**实验三 结构化数据存储实验**

**一、实验目的**

1. 理解真值与机器数
2. 掌握不同类型变量在内存中的存储情况
3. 理解数据存储的字对齐
4. 掌握复杂类型数据的存储空间分配与访问
5. 理解不同数据类型转换

**二、实验环境**

PC机，Linux 32-bit i386，C/汇编语言

gcc、objdump、gdb

**三、实验学时**

4学时

**四、实验内容**

4.1 实验内容一

4.1.1 代码及编程思路

代码：

#include <stdio.h>

int main()

{ int ai=100,bi=2147483648,ci=-100;

unsigned au=100,bu=2147483648,cu=-100;

printf("ai=%d,bi=%d,ci=%d\n",ai,bi,ci);

printf("au=%u,bu=%u,cu=%u\n",au,bu,cu);

}

编程思路：

通过带符号数与无符号数的输出来理解整数的编码对应的真值与机器数

4.1.2 实验步骤

（1）编写程序

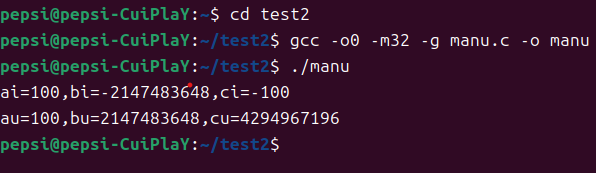
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

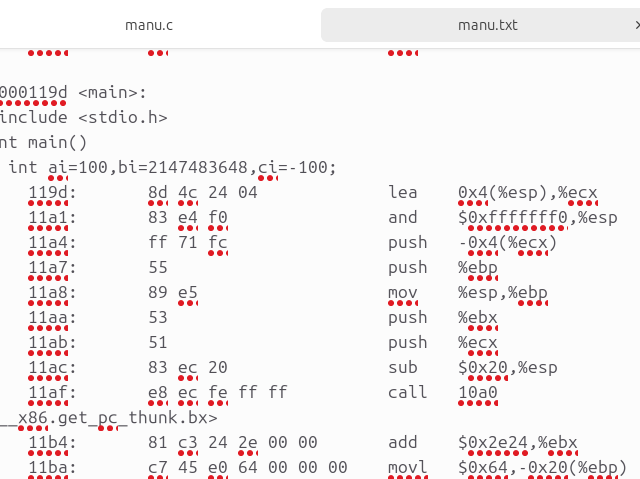
（3）使用objdump指令进行反汇编

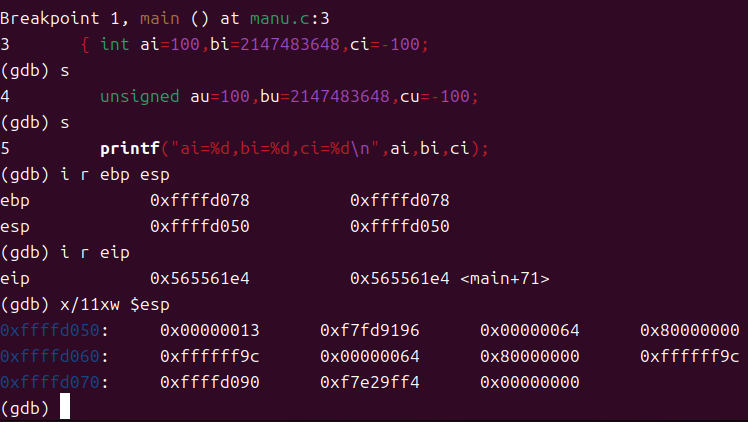
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.1.3 程序运行截屏







4.1.4 出现问题及解决办法

无

4.2 实验内容二

4.2.1 代码及编程思路

代码：

#include <stdio.h>

int main()

{ int ai=100,bi=-100;

float af=100,bf=-100;

printf("ai=%d, bi=%d\n",ai,bi);

printf("af=%f, bf=%f\n",af,bf);

return 0;

}

编程思路：

通过整数与浮点数的反汇编内容来理解浮点的编码对应的真值与机器数

4.2.2 实验步骤

（1）编写程序

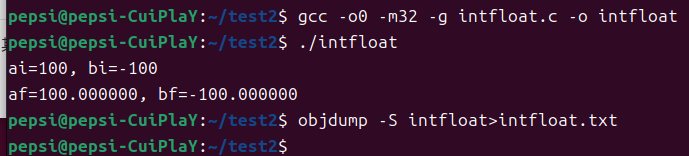
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

