汇编与接口技术实验环境说明

版本：1.0



华为技术有限公司

目录

[1汇编与接口技术实验环境介绍 2](#_Toc50558376)

[1.1实验介绍 2](#_Toc50558377)

[1.1.1关于本实验 2](#_Toc50558378)

[1.1.2教学目标 2](#_Toc50558379)

[1.2 ARMv8原生实验环境搭建 2](#_Toc50558380)

[1.2.1购买华为鲲鹏云服务器 2](#_Toc50558381)

[1.2.2环境配置 6](#_Toc50558382)

[1.2.3测试汇编代码 7](#_Toc50558383)

[1.3华为CloudIDE开发环境配置 8](#_Toc50558384)

[1.3.1开通CloudIDE 8](#_Toc50558385)

[1.3.2测试汇编代码 11](#_Toc50558386)

# 汇编与接口技术实验环境介绍

## 实验介绍

### 关于本实验

ARMv8汇编实验环境的搭建。

### 教学目标

掌握在鲲鹏云服务器上搭建ARMv8 汇编实验环境。

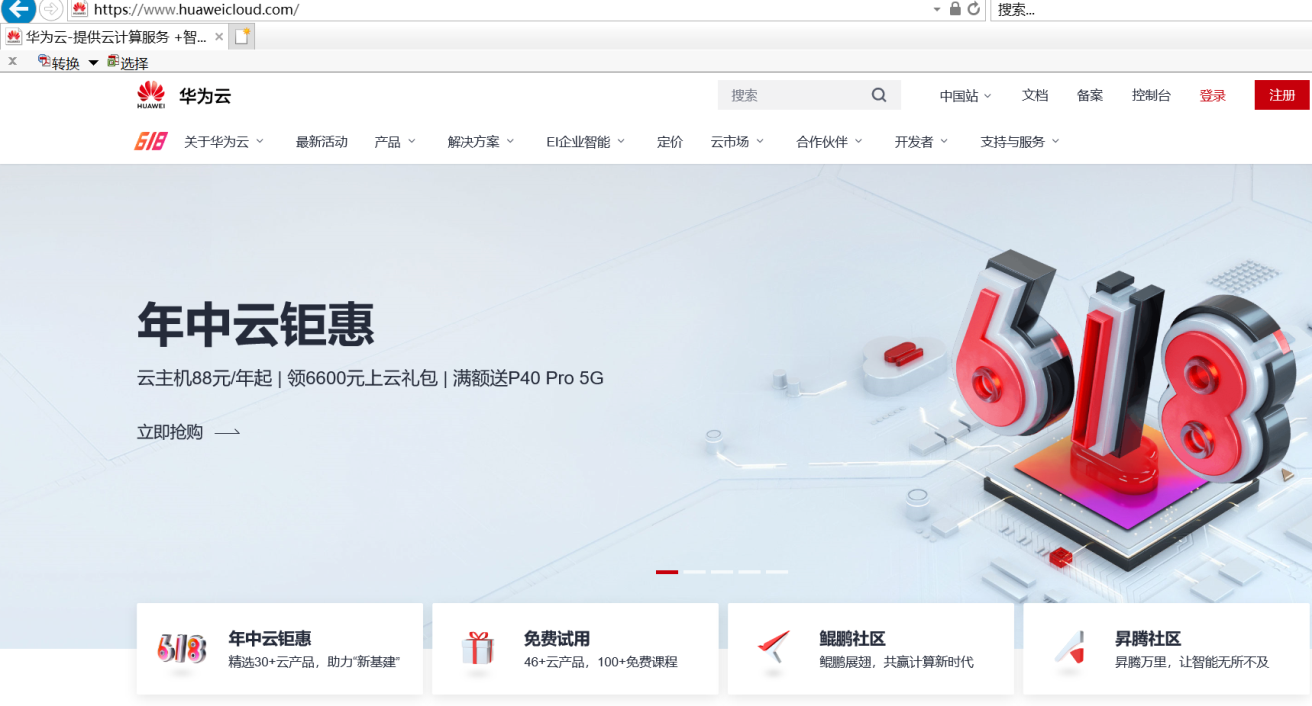
