          mov    $0x10,%eax
    11f9:       48 83 e8 01             sub    $0x1,%rax
    11fd:       48 01 d0                add    %rdx,%rax
    1200:       b9 10 00 00 00          mov    $0x10,%ecx
    1205:       ba 00 00 00 00          mov    $0x0,%edx
    120a:       48 f7 f1                div    %rcx
    120d:       48 6b c0 10             imul   $0x10,%rax,%rax
    1211:       48 29 c4                sub    %rax,%rsp
    1214:       48 89 e0                mov    %rsp,%rax
    1217:       48 83 c0 03             add    $0x3,%rax
    121b:       48 c1 e8 02             shr    $0x2,%rax
    121f:       48 c1 e0 02             shl    $0x2,%rax
    1223:       48 89 45 d8             mov    %rax,-0x28(%rbp)
    1227:       b8 64 00 00 00          mov    $0x64,%eax
    122c:       48 98                   cltq
    122e:       48 83 e8 01             sub    $0x1,%rax
    1232:       48 89 45 d0             mov    %rax,-0x30(%rbp)
    1236:       b8 64 00 00 00          mov    $0x64,%eax
    123b:       48 98                   cltq
    123d:       49 89 c0                mov    %rax,%r8
    1240:       41 b9 00 00 00 00       mov    $0x0,%r9d
    1246:       b8 64 00 00 00          mov    $0x64,%eax
    124b:       48 98                   cltq
    124d:       48 89 c6                mov    %rax,%rsi
    1250:       bf 00 00 00 00          mov    $0x0,%edi
    1255:       b8 64 00 00 00          mov    $0x64,%eax
    125a:       48 98                   cltq
    125c:       48 c1 e0 02             shl    $0x2,%rax
    1260:       48 8d 50 03             lea    0x3(%rax),%rdx
    1264:       b8 10 00 00 00          mov    $0x10,%eax
    1269:       48 83 e8 01             sub    $0x1,%rax
    126d:       48 01 d0                add    %rdx,%rax
    1270:       bf 10 00 00 00          mov    $0x10,%edi
    1275:       ba 00 00 00 00          mov    $0x0,%edx
    127a:       48 f7 f7                div    %rdi
    127d:       48 6b c0 10             imul   $0x10,%rax,%rax
    1281:       48 29 c4                sub    %rax,%rsp
    1284:       48 89 e0                mov    %rsp,%rax
    1287:       48 83 c0 03             add    $0x3,%rax
    128b:       48 c1 e8 02             shr    $0x2,%rax
    128f:       48 c1 e0 02             shl    $0x2,%rax
    1293:       48 89 45 c8             mov    %rax,-0x38(%rbp)
    1297:       48 8b 55 c8             mov    -0x38(%rbp),%rdx
    129b:       48 8b 45 d8             mov    -0x28(%rbp),%rax
    129f:       48 89 d6                mov    %rdx,%rsi
    12a2:       48 89 c7                mov    %rax,%rdi
    12a5:       e8 a0 fe ff ff          callq  114a <init>
    12aa:       c7 45 e4 00 00 00 00    movl   $0x0,-0x1c(%rbp)
    12b1:       eb 2d                   jmp    12e0 <madd+0x138>
    12b3:       48 8b 45 d8             mov    -0x28(%rbp),%rax
    12b7:       8b 55 e4                mov    -0x1c(%rbp),%edx
    12ba:       48 63 d2                movslq %edx,%rdx
    12bd:       8b 0c 90                mov    (%rax,%rdx,4),%ecx
    12c0:       48 8b 45 c8             mov    -0x38(%rbp),%rax
    12c4:       8b 55 e4                mov    -0x1c(%rbp),%edx
    12c7:       48 63 d2                movslq %edx,%rdx
    12ca:       8b 04 90                mov    (%rax,%rdx,4),%eax
    12cd:       01 c1                   add    %eax,%ecx
    12cf:       48 8b 45 d8             mov    -0x28(%rbp),%rax
    12d3:       8b 55 e4                mov    -0x1c(%rbp),%edx
    12d6:       48 63 d2                movslq %edx,%rdx
    12d9:       89 0c 90                mov    %ecx,(%rax,%rdx,4)
    12dc:       83 45 e4 01             addl   $0x1,-0x1c(%rbp)
    12e0:       b8 64 00 00 00          mov    $0x64,%eax
    12e5:       39 45 e4                cmp    %eax,-0x1c(%rbp)
    12e8:       7c c9                   jl     12b3 <madd+0x10b>
    12ea:       90                      nop
    12eb:       48 89 dc                mov    %rbx,%rsp
    12ee:       48 8b 5d f8             mov    -0x8(%rbp),%rbx
    12f2:       c9                      leaveq
    12f3:       c3                      retq
```

4.2 实验内容二

为了保证程序运行次数尽可能少（懒），实现顺序会与提供的有所不同

1. 使用 **gdb** 命令打开 **GDB**

