

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： ARM指令系统的理解**

**院 系 ：计算机科学与技术**

**专业班级 ：**

**学 号 ：**

**姓 名 ：**

**指导教师 ：**

**2025 年 4 月 24 日**

**一、实验目的与要求**

通过在ARM虚拟环境下调试执行程序，了解 ARM的指令系统。

实验环境：ARM 虚拟实验环境 QEMU

工具：gcc, gdb 等

**二、实验内容**

**任务1、C与汇编的混合编程**

**任务2、内存拷贝及优化实验**

程序及操作方法 见 <ARM实验任务.pdf>

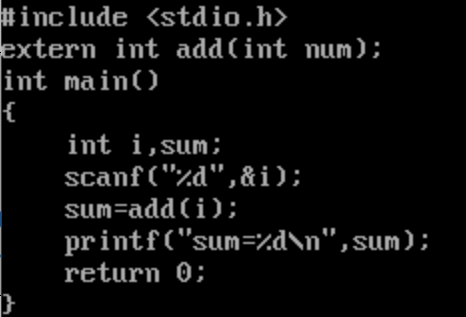
**三、实验记录及问题回答**

**（1）实验任务的实验结果记录**

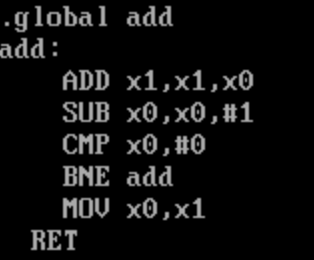
