

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 实验基础**

**院 系 ：计算机科学与技术**

**专业班级 ： 本硕博202001**

**学 号 ： U202115674**

**姓 名 ： 姚晨炫**

**指导教师 ： 许向阳**

**2022 年 9 月 26 日**

**一、实验目的与要求**

(1) 熟练掌握程序开发平台(VS2019) 的基本用法，包括程序的编译、链接和调试，在调试时能熟练使用反汇编窗口、寄存器窗口、监视窗口、内存窗口、调用堆栈窗口等观察相关信息；

(2) 熟练掌握程序开发平台(Ubuntu) 的基本用法，包括程序的编辑、编译、链接和调试；

(3) 掌握 ARM 环境下程序开发的基本用法，比较Intel CPU 与 ARM机器指令的异同；

(4) 了解 32/64位环境下程序设计的不同特点及配套的开发工具；

(5) 熟悉编程的基础知识，包括数据在计算机内的表现形式、寻址方式、常用指令等；

(6) 熟悉程序运行的基本原理；

(7) 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；

(8) 加深对转移指令及一些常用的汇编指令的理解。

通过在不同开发环境下使用不同的开发工具，了解汇编语言表示的异同点；了解不同CPU下机器指令的异同点；从机器角度理解程序工作的基本原理。

**二、实验内容**

**任务1.1 编写一个程序，完成数据存储压缩和解压。**

定义了 结构 student ，以及结构数组变量s[3];

struct student {

char name[8];

short age;

float score;

char remark[200]; // 备注信息

};

student old\_s[3]; // old\_s[0].name 为自己的姓名;

student new\_s[3];

说明：不同学生的备注长短不同，student结构比较占空间。编写程序，将 old\_s[3] 中的所有信息依次紧凑存放到一个字符数组中，然后从 message 转换到结构数组 new\_s[3]中。打印压缩前、解压后的结果，以及压缩前、压缩后存放数据的长度。

要求：

① 提供两种格式的压缩函数

int pack\_student\_bytebybyte(student\* s, int sno, char \*buf); 实现一个字节一个字节的向buf中写数据；

int pack\_student\_whole(student\* s, int sno, char \*buf); 要求对short、float字段都只能用一条语句整体写入，用strcpy实现串的写入。

s为结构数组的起始地址； sno 为学生人数； buf 为压缩区域存储区的首地址;返回压缩后的字节数。

② 提供一个解压函数

int restore\_student(char \*buf, int len, student\* s);

buf 为压缩区域存储区的首地址;len为buf中存放数据的长度；s为存放解压数据的结构数组的起始地址；返回解压的人数。

③ 调用函数时，除了使用参数提供的信息外，不准使用其他的外部信息。

第一个记录压缩（即old\_s[0]）要调用pack\_student\_bytebybyte；第2-3个记录压缩要调用pack\_student\_whole。

**任务1.2 数据的表示和存储**

编写一个C语言程序，使用VS2019进行编译、链接和调试。在报告中，总结数据表示和存储的规律，数据存储单元的地址表达形式和地址计算方法，数据类型之间转换的规律，并使用语句示例（不需要完整的程序）说明。定义的变量应包含有 char、short、unsigned short、int、unsigned int、float、double，以及由这些类型组成的数组、结构。

**任务2.1 gcc 和 gdb 的使用**

使用vi 等工具编辑，使用gcc进行编译、链接，使用gdb进行调试。通过实验举例说明 Intel 格式与 AT&T 格式中汇编语句的差别。

**任务2.2 编写完成指定功能的程序**

在Ubuntu 环境下，编写C语言程序，完成如下功能：

1. 判断有符号数(short)加法运算是否产生溢出，要求输入两个数，并给出运算结果和相应的提示；
2. 判断无符号数(unsigned short)加法运算是否产生溢出，要求输入两个数，并给出运算结果和相应的提示；
3. 输入一个浮点数(float) ，按照从高地址到低地址字节的顺序，以十六进制形式显示各字节的内容，显示数据的正负号、以二进制形式显示数据的阶码、尾数。

