基础代码示例实验手册

版本 1.0

华为技术有限公司

目录

1 课程介绍

1.1 简介

本手册适用于学习 ARM 平台汇编课程的学生进行实验练习，完成本实验手册后，您将能更加充分理解 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。

1.2 内容描述

本实验指导书通过在华为鲲鹏云服务器上，编译运行 3 个不同功能的示例程序。完成实验操作后，读者会掌握基本的汇编程序编写，ARMv8 开发编译环境的配置以及加深对 ARM 平台的了解。

1.3 读者知识背景

本课程为 ARM 平台汇编基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具备基本的 Linux 命令能力。

1.4 实验环境说明

- 华为鲲鹏云主机、openEuler20.03 操作系统；
- 安装 gcc7.3+版本；
- 每套实验环境可供 1 名学员上机操作。

2 实验 1：hello-world 示例程序

2.1 实验介绍

2.1.1 关于本实验

实现 ARM 平台精简指令集（RISC）编写的 hello-world 程序的编译和运行。

2.1.2 教学目标

掌握 GNU ARM 平台汇编代码的编写以及编译运行方式。

2.1.3 实验内容介绍

在本例子中，两次使用软中断指令 svc 来进行系统调用，系统调用号通过 x8 寄存器传递。在第一次使用 svc 指令来在屏幕上打印一个字符串“Hello”：x0 寄存器用于存放标准屏幕输出 stdout 描述符 0，表明将向屏幕输出一些内容；x1 寄存器用于存放待输出的字符串的首地址 msg；x2 寄存器用于存放待输出字符串的长度 len；x8 寄存器用于存放系统功能调用号 64，即 64 号系统功能即系统写功能 sys_write()，写的目标在 x0 中定义；svc #0 表示是一个系统功能调用。

第二次使用 svc 指令来退出当前程序：x0 寄存器用于存放退出操作码 123，不同的退出操作码将对应不同的退出操作；x8 寄存器用于存放系统功能调用号 93，即 93 号系统功能即系统退出功能 sys_exit()，退出操作码在 x0 中定义；svc #0 表示是一个系统功能调用。

注意：像这种系统功能调用的方式和功能号，都是基于 Arm64 处理器体系结构以及之上所运行的 linux kernel 甚至 BIOS 来共同支持，而不仅仅是 Arm64 架构自身所能完成的。

在.data 部分，加载 msg 和 len 实际上使用的是文字池的方法，即将变量地址放在代码段中不会执行到的位置（因为第二次使用 svc 指令来退出当前程序之后，是不可能将 svc #0 指令之后的内容来当做指令加以执行的），使用时先加载变量的地址，然后通过变量的地址得到变量的值。

本代码是 Aarch64 体系结构的汇编代码，需要在 ArmV8 处理器上运行。寄存器 Xn 都是 Aarch64 体系结构中的寄存器，svc 是 Aarch64 体系结构中的指令。

2.2 实验任务操作指导

2.2.1 创建示例程序源码

以下步骤以在华为鲲鹏云服务器上执行为例。

步骤 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。

```
mkdir hello
cd hello
```

步骤 2　创建示例程序源码 hello.s

执行以下命令，创建示例程序源码 hello.s。

```
vim hello.s
```

代码内容如下：

```
.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
```

```
.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
~
```

步骤 3 进行编译运行

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。

```
as hello.s –o hello.o
ld hello.o –o hello
./hello
```

```
[root@ecs-huawei hello]# ls
hello.s
[root@ecs-huawei hello]# as hello.s -o hello.o
[root@ecs-huawei hello]# ls
hello.o  hello.s
[root@ecs-huawei hello]# ld hello.o -o hello
ld: warning: cannot find entry symbol _start; defaulting to 00000000004000b0
[root@ecs-huawei hello]# ls
hello  hello.o  hello.s
[root@ecs-huawei hello]# ./hello
Hello World!
[root@ecs-huawei hello]#
```

