

<<嵌入式系统>>

实验报告

2022 年 1 月

综合实验：ARM-LINUX 混合编程

一、实验目的：

- 1、掌握 ARM-LINUX 汇编语言编程技术。
- 2、掌握 ARM-LINUX C/C++/ 汇编混合编程技术。
- 3、掌握使用 Makefile 控制工程自动编译。
- 4、掌握 ARM-LINUX 交叉编译、交叉调试方法。

二、实验要求：

编写程序实现以下功能：

- 1、汇编语言实现两个长整数相乘。
- 2、编写 C/C++ 程序调用汇编相乘程序，并输出结果。
- 3、编写 Makefile 控制自动从源代码生成可执行程序。
- 4、在智能小车上运行和调试程序

三、实验原理

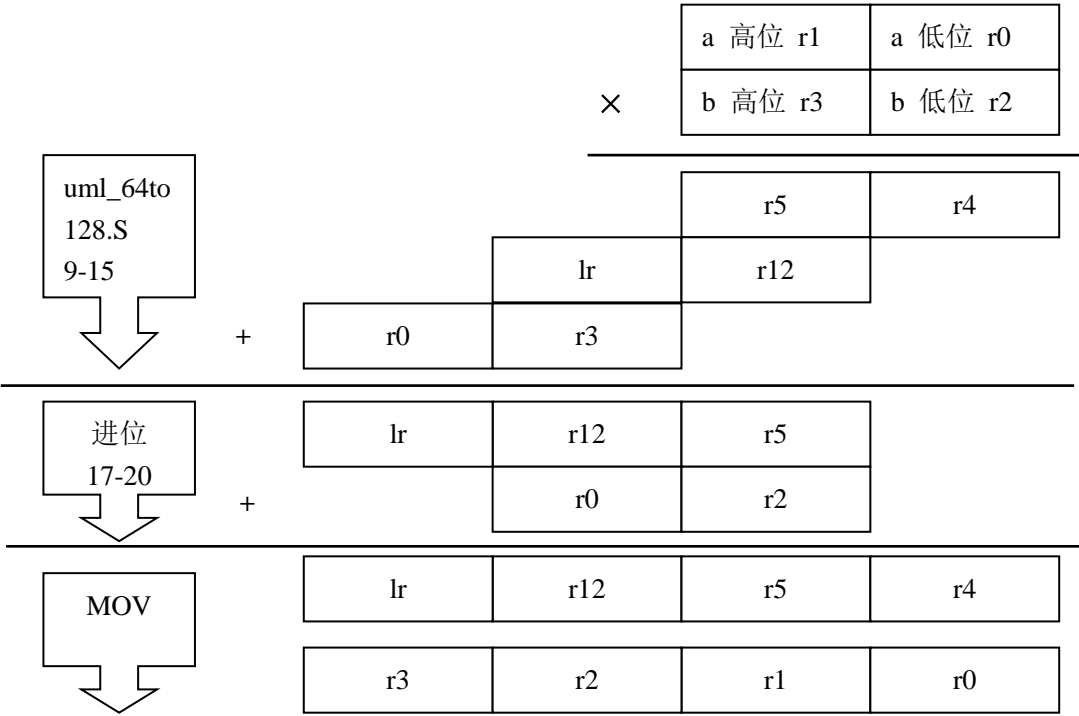
混合语言编程原理与方法

1. **内嵌编译。**在 C/C++ 源程序中，可以嵌入一段 ARM 汇编代码。以 `_arm{}` 标识。
2. **C/C++ 和汇编程序互相调用。**
 - a) **C/C++ 调用汇编程序。**汇编程序使用 `EXPORT` 指示符声明本程序可以被其它程序调用。C 语言程序中使用 `extern` 关键词声明该汇编程序可以被调用，C++ 语言程序可以使用 `extern "C"` 来声明该汇编程序可以被调用。
 - b) **汇编程序调用 C 程序。**在汇编程序中使用 `IMPORT` 指示符声明将要调用 C 程序。
 - c) **汇编程序调用 C++ 程序。**在 C++ 程序中使用关键词 `extern` 声明被调用的 C++ 程序。在汇编程序中使用 `IMPORT` 指示符声明将要调用 C++ 程序。在汇编程序中将参数存放在数据栈中，存放参数的数据栈的单元地址放在 `R0` 寄存器中。
3. **C 程序调用 C++ 程序。**在 C 程序中使用关键词 `extern` 声明调用的 C++ 函数，在 C++ 程序中使用关键词 `extern "C"` 声明被调用的 C++ 函数。

四、程序设计

1、算法描述

使用活动图或者流程图辅以关键步骤说明。



2、源代码及主要步骤注释说明。

main.c

```
1. #include <stdio.h>
2.
3. typedef struct { unsigned r0,r1,r2,r3; } return_type;
4.
5. // declare ARM program
6. extern int uml_64to128 (unsigned long long i, unsigned long long j, return_t
    type* rcv);
7.
8. int main ()
9. {
10.     unsigned long long a, b;
11.     return_type r;
12.
13.     scanf("%llu", &a);
14.     scanf("%llu", &b);
15.
16.     // call ARM program
17.     uml_64to128(a, b, &r);
18.
19.     printf("0X%08x%08x%08x%08x\n", r.r3, r.r2, r.r1, r.r0);
20.     return 0;
21. }
```

