实验6 RISC-V汇编器与模拟器实验

一、实验目的

- 学习RISC-V的RV32I指令集，熟悉其指令格式、汇编指令助记符，掌握机器指令编码方法；
- 学习RV32I汇编程序设计，学会使用RISC-V交叉编译器、汇编器，将高级语言程序翻译成汇编语言程序，进而翻译成二进制文件；同时，学会反汇编方法；
- 了解使用RISC-V的模拟器运行程序的方法。

二、实验内容与原理

- **实验内容：** 首先安装一个装有RISC-V交叉编译器的Linux虚拟机，然后学会使用Linux命令进行程序的编译、汇编链接、运行以及反汇编。

  - 1、计算机的三种语言
  - 2、编译过程
  - 3、安装交叉编译器的Linux虚拟机
  - 4、RISC-V交叉编译过程
  - 5、RISC-V汇编语言程序

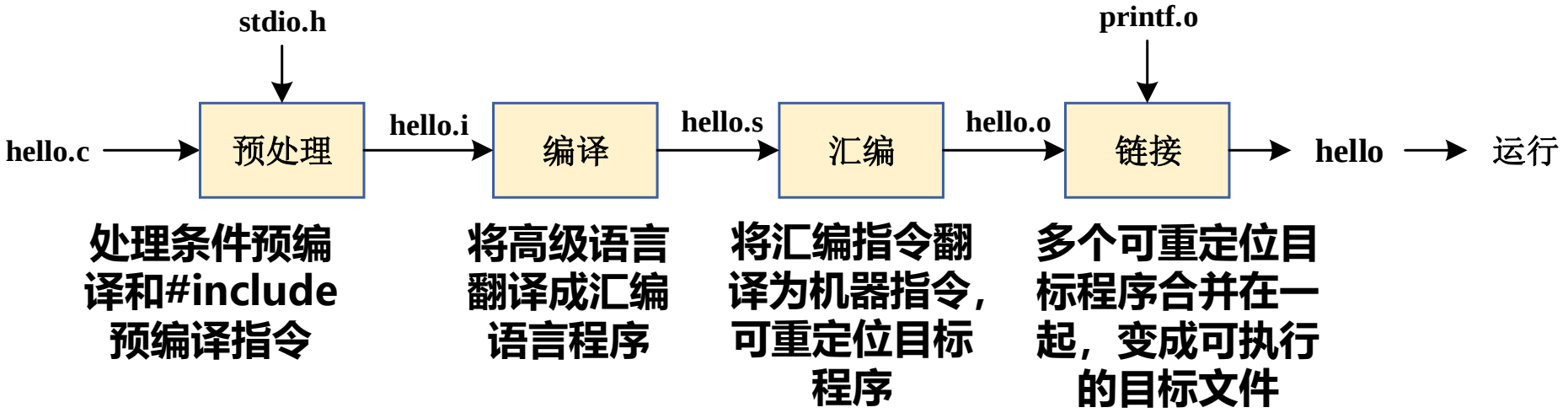
1、计算机的三种语言

- （1）机器语言：用机器指令编写程序，代码语言
  - 机器指令：能被计算机硬件识别并直接执行的0、1代码串。
- （2）汇编语言：用汇编指令编写程序，符号语言
  - 汇编指令：用助记符来表示机器指令
  - 汇编器：将汇编语言程序翻译成机器语言程序的软件
- （3）高级语言：用面向用户的自然语言编写程序，符号语言，可读性好，编程效率高，可移植性好
  - 编译器：将高级语言程序翻译成二进制机器语言程序的软件

2、编译过程

```
#include <stdio.h>
int main() {
        printf("hello world\n");
        return 0;
}
```