## ARMv8原生实验环境搭建

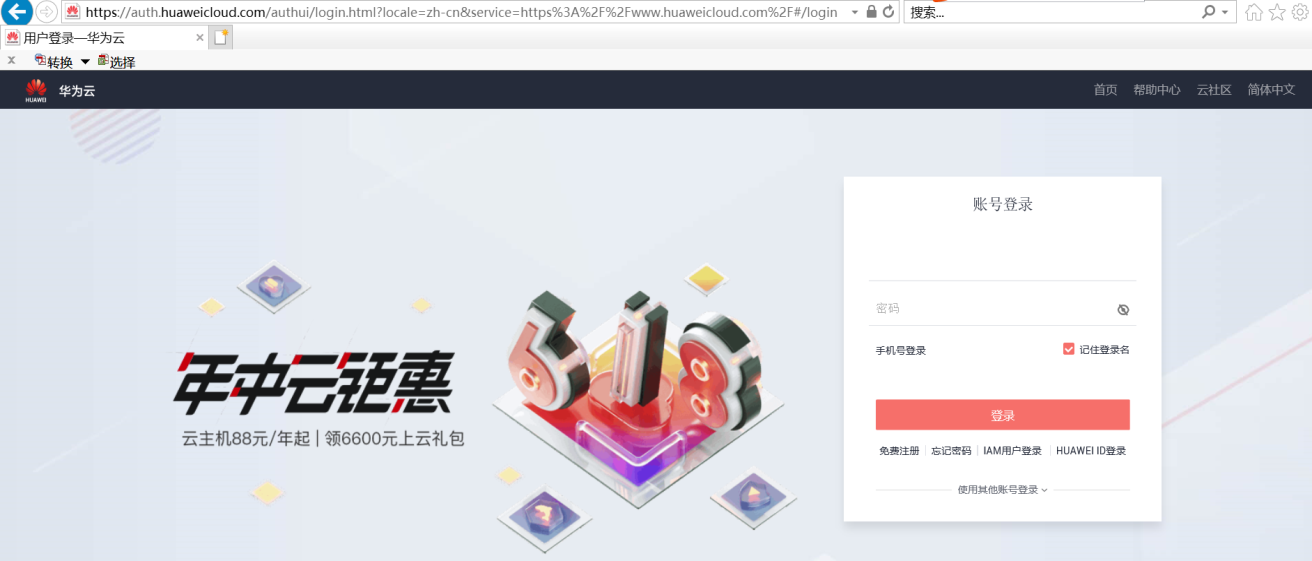
开发者可以搭建基于 ARMv8 架构的开发环境，即原生实验环境。原生实验环境既可以使用如亚马逊(Amazon)、飞腾、华为等公司生产的物理服务器、也可以使用亚马逊(Amazon)云服务器EC2或华为公司鲲鹏系列云服务器。本次使用华为鲲鹏云服务器作为实验环境。

### 购买华为鲲鹏云服务器

登录华为云

打开浏览器，输入华为云的域名：<https://www.huaweicloud.com，点击右上角登录按钮>，输入用户名与密码。





立即购买弹性云服务器（ECS）

选择产品 ->基础服务 ->弹性云服务器 ECS ->立即购买。



点击立即购买。

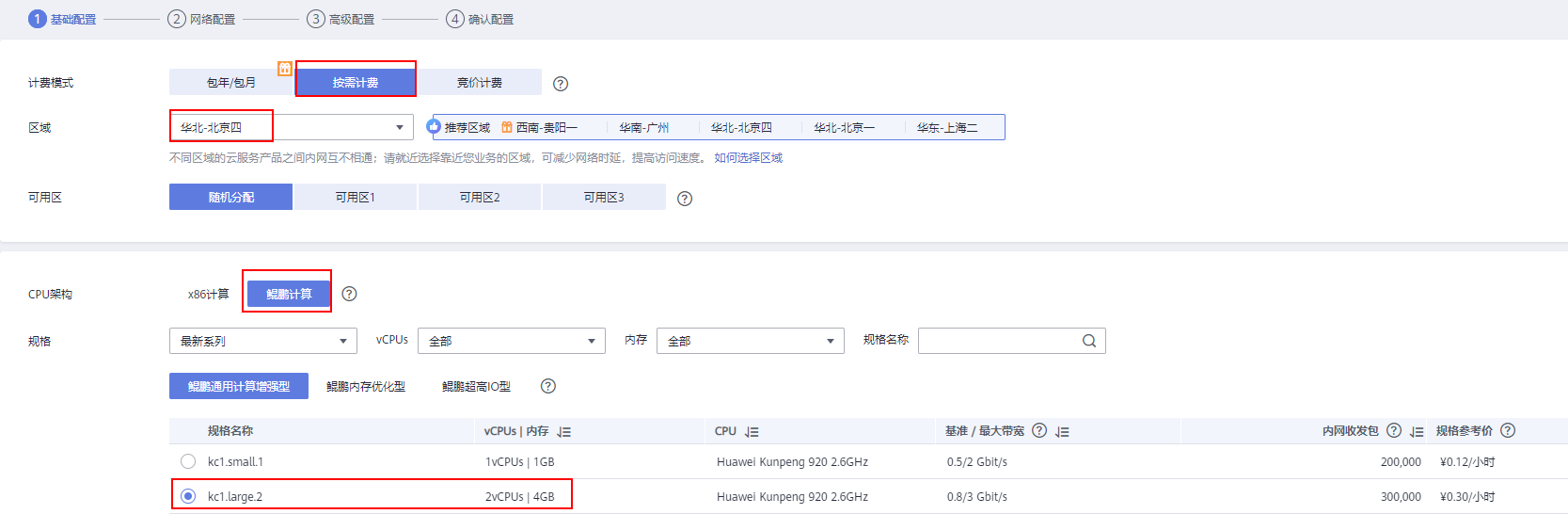


基础配置

按照下表配置进行购买即可。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 计费模式 | 区域 | CPU架构 | 规格 | 镜像 | 系统盘 |
| 按需计费 | 华北-北京四 | 鲲鹏计算 | kc1.large.2 | 公共镜像：openEuler 20.03 | 至少40GB |