**任务3.1 安装QEMU等环境，编译执行示例程序。**

需要编译执行的程序包括：

（1）“ARM虚拟环境安装说明”文档中“1.4.1”的程序（一个显示Hello World!的汇编语言程序）

（2）“ARM虚拟环境安装说明”文档中“2.2.1”（一个测试内存拷贝函数的执行时间的C语言与汇编语言混合编程的程序）和“2.2.3”（对前面“2.2.1”程序的优化）的程序。

指出 ARM 指令 与 Intel 指令的异同点。

**任务3.2 对任务1.1 中的程序，在VS2019下用x64平台编译，与32位平台下编译的结果进行比较，用语句示例指出它们的异同点。**

**三、实验记录及问题回答**

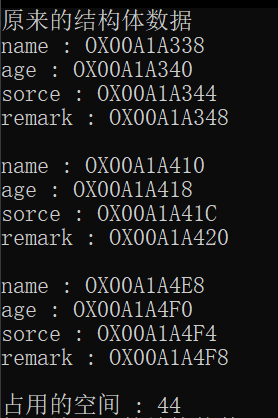
**（1）实验任务 1.1 的算法思想及运行结果**

pack\_student\_bytebybyte利用的算法思想是将原结构体内部的指针强行转换为char指针，利用for循环，将结构内部的数据全部读取出来。

pack\_student\_whole利用的算法思想是将原结构体内部的指针以char语句的形式来读，利用strcpy读取内存中的内容，将其复制到message数组中。

restore\_student利用的算法思想是利用char指针将message数组中的每一个字节读出，然后写入结构体内，相当于 pack\_student\_bytebybyte的逆函数。

**运行结果：**

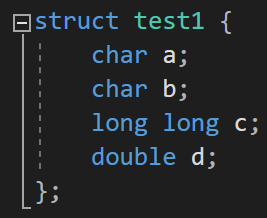
**文本

描述已自动生成**

其中原始数据一共占用了8+4+4+200=216个字节，压缩后占用的空间为3+2+4+4+2+3+2+4+4+2+3+2+4+3+2=44个字节，可见完成了压缩数据的任务。

**（2）实验任务 1.2 的实验记录及问题回答**

我做了关于结构体对齐的相关实验，定义的结构体如下



打印出来的地址为

文本

描述已自动生成

可以看到longlong类型的c存放的地址于char类型的b直接留下了六个字节的空

计算机系统对基本数据类型的合法地址做出了一些限制，要求某种对象的地址必须是2，4，或者8的倍数，这种对齐简化了形成处理器和内存系统之间接口的硬件设计，这样我们就可以利用一次内存读取来读取我们想要的数据了，否则可能会因为数据分在两个内存块中而需要两次的内存读取

编译器会在汇编代码中放入命令，指明全局数据所需要的对齐。

对于包含结构的代码，编译器可能需要在字段的分配中插入间隙，来保证每一个结构元素都满足它的对齐要求。结构对它本身的起始地址也会有一些对齐要求。一般来说都是对结构体内最大的那个数据类型来对齐的。在上述的结构体中longlong类型是最大的数据类型，占用了八个字节因此他的其实地址必必须是八的倍数，中间就会空出来六个字节

**（3）实验任务 2.1 的实验记录及问题回答**

使用Intel格式的反汇编

文本

描述已自动生成

使用AT&T格式的反汇编

文本

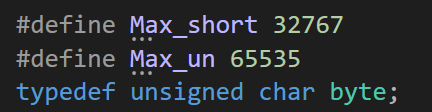
描述已自动生成

其中的区别为

1. 前缀 AT&T中寄存器有前缀%，立即数有前缀$
2. 操作方向 AT&T格式的后的第一个操作数为源操作数，第二个为目的操作数，intel格式则相反
3. 内存操作 AT&T内存寻址用的是()，intel内存寻址用的是[]
4. 指令后缀AT&T 会在指令后面添加(‘b’, ‘w’, ‘l’ 或 ‘q’) 指定操作数的大小而Intel 会在内存操作数上指定 byte ptr, word ptr, dword ptr
5. 寻址 AT&T格式的语法为disp(BASE,index,scale)，Intel 语法：[base + index\*scale + disp]