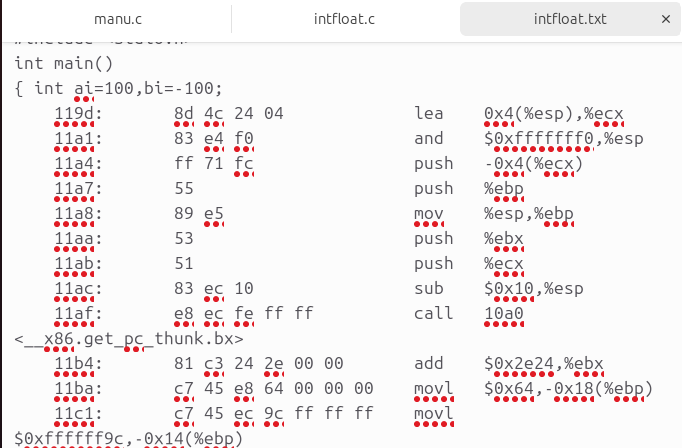
（3）使用objdump指令进行反汇编

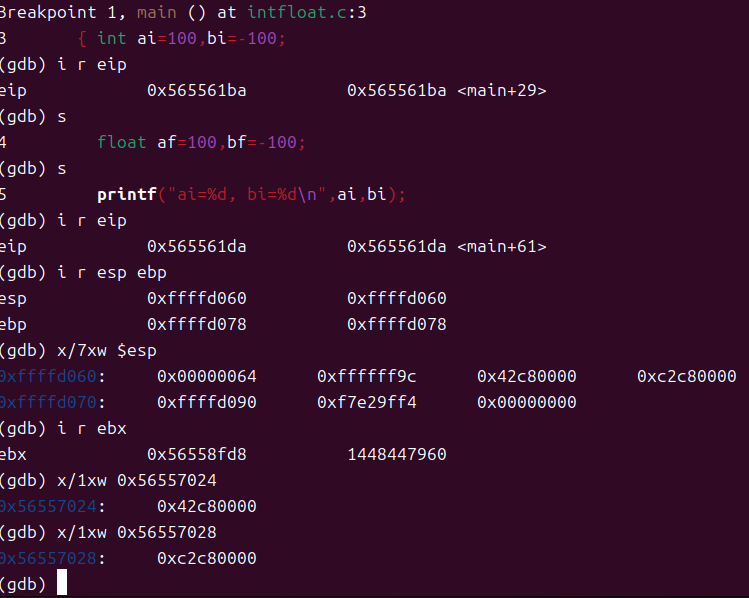
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.2.3 程序运行截屏







4.2.4 出现问题及解决办法

无

4.3 实验内容三

4.3.1 代码及编程思路

代码：

#include "stdio.h"

void main( )

{ char a=100;

short b=100;

int c=100;

int d=0x12345678;

printf("a=%0xH,b=%0xH,c=%0xH,d=%0xH\n",a,b,c,d);

}

编程思路：

通过程序的反汇编内容来理解数据存储的宽度和排列方式

4.3.2 实验步骤

（1）编写程序

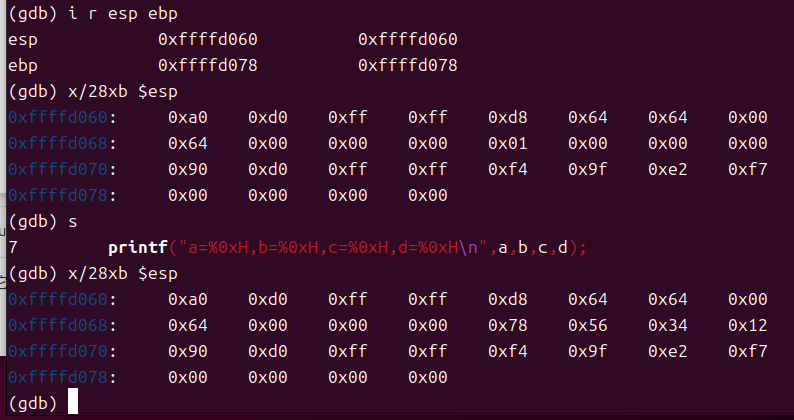
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