参考截图如下：



可选择较小资源,成本更低.对于微机原理实验,kc1.small1.1足够使用。



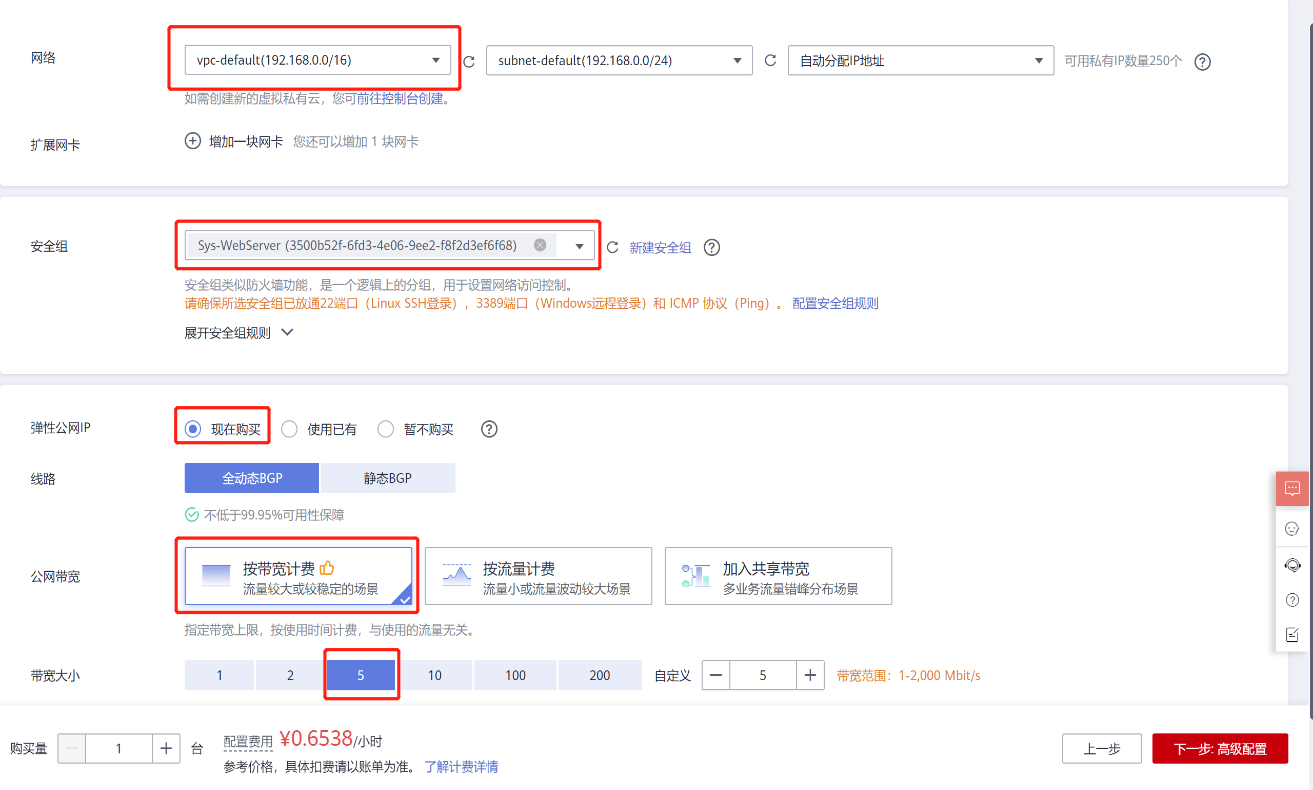
点击下一步：网络配置。

网络配置

按照下表进行网络配置

|  |  |  |
| --- | --- | --- |
| 网络 | 安全组 | 弹性公网IP |
| 默认的VPC | Sys-default | 现在购买，带宽大小选择1Mbits/s |