- 高级语言程序→可执行的二进制文件（机器语言程序）：经过4个阶段



3、安装交叉编译器的Linux虚拟机

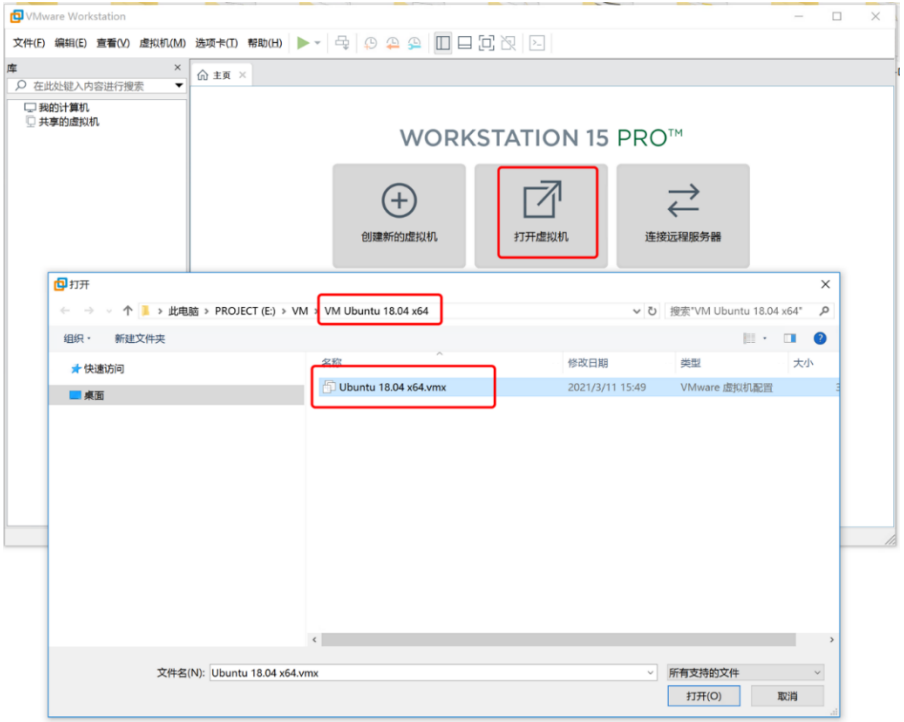
- 交叉编译: 在一种平台上运行编译器程序，生成的目标程序可以在另一种目标平台上运行，但是不能在该编译平台上运行。
  - 宿主机：运行交叉编译器的计算机
  - 目标机：能运行交叉编译生成的目标文件的计算机
- RISC-V的交叉编译工具链大多在Linux环境下
- 首先安装一个Linux虚拟机，然后在Linux虚拟机中安装RISC-V的交叉编译工具链 。