```
2022212408@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)
```

2. 使用 **file work1** 命令运行之前编译的程序

```
(gdb) file work1
Reading symbols from work1...
(gdb)
```

3. 使用 **break** 命令添加断点

```
(gdb) break madd
Breakpoint 1 at 0x5555555551b1: file main.c, line 18.
(gdb)
```

4. 使用 **info break** 命令查看所有的断点

```
(gdb) break madd
Breakpoint 1 at 0x5555555551b1: file main.c, line 18.
(gdb) break init
Breakpoint 2 at 0x555555555156: init. (8 locations)
(gdb) break 25
Breakpoint 3 at 0x5555555552ea: file main.c, line 27.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555555551b1 in madd at main.c:18
2       breakpoint     keep y   <MULTIPLE>
2.1                         y   0x0000555555555156 in init at main.c:8
2.2                         y   0x00007ffff7e20350 in init at fmtmsg.c:211
2.3                         y   0x00007ffff7e6b850 in init at strsignal.c:81
2.4                         y   0x00007ffff7e6ba1e in init at strsignal.c:86
2.5                         y   0x00007ffff7ef8fa0 in init at backtrace.c:53
2.6                         y   0x00007ffff7ef8ffa in init at backtrace.c:63
2.7                         y   0x00007ffff7ef9110 in init at backtrace.c:54
2.8                         y   0x00007ffff7ef9162 in init at backtrace.c:63
3       breakpoint     keep y   0x00005555555552ea in madd at main.c:27
(gdb)
```

5. 使用 **delete** 命令删除断点

```
(gdb) delete 2
(gdb) delete 3
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555555551b1 in madd at main.c:18
(gdb)
```

6. 使用 **list** 命令查看源代码，并在循环中设置断点

```
(gdb) list 22
17      void madd()
18      {
19              int a[N], b[N];
20              init(a, b);
21
22              for (int i = 0; i < N; i ++ )
23              {
24                      a[i]  = a[i] + b[i];
25              }
26
(gdb) list 27
22              for (int i = 0; i < N; i ++ )
23              {
24                      a[i]  = a[i] + b[i];
25              }
26
27              return;
28      }
29
30      int main()
31      {
(gdb) break 24
Breakpoint 4 at 0x5555555552b3: file main.c, line 24.
(gdb)
```

7. 使用 **run** 命令运行程序，并在第一处断点暂停

```
(gdb) run
Starting program: /students/2022212408/work1

Breakpoint 1, madd () at main.c:18
18      {
(gdb)
```

8. 使用 **stepi** 命令执行一条指令

```
(gdb) stepi
0x00005555555551bc      19              int a[N], b[N];
(gdb)
```

9. 使用 **continue** 命令继续运行程序，在下个断点处暂停

```
(gdb) continue
Continuing.

Breakpoint 4, madd () at main.c:24
24                          a[i]  = a[i] + b[i];
(gdb)
```

10. 使用 **clear** 命令删除函数 **madd()** 的断点

```
(gdb) clear madd

(gdb) info break
Deleted breakpoint 1 Num     Type           Disp Enb Address            What
4       breakpoint     keep y   0x00005555555552b3 in madd at main.c:24
        breakpoint already hit 1 time
(gdb)
```

11. 使用 **disassemble** 命令查看函数 **madd()** 的汇编代码