通过上述代码运行，可以看出，编写的 hello-wolrd 示例程序已经在华为鲲鹏云服务器上通过编译和运行，并成功输出结果。

2.2.2 思考题及答案

- 思考下同样的代码在 X86 平台能否运行，为什么？

参考答案：

不能，因为 X86 平台使用的是复杂指令集(CISC)，而我们实验中使用到的华为鲲鹏云服务器是基于 ARM 平台的，使用的是精简指令集(RISC)，二者的汇编指令差异较大。

3 实验 2：使用 C 语言代码调用汇编程序

3.1 实验介绍

3.1.1 关于本实验

实现 ARM 平台上通过 C 语言源码来调用汇编源码中的代码。

3.1.2 教学目标

掌握在 ARM 平台上使用 C 语言源码来调用汇编源码的方法。

3.1.3 实验内容介绍

该汇编代码是针对 Aarch64 架构的。在汇编程序中，用.global 定义一个全局函数 strcpy1，然后该函数就可以在 C 代码中用 extern 关键字加以声明，然后直接调用。

3.2 实验任务操作指导

3.2.1 创建示例程序源码

以下步骤以在华为鲲鹏云服务器上执行为例。

步骤 1　创建目录

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

```
mkdir called
cd called
```

步骤 2　创建 globalCalling.c 源代码

执行以下命令，创建示例调用 C 语言程序源码 globalCalling.c。

```
vim globalCalling.c
```

代码内容如下：

```
/* globalCalling.c*/
#include <stdio.h>
extern void strcpy1(char *d, const char *s);
```

```
int main()
{
      const char *srcstring="Source string";
      char dststring[]="Destination string";

      printf("Original Status: %s      %s\n",srcstring,dststring);
    strcpy1(dststring,srcstring);
    printf("Modified Status: %s      %s\n",srcstring,dststring);
      return 0;
}
```

```
/* globalCalling.c*/
#include <stdio.h>
extern void strcpy1(char *d, const char *s);
int main()
{
    const char *srcstring="Source string";
    char dststring[]="Destination string";

    printf("Original Status: %s    %s\n",srcstring,dststring);
   strcpy1(dststring,srcstring);
   printf("Modified Status: %s    %s\n",srcstring,dststring);
    return 0;
}
~
~
```

步骤 3 创建 globalCalled.S 源代码

执行以下代码命令，创建被调用的汇编语言程序源码 globalCalled.S。

```
vim globalCalled.S
```

代码内容如下：

```
/* globalCalled.S */
.global strcpy1

# Start the function: strcpy1
strcpy1:
LDRB w2,[X1],#1
STR w2,[X0],#1
CMP w2,#0      //ascii code "NUL" is the last character of a string;
BNE strcpy1
RET
```

```
/* globalCalled.S */
.global strcpy1

# Start the function: strcpy1
strcpy1:
LDRB w2,[X1],#1
STR w2,[X0],#1
CMP w2,#0   //ascii code "NUL" is the last character of a string;
BNE strcpy1
RET
```

3.2.2 进行编译运行

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。

```
gcc globalCalling.c globalCalled.S -o called
./called
```

```
[root@ecs-huawei called]# pwd
/root/called
[root@ecs-huawei called]# ls
globalCalled.S  globalCalling.c
[root@ecs-huawei called]# gcc globalCalling.c globalCalled.S -o called
[root@ecs-huawei called]# ls
called  globalCalled.S  globalCalling.c
[root@ecs-huawei called]# ./called
Original Status: Source string   Destination string
Modified Status: Source string   Source string
[root@ecs-huawei called]#
```

通过上述代码运行，可以看出，编写的使用 C 语言代码调用汇编程序已经在华为鲲鹏云服务器上通过编译和运行，并成功输出结果：

Original Status: Source string Destination string

Modified Status: Source string Source string

3.3 思考题及答案

- 除了使用调用汇编脚本的方式，还有哪些方法可以使用 C 语言调用汇编代码?