3、安装交叉编译器的Linux虚拟机

1）下载并安装VMware Workstation Player，要求版本16

2）拷贝安装好RISC-V交叉编译器的Linux虚拟机文件夹“VM Ubuntu 18.04 x64”到本地磁盘（25G的空闲空间）

3）打开Vmware→“打开虚拟机”→选择前述“VM Ubuntu 18.04 x64”文件夹中的“Ubuntu 18.04 x64.vmx”并打开

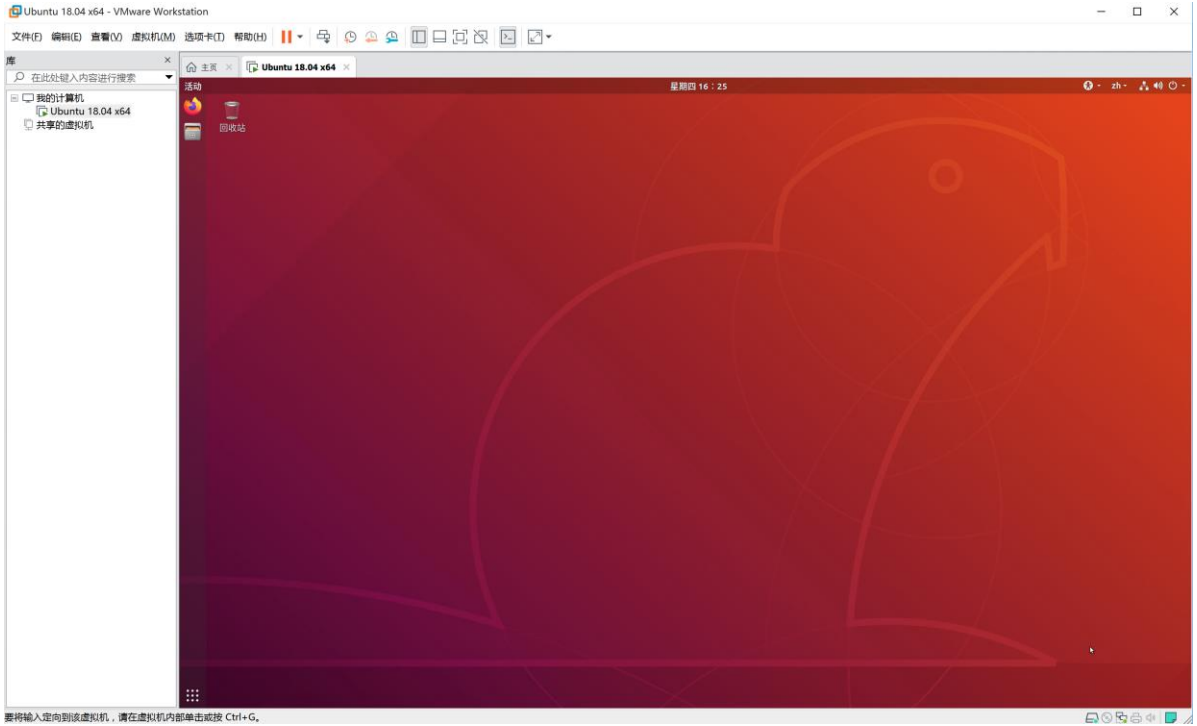


3、安装交叉编译器的Linux虚拟机

4）配置虚拟机的内存和CPU核心数

5）点击“开启此虚拟机”，启动虚拟机。

6）开机后，默认登录用户：hdu，默认密码：hdu



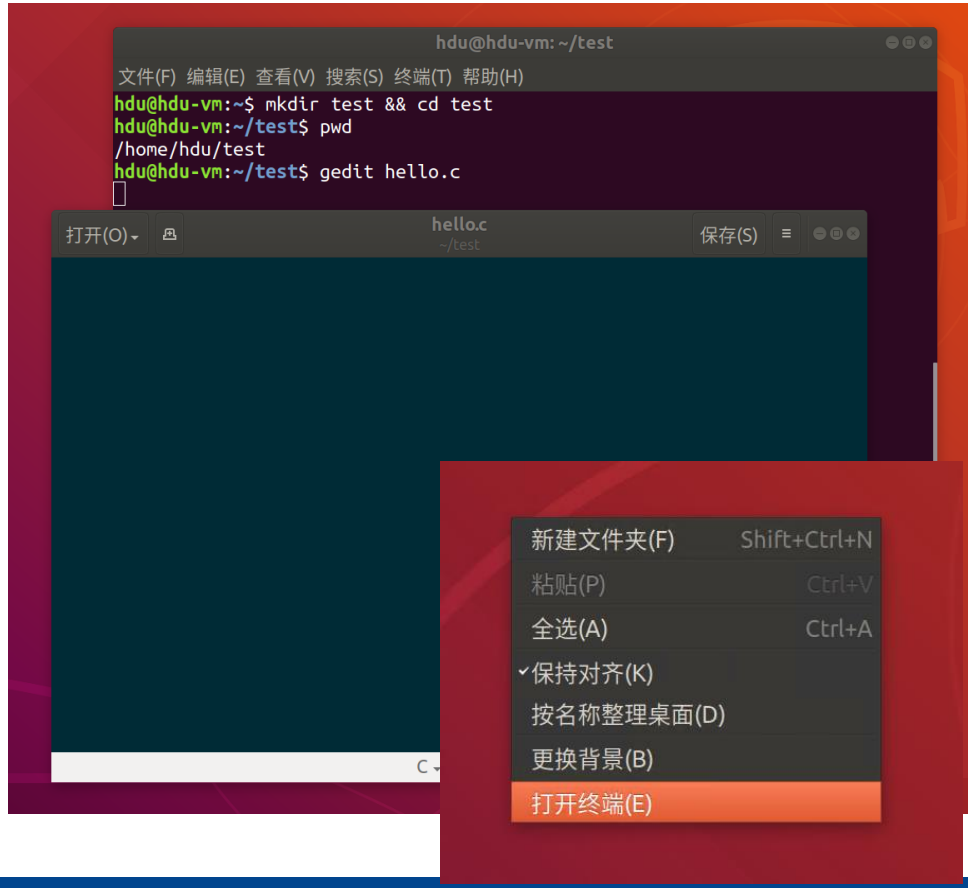
4、RISC-V交叉编译过程

- 1）新建一个hello.c程序的源文件

①右键点击“打开终端”，默认进入用户hdu的根目录“/home/hdu”

