

南开大学

汇编语言与逆向技术课程实验报告

实验四： ARM 平台-HelloWorld



学 院 网络空间安全学院
专 业 信息安全、法学双学位
学 号 2212000
姓 名 宋奕纬
班 级 1061

一、实验目的

- 1、理解 GNU ARM 汇编代码运行环境的搭建、配置及编译运行，掌握在华为

鲲鹏云服务器上进行环境配置

- 2、命令行输出 “HelloWorld”

二、实验原理

- 1、实验环境：

华为鲲鹏云主机、

openEuler20.03 操作系统；

- 2、实验原理：vim 基本模式介绍

- (1) 命令模式

Vim 默认模式，命令模式下的按键都会被当做命令执行

任何模式下都可以通过 esc 回到命令模式

- (2) 插入模式

必须进入插入模式才允许编辑内容

从命令模式按 i/o/a 键进入插入模式

- (3) 末行模式

常用于文件保存、退出及列出行号等操作

从命令模式按 shift+: 进入末行模式

:w [保存]

:q [退出]

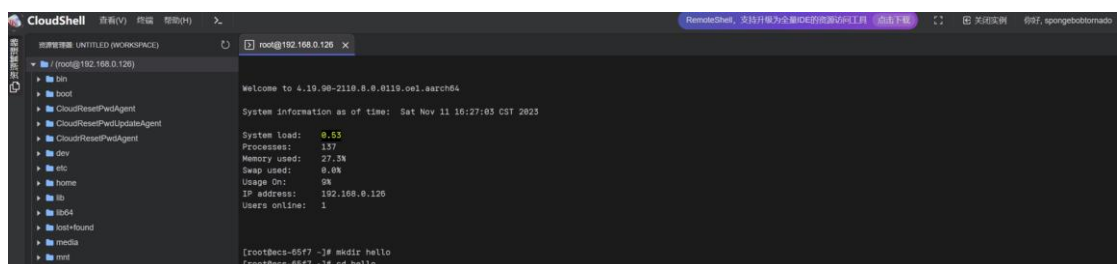
:wq [保存并退出]

:q! [不保存强制退出]

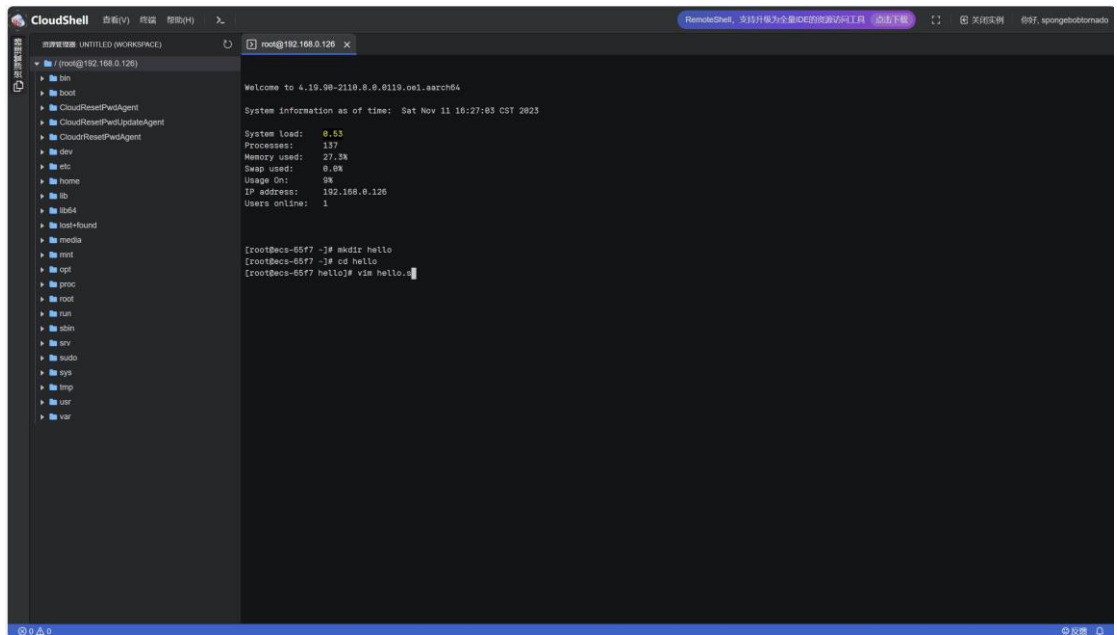
三、实验过程

- 1、创建 hello 目录

执行以下命令，创建 hello 目录，存放该程序的所有文件，并进入 hello 目录。



2、创建示例程序代码 hello.s



The screenshot shows a CloudShell terminal window. The left sidebar displays a file tree with directories like bin, boot, CloudResetPwAgent, etc. The main terminal area shows the following output:

