

**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 ： U202115666**

**姓 名 ： 刘文博**

**指导教师 ： 许向阳**

**2022 年 10 月 1 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**

**算法思想：**

对于pack\_student\_bytebybyte函数使用一个char型指针遍历结构体中的数据，数据有效则读入，每当遍历到第一个0就读入然后从下一数据类型的首字节开始遍历，遍历的同时用cntbuf记录字节数。

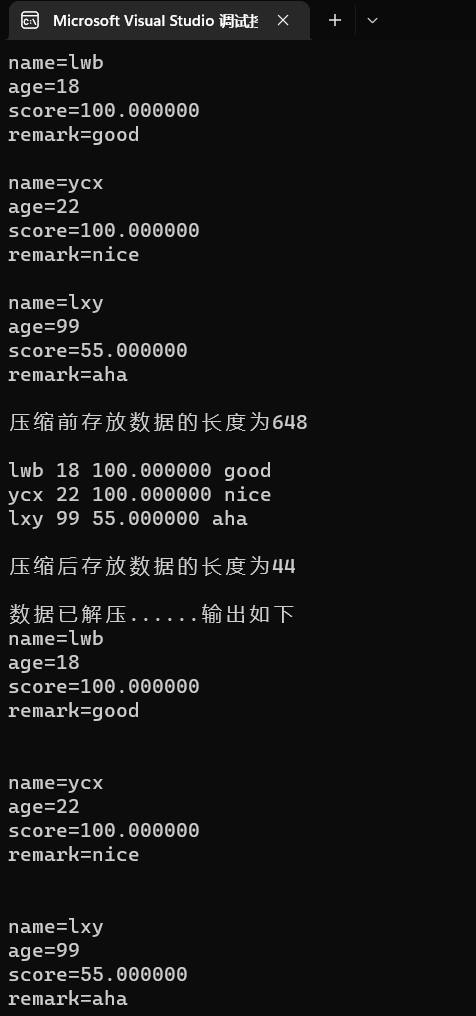
对于pack\_student\_whole函数，字符数组调用strcpy函数直接复制，age、score则用一p指针移动到这个数据的首字节，再强制转换把剩下的字节作为整体读入。最后用指向压缩数组末尾的指针减去基址buf的值得到字节数。

对于restore\_student函数可以用与压缩函数相反的方法进行解压。具体步骤如下



**运行结果：**

运行结果如下页图所示



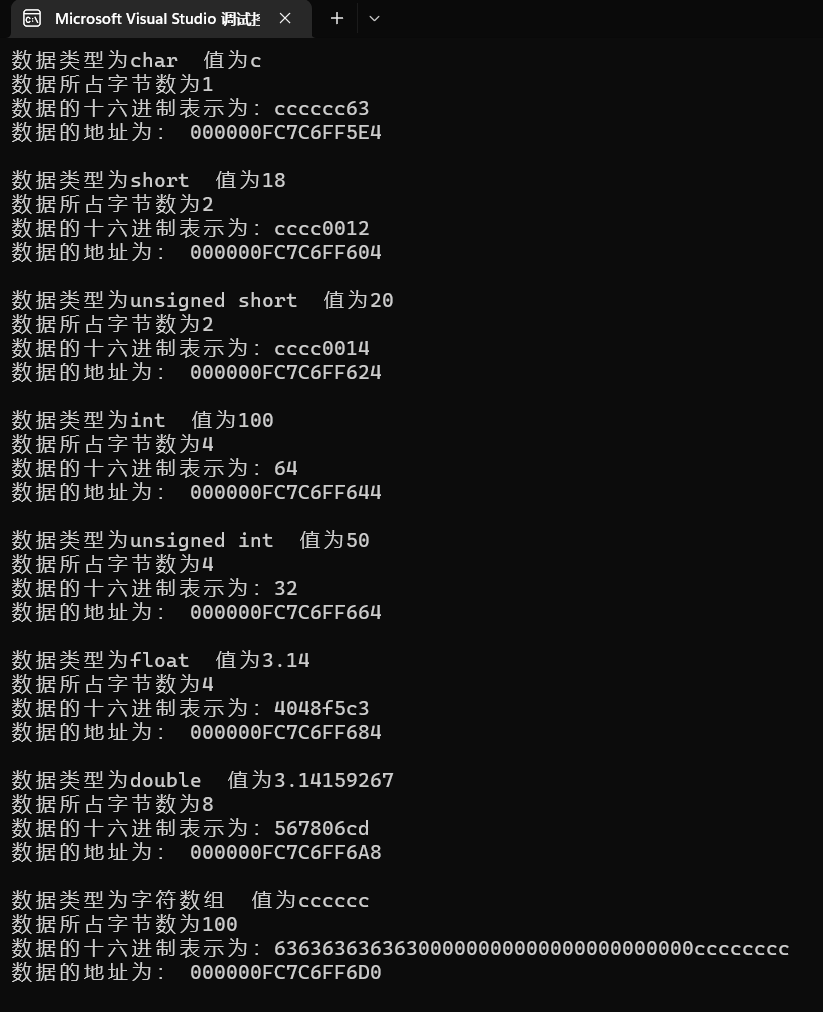
**（2）实验任务 1.2 的实验记录及问题回答**

**实验记录**

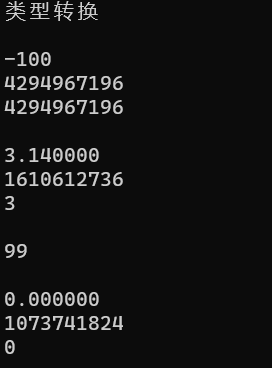
编写一个程序定义了char、short、unsigned short、int、unsigned int、float、double类型的变量，以及由这些类型组成的数组、结构。