**任务1、C与汇编的混合编程**

## 1、C语言调用汇编实现累加值求值

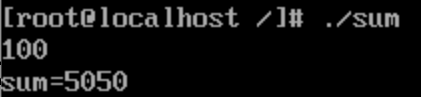
sum.c文件内容如下：



add.s文件内容如下：

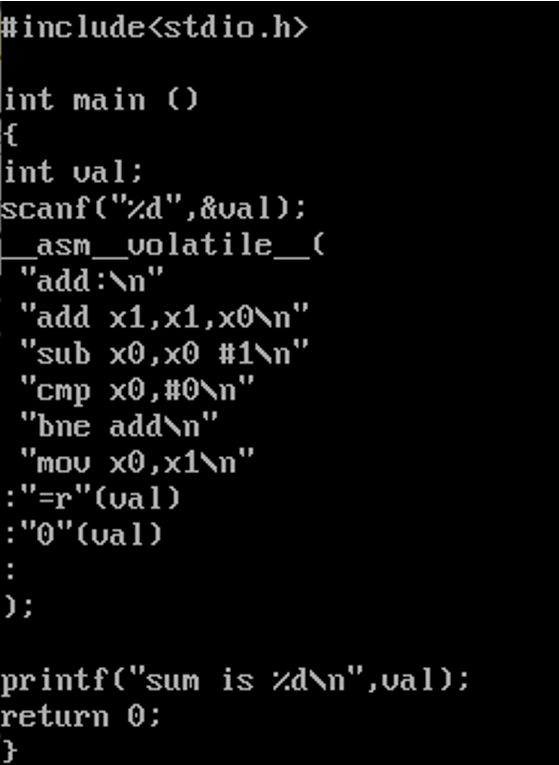


执行结果如下：

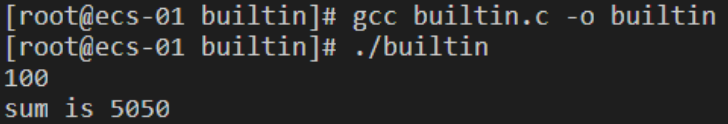


2.C语言内嵌汇编

Builtin.c内容如下：



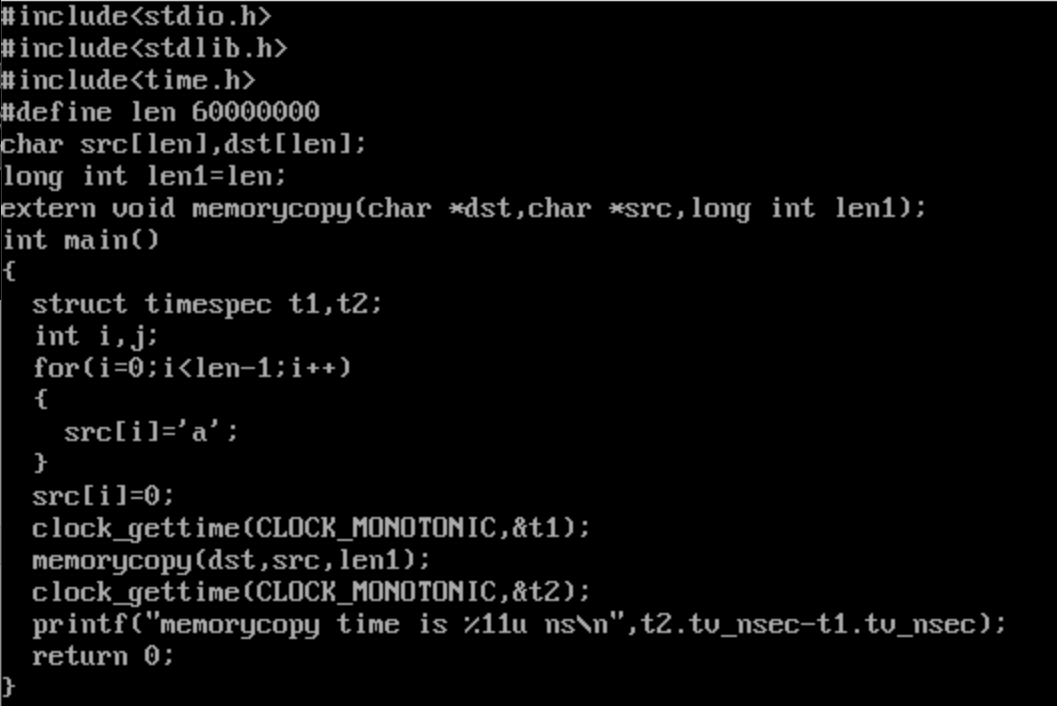
执行结果：



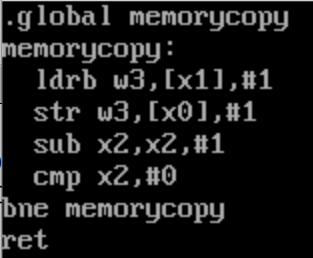
格式为\_\_asm\_\_ \_\_volatile\_\_ (“asm code”:output：input);