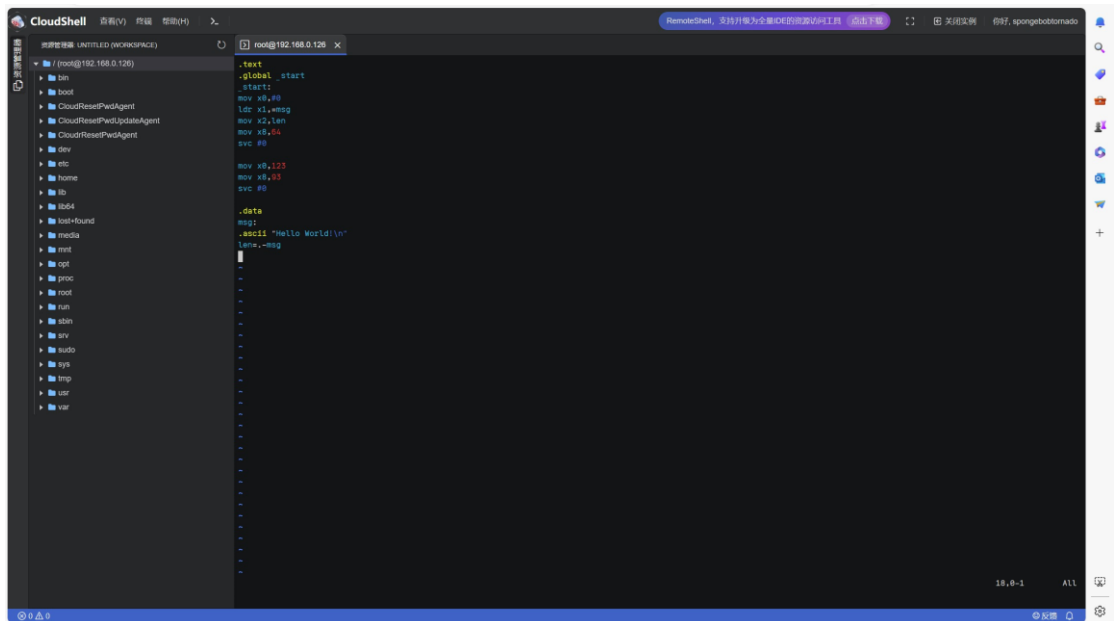
```
Welcome to 4.19.08-2118.8.0.6119.oel.aarch64

System Information as of time: Sat Nov 11 16:27:03 CST 2023

System load: 0.53
Processes: 137
Memory used: 27.3%
Swap used: 0.0%
Usage On: 0%
IP address: 192.168.0.126
Users online: 1

[root@ecs-65f7 ~]# mkdir hello
[root@ecs-65f7 ~]# cd hello
[root@ecs-65f7 hello]# vim hello.s
```

3、主代码部分书写



The screenshot shows the same CloudShell terminal window, but now the file `hello.s` is open in the editor. The code is as follows:

```
.text
.global _start
_start:
    mov r0, #0
    ldr r1, msg
    svc #0

    mov r0, 123
    mov r0, 01
    svc #0

.data
msg:
    .ascii "Hello World!\n"
    .len .msg
```

4、编译运行

保存示例源码文件，然后退出 vim 编辑器。在当前目录中依次执行以下命令，进行代码编译运行。

```
[root@ecs-65f7 ~]# as hello.s -o hello.o
[root@ecs-65f7 ~]# ld hello.o -o hello
ld: cannot open output file hello: Is a directory
[root@ecs-65f7 ~]# ld hello.o -o hello
ld: cannot open output file hello: Is a directory
[root@ecs-65f7 ~]# rm -rf hello
[root@ecs-65f7 ~]# ld hello.o -o hello
[root@ecs-65f7 ~]# ./hello
Hello World!
```

出现报错，网上搜索资料后知道目录下出现同名的文件，故用命令 `rm -rf test` 删除 (也可以更改文件名)，修改后正常运行，输出 `hello`。

四、实验结论及心得体会

1、 代码解析：

代码	解析
.text	表示接下来的指令为代码段
.global _start	声明一个全局标签_start，作为程序的入口点。
_start:	程序的入口点标签。
mov x0,#0	把#0 存储到寄存器 x0 中，用于存储函数的返回值，表示输出到标准输出。
ldr x1,msg	把字符串 msg 的地址加载到 x1 寄存器中，用于输出该字符串。
mov x2,len	把 len 的值存储到寄存器 x2 中，len 表示字符串的长度。
mov x8,64	ARM64 汇编语言函数前 8 个参数使用 x0-x7 寄存器传递，多于 8 个的参数均通过堆栈传递，并且返回值通过 x0 寄存器（返回。在使用软中断进行系统调时，系统调用号通过 x8 寄存器传递，用 svc 指令产生软中断，实现从用户模式到管理模式的切换。 1、svc #0 触发一个 Supervisor Call（SVC）异常，用于实现操作系统的系统调用。 2、系统调用号： 64: sys_write() 用于输出 93: sys_exit() 用于退出 3、mov x0, 123 表示 exit code
svc #0	
mov x0,123	
mov x8,93	
svc #0	

代码	解析
.data	表示接下来的数据属于数据段。
msg:	字符串 msg 的起始位置标签，标识字符串的起始位置。
.ascii "Hello World!\n"	定义并存储字符串"Hello World!\n"
len=.-msg	计算字符串的长度，并赋值给 len 标签。

2、问题：同样的代码能否在 x86 平台运行，为什么？

答：不能，因为二者处理器架构不同，有不同的指令集和硬件架构（寄存器等），x86 处理器不能理解 arm 指令集。故 arm 汇编下的代码无法在 x86 平台下直接运行。