同时举出几个类型转换的例子

实验记录如下图所示







**问题回答**

通过实验记录可以看出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如下

数据表示的规律：

char类型将字符与ASCII码对应，实际上相当于一字节的整数；

short类型占用两个字节，在计算机内表示为二进制数

unsigned short类型与short类似，但没有符号位只能表示正整数和0，相应的数据表示范围拓宽二倍

int类型占用两个字节，在计算机内表示为二进制数

unsigned int类型与int类似，但没有符号位只能表示正整数和0，相应的数据表示范围拓宽二倍

float类型采用IEEE754标准，有符号位，指数位和尾数位，其中尾数位能够表示一定精度内的0~2的数，指数位的计算方式想当于一个unsigned二进制整数值减去一个偏移量。

double 占用八个字节，所能表示的数据范围和精度较float均有很大提升

字符数组占用的字节数为字符数组的长度，未初始化的部分存储不确定的初值。

数据存储单元的地址表达形式和地址计算方法：

一般而言，相邻定义数据时其地址的差值为32个字节，当定义的数据长度超过4字节则差值为36字节，数组取地址时取到的是其第一个元素的地址。地址计算方法下标为i的元素的地址等于首个元素的地址加上i\*数组元素类型对应的字节数，当定义结构体时，会有特殊的对齐原则。

类型转换规律：

如果不加以强制类型转换，会根据输出时指定的方式（根据占位符）对数据进行截取和解读。

实验中还通过一个特殊的例子发现float类型数据在printf时会被转化为double类型再输出，如果这时候指定输出类型为int，则会截取要输出的double数的低32位进行解读并输出。