- 命令行 “mkdir test && cd test”：创建一个名为“test”的目录并进入
- “pwd”：查看当前所处的目录

② “gedit hello.c”：使用gedit编辑器在当前目录下编辑hello.c文件，若不存在该文件会自动创建

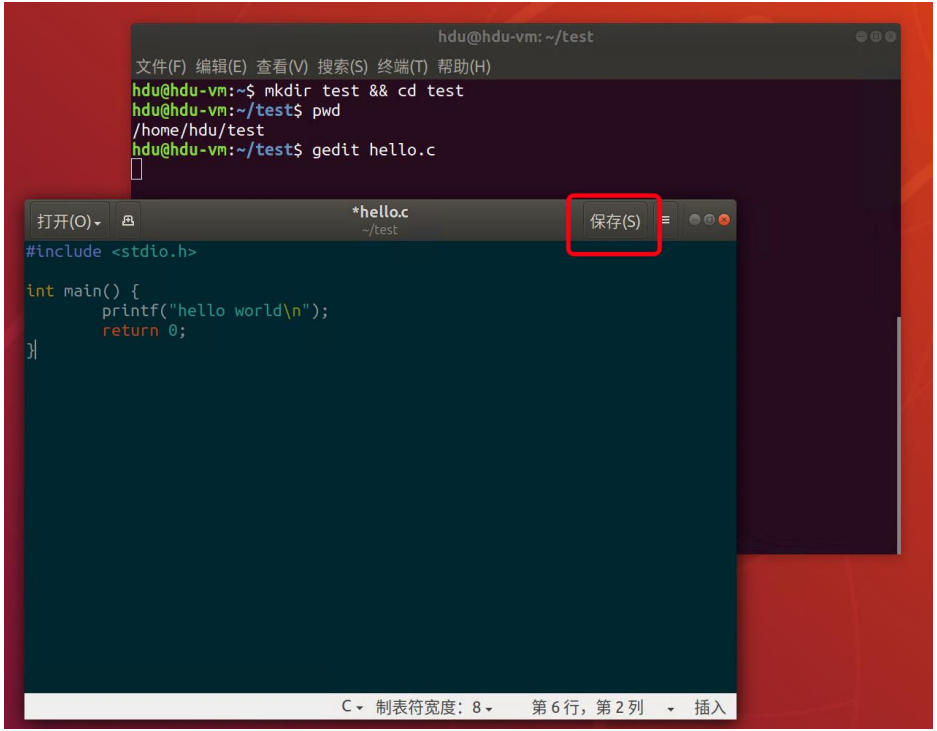


4、RISC-V交叉编译过程

- 1）新建一个hello.c程序的源文件

③弹出编辑框，输入C语言代码、保存、关闭编辑器窗口

```
#include <stdio.h>
int main() {
        printf("hello world\n");
        return 0;
}
```



4、RISC-V交叉编译过程

- 2）编译并运行32位程序

①gccrv32 hello.c -o hello：编译hello.c文件，生成的文件为hello

- 一次编译生成了hello：中间经过了预编译、编译、汇编和链接4个阶段
- 命令选项“-o”：“输出”，其后是输出产生的文件名

②runrv32 hello：运行可执行文件“hello”，显示结果

4、RISC-V交叉编译过程

- 3）反汇编目标文件

① dumprv32 -d hello > dump32.txt：反编译hello文件，并将结果保存到文本文件dump32.txt，“>”指定输出到哪个文件

② gedit dump32.txt：打开dump32.txt

- 包括指令内存地址、机器指令代码、汇编指令等

```
hdu@hdu-vm: ~/test
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
hdu@hdu-vm:~$ mkdir test && cd test
hdu@hdu-vm:~/test$ pwd
/home/hdu/test
hdu@hdu-vm:~/test$ gedit hello.c
hdu@hdu-vm:~/test$ gccrv32 hello.c -o hello
hdu@hdu-vm:~/test$ ls
hello  hello.c
hdu@hdu-vm:~/test$ runrv32 hello
hello world
hdu@hdu-vm:~/test$ dumprv32 -d hello > dump32.txt
hdu@hdu-vm:~/test$ ls
dump32.txt  hello  hello.c
hdu@hdu-vm:~/test$ gedit dump32.txt
```

```
打开(O)   dump32.txt   ~/test   保存(S)

hello:     文件格式 elf32-littleriscv


Disassembly of section .text:

00010074 <register_fini>:
   10074:       00000793                li      a5,0
   10078:       00078863                beqz    a5,10088 <register_fini+0x14>
   1007c:       00011537                lui     a0,0x11
   10080:       93450513                addi    a0,a0,-1740 # 10934 <__libc_fini_array>
   10084:       2a80206f                j       1232c <atexit>
   10088:       00008067                ret

0001008c <_start>:
   1008c:       00005197                auipc   gp,0x5
   10090:       d8418193                addi    gp,gp,-636 # 14e10 <__global_pointer$>
   10094:       04418513                addi    a0,gp,68 # 14e54 <__malloc_max_total_mem>
   10098:       09c18613                addi    a2,gp,156 # 14eac <__BSS_END__>

纯文本  制表符宽度：8  第 1 行，第 1 列  插入
```