参考截图如下：



V

V

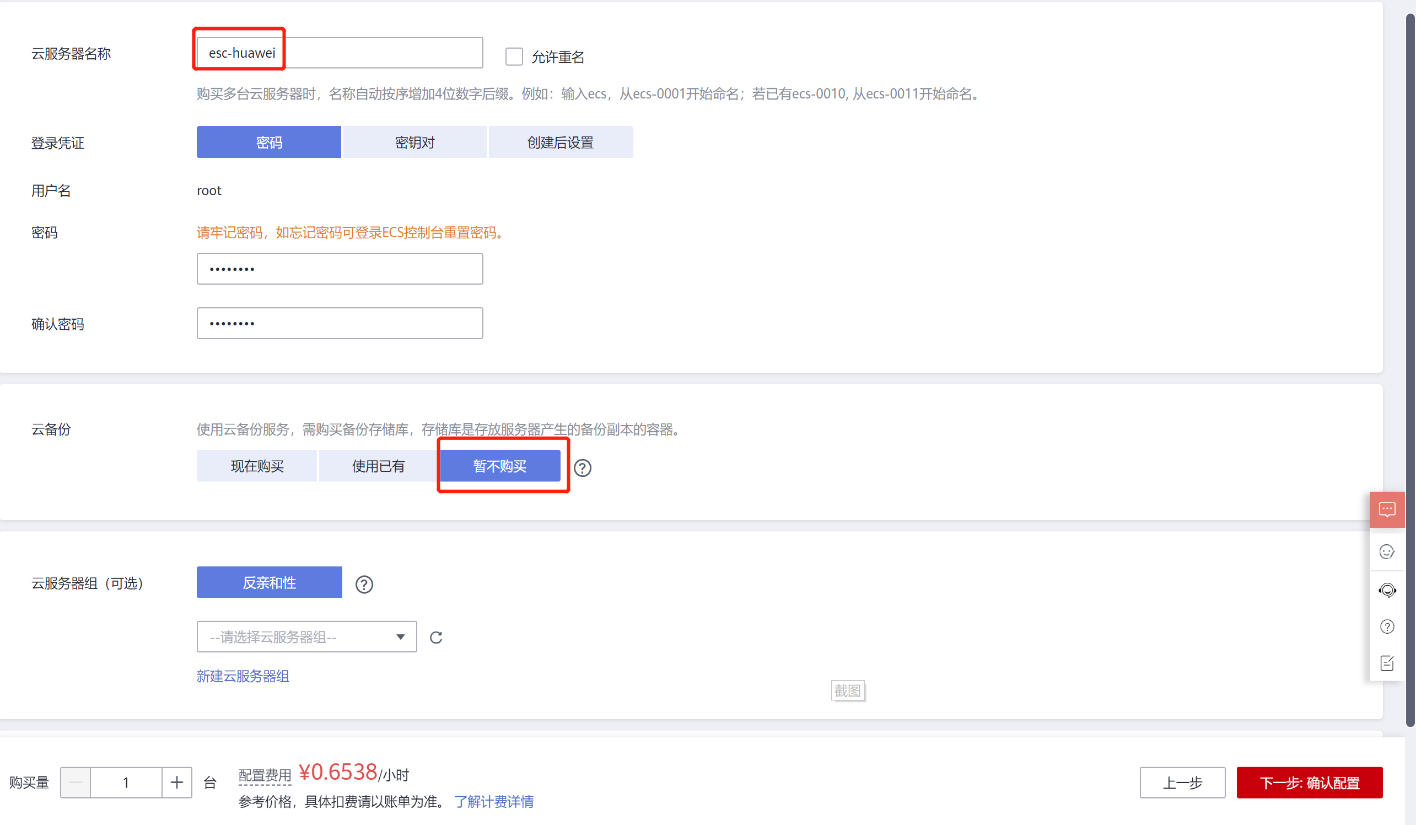
V

选择较小带宽，可以降低费用。

点击下一步：高级配置。

高级配置

设置云服务器名称和密码，云备份选择暂不购买。



V

V

确认配置

勾选我已阅读并同意《华为镜像免责声明》，点击立即购买。



V

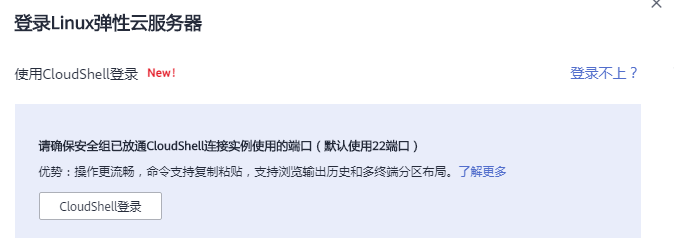
### 环境配置

登录ECS

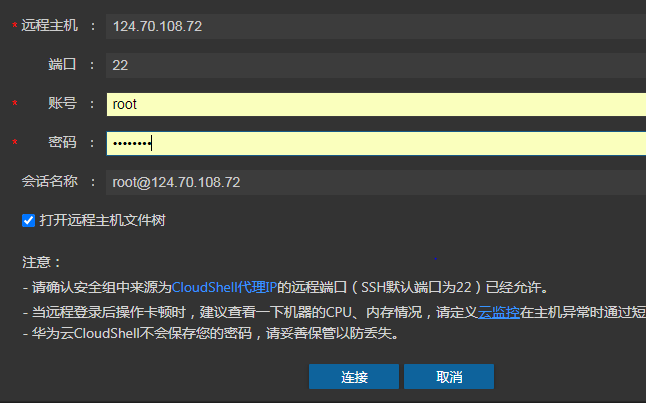
创建好ECS后，可以在控制台 ->弹性云服务器，点击远程登录。



选择第一项登录



输入用户名和密码即可登陆。用户名root 密码



下载GDB工具

GDB GDB工具是Unix/Linux 系统上**标配的调试工具**，主要有如下几个功能：

1、启动程序，可以按照程序员自定义的要求随心所欲的运行程序。

2、设置断点，可让被调试的程序在所指定的[断点](https://baike.baidu.com/item/%E6%96%AD%E7%82%B9)处停住。（断点可以是[条件表达式](https://baike.baidu.com/item/%E6%9D%A1%E4%BB%B6%E8%A1%A8%E8%BE%BE%E5%BC%8F" \t "_blank)）