uml_64to128.S

```
1. ; 参考课本代码 p155
2. ; 例 5-18
3. .section .text, "x"
4. .global uml_64to128
5.
6. uml_64to128:
7.     STMFD sp!,{r4,r5,fp,lr}
8.     LDR fp,[sp,#16]
9.     UMULL r4,r5,r0,r2 ; i 低位 * j 低位, 结果存入 r4, r5
10.    UMULL r12,lr,r0,r3 ; i 低位 * j 高位, 结果存入 lr, r12
11.    UMULL r3,r0,r1,r3 ; i 高位 * j 高位, 结果存入 r0, r3
12.
13.    ADDS r5,r5,r12 ; 调整 63-32 位
```

```

14.     ADCS r12,lr,r3 ; 调整 95-64 位
15.     ADC lr,r0,#0 ; 调整 127-96 位
16.
17.     UMULL r2,r0,r1,r2 ; i 高位 * j 低位, 结果存入 r0, r2
18.     ADDS r5,r5,r2 ; 调整 63-32 位
19.     ADCS r12,r12,r0 ; 调整 95-64 位
20.     ADC lr,lr,#0 ; 调整 127-96 位
21.
22.     MOV r0,r4
23.     MOV r1,r5
24.     MOV r2,r12
25.     MOV r3,lr
26.
27. ; store in stack and return
28.     STR r0, [fp]
29.     STR r1, [fp, #4]
30.     STR r2, [fp, #8]
31.     STR r3, [fp, #12]
32.     LDMFD sp!,{r4,r5,fp,pc}

```

Makefile

```

1. CC = arm-linux-gcc
2. LD = arm-linux-ld
3. EXEC = uml
4. OBJS = main.o uml_64to128.o
5.
6. CFLAGS +=
7. LDFLAGS +=
8.
9. all: $(EXEC)
10.
11. $(EXEC): $(OBJS)
12.     $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS$(LDLIBS_.$@))
13.     cp $(EXEC) /tftpboot/
14.
15. clean:
16.     -rm -f $(EXEC) *.elf *.gdb *.o

```

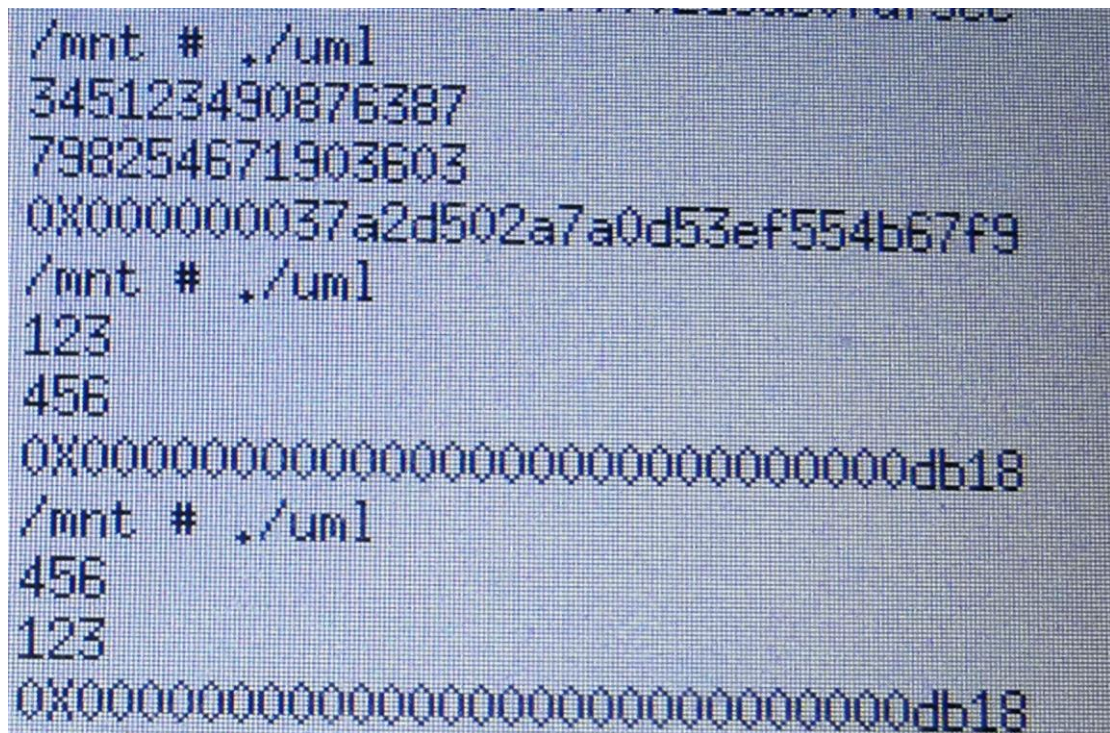
五、实验步骤

对实验的每一个步骤进行详细说明

1. 打开虚拟机。
2. 在虚拟机上编写程序源码。
3. 在主机编译，生成可执行文件 uml。
4. 连接实验设备与 PC 机器的串口和网口。
5. 打开超级终端工具，启动实验设备，等待启动完成后 Enter 进入命令行。
6. 设置虚拟机 IP 地址 192.168.1.180 并启动 nfs 网络服务。
7. 在超级终端上，使用 nfs 工具挂 Fedora 上的 tftpboot 目录到实验设备 linux 下的 /mnt/nfs/ 目录。
8. 在小车上进入目录并运行可执行文件，输入几个样例，查看结果。

六、实验结果

在智能小车屏展示实验结果



```
/mnt # ./uml
345123490876387
798254671903603
0X000000037a2d502a7a0d53ef554b67f9
/mnt # ./uml
123
456
0X00000000000000000000000000000000db18
/mnt # ./uml
456
123
0X00000000000000000000000000000000db18
```