如果考虑强制类型转换

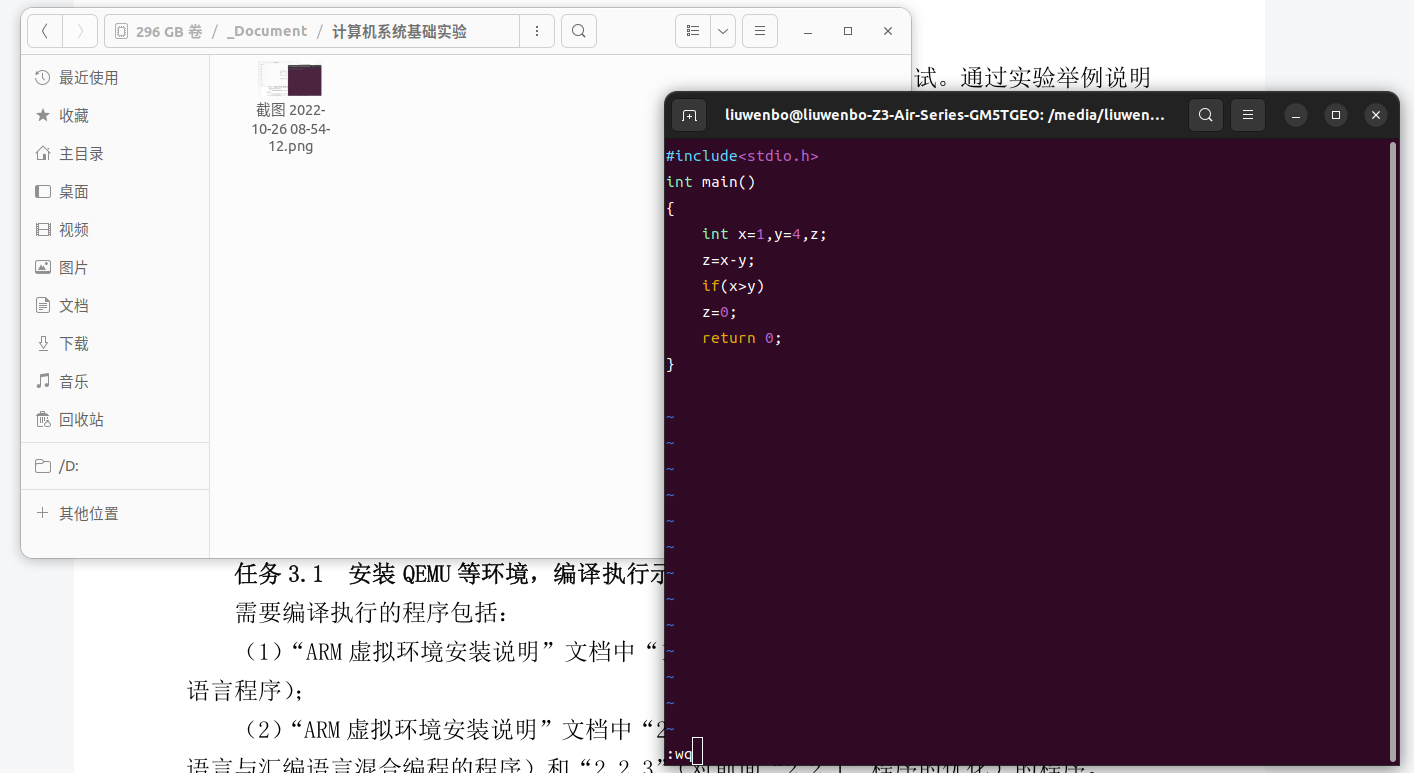
Int转换为unsigned int 数据内容不会改变

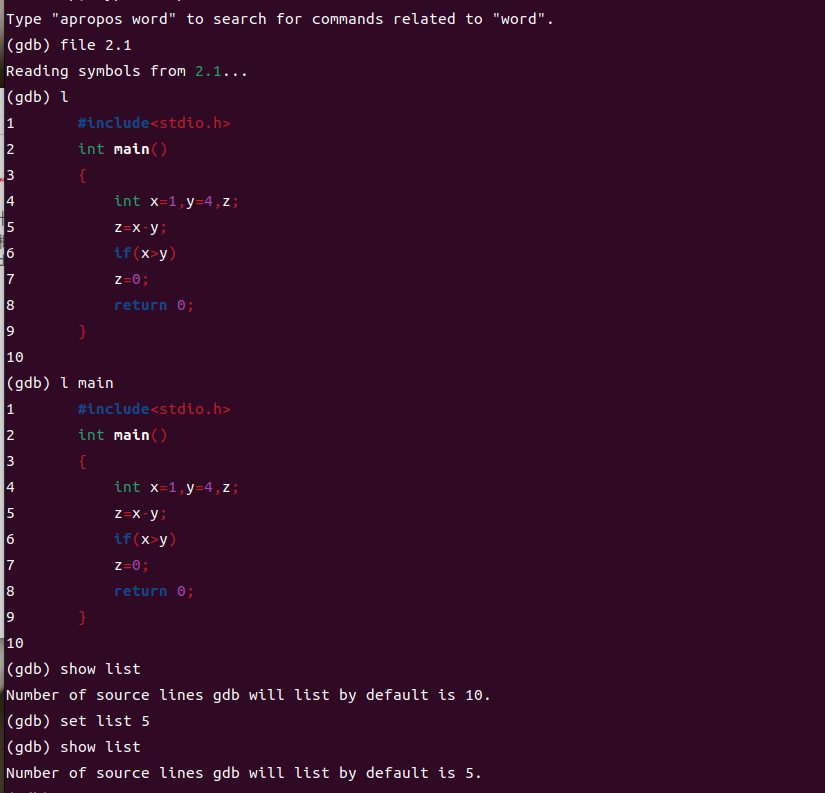
Float转换为int 数据内容会改变且只会截取整数部分。

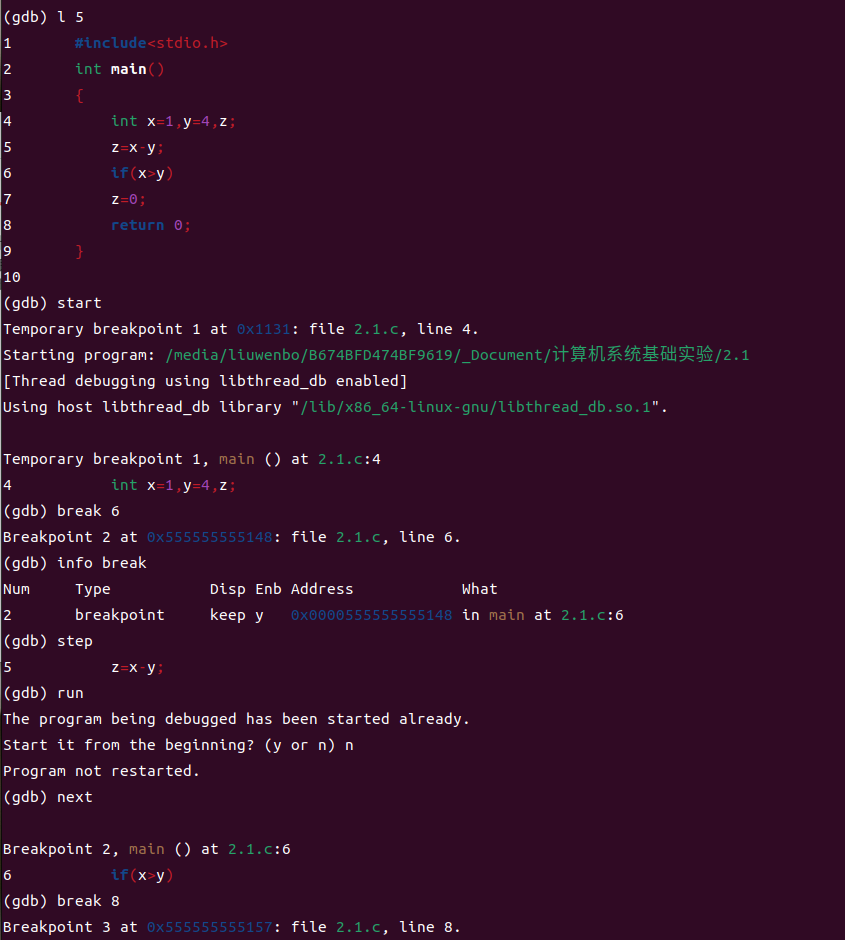