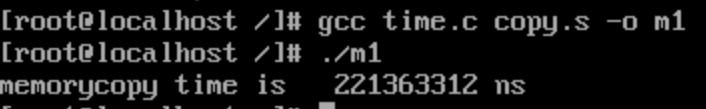
## 任务2、内存拷贝以及优化实验

1．基础代码



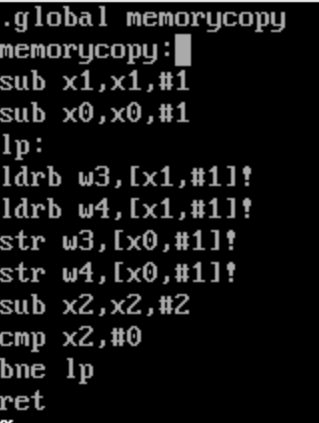
copy.s文件内容

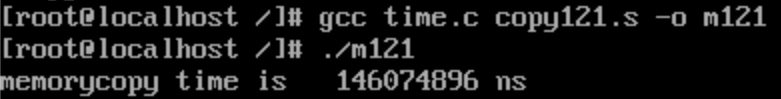




2.优化

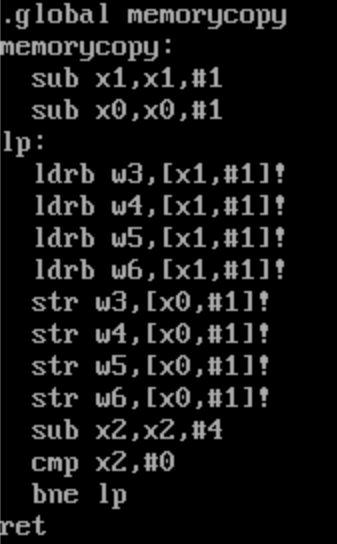
copy121.c内容：



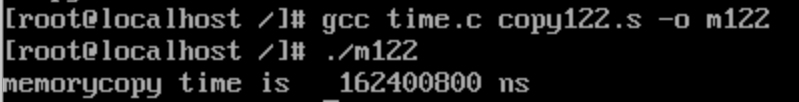


3.四倍展开

基础代码：

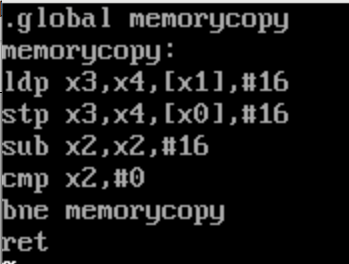


**编译运行结果**

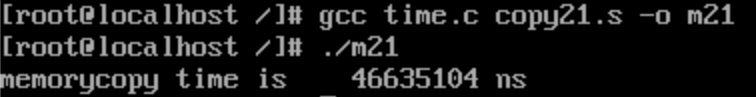


**4.** **内存突发传输方式优化**

**copy21.s文件内容**



**编译运行结果：**



**（2）ARM 指令及功能说明**

**LDR 字数据加载指令**

（注：ARMv8-A处理器体系结构的执行状态有两种：AArch32状态和AArch64状态，用的寄存器名称不同。AArch32状态下32位通用寄存器是R0-R12，AArch64状态下64位通用寄存器用X0-X30(对应低32位W0-W30)）

LDR指令的格式为：

LDR{条件} 目的寄存器，<存储器地址>

LDR指令用于从存储器中将一个32位的字数据传送到目的寄存器中。该指令通常用于从存储器中读取32位的字数据到通用寄存器，然后对数据进行处理。当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

LDR R0，[R1] ；将存储器地址为R1的字数据读入寄存器R0。

LDR R0，[R1，R2] ；将存储器地址为R1+R2的字数据读入寄存器R0。

LDR R0，[R1，＃8] ；将存储器地址为R1+8的字数据读入寄存器R0。

LDR R0，[R1，R2] ！ ；将存储器地址为R1+R2的字数据读入寄存器R0，并将新地址R1＋R2写入R1。

LDR R0，[R1，＃8] ！ ；将存储器地址为R1+8的字数据读入寄存器R0，并将新地址R1＋8写入R1。

LDR R0，[R1]，R2 ；将存储器地址为R1的字数据读入寄存器R0，并将新地址R1＋R2写入R1。

LDR R0，[R1，R2，LSL＃2] ！ ；将存储器地址为R1＋R2×4的字数据读入寄存器R0，并将新地址R1＋R2×4写入R1。

LDR R0，[R1]，R2，LSL＃2 ；将存储器地址为R1的字数据读入寄存器R0，并将新地址R1＋R2×4写入R1。

**LDRB 字节数据加载指令**

LDRB指令的格式为：

LDR{条件}B 目的寄存器，<存储器地址>

LDRB指令用于从存储器中将一个8位的字节数据传送到目的寄存器中，同时将寄存器的高24位清零。该指令通常用于从存储器中读取8位的字节数据到通用寄存器，然后对数据进行处理。当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

LDRB R0，[R1] ；将存储器地址为R1的字节数据读入寄存器R0，并将R0的高24位清零。

LDRB R0，[R1，＃8] ！ ；将存储器地址为R1＋8的字节数据读入寄存器R0，并将新地址R1＋8写入R1。

**LDRH 半字数据加载指令**

LDRH指令的格式为：

LDR{条件}H 目的寄存器，<存储器地址>

LDRH指令用于从存储器中将一个16位的半字数据传送到目的寄存器中，同时将寄存器的高16位清零。该指令通常用于从存储器中读取16位的半字数据到通用寄存器，然后对数据进行处理。当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