3、单步调试，连续调试。

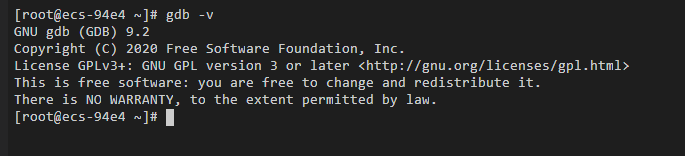
4、查看执行结果。

安装方法，可参考以下文件

“GDB安装教程.pdf” (从GNU官网下载，比较慢)

“华为云ECS配置远程登陆换源安装GDB.PDF” （比较快）

安装完成后，在命令行中输入gdb -v，可查看版本信息。



### 测试汇编代码

### 在linux环境下编写汇编代码，需要用vim文本编辑器，需要下载安装vim编辑器。也可以在左侧文件栏点右键新建文件。

### 使用Vim编辑器调试

### 输入 vim hello.s

vim hello.s

输入以下代码:

.text

.global tart1

tart1:

mov x0,#0

ldr x1,=msg

mov x2,len

mov x8,64

svc #0

mov x0,123

mov x8,93

svc #0

.data

msg:

.ascii "Hello World!\n"

len=.-msg

保存文件，然后通过运行以下命令将其编译为二进制文件

ashello.s -o hello.o

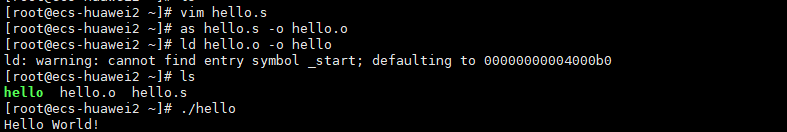
使用以下命令进行链接，输出可执行文件

ld hello.o -o hello

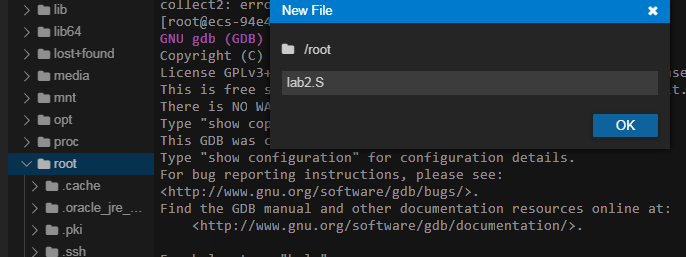
使用以下命令执行hello程序。

./hello

测试流程如下图所示：



示例2：在root文件夹下，新建lab2.S



点击确定

输入以下代码

.data

xx:

.byte 6

yy:

.byte

.text

.global \_start

\_start:

ldr x1,#xx

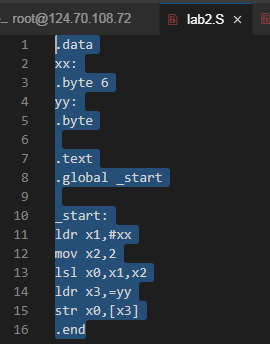
mov x2,2

lsl x0,x1,x2

ldr x3,=yy

str x0,[x3]

