華中科技大學

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 ： 计算机本硕博 2301 班

学　　号 ： U202315763

姓　　名 ： 王家乐

指导教师 ： 李海波

2024 年 11 月 5 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〉

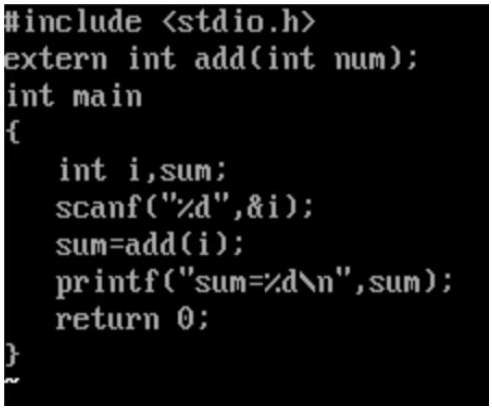
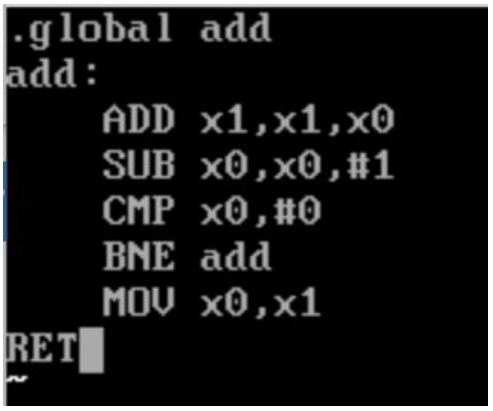
三、实验记录及问题回答

（1）实验任务的实验结果记录

1、C 与汇编的混合编程

1.1、C 语言调用汇编实现累加和求值

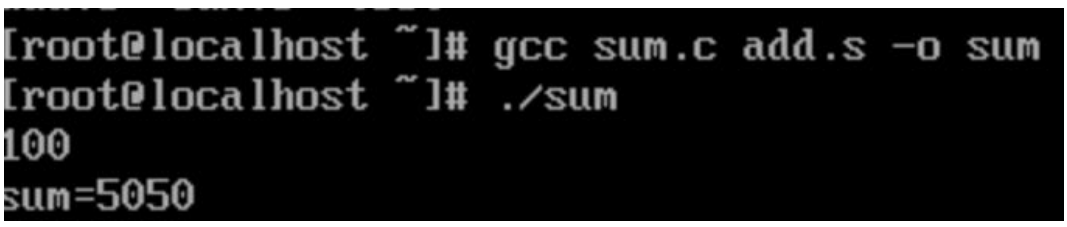
输入一个正整数，输出从 0 到该正整数的所有正整数的累加和，输入输出功能在 C 代码中实现，计算功能通过调用汇编函数实现。需要传入的参数是输入的正整数，汇编传出的参数为累加和，因此只用到一个 x0 寄存器即可实现参数传递功能。

执行命令 vi sum.c 编写 C 程序	执行命令 vim add.s 编写汇编程序
	

```
#include <stdio.h>
extern int add(int num);
int main
{
   int i,sum;
   scanf("%d",&i);
   sum=add(i);
   printf("sum=%d\n",sum);
   return 0;
}
~
```

```
.global add
add:
    ADD x1,x1,x0
    SUB x0,x0,#1
    CMP x0,#0
    BNE add
    MOV x0,x1
RET
~
```

使用 gcc 编译生成可执行文件，程序执行结果正确



```
[root@localhost ~]# gcc sum.c add.s -o sum
[root@localhost ~]# ./sum
100
sum=5050
```

1.2、C 语言内嵌汇编

执行命令 vi builtin.c 编写 C 程序

```
#include <stdio.h>
int main(){
int val;
scanf("%d",&val);
__asm__ __volatile__(
"add:\n"
"ADD x1,x1,x0\n"
"SUB x0,x0,#1\n"
"CMP x0,#0\n"
"BNE add\n"
"MOV x0,x1\n"
:"=r"(val)
:"0"(val)
:
);
printf("sum is %d \n",val);
return 0;
}
```

使用 gcc 编译生成可执行文件，程序执行结果正确

```
[root@localhost ~]# gcc builtin.c -o builtin
[root@localhost ~]# ./builtin
100
sum is 5050
```

2、内存拷贝及优化

优化效果通过计算对应代码段的执行时间来判断。具体方案是通过 C 语言调用汇编，在 C 代码中计算时间，在汇编代码中设计不同的方案，对比每种方案的执行时间，判断优化效果。本示例的优化针对内存读写，示例程序功能是内存拷贝，拷贝功能在汇编函数中实现。

执行命令 vi time.c 编写 C 语言计时程序

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define len 60000000
char src[len],dst[len];
long int len1=len;
extern void memorycopy(char *dst,char *src,long int len1);
int main()
{
struct timespec t1,t2;
int i,j;
for(int i=0;i<len;i++)
{
src[i]='a';
}
src[i]=0;
clock_gettime(CLOCK_MONOTONIC,&t1);
memorycopy(dst,src,len1);
clock_gettime(CLOCK_MONOTONIC,&t2);
printf("memorycopy time is %11u ns\n",t2.tv_nsec-t1.tv_nsec);
return 0;
}
```

执行命令 vi copy.s 编写优化前的原始汇编代码