4、RISC-V交叉编译过程

- **编译、执行、反汇编命令总结：假设：**
  - 已有的源程序文件：source.c
  - 编译生成的文件名（-o后的第一个参数）：compiled.o
  - 存放反编译结果的文件：dump.txt
- 编译32位程序的命令：gccrv32 source.c -o compiled.o
- 执行32位程序的命令：runrv32 compiled.o
- 反汇编32位程序的命令：dumprv32 -d compiled.o > dump.txt
- 编译64位程序的命令：gccrv64 source.c -o compiled.o
- 执行64位程序的命令：runrv64 compiled.o
- 反汇编64位程序的命令：dumprv64 -d compiled.o > dump.txt

4、RISC-V交叉编译过程

- 4）分步编译过程
  - gccrv32 hello.c -o hello：一次性生成了可执行文件hello
  - 使用不同的编译选项，可以分步编译
- 预处理：gccrv32 -E hello.c -o hello.i，生成hello.i文件
- 编译：gccrv32 -S hello.i -o hello.s，生成汇编语言程序hello.s文件
- 汇编：gccrv32 -c hello.s -o hello.o，生成可重定位目标文件hello.o
- 链接：gccrv32 hello.o -o hello，生成可执行文件hello
- 运行：runrv32 hello

- 如果是编译为64位的RISC-V程序并运行：命令中的“32”改成“64”

5、RISC-V汇编语言程序

- .s文件：汇编语言源程序
- .o文件：可重定位目标文件，包含了若干节（section）：
  - .text节：机器指令码（即程序）
  - .data节：已初始化静态数据
  - .bss节：未初始化静态数据
  - .rodata节：只读数据（常量）
- hello.s的部分汇编指示符和指令的解释

```
    .section .rodata            #指示下面是只读数据区域
    .align   2                  #指示按照 2²=4 字节边界对齐
.LC0:                           #定义标号 LC0，代表下面 string 常量字符串的首地址
    .string  "hello world"      #定义字符串
    .text                       #代码段从此开始
    .align   2                  #指示代码段按照 2²=4 字节边界对齐
    .globl   main               #声明 main 是全局符号
    .type main, @function       #定义 main 是一个函数类型
main:                           # main 符号，表明主程序开始
    addi     sp,sp,-16          #创建本程序的栈帧（堆栈空间），16B
    sw       ra,12(sp)          #保存返回地址到堆栈
    sw       s0,8(sp)           #保存帧指针 s0（fp）到堆栈（fp 后续用于创建局部栈帧）
    addi     s0,sp,16           #s0 指向本程序的栈帧后面
    lui   a5,%hi(.LC0)          #%hi 表示取符号 LC0 的地址高 20 位，低 12 位清零
    addi     a0,a5, %lo(.LC0)   #%lo 表示取符号 LC0 的地址低 12 位，高 20 位清零
    call     puts               #调用子程序 puts，输入参数在 a0 寄存器（要输出的字符串首
                                地址）
    li       a5,0               #参数 a5 清零
    mv       a0,a5              #参数 a0 清零
    lw    ra,12(sp)             #弹出返回地址
    lw    s0,8(sp)              #弹出 s0，恢复现场
    addi     sp,sp,16           #释放本程序的堆栈空间
    jr       ra                 #返回到操作系统
```