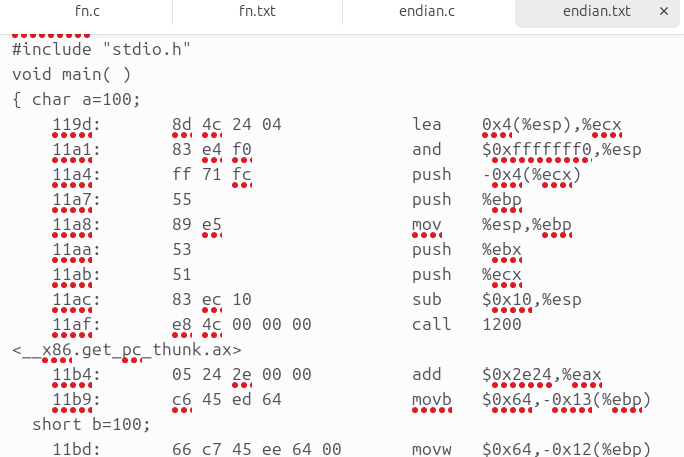
（3）使用objdump指令进行反汇编

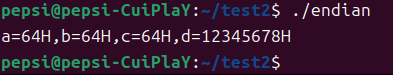
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.3.3 程序运行截屏







4.3.4 出现问题及解决办法

出现的内容可能与视频不一致，原因是存放方式不一样

4.4 实验内容四

4.4.1 代码及编程思路

代码：

#include "stdio.h"

int main()

{ struct record{

char a;

int b;

short c;

char d;

}R[2];

R[0].a=1; R[0].b=2; R[0].c=3; R[0].d=4;

R[1].a=5; R[1].b=6; R[1].c=7; R[1].d=8;

printf("数据存储时的边界对齐");

return 0;

}

编程思路：

通过程序的反汇编内容来理解数据存储的对齐方式

4.4.2 实验步骤

（1）编写程序

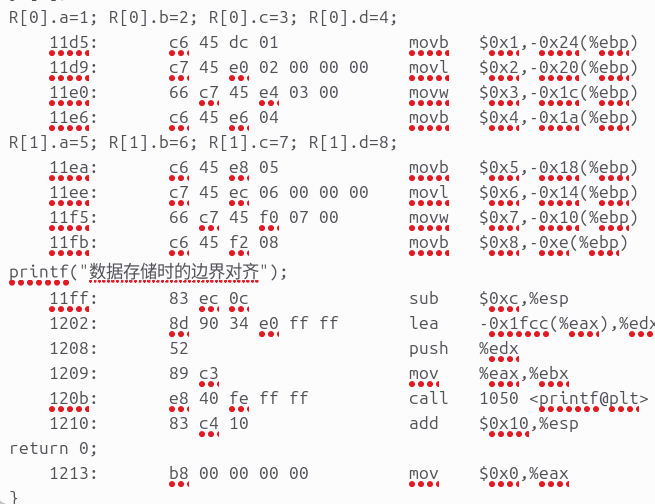
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

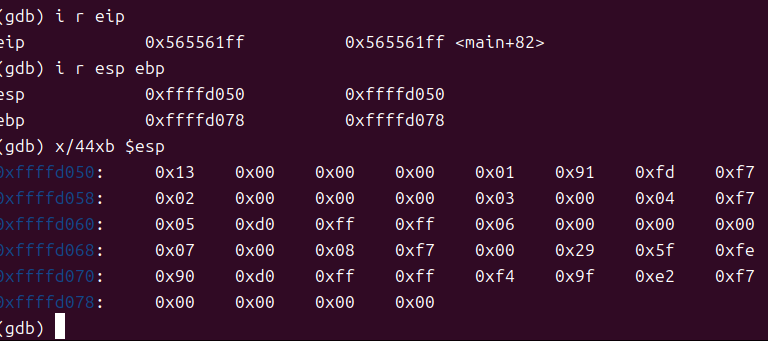
（3）使用objdump指令进行反汇编

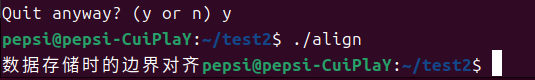
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.7.3 程序运行截屏







4.4.4 出现问题及解决办法

无

4.5 实验内容五

4.5.1 代码及编程思路

代码：

#include "stdio.h"

int main()

{ short si =- 100;

unsigned short usi=si;

int i=usi;

unsigned ui=usi;

int i1=si;

unsigned ui1=si;

int i2=0x12345678;

short si2=i2;

unsigned short usi2=i2;

int i3=si2;

int i4=4294967296;

printf("si=%d,usi=%u,i=%d,ui=%u,i1=%d,ui1=%u\n",si,usi,i,ui,i1,ui1);

printf("i2=%d,si2=%d,usi2=%u,i3=%d,i4=%d\n", i2,si2,usi2,i3,i4);

return 0;

