

**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 ： CS2306**

**学 号 ： U202315696**

**姓 名 ： 彭冲**

**指导教师 ： 朱虹**

**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**

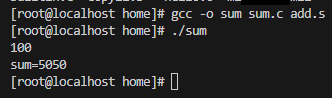
**任务2、内存拷贝及优化实验**

程序及操作方法 见 <ARM实验任务.pdf>

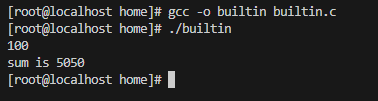
**三、实验记录及问题回答**

**（1）实验任务的实验结果记录**

**2.1 C与汇编的混合编程**

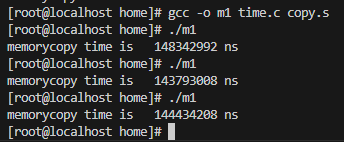


图表 1 C语言调用汇编实现累加和求值

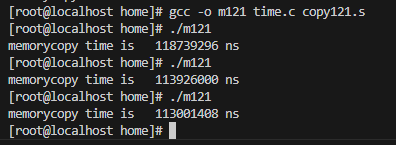


图表 2 C语言内嵌汇编

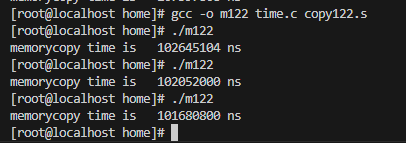
**2.2 内存拷贝及优化**



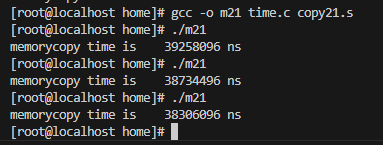
图表 3 基础代码



图表 4 2倍循环展开优化



图表 5 4倍循环展开优化



图表 6 内存突发传输方式优化

**（2）ARM 指令及功能说明**

**2.1 C与汇编的混合编程**

本部分使用汇编编写了累加求和函数，汇编源码为：

1. .global add

2. add:

3. mov x1,#0

4. loop:

5. add x1,x1,x0

6. sub x0,x0,#1

7. cmp x0,#0

8. bne loop

9. mov x0,x1

10. ret

sum.c中add函数原型为int add(int num)。在ARM64(AArch64)调用约定中，参数 num由x0寄存器传递，返回值由x0寄存器返回。

**.global add**

声明add为全局符号，可在其他模块中调用。

**add:**

**mov x1,#0**

查阅arm a64指令集手册，对应的指令格式为MOV <Xd|SP>, #<imm>，将立即数写入到寄存器。这里将寄存器x1置为0，作为累加器。

**loop:**

**add x1,x1,x0**

对应的指令格式为ADD <Xd|SP>, <Xn|SP>, #<imm>{, <shift>}，将两个寄存器或寄存器与立即数的值相加，并将结果写入目标寄存器。这里将x1和x0的值相加，结果存回 x1，实现累加操作。

**sub x0,x0,#1**

对应的指令格式为 SUB <Xd|SP>, <Xn|SP>, #<imm>，用于将寄存器的值与立即数相减，并将结果写入目标寄存器。这里将x0的值减1，结果存回x0。x0既是累加数，同时作为循环计数器，自减1。

**cmp x0,#0**

对应的指令格式为 CMP <Xn|SP>, #<imm>，本质上等价于SUBS指令，但结果不保存，只影响条件标志位。这里将x0与0比较，为后续条件跳转做准备。

**bne loop**

对应的指令格式为 B.<cond> <label>，BNE 表示“不等于”时跳转。这里如果x0不等于0，则跳转回loop标签，继续循环。

**mov x0,x1**

对应的指令格式为 MOV <Xd|SP>, <Xm|SP>，用于将一个寄存器的值复制到另一个寄存器。这里将 x1 的累加结果赋值给 x0，作为返回值。

**ret**

从当前子程序返回到调用者。

**2.2 内存拷贝及优化**

**1.基础代码**

1. .global memorycopy

2. memorycopy:

3. ldrb w3,[x1],#1

4. str w3,[x0],#1

5. sub x2,x2,#1

6. cmp x2,#0

7. bne memorycopy

8. ret

time.c中函数memorycopy原型为void memorycopy(char\*dst, char\*src, long int len1)，在ARM64(AArch64)调用约定中，3个参数从左至右依次由x0,x1,x2寄存器传值。

关键指令：

**ldrb w3, [x1], #1**

对应的指令格式：LDRB <Wt>, [<Xn|SP>], #<imm>。此处为后索引方式，从地址x1（即src）指向的内存读取一个字节（8位）到w3，然后x1自动加1，实现源地址递增。

**str w3, [x0], #1**

对应的指令格式：STR <Wt>, [<Xn|SP>], #<imm>。此处为后索引方式，将w3的值存储到x0（即dst）指向的内存地址，然后x0自动加1，实现目标地址递增。

通过这两条指令，每次搬运一个字节，直到复制完指定字节数，运行时间约145ms。

**2. 2倍展开**

关键指令：

**ldrb w3,[x1,#1]!**

**ldrb w4,[x1,#1]!**

对应的指令格式：LDRB <Wt>, [<Xn|SP>, #<imm>]!。此处为自动索引方式，从x1+1地址读取一个字节到w3和w4，每次将x1加1，实现两字节读取。

**str w3, [x0, #1]!**

**str w4, [x0, #1]!**

对应的指令格式：STR <Wt>, [<Xn|SP>, #<imm>]!。此处为自动索引方式，将w3的值存储到x0+1指向的内存地址，然后x0自动加1，实现两字节写入。

通过这两条指令，每次搬运4个字节，直到复制完指定字节数，运行时间约115ms，相比基础代码提升26.1%。

**3. 4倍展开**

与上述2倍展开同理，每次搬运4个字节，直到复制完指定字节数，运行时间约102ms，相比基础代码提升42.2%。

**4. 内存突发传输方式优化**

使用位宽更长的ldp和stp指令，对应的格式：对应的指令格式为 LDP <Xt1>, <Xt2>, [<Xn|SP>] , #<imm>和STP <Xt1>, <Xt2>, [<Xn|SP>] , #<imm>，每次搬运16个字节，直到复制完指定字节数，运行时间约39ms，相比基础代码提升271.8%。

**四、体会**

通过本次基于QEMU的ARM虚拟环境搭建与汇编编程实验，我对ARM架构开发环境部署、汇编语言特性及代码优化有了更深刻的理解，同时也积累了Linux系统操作和调试工具的使用经验。

openEuler系统的安装与yum源配置让我接触到国产开源操作系统的成熟生态，其针对ARM架构的优化提升了开发效率。openEuler不仅是一个​​针对ARM架构优化的操作系统​​，更代表国产开源生态的技术突破。openEuler的成熟度与开放性让我对国产操作系统的未来充满信心。