5、RISC-V汇编语言程序

- 例：编写汇编语言程序，并将其翻译成机器语言程序
- 1）编辑汇编源程序
  - 输入汇编程序，保存为test.s： 用gedit test.s命令或者任何一个文本编辑器
- 程序功能：计算（-3）*8+1000-7000
- 对应的C程序片段是：

```
main:
li    x5,  -3          #-3→x5
li    x6,  1000        #1000→x6
li    x7,  7000        #7000→x7
slli  x8,  x5,  3      #x5 << 3→x8
add   x8,  x8,  x6     #x8+x6→x8
sub   x8,  x8,  x7     #x8-x7→x8
jr    ra
```

```
int a,b,c,w;
a = -3;          b = 1000;          c = 7000;
w = 8*a+b-c;
```

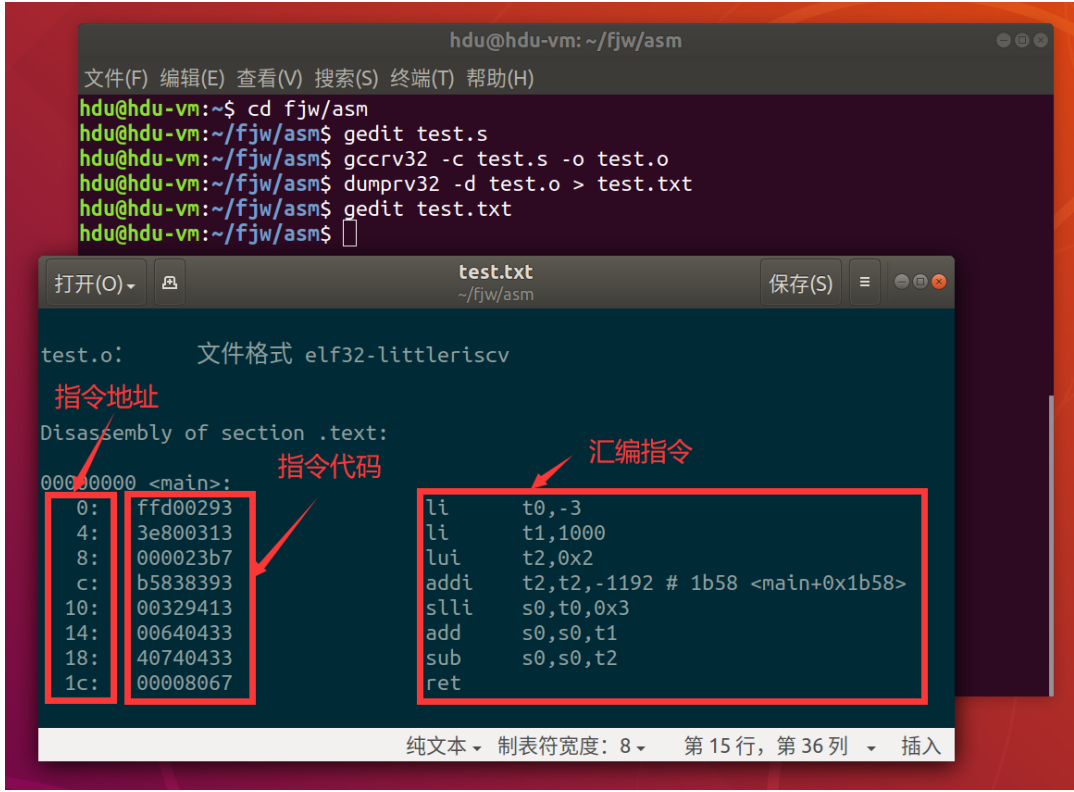
5、RISC-V汇编语言程序

2）汇编程序，变成可执行文件

- gccrv32 -c test.s -o test.o：将汇编语言源程序翻译成二进制文件

3）反汇编可执行文件，查看机器指令码

- dumprv32 -d test.o > test.txt：将.o文件的二进制代码反汇编，输出到test.txt文件中
- gedit test.txt：查看test.txt内容



三、实验要求

1. 安装好带有RISC-V交叉编译器和模拟器的Linux虚拟机，并用hello.c程序验证，确保能使用。

2. 将下面一段汇编语言程序acc.s，借助汇编器和反汇编器，将其翻译成机器语言程序；分析其功能；分析jLoop1是用什么指令实现的?

```
main:
        add   t0,x0,    x0
        add   t1,x0,    x0
        addi  t2,x0,    10
L1:       lw    t3,0x40(t1)
        add   t0,t0,t3
        addi  t1,t1,4
        addi  t2,t2,-1
        beq   t2,x0,    L2
        j   L1
L2:       sw t0,    0x80(x0)
```