.end



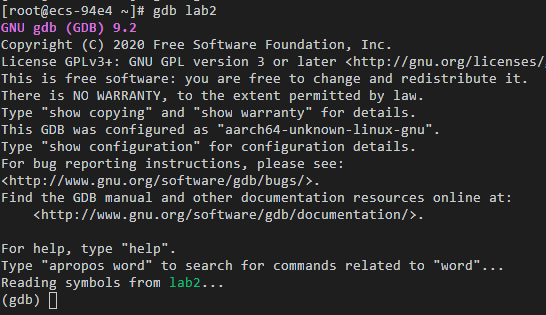
使用Gcc编译器，对lab2.S 进行编译。

gcc -g lab2.S -nostartfiles -o lab2

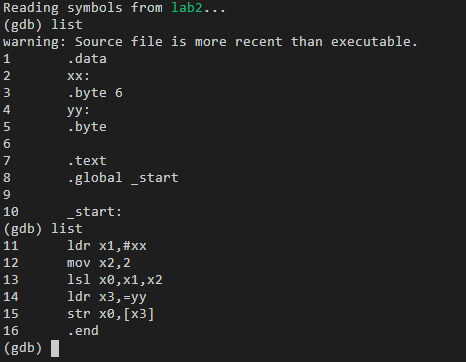


采用gdb来调试lab2.o文件。

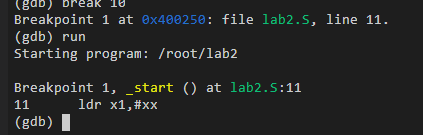
在命令行输入：gdb lab2 1，回车后出现：



2，用多个list命令查看多行源代码

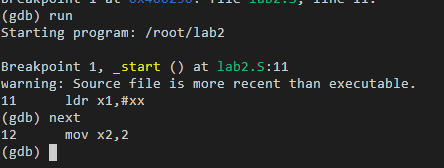


3，在代码的第10行插入第一个断点

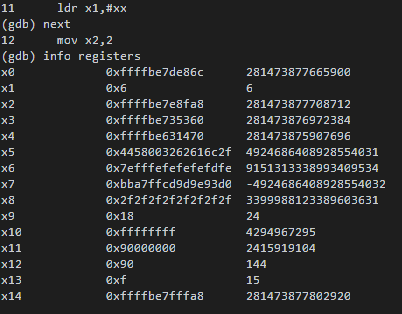


在此基础上，再使用run命令，则表示程序先执行到第10行处就暂停。

使用next 指令

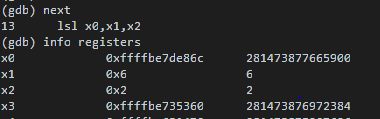


输入info registers命令查看寄存器信息



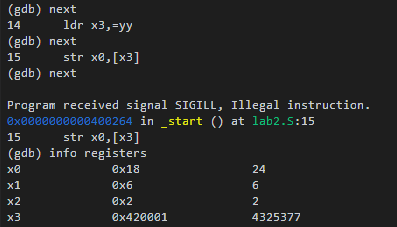
通过LDR X1 ,#xx指令，将变量xx的值赋值给x1寄存器

继续next，然后再输入info registers命令



mov x2，2指令，将立即数2 给X2寄存器赋值 。

点击next 执行 lsl x0,x1,x2 实现将x1左移下x2位，然后赋值到x0寄存器



可以发现 0x06，左移两位，数值为0x18。执行结果正确

ldr x3,=yy 将data段变量yy地址赋值给x3寄存器

str x0,x3将寄存器x0的值，赋值到x3寄存器中的地址中，即将结果传回yy变量。

输入 x/x &yy 查看 yy变量的值。



示例2 调试c与汇编混合编程。

C语言-ArmV8汇编混合编程，实现两个数的相加：

#include <stdio.h>

#include <stdlib.h>

typedef unsigned int u32;

extern int plus(u32, u32);

int main()

{

u32 a = 2;

u32 b = 3;

printf("%d\n", plus(a, b));

return 0;

}

-------------plus.S-----汇编代码函数plus--------------------

#include "plus.h"

ENTRY(plus)

add w0, w1, w0

ret

ENDPROC(plus)

-----------------plus.h----------汇编代码的头文件------------------

#ifndef ENTRY

#define ENTRY(name) \

.globl name ; \

.align 4 ; \

name:

#endif

/\* If symbol 'name' is treated as a subroutine (gets called, and returns)

\* \* then please use ENDPROC to mark 'name' as STT\_FUNC for the benefit of

\* \* static analysis tools such as stack depth analyzer.