**（4）实验任务 2.2 的运行结果记录**

相关定义



程序运行的结果

文本

描述已自动生成

其中float对应的地址

图形用户界面, 文本, 网站

描述已自动生成

表明成功的打印了float的相关内容。

**（5）实验任务 3.1 的实验记录及问题回答**

（1）实现的hello程序截图

文本

描述已自动生成

（2）任务的c语言代码和编译语言代码及其运行

优化前后的运行时间

C语言代码和汇编代码

文本

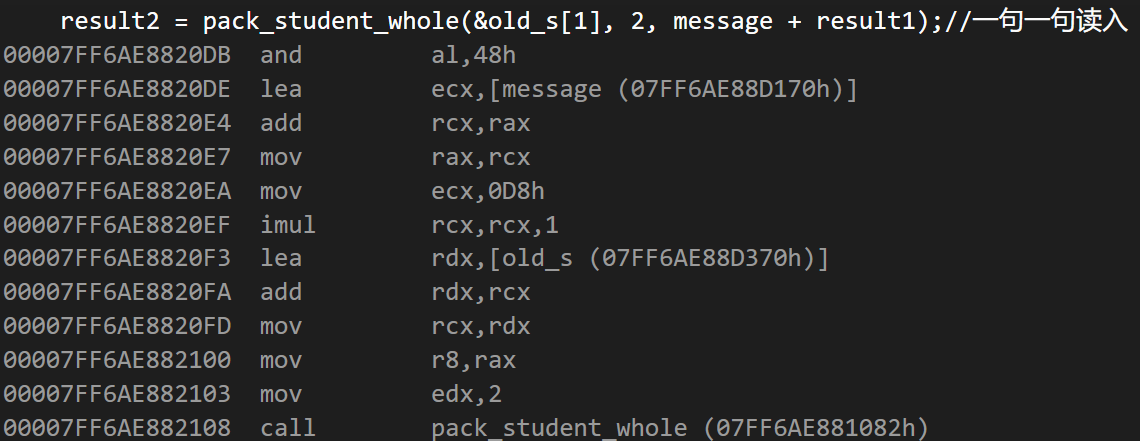
描述已自动生成 文本

描述已自动生成

其中arm指令的长度是固定的 arm架构的指令集比intel精简 arm架构的寻址方式更加多样

**（6）实验任务 3.2 的实验记录及问题回答**

使用64位编译的结果

****

使用32位编译的结果

**文本

描述已自动生成**

1. X86\_64中操作的地址不一定非得是64位，也可以是32位（rax ecx都可以使用）而x86\_32只能使用32位地址。
2. 部分x86\_32指令在x86\_64中不再支持
3. 64位没有像32位里ebp的指针，而直接使用rsp。

**四、体会**

本次实验掌握了VS2019的基本用法，进行了程序的编译、链接和调试。学习并查看了反汇编窗口、寄存器窗口、监视窗口等窗口的功能。真真正正明白了各种类型的数据在内存中的存储方式。在内存中，数据就是一些排列的数，不同的取数方式会得到不同的结果。我们可以有不同的方式找到我们存储的数据，我们能够在“茫茫数字”中找到我们存的那个变量，其实还有一些神奇！同时这是遵守一定的规则的，这也体现出计算机的严密之处。实验1.2也自己实现了数据的压缩和解压，学习了结构体、short、float、字符数组等的存储方式，明白了变量声明和定义在内存中的表现。

通过实了解32位和64位环境下程序设计的不同特点，安装了QEMU环境，通过完成指定的程序设计，观察并理解了汇编语言在不同环境下的基本特点。在QEMU环境里领略的ARM指令集，开拓了我的视野。

**五、源码**

实验任务 1.1 的源程序、2.2 的源程序