**（3）实验任务 2.1 的实验记录及问题回答**

**实验记录**

首先使用使用vim作为编辑器再linux系统中写了一段简单的代码

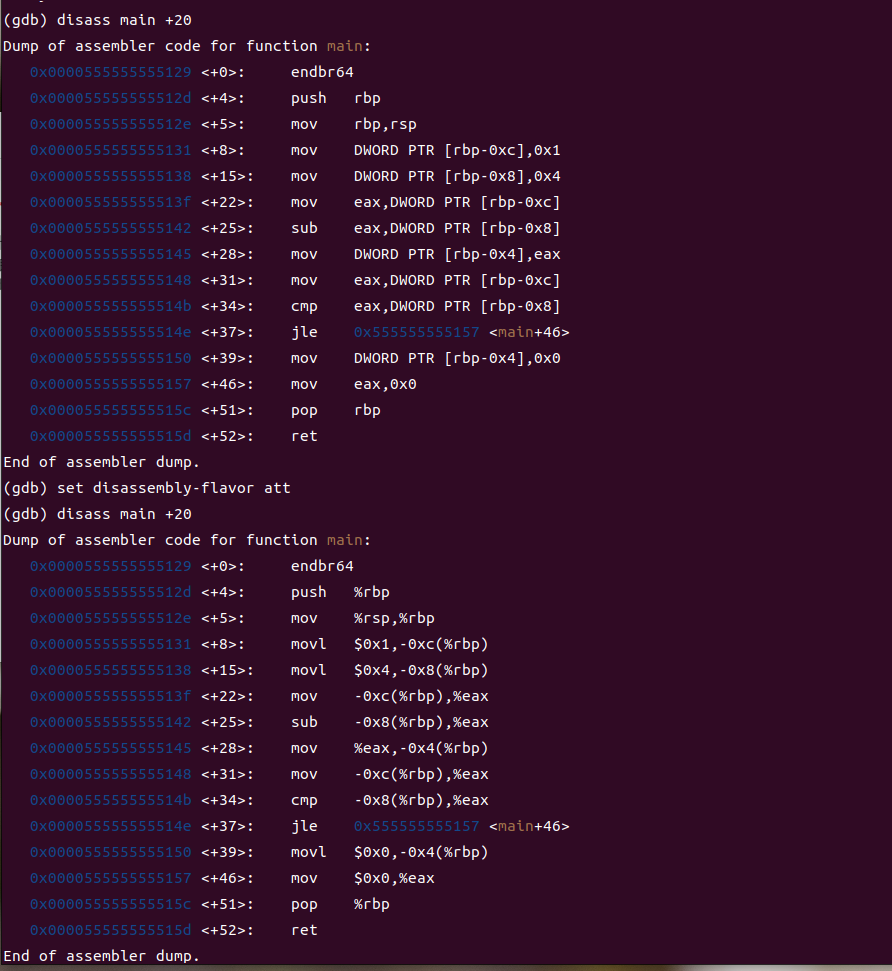


之后使用gcc进行编译、链接，使用gdb进行调试。



**问题回答**

通过实验发现 Intel 格式与 AT&T 格式中汇编语句有如下差别。

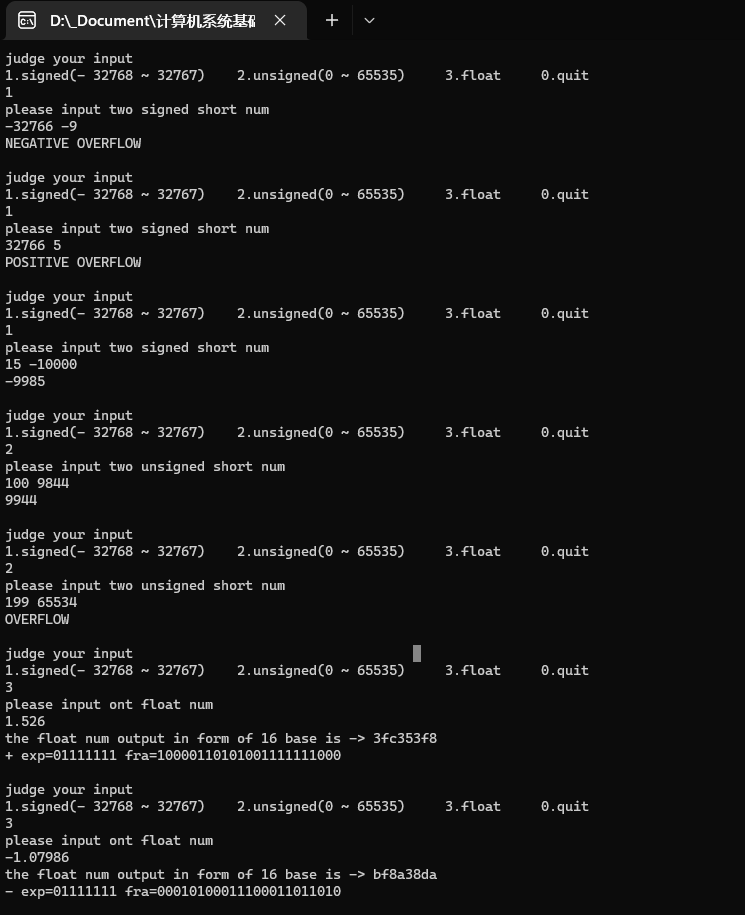


1. intel格式的操作数没有前缀而att格式中寄存器有%前缀，立即数有**$前缀**
2. intel中基寄存器用[]括起来而att中用（）括起来
3. 间接寻址方式不同，intel采用segreg:[base+index\*scale+disp]，而att采用%segreg:disp(base,index,scale)
4. Intel中用dword ptr指出操作数的字节，而att则用mov movl之类的指令进行区别
5. Intel格式中目的操作数在前，而att目的操作数在后。