\*

\* \*/

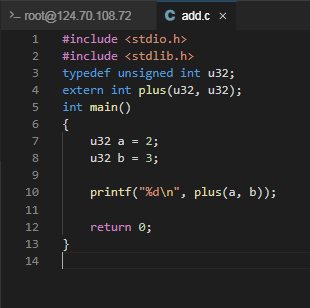
#ifndef ENDPROC

#define ENDPROC(name) \

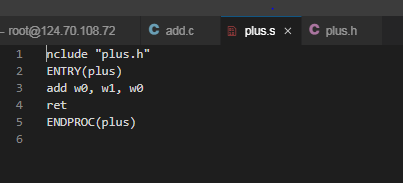
.type name, @function ; \

.size name, .-name

#endif



add.c 文件截图

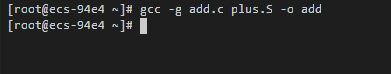


plus.S截图

对例1代码进行带调试信息的编译：

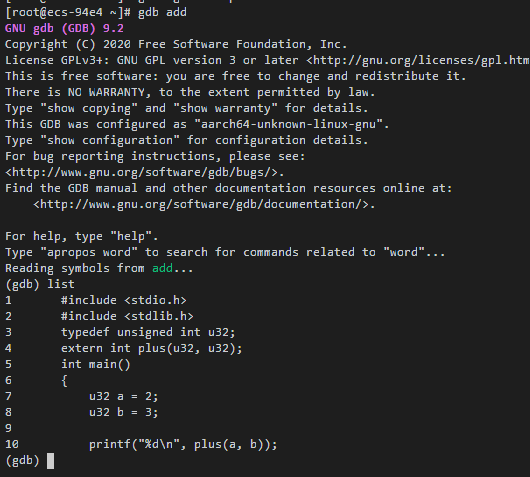
gcc –g add.c plus.S –o add

生成带调试信息的可执行文件add

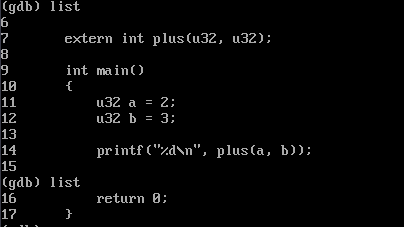


输入 gdb add 命令，对add进行调试，

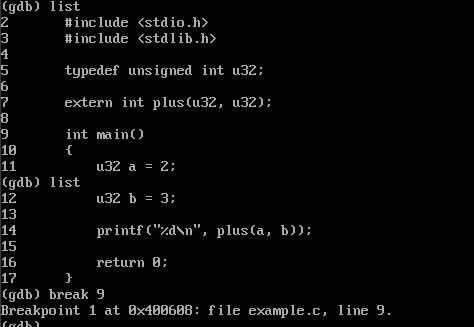
输入list观看代码



2，用多个list命令查看多行源代码

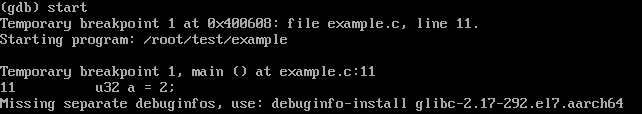


3，在代码的第9行插入第一个断点

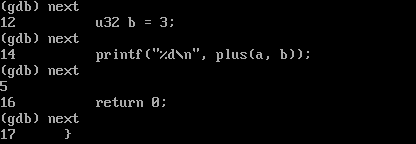


在此基础上，再使用run命令，则表示程序先执行到第9行处就暂停。

4，采用start命令来运行第一行代码

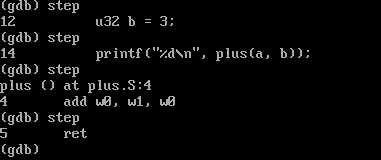


5，采用next命令来单步执行



注意观察：next命令不展开plus函数内部的代码来进行单步，直接跳过。

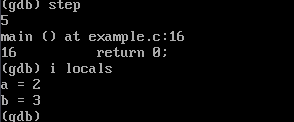
6，采用step命令来单步执行



注意观察：step命令展开plus函数内部的代码来进行单步。

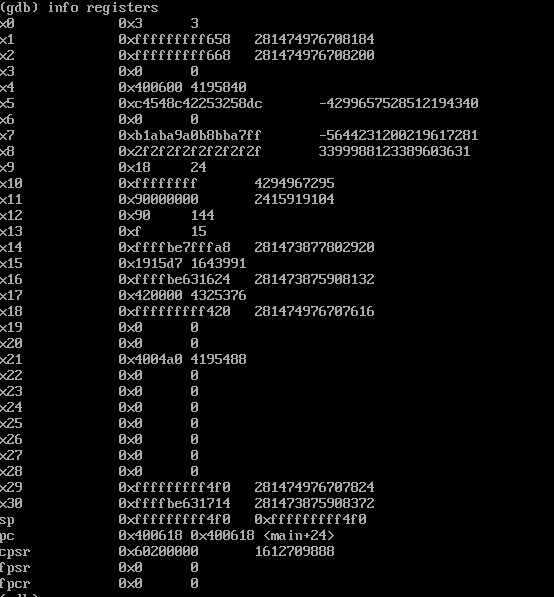
1. 在程序单步执行过程中，可以：

* 使用info locals命令来查看当前函数中局部变量的值

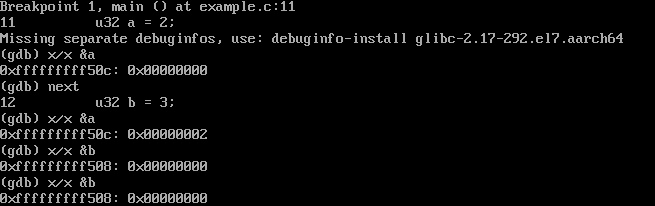


当前函数main中的局部变量为a和b。

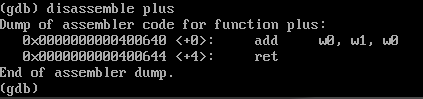
* 使用info registers命令来查看处理器中各寄存器的值。



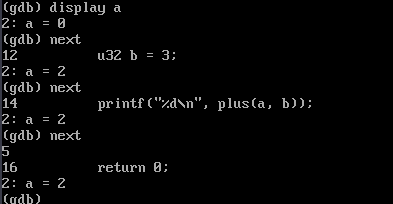
* 使用x命令来查看example程序中变量a和b所在的内存地址（用&a和&b表示）以及取值情况，在下图中采用了x/x &a以及x/x &b命令。