LDRH R0，[R1] ；将存储器地址为R1的半字数据读入寄存器R0，并将R0的高16位清零。

LDRH R0，[R1，＃8] ；将存储器地址为R1＋8的半字数据读入寄存器R0，并将R0的高16位清零。

LDRH R0，[R1，R2] ；将存储器地址为R1＋R2的半字数据读入寄存器R0，并将R0的高16位清零。

**STR 字数据存储指令**

STR指令的格式为：

STR{条件} 源寄存器，<存储器地址>

STR指令用于从源寄存器中将一个32位的字数据传送到存储器中。该指令在程序设计中比较常用，且寻址方式灵活多样，使用方式可参考指令LDR。

指令示例：

STR R0，[R1]，＃8 ；将R0中的字数据写入以R1为地址的存储器中，并将新地址R1＋8写入R1。

STR R0，[R1，＃8] ；将R0中的字数据写入以R1＋8为地址的存储器中。

STR R0，[R1，＃8] ！ ；将R0中的字数据写入以R1为地址的存储器中，并将新地址R1＋8写入R1。

**STRB 字节数据存储指令**

STRB指令的格式为：STR{条件}B 源寄存器，<存储器地址>

STRB指令用于从源寄存器中将一个8位的字节数据传送到存储器中。该字节数据为源寄存器中的低8位。

指令示例：

STRB R0，[R1] ；将寄存器R0中的字节数据写入以R1为地址的存储器中。

STRB R0，[R1，＃8] ；将寄存器R0中的字节数据写入以R1＋8为地址的存储器中。

**STRH 半字数据存储指令**

STRH指令的格式为：

STR{条件}H 源寄存器，<存储器地址>

STRH指令用于从源寄存器中将一个16位的半字数据传送到存储器中。该半字数据为源寄存器中的低16位。

指令示例：

STRH R0，[R1] ；将寄存器R0中的半字数据写入以R1为地址的存储器中。

STRH R0，[R1，＃8] ；将寄存器R0中的半字数据写入以R1＋8为地址的存储器中。

**LDP/STP指令**

是 LDP/STP 的衍生, 可以同时读/写两个寄存器，并访问16个字节的内存数据,

指令示例：

LDP x3,x4,[x1,#16] ；读取x1+16地址后的16个字节的数据写入x3、x4寄存器中。

LDP x3,x4,[x1],#16 ；读取x1地址后的16个字节的数据写入x3、x4寄存器中，并更新x1=x1+16。

LDP x9,x10,[x1,#64]! ；读取x1+64地址后的16个字节的数据写入x9、x10寄存器中。并将新地址x1＋64写入x1。

STP x3,x4,[x0,#16] ；将x3、x4中的数据写入以x0+16地址后的16个字节地址中

STP x3,x4,[x0],#16 ；将x3、x4中的数据写入以x0地址后的16个字节地址中

并更新x0=x0+16。

STP x9,x10,[x0,#64]! ；将x9、x10中的数据写入以x0+64地址后的16个字节地址 中，并将新地址x0＋64写入x0。

**四、体会**

在本次实验中我分析C代码如何编译成对应的ARM汇编指令，对比高层语言和底层机器指令的映射关系，增强了对编译器行为的理解，还掌握了嵌入式汇编的基本语法及寄存器约束的用法，例如如何使用 \_asm\_\_volatile\_嵌入ARM指令，如何通过输入输出约束将C语言变量和汇编寄存器映射，学会分析代码中内存访问的瓶颈点，理解对齐访问对性能的影响，以及如何优化代码以充分利用CPU的流水线和缓存，在实验中我们将ARM指令集、内存访问机制、编译原理等知识通过实验加以验证和实践，加深了对理论知识的理解，从代码设计、调试到优化的整个过程，提高了分析和解决实际问题的能力。

综上所述, 通过本次实验，我不仅掌握了ARM指令系统和优化编程的技巧，还提升了C与汇编混合编程和调试分析能力，这将对嵌入式开发和系统编程相关的学习与实践产生重要影响