参考答案：

可以使用 C 语言中的 asm 关键字来在 C 语言源码中嵌入汇编代码。

4 实验 3：使用 C 语言代码内嵌汇编程序

4.1 实验介绍

4.1.1 关于本实验

实现在 ARM 平台上通过 C 语言代码内嵌汇编代码的方式，将一个整数类型值，以字节为单位从小尾端转到大尾端或者相反的功能。

4.1.2 教学目标

掌握在 ARM 平台上实现 C 语言代码中内嵌汇编代码的方法。

4.1.3 内容介绍

通过 C 语言代码内嵌汇编代码，将一个整数类型值，以字节为单位从小尾端转到大尾端或者相反的功能。例如小尾端时 32bit 整数值用 16 进制表示为 0x12345678，将其以字节为单位转换为大尾端存储后，该值为 0x78563412。

4.2 实验任务操作指导

4.2.1 创建示例程序源码

以下步骤以在华为鲲鹏云服务器上执行为例。

步骤 1　创建目录

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

```
mkdir builtin
cd builtin
```

步骤 2　创建 C 语言内嵌汇编程序源代码

执行以下命令，创建 C 语言内嵌汇编程序源码 globalBuiltin.c。

```
vim globalBuiltin.c
```

代码内容如下：

```
/* globalBuiltin.c*/
```

```
#include <stdio.h>
int main()
{
int val=0x12345678;

 __asm__ __volatile__(
      "mov x3,%1\n"
    "mov w3,w3, ror #8\n"
    "bic w3,w3, #0x00ff00ff\n"

    "mov x4,%1\n"
    "mov w4,w4, ror #24\n"
    "bic w4,w4, #0xff00ff00\n"

    "add w3,w4,w3\n"
    "mov %0,x3\n"

    :"=r"(val)
    :"0"(val)
    :"w3","w4","cc"
    );

printf("out is %x \n",val);
return 0;
}
```

```
#include <stdio.h>
int main()
{
    int val=0x12345678;

    __asm__ __volatile__(
        "mov x3,%1\n"
        "mov w3,w3, ror #8\n"
        "bic w3,w3, #0x00ff00ff\n"

        "mov x4,%1\n"
        "mov w4,w4, ror #24\n"
        "bic w4,w4, #0xff00ff00\n"

        "add w3,w4,w3\n"
        "mov %0,x3\n"

        :"=r"(val)
        :"0"(val)
        :"w3","w4","cc"
    );

    printf("out is %x \n",val);
    return 0;
}
```

4.2.2 进行编译

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。

步骤 1　预处理

```
gcc -E globalBuiltin.c -o globalBuiltin.i
```

步骤 2　编译

```
gcc -S globalBuiltin.i -o globalBuiltin.s
```

步骤 3　汇编

```
gcc -c globalBuiltin.s -o globalBuiltin.o
```

步骤 4　生成可执行文件

```
gcc globalBuiltin.o -o globalBuiltin
```

命令和生成的文件如下：

```
[root@ecs-hw ~]# mkdir builtin
[root@ecs-hw ~]# cd builtin
[root@ecs-hw builtin]# vim globalBuiltin.c
[root@ecs-hw builtin]# [root@ecs-hw builtin]# gcc -E globalBuiltin.c -o globalBuiltin.i
[root@ecs-hw builtin]# gcc -S globalBuiltin.i -o globalBuiltin.s
[root@ecs-hw builtin]# gcc -c globalBuiltin.s -o globalBuiltin.o
[root@ecs-hw builtin]# gcc globalBuiltin.o -o globalBuiltin
[root@ecs-hw builtin]# ls
globalBuiltin  globalBuiltin.c  globalBuiltin.i  globalBuiltin.o  globalBuiltin.s
```

4.2.3 进行运行

运行生成的 globalBuiltin 文件，查看输出结果。

命令如下：

```
./globalBuiltin
```

结果如下：

```
[root@ecs-hw builtin]# ./globalBuiltin
out is 78563412
```

通过上述代码运行，可以看出，编写的 C 语言代码内嵌汇编程序已经在华为鲲鹏云服务器上通过编译和运行，并成功输出结果：out is 78563412。

4.2.4 思考题及答案

- 在 C 代码中内嵌的汇编语句的基本格式为是什么？

参考答案：

__asm__ __volatile__ (“asm code”
： 输出操作数列表
： 输入操作数列表
： clobber 列表
)

说明：《汇编与接口技术》课程配套实验手册中的实验内容由北京交通大学计算机与信息技术学院赵宏智老师提供，华为公司负责实验手册文档的编写。