三、实验要求

3. move.s：

- 子程序BankMove：将主存一个地址连续的数据块（数组）复制到主存另一个区域，3个入口参数：
  - a0：源数据区域的首地址
  - a1：目标数据区域的首地址
  - a2：复制的数据个数（数组长度）
- 主程序main：调用BankMove，从内存区域30H复制10个数据到60H

```
BankMove:
    add     t0,   a0,   zero;      #t0=源数据区域首址
    add     t1,   a1,   zero;      #t1=目的数据区域首址
    add     t2,   a2,   zero;      #t2=数据块长度
L1: lw      t3,   0(t0);           #t3=取出数据
    sw      t3,   0(t1);           #存数据
    addi    t0,   t0,   4;         #移动源数据区指针
    addi    t1,   t1,   4;         #移动目的数据区指针
    addi    t2,   t2,   -1;        #计数值-1
    bne     t2,   zero, L1;        #计数值≠0，则没有复制完，转循环体首部
    jr      ra                     #复制完成，则子程序返回
main:
    addi    a0,   zero, 0x30;      #a0=0000_0030H，源数据区域首址
    addi    a1,   zero, 0x60;      #a1=0000_0060H，目的数据区域首址
    addi    a2,   zero, 10;        #a2=0000_000AH，复制的数据个数
    jal     BankMove               #子程序调用
```

(1)请对上述程序进行汇编与反汇编；

(2)说明子程序调用和返回的具体实现过程：解析子程序调用jal和返回指令jr的机器代码，计算它们的转移地址，写出指令执行的具体操作与结果，指出跳转目标地址的指令。

三、实验要求

4. 对于下面一段C语言程序，请动手编写对应的RV32I汇编程序sum.s，并进行汇编和反汇编。

```
int sum(int n)
{
    int i,s=0;
    for(i=0;i<=n;i++)
        s += i;
    return(s);
}
```

```
int main()
{
    int x=100;    int y;
    y = sum(x);
    return 0;
}
```

5. 直接对上面的C程序sum.c进行编译，并反汇编，与自己编写的汇编语言程序对比，有什么不同？请进行分析。

四、实验步骤

1. 下载虚拟机软件VMware Workstation Player 16并安装。
2. 按照前述步骤，拷贝并打开装有RISC-V交叉编译器和模拟器的Linux虚拟机，进行设置。
3. 编辑hello.c文件，进行一步到位的编译和分步编译测试，确保RISC-V交叉编译器和模拟器能正常使用。
4. 用test.s文件进行汇编器和反汇编器的测试。
5. 将指定的汇编语言程序拷贝到acc.s文件，使用命令对其汇编，然后反汇编，分析其功能。

四、实验步骤

6. 对move.s汇编程序进行汇编和反汇编，查看子程序调用与返回指令的机器指令代码，分析具体的字段编码，计算其偏移量与目标地址，得出结论。
7. 编写sum.s的汇编语言程序，使用命令对其汇编，然后反汇编。
8. 将sum.c的C程序进行编译，生成二进制文件后，再进行反汇编，与上述反汇编代码对比与分析。

五、思考与探索（至少完成1道）

1. 测试用例hello.c中，如果改成“printf("hello %s\n","world");”，重新测试，汇编程序有什么不同?
2. acc.s程序在功能不变的情况下，可以优化得更简短吗？如果可以，说明你的方法。
3. move.s程序中，BankMove子程序复制的数据是字节、半字还是字？你是如何判断的？它为何不直接使用a0~a2完成复制，而是要将其装入t0~t2后再进行处理?
4. 写出调用BankMove子程序，复制从内存单元1000 0000H到1000 100H的20个字数据的主程序。

五、思考与探索（至少完成1道）

5. 仔细分析你的sum.s程序，说明你是如何保证边界条件满足的?
6. 华为公司设计与生产了海思麒麟、鲲鹏及昇腾处理器芯片，请查找资料，说明它们是基于什么指令集架构的处理器？它们之间有何区别？对应的指令集架构与RISC-V有何异同点?
7. 查阅文献或者官网资料，了解国产龙芯处理器的指令集架构和应用领域，与RISC-V指令格式作对比。
8. 谈谈你在实验中碰到了哪些问题？又是如何解决的?