* 可以使用disassemble命令来查看函数如plus的汇编代码

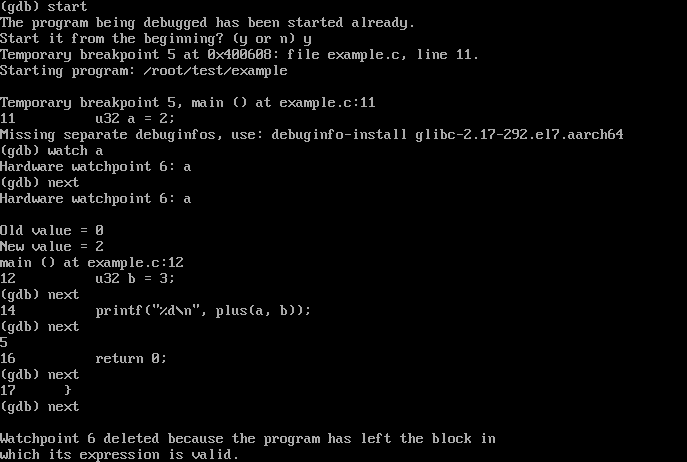


* 可以使用display命令来在每一步查看待观察的变量值，比如display a



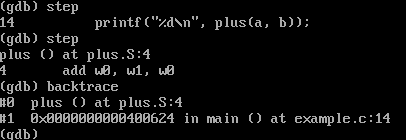
在每一次next之后，都会显示变量a的值。

* 使用watch命令来在变量a的值的变化情况



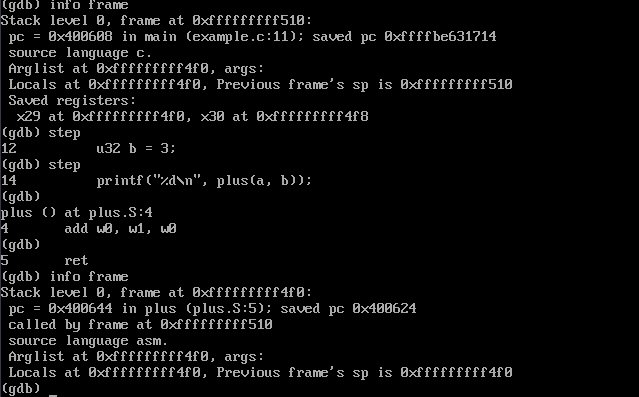
可以看出，在使用watch a命令之后，只有当变量a的值发生变化时才在屏幕上打印出变化前的a值和变化后的a值，其它时刻不在屏幕上打印a的值。在监控效率上，使用watch命令要比display命令的效率高的多。

* 使用backtrace命令来在观察当前函数被调用的情况



当前函数（用“#0”标志）plus被上一级函数（用“#1”）main所调用。

* 使用info frame命令来查看当前栈帧的信息



示例3 多项式运算

试编制一程序，求出下列公式中的w值，并在屏幕输出显示。

；其中x，y，z的值分别存放在定义变量x,y,z中，结果

新建lab3.c

#include <stdio.h>

#include <stdlib.h>

typedef unsigned int u32;

extern int polymath(u32, u32, u32);

int main()

{

    u32 x = 2;

    u32 y = 3;

    u32 z = 1;

    printf("%d\n", polymath(x,y,z));

    return 0;

}

新建polymath.S

**注意：用gcc 编译汇编的文件，汇编文件扩展名必须是S，不能是s，s只做编译未做预处理。**

#include "polymath.h"

ENTRY(polymath)

add w0, w1, w0

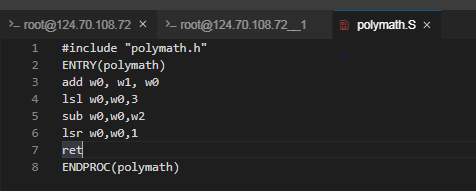
lsl w0,w0,3

sub w0,w0,w2

lsr w0,w0,1

ret

ENDPROC(polymath)

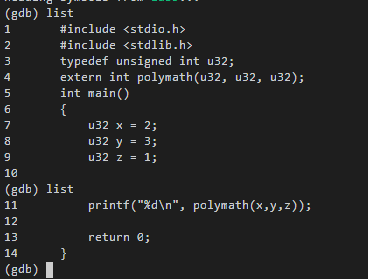


使用gcc 对上述文件编译



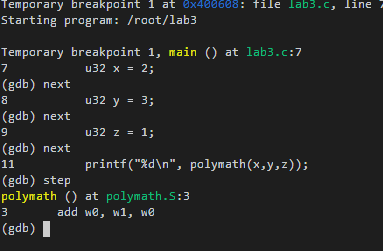
输入gdb lab3命令， 对lab3进行调试

输入list 命令，观看程序代码

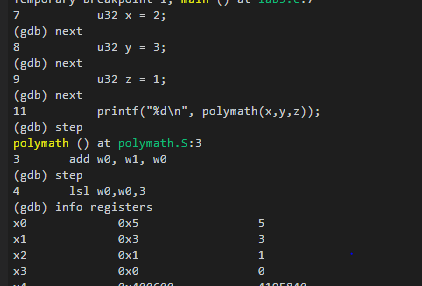


输入 start 命令，从main函数执行程序

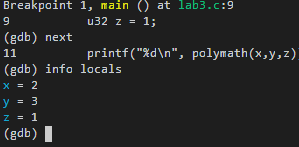
连续next 单步调试。用step指令，进入函数内部，单步调试。



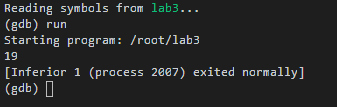
通过info registers 查看运行结果



info locals 查看变量值



程序运行结果在屏幕显示



x=2,y=3,z=1

=19