```
(gdb) disassemble
Dump of assembler code for function madd:
   0x00005555555551a8 <+0>:     push   %rbp
   0x00005555555551a9 <+1>:     mov    %rsp,%rbp
   0x00005555555551ac <+4>:     push   %rbx
   0x00005555555551ad <+5>:     sub    $0x38,%rsp
   0x00005555555551b1 <+9>:     mov    %rsp,%rax
   0x00005555555551b4 <+12>:    mov    %rax,%rbx
   0x00005555555551b7 <+15>:    mov    $0x64,%eax
   0x00005555555551bc <+20>:    cltq
   0x00005555555551be <+22>:    sub    $0x1,%rax
   0x00005555555551c2 <+26>:    mov    %rax,-0x18(%rbp)
   0x00005555555551c6 <+30>:    mov    $0x64,%eax
   0x00005555555551cb <+35>:    cltq
   0x00005555555551cd <+37>:    mov    %rax,%r10
   0x00005555555551d0 <+40>:    mov    $0x0,%r11d
   0x00005555555551d6 <+46>:    mov    $0x64,%eax
```

12. 使用 **info reg** 命令查看当前所有寄存器的值

```
(gdb) info reg
rax            0x7fffffffe900      140737488349440
rbx            0x7fffffffeaa0      140737488349856
rcx            0x10                16
rdx            0x0                 0
rsi            0x7fffffffe760      140737488349024
rdi            0x7fffffffe900      140737488349440
rbp            0x7fffffffeae0      0x7fffffffeae0
rsp            0x7fffffffe760      0x7fffffffe760
r8             0x64                100
r9             0x0                 0
r10            0x64                100
r11            0x0                 0
r12            0x555555555040      93824992235584
r13            0x7fffffffebe0      140737488350176
r14            0x0                 0
r15            0x0                 0
rip            0x5555555552ba      0x5555555552ba <madd+274>
eflags         0x297               [ CF PF AF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

13. 使用 **print** 命令输出 **%rax** 寄存器的内容

```
(gdb) print /x $rax
$1 = 0x7fffffffe900
(gdb)
```

14. 使用 **x/20** 命令检查函数 **madd()** 的前 **20** 个字节

```
(gdb) x/20 madd
0x5555555551a8 <madd>:  0xe5894855      0xec834853      0xe0894838      0xb8c38948
0x5555555551b8 <madd+16>:       0x00000064      0x83489848      0x894801e8      0x64b8e845
0x5555555551c8 <madd+32>:       0x48000000      0xc2894998      0x0000bb41      0x64b80000
0x5555555551d8 <madd+48>:       0x48000000      0xc2894898      0x000000b9      0x0064b800
0x5555555551e8 <madd+64>:       0x98480000      0x02e0c148      0x03508d48      0x000010b8
(gdb)
```

15. 使用 **watch** 命令监视变量 **i**，继续运行，程序在 **i** 的值发生变化时暂停

```
(gdb) watch i
Hardware watchpoint 6: i


(gdb) continue
Continuing.

Hardware watchpoint 6: i

Old value = 0
New value = 1
0x000055555555552e0 in madd () at main.c:22
22              for (int i = 0; i < N; i ++ )
(gdb)
```

16. 使用 **kill** 命令停止程序

```
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 387088) killed]
(gdb)
```

17. 使用 **nexti** 命令执行下一个函数

```
(gdb) nexti
22              for (int i = 0; i < N; i ++ )
(gdb)
```

18. 使用 **quit** 命令退出 **GDB**

```
(gdb) quit
2022212408@bupt1:~$
```

4.3 实验内容三

使用 **objdump -S -d work1** 命令生成汇编程序，我们可以很直观地找到 **a[i] = a[i] + b[i]** 的汇编代码，如下图所示

```
2022212408@bupt1:~$ objdump -S -d work1

work1:     file format elf64-x86-64


                a[i]  = a[i] + b[i];
  401270:       48 8b 45 d8             mov    -0x28(%rbp),%rax
  401274:       8b 55 e4                mov    -0x1c(%rbp),%edx
  401277:       48 63 d2                movslq %edx,%rdx
  40127a:       8b 0c 90                mov    (%rax,%rdx,4),%ecx
  40127d:       48 8b 45 c8             mov    -0x38(%rbp),%rax
  401281:       8b 55 e4                mov    -0x1c(%rbp),%edx
  401284:       48 63 d2                movslq %edx,%rdx
  401287:       8b 04 90                mov    (%rax,%rdx,4),%eax
  40128a:       01 c1                   add    %eax,%ecx
  40128c:       48 8b 45 d8             mov    -0x28(%rbp),%rax
  401290:       8b 55 e4                mov    -0x1c(%rbp),%edx
  401293:       48 63 d2                movslq %edx,%rdx
  401296:       89 0c 90                mov    %ecx,(%rax,%rdx,4)
```