```
.global memorycopy
memorycopy:
ldrb w3,[x1],#1
str w3,[x0],#1
sub x2,x2,#1
cmp x2,#0
bne memorycopy
ret
```

编译并运行，执行时间为 206401191ns

```
[root@localhost ~]# gcc time.c copy.s -o m1
[root@localhost ~]# ./m1
memorycopy time is   206401191 ns
```

创建 2 倍展开优化 copy121.s 文件

```
.global memorycopy
memorycopy:
sub x1,x1,#1
sub x0,x0,#1
lp:
ldrb w3,[x1,#1]!
ldrb w4,[x1,#1]!
str w3,[x0,#1]!
str w4,[x0,#1]!
sub x2,x2,#2
cmp x2,#0
bne lp
ret
```

编译并运行，执行时间为 152543555ns

```
[root@localhost ~]# gcc time.c copy121.s -o m121
[root@localhost ~]# ./m121
memorycopy time is   152543555 ns
```

创建 2 倍展开优化 copy122.s 文件

```
.global memorycopy
memorycopy:
sub x1,x1,#1
sub x0,x0,#1
lp:
ldrb w3,[x1,#1]!
ldrb w4,[x1,#1]!
ldrb w5,[x1,#1]!
ldrb w6,[x1,#1]!
str w3,[x0,#1]!
str w4,[x0,#1]!
str w5,[x0,#1]!
str w6,[x0,#1]!
sub x2,x2,#4
cmp x2,#0
bne lp
ret
```

编译并运行，执行时间为 147881698ns

```
[root@localhost ~]# gcc time.c copy122.s -o m122
[root@localhost ~]# ./m122
memorycopy time is   147881698 ns
```

创建内存突发传输优化 copy21.s 文件

```
.global memorycopy
memorycopy:
ldp x3,x4,[x1],#16
stp x3,x4,[x0],#16
sub x2,x2,#16
cmp x2,#0
bne memorycopy
ret
```

编译并运行，执行时间为 45528429ns

```
[root@localhost ~]# gcc time.c copy21.s -o m21
[root@localhost ~]# ./m21
memorycopy time is    45528429 ns
```

一次对 16 字节读写程序执行效率明显优于单字节读写

（2）ARM 指令及功能说明

查阅《ISA_A64_xml_A_profile-2023-09.pdf》，指出 10 条不同指令（存数、取数、算术运算、转移指令、函数调用等都应覆盖）的功能。

1. LDR (Load Register)

   功能：从指定的内存地址加载数据到寄存器中。用于数据读取。

   示例：LDR X0, [X1, #8] 从地址 X1+8 处加载数据到 X0。

2. STR (Store Register)

   功能：将寄存器中的数据存储到指定的内存地址。用于数据写入。

   示例：STR X0, [X1, #8] 将 X0 中的数据存储到地址 X1+8 处。

3. ADD (Addition)

   功能：将两个寄存器的值相加，并将结果存入目标寄存器。用于整数加法运算。

   示例：ADD X0, X1, X2 将 X1 和 X2 的值相加并存入 X0。

4. SUB (Subtract)

   功能：将两个寄存器的值相减，并将结果存入目标寄存器。用于整数减法运算。

   示例：SUB X0, X1, X2 计算 X1-X2 并存入 X0。

5. MOV (Move)

   功能：将一个寄存器的值传送到另一个寄存器。用于数据传送。

   示例：MOV X0, X1 将 X1 中的值复制到 X0。

6. MUL (Multiply)

   功能：计算两个寄存器值的乘积，并将结果存入目标寄存器。用于整数乘法运算。

   示例：MUL X0, X1, X2 将 X1 和 X2 相乘并存到 X0。

7. CMP (Compare)

   功能：比较两个寄存器的值，为条件判断设置标志。常用于分支判断。

   示例：CMP X1, X2 比较 X1 和 X2 的值。

8. B (Branch)

   功能：无条件跳转到指定的标签地址。用于控制流程。

   示例：B label 跳转到 label。

9. BL (Branch with Link)

   功能：跳转到子程序地址，同时将返回地址存入链接寄存器（LR）。用于函数调用。

   示例：BL function 跳转到 function 函数地址。

10. RET (Return from Subroutine)

    功能：从子程序返回到调用地址，返回地址从链接寄存器（LR）读取。

    示例：RET 从当前子程序返回。

四、体会

在本次实验中，我深入理解了 ARM 指令集的工作机制及其在不同场景下的优化效果。从任务设计和调试过程中，我掌握了基本的 ARM 汇编指令和寄存器操作，还在 C 与汇编的混合编

程中积累了许多实践经验。

内存拷贝的优化实验让我更直观地理解了如何通过汇编级的优化手段来提升程序性能。通过对比不同优化方法（如指令展开、突发传输等）对执行时间的影响，我认识到细微的代码优化能带来显著的性能差异。在现代计算中，尤其是在嵌入式系统和性能敏感的应用中，充分利用指令集的潜力至关重要。实验中，突发传输优化将内存拷贝的效率提升了几个数量级，这深刻印证了硬件和指令优化在性能提升上的重要作用。

本次实验不仅强化了我对 ARM 指令系统的理解，还让我体会到了低级编程的魅力和挑战，为今后更复杂的底层编程和系统优化打下了坚实的基础。