**（4）实验任务 2.2 的运行结果记录**

在ubuntu环境下使用vscode编写了2.2的源代码

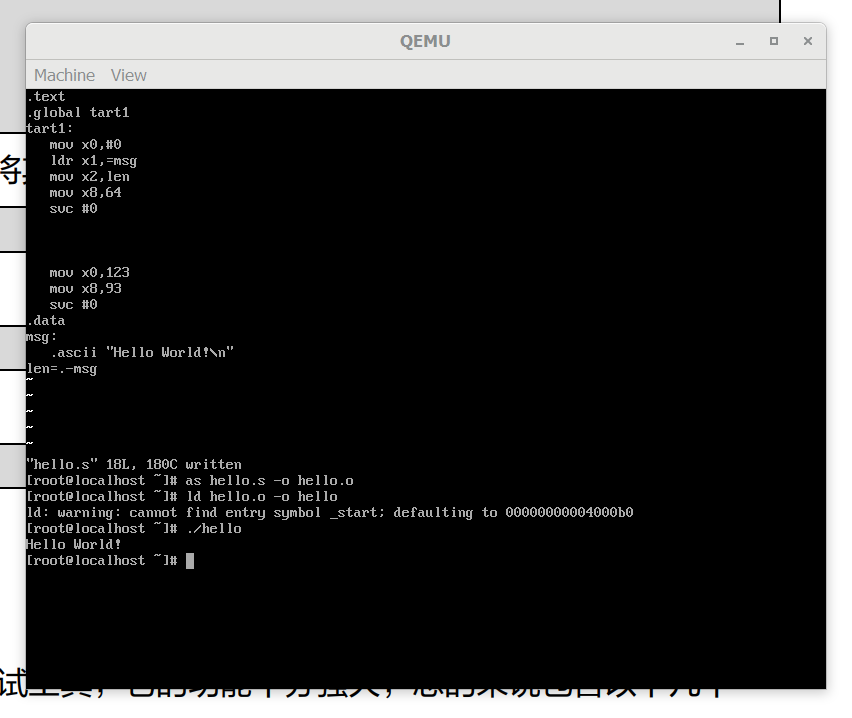
运行结果如下



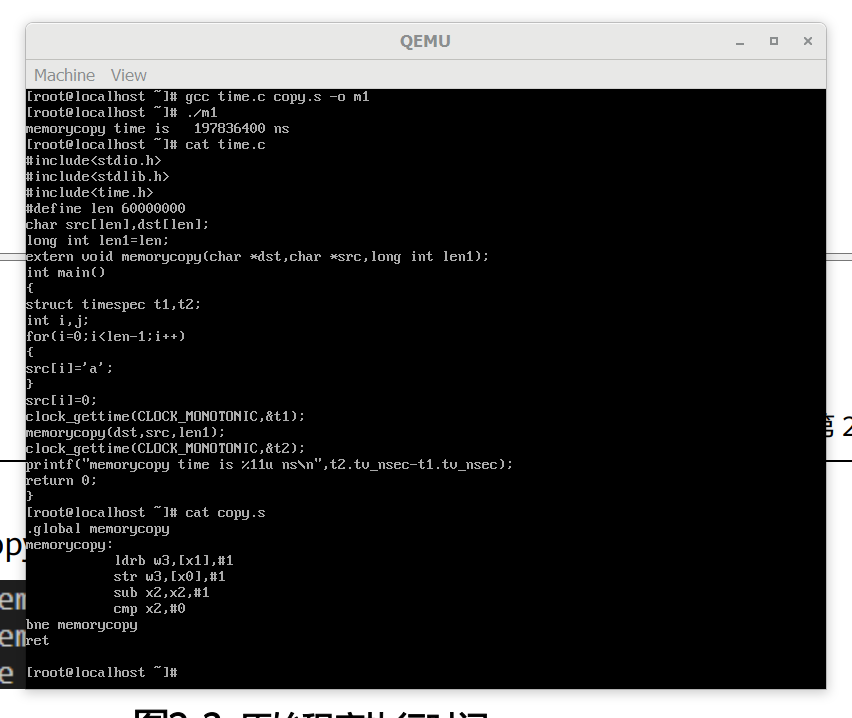
**（5）实验任务 3.1 的实验记录及问题回答**

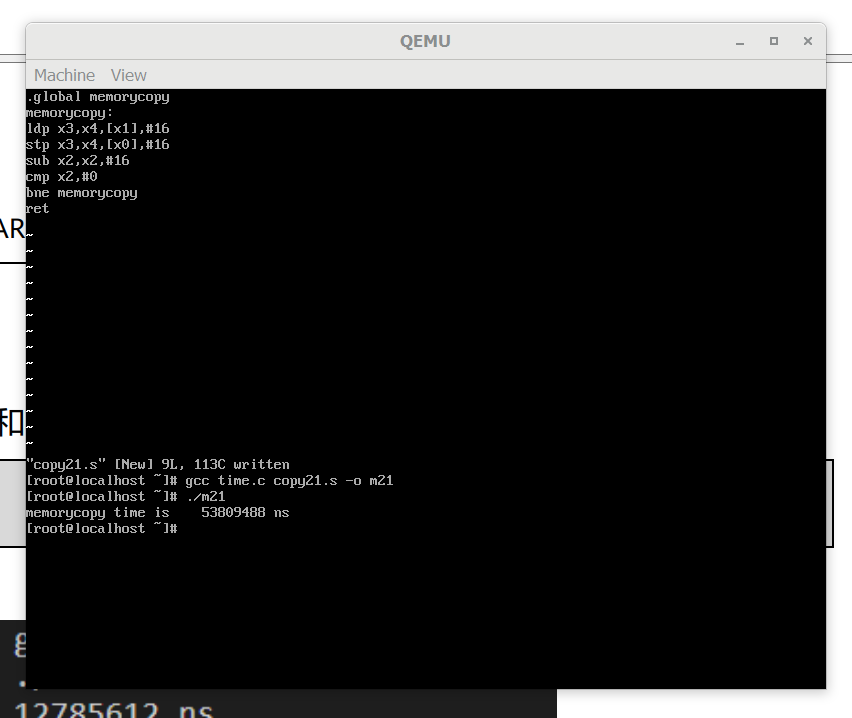
**实验记录：**

首先使用qemu配置OpenEuler虚拟机，然后分别用vi编辑1.4.1，2.2.1，2.2.3的源代码，之后用gcc编译执行，实验结果如下



编译执行1.4.1



编译执行2.2.1

编译执行2.2.3

**问题回答：**

通过实验可以发现ARM指令和Intel指令有如下异同

ARM中指令较少，其指令是等长的，各种指令的使用频率大致相同，其中立即数要加上#，而Intel指令较多，指令可以不等长，各种指令的使用频率区别很大，立即数不需要加上#。但二者都是设计制造微处理器的典型技术