从 **add %eax, %ecx**，我们可以看出 **a[i]** 被存放在 **%ecx** 寄存器中，**b[i]** 被存放在 **%eax** 寄存器中

4.4 实验内容四

1. 用 **GDB** 打开之前编译的程序

```
2022212408@bupt1:~$ gdb work1
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 work1...
(gdb)
```

2. 添加条件断点，在 **i == 48** 时暂停程序

```
(gdb) break 24 if i == 48
Breakpoint 1 at 0x401270: file main.c, line 24.
(gdb) run
Starting program: /students/2022212408/work1

Breakpoint 1, madd () at main.c:24
24                      a[i]  = a[i] + b[i];
(gdb) print i
$1 = 48
(gdb)
```

3. 此时由于不知道 **stepi** 命令应该执行几步，所以先使用 **disassemble/m** 命令查看汇编代码，发现应执行 **8** 步，并且操作数寄存器为 **%ecx** 和 **%eax**

```
(gdb) disassemble/m
Dump of assembler code for function madd:

23              {
24                      a[i]  = a[i] + b[i];
=> 0x0000000000401270 <+267>:   mov    -0x28(%rbp),%rax
   0x0000000000401274 <+271>:   mov    -0x1c(%rbp),%edx
   0x0000000000401277 <+274>:   movslq %edx,%rdx
   0x000000000040127a <+277>:   mov    (%rax,%rdx,4),%ecx
   0x000000000040127d <+280>:   mov    -0x38(%rbp),%rax
   0x0000000000401281 <+284>:   mov    -0x1c(%rbp),%edx
   0x0000000000401284 <+287>:   movslq %edx,%rdx
   0x0000000000401287 <+290>:   mov    (%rax,%rdx,4),%eax
   0x000000000040128a <+293>:   add    %eax,%ecx
   0x000000000040128c <+295>:   mov    -0x28(%rbp),%rax
   0x0000000000401290 <+299>:   mov    -0x1c(%rbp),%edx
   0x0000000000401293 <+302>:   movslq %edx,%rdx
   0x0000000000401296 <+305>:   mov    %ecx,(%rax,%rdx,4)
```

4. 按照题目要求输出指定内容

```
(gdb) stepi 8
0x000000000040128a      24                      a[i]  = a[i] + b[i];
(gdb) print $ecx
$15 = -2
(gdb) print /x $ecx
$16 = 0xfffffffe
(gdb) print $eax
$17 = 56
(gdb) print /x $eax
$18 = 0x38
(gdb) stepi
0x000000000040128c      24                      a[i]  = a[i] + b[i];
(gdb) print $ecx
$19 = 54
(gdb) print /x $ecx
$20 = 0x36
(gdb) print $eax
$21 = 56
(gdb) print /x $eax
$22 = 0x38
(gdb)
```

5 总结体会

- 在本次实验中，我深入了解了 ***Linux*** 环境下的常用命令、***gcc*** 编译器、***gdb*** 调试工具以及 ***objdump*** 反汇编工具，更加深刻地认识到了汇编程序与源程序之间的对应关系，并通过实验加强了对相关命令的掌握。
- 遇到的问题：一开始我以为 **stepi** 和 **nexti** 命令的操作对象是源代码，导致在实验时并没有及时发现使用 **stepi** 和 **nexti** 命令时代码并没有运行至下一行的根本原因
- 解决方法：通过询问大语言模型，在 ***Stack Overflow*** 上查询相关问题，最终发现这两条命令的操作对象是汇编代码，解决了问题
- 建议：希望老师能提早把实验需要更改的地方通知给大家，~~今天都写了一半了才通知实验有更新~~