}

编程思路：

通过整数类型之间的转换以及生成反汇编内容来理解整数数据类型的转换

4.5.2 实验步骤

（1）编写程序

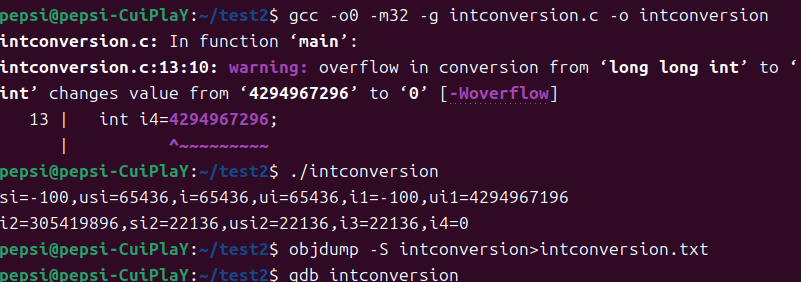
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

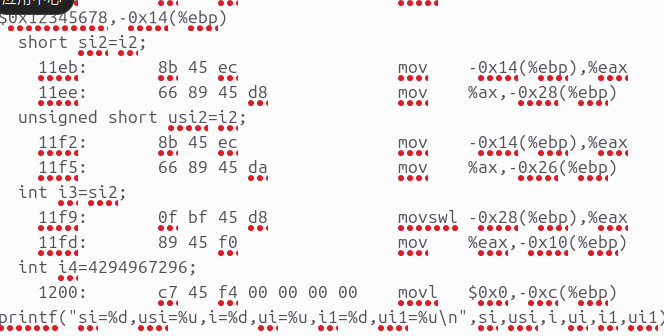
（3）运行可执行文件

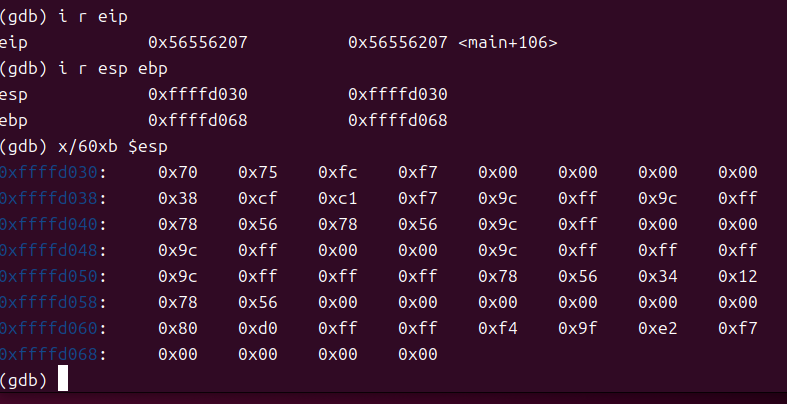
（4）使用objdump指令进行反汇编

（5）使用gdb指令对程序进行调试

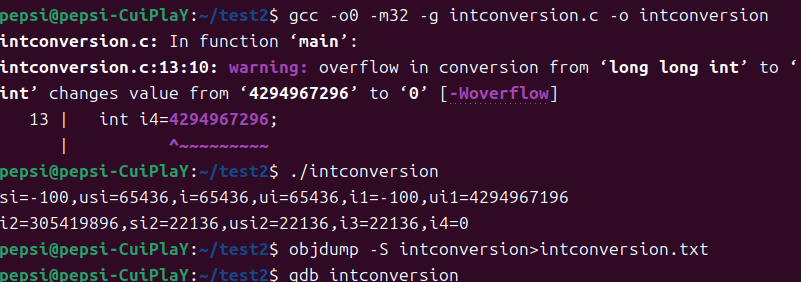
4.5.3 程序运行截屏







4.5.4 出现问题及解决办法



程序出现这种警告，无需担心，只是提示，本内容就是在此警告下理解整数数据类型的转换

4.6 实验内容六

4.6.1 代码及编程思路

代码：

#include <stdio.h>

int main()

{ int i1=0x7fffffff,i2,itemp;

float f1=0x987654321,f2,ftemp;

ftemp=i1;

i2=ftemp;

itemp=f1;

f2=itemp;

printf("i1=%d,i2=%d\nf1=%f,f2=%f\n",i1,i2,f1,f2);

return 0;