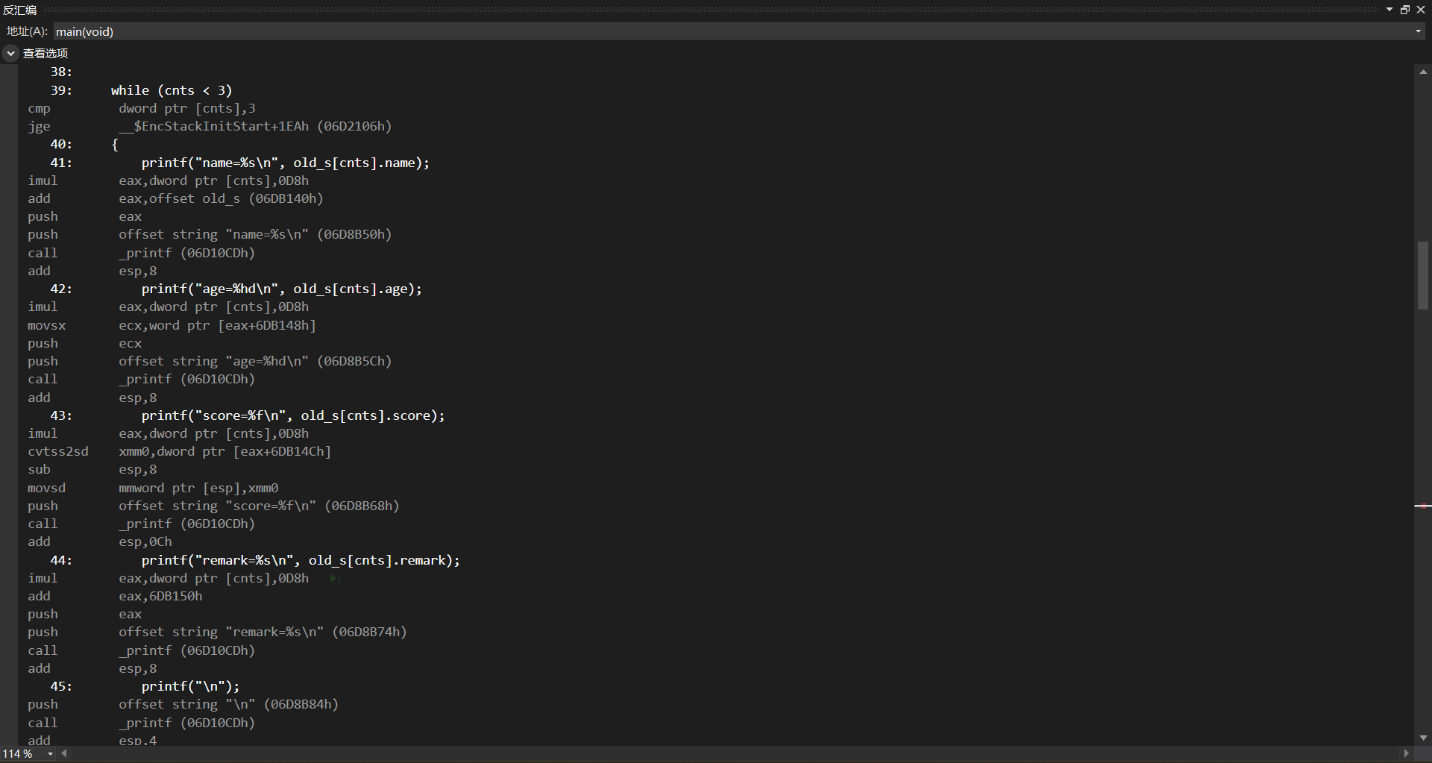
**（6）实验任务 3.2 的实验记录及问题回答**

**实验记录：**

使用vs2022分别在x64平台和win32平台编译1.1中的代码，使用vs的反汇编窗口查看了main函数中的一段代码反汇编之后的反汇编代码，结果如下



X64下的反汇编



Win32下的反汇编

**问题回答：**

通过观察同一段代码在x64和win32平台下的反汇编代码，可以得出以下异同点

64位有16个寄存器，32位只有8个。但是32位前8个都有不同的命名，分别是e \_ ，而64位前8个使用了r代替e，也就是r \_。e开头的寄存器命名依然可以直接运用于相应寄存器的低32位。而剩下的寄存器名则是从r8 - r15，其低位分别用d，w,b指定长度。

32位使用栈帧来作为传递的参数的保存位置，而64位使用寄存器，分别用rdi,rsi,rdx,rcx,r8,r9作为第1-6个参数。rax作为返回值

64位没有栈帧的指针，32位用ebp作为栈帧指针，64位取消了这个设定，rbp作为通用寄存器使用

64位支持一些形式的以PC相关的寻址，而32位只有在jmp的时候才会用到这种寻址方式。

mov指令和push pop扩展了movq系列的mov和pushq以及popq用来操作quad word。

**四、体会**

通过本次实验我对数据在计算机中的表示以及存储有了了解，对汇编语言有了认识，明白了高级语言中的操作在底层是具体通过什么指令来完成的。这对我在编程时考虑计算机的效率和代码的正确性有很大的帮助。

**五、源码**

实验任务 1.1 的源程序、2.2 的源程序

﻿**实验1.1的源程序如下**

// 计算机系统实验1.1.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#pragma warning(disable:4996)

#include<stdio.h>

#include<string.h>

struct student {

char name[8];

short age;

float score;

char remark[200]; // 备注信息

};//student结构占用字节多 13\*16+8=216 8+4(2)+4+200

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

student new\_s[3];

int pack\_student\_bytebybyte(student\* s, int sno, char\* buf);//逐字节压缩

int pack\_student\_whole(student\* s, int sno, char\* buf);//不逐字节读入 使用strcpy

int restore\_student(char\* buf, int len, student\* s);//修复函数

int coutmessage(char\* buf, int len);//输出函数

int main()

{

/\*初始化\*/

int cnts = 0;

char message[505];

strcpy(old\_s[0].name, "lwb");

old\_s[0].age = 18;

old\_s[0].score = 100;

strcpy(old\_s[0].remark, "good");

strcpy(old\_s[1].name, "ycx");

old\_s[1].age = 22;

old\_s[1].score = 100;

strcpy(old\_s[1].remark, "nice");

strcpy(old\_s[2].name, "lxy");

old\_s[2].age = 99;

old\_s[2].score = 55;

strcpy(old\_s[2].remark, "aha");

//这里直接给出结构体的数据 避免了实验过程中多次的输入

while (cnts < 3)

{

printf("name=%s\n", old\_s[cnts].name);

printf("age=%hd\n", old\_s[cnts].age);

printf("score=%f\n", old\_s[cnts].score);

printf("remark=%s\n", old\_s[cnts].remark);

printf("\n");

cnts++;

}//输出压缩前的结果

cnts = 0;

printf("压缩前存放数据的长度为%u\n\n", sizeof(student) \* 3);//输出压缩前存放数据的长度

int cntbuf = pack\_student\_bytebybyte(old\_s, 1, message);//用第一种压缩函数压缩第一条数据

cntbuf += pack\_student\_whole(old\_s + 1, 2, message + cntbuf);//用第二种压缩函数压缩后两条数据

coutmessage(message, cntbuf);//输出压缩后的数据

printf("\n压缩后存放数据的长度为%d\n\n", cntbuf);//输出压缩后存放数据的长度

int sno = restore\_student(message, cntbuf, new\_s);//用解压函数解压数据

printf("数据已解压......输出如下\n");

while (cnts < sno)