}

编程思路：

通过整数类型与浮点数类型之间的转换以及生成反汇编内容来理解整数数据类型与浮点数数据类型的转换

4.6.2 实验步骤

（1）编写程序

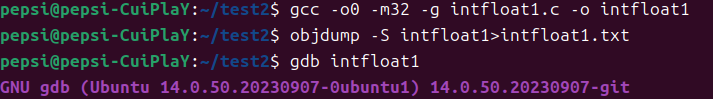
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

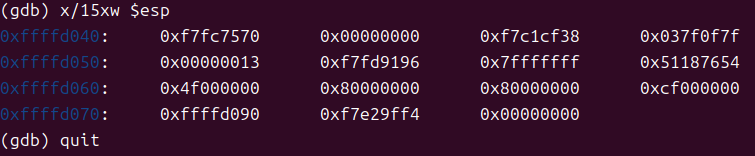
（3）使用objdump指令进行反汇编

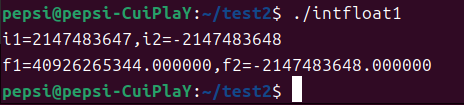
（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.6.3 程序运行截屏







4.6.4 出现问题及解决办法

无

4.7 实验内容七

4.7.1 代码及编程思路

代码：

#include <stdio.h>

int f1(unsigned int n)

{ int sum=1,power=1;

int i;

for(i=0;i<=n-1;i++)

{ power\*=2;

sum+=power;

}

return sum;

}

int main()

{ int sum;

sum=f1(0);

printf("sum=%d\n",sum);

return 0;

}

编程思路：

通过整数类型的转换以及生成反汇编内容来理解C语言中的自动类型转换

4.7.2 实验步骤

（1）编写程序

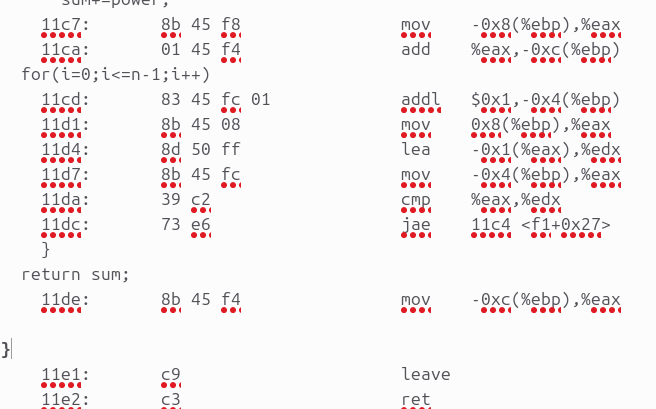
（2）使用指令分别对源程序进行预处理、编译、汇编和链接，最后生成可执行文件

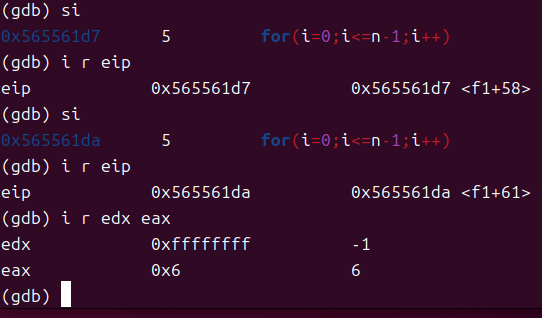
（3）使用objdump指令进行反汇编

（4）使用gdb指令对程序进行调试

（5）运行可执行文件

4.7.3 程序运行截屏





4.7.4 出现问题及解决办法

程序会陷入死循环，应该将f1中unsigned int n修改为int n

**五、实验收获**

吐槽一下实验过程中的心路，如：通过实验，提醒自己以后编程要注意什么

搞清楚什么是真值，以及怎样变成原码、反码和补码。动手把一些数字或者字母转换成这些码，看看它们在计算机里到底是怎么表示的。看看整数、小数、字母等东西在电脑里面是怎么存的。写点小程序，输出各种变量的地址，观察它们在内存中的位置关系。学习下电脑喜欢怎么存放数据，为什么得按照一定的规则来提高速度。通过这次实验，你就能更清楚地了解计算机是怎么处理我们的数据的，以及为啥我们得小心翼翼地对待内存。理论要联系实际，编程时需要细心，并且要经常回顾和总结，才能不断提高自己的编程能力。