{

printf("name=%s\n", new\_s[cnts].name);

printf("age=%hd\n", new\_s[cnts].age);

printf("score=%f\n", new\_s[cnts].score);

printf("remark=%s\n", new\_s[cnts].remark);

printf("\n\n");

cnts++;

}//输出解压后的结果

return 0;

}

int pack\_student\_bytebybyte(student\* s, int sno, char\* buf)

{

int cnts = 0, cntname = 0, cntage = 0, cntscore = 0, cntremark = 0, cntbuf = 0;

char\* p = (char\*)s;

char\* p0 = buf;

while (cnts < sno)

{

//读入name数组

cntname = 0;

while (cntname < 8)

{

if (\*p)

{

\*p0 = \*p;

cntname++;

cntbuf++;

p++;

p0++;

}

else

{

\*p0 = 0;

cntbuf++;

p += (8 - cntname);

p0++;

break;

}

}

//读入short

cntage = 0;

while (cntage < 2)

{

\*p0 = \*p;

cntage++;

cntbuf++;

p++;

p0++;

}

p += 2;

//读入float

cntscore = 0;

while (cntscore < 4)

{

\*p0 = \*p;

cntscore++;

cntbuf++;

p++;

p0++;

}

//读入remark数组

cntremark = 0;

while (cntremark < 200)

{

if (\*p)

{

\*p0 = \*p;

cntremark++;

cntbuf++;

p++;

p0++;

}

else

{

\*p0 = 0;

cntbuf++;

p += (200 - cntremark);

p0++;

break;

}

}

cnts++;

}

return cntbuf;//返回压缩后的字节数

}

int pack\_student\_whole(student\* s, int sno, char\* buf)

{

int cnts = 0;

char\* p0 = buf;

char\* p = NULL;

while (cnts < sno)

{

p = s[cnts].name;

strcpy(p0, p);

p0 += (strlen(p) + 1);//读入name数组

p = (char\*)&s[cnts].age;

\*((short\*)p0) = \*((short\*)p);

p0 += 2;//读入age

p = (char\*)&s[cnts].score;

\*((float\*)p0) = \*((float\*)p);

p0 += 4;//读入score

p = s[cnts].remark;

strcpy(p0, p);

p0 += (strlen(p) + 1);//读入remark数组

cnts++;

}

return p0 - buf;

}

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

{

int cnts = 0;

char\* p = buf;//遍历message数组

while (p - buf < len)//当从message数组中读入的数据不够len

{

strcpy(s[cnts].name, p);//读入name

p += strlen(s[cnts].name) + 1;//p指针移动到age处

s[cnts].age = \*((short\*)p);//读入age

p += 2;//p指针移动到score处

s[cnts].score = \*((float\*)p);//读入score

p += 4;//p指针移动到remark处

strcpy(s[cnts].remark, p);//读入remark

p += strlen(s[cnts].remark) + 1;//p指针移动到下一name处

cnts++;//增加学生数

}

return cnts;//返回学生数

}

int coutmessage(char\* buf, int len)

{

int cnts = 0;

char\* p = buf;//遍历message数组

while (p - buf < len)//当从message数组中读入的数据不够len

{

printf("%s ", p);

p += strlen(p) + 1;//p指针移动到age处

printf("%hd ", \*((short\*)p));

p += 2;//p指针移动到score处

printf("%f ", \*((float\*)p));

p += 4;//p指针移动到remark处

printf("%s\n", p);

p += strlen(p) + 1;

cnts++;//增加学生数

}

return cnts;//返回学生数

}

﻿

**实验2.2的源程序如下**

#pragma warning(disable:4996)

#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{

int op = 1;

short a, b, ab;

unsigned short c, d, cd;

float e;

int\* f;

unsigned int g = 1;

while (op)

{

printf("\n");

printf("judge your input\n");

printf("1.signed(- 32768 ~ 32767) 2.unsigned(0 ~ 65535) 3.float 0.quit\n");

scanf("%d", &op);

switch (op)

{

case 1:

printf("please input two signed short num\n");

scanf("%hd%hd", &a, &b);

ab = a + b;

if (a > 0 && b > 0 && ab < 0) printf("POSITIVE OVERFLOW\n");

else if (a < 0 && b < 0 && ab>0) printf("NEGATIVE OVERFLOW\n");

else printf("%hd\n", ab);

break;

case 2:

printf("please input two unsigned short num\n");

scanf("%hu%hu", &c, &d);

cd = c + d;

if (cd < c) printf("OVERFLOW\n");

else printf("%hu\n", cd);

break;

case 3:

g <<= 31;

printf("please input ont float num\n");

scanf("%f", &e);

f = (int\*)(&e);

printf("the float num output in form of 16 base is -> ");

printf("%x\n", \*f);

if (\*f & g) printf("- ");

else printf("+ ");

printf("exp=");

for (int i = 1; i < 9; i++)

{

g >>= 1;

if (g & \*f) printf("1");

else printf("0");

}

printf(" fra=");

for (int i = 1; i < 24; i++)

{

g >>= 1;

if (g & \*f) printf("1");

else printf("0");

}

printf("\n");

break;

defalut:

op = 0;

}

}

}