计算机系统基础实验总结

院系：计算机学院　学号：　　姓名：

成绩：

LAB1

实验概述

Lab1 数据表示，需要我们在 32 位 Linux 虚拟机上使用有限种类的运算符（按位取反，逻辑与，或，非等），并尽可能的精简步骤来补充 bits.c 中的函数内容，以实现规定的函数功能。函数补充完成后可输入特定指令提交评阅。bits.c 中的函数分为位操作函数，补码运算函数，浮点数表示函数几类，完成实验以掌握各种数据类型在计算机中的表示方法，与 C 语言数据类型的位级表示及操作。

实验收获总结

在对 lab1 中十五个位运算

函数的补充过程中，我初步了解了 Linux 系统的一些命令和操作，也加深了位的表示，二进制补码运算，浮点数 IEEE754 表示等方面的理解。这次试验对我的理论课学习有很大帮助，更引起了我在学习理论与实验课的兴趣。在之后的实验和学习中我希望能学习到更多的有关计算机系统的知识并加以实践应用。

具体题目分析

1. *bitOr*（只利用 ~ 和 & 操作，将数 x 和 y 相或）括号内为题目翻译

   思路：由德摩根定律可知，取或，等价于将两个数的取反值~x, ~y 相与后，再取反。

   代码：int bitOr(int x, int y) {

   return ~((~x)&(~y));

   }

2. *evenBits*（返回值为所有位均是 1 的数）

   思路:所有位均为一，即 0x55555555，即将 0x55 左移八位，再左移十六位。

   代码：int evenBits(void) {

   int a = 0x55 << 8 | 0x55;

   return a << 16 | a;}

3. *isTmin*（判断一个数 x 是否是 int 型最小值 ）

   思路：int 型最小值（补码是 0x80000000）有性质：tmin + tmin = 0. 所以 tmin = ~tmin+1.

   但是要特判去掉同样满足的 0 。

   代码：int isTmin(int x) {

```
return !(x+x)&!!(x);

}
```

4. *allEvenBits*(判断一个二进制数偶数位是否全为 1)

思路：若一个二进制数偶数位为 1，奇数位为 0，则这个数为 0x55555555。

先将 x=x&0x55555555，将这个数奇数为变为 0，之后 x^0x55555555 判断该数是否为 0x55555555。

代码： int allEvenBits(int x) {

```
int y=0x55;
 y |= y<<8;
 y |= y<<16;
 x &= y;
 return !(x^y);
 }
```

5. *anyEvenBit*（判断一个二进制数任意偶数位是否有 1）

思路：判断偶数位是否含有 1，只需要将所有偶数位与 1 相与，奇数位与 0 相与。若结果为 0，则偶数位没有 1。

代码：int anyEvenBit(int x) {

```
int mask = 0x55 | 0x55 << 8;
mask = mask | mask << 16;
return !!(x&mask);}
```

6. *fitsBits*（判断 x 是否能用 n 位补码表示）

思路：判断其[n+1,3]区间上的数是否全为 1，或 0，即可。

所有数位都为 0，即判断!x 是否为 1。

判断一个数都为 1，只需要将这个数取反~x，判断取反之后是否为 0 便可。

代码：int fitsBits(int x, int n) {

```
 int a = n + 31;
    int b = x >> 31;
    x = x >> a;
    x = x ^ b;
    return !x;
}
```

7. *float_neg*（浮点数取反）

思路：取反即可，但需判断是否为 NaN（if 判断）。若是 NaN 数，返回原值，否则返回原数符号位取反对应的数

代码：

```
unsigned float_neg(unsigned uf) {
    int tmp =0,ret=0;
    ret = uf ^ 0x80000000; // sign reverse
    tmp = uf & 0x7fffffff; //
    if(tmp > 0x7f800000)     // NaN
      ret = uf;
      return ret;
      }
```

8. *isAsciiDigit*（判断 x 是否可以是表示数字的 Ascii 码）

思路：判断 x 是否满足 **0x30 <= x <= 0x39**，若满足，返回 **1**。

代码：

```
int isAsciiDigit(int x) {
    int is_upper = !((x & ~0x0F) ^ 0x30);
    int is_lower = !((x & 0x0F) + 0x06 & 0xF0);
```

return is_upper & is_lower;

}

9. *isLess*（判断 x<y）

思路：x 和 y 的符号位不同：如果 x 的符号位为 1 则满足；

x 和 y 的符号位相同：如果 y-x 的符号位为 0 则满足（此时可能相等）。

代码: int isLess(int x, int y) {

int not_y=~y;

return ((((x+ not_y+1)&(x^not_y))|(x&not_y))>>0x1F)&1;}

10. *multFiveEighths*（计算 x*5/8）

思路：x 左移二位加 x 即为乘五，再右移三位即为除以八。

代码：int multFiveEighths(int x) {

x = x + (x<<2);

return (x+(x>>31&7))>>3;}

11. *replaceByte*（用字节数 c 来代替 n 中第 x 字节数）

思路：首先去除 x 的第 n 字节数，与~(0xff<<(n*8))相与，然后与 c<<(n*8)相或。

代码：
```
int replaceByte(int x, int n, int c) {
int mask_ = 0xff << (n<<3);
c <<= (n<<3);
return (x&(~mask_))|c;
}
```

12. *bitParity*（若 x 中含有奇数个 0 返回 1，偶数个 0 返回 0）

思路：偶数与偶数之差为偶数，偶数与奇数之差为奇数。所以 32 位二进制数中 1 和 0 的个数与奇偶性相同。

将 32 位二进制中所有数字进行异或计算。若有偶数个 1 则异或结果为 0，反之。

代码：int bitParity(int x) {

```
    x^=x>>16;
    x^=x>>8;
    x^=x>>4;
    x^=x>>2;
    x^=x>>1;
    return x&1;
}
```

13. *float_half*

思路:先判断是否为 NaN 数

阶码只有最后一位为 1，除 2 之后要变为非规格化数，按非规格化数处理，即处理阶码下溢情况阶码全 0，那要么是 0，要么是非规格化数，直接右移一位同时保留符号位规格化数，正常处理，阶码减一

代码：

```
unsigned float_half(unsigned uf) {
  unsigned s=uf&0x80000000;
    unsigned exp=uf&0x7f800000;
    int lsb=((uf&3)==3);
    if (exp==0x7f800000)
        return uf;
    if (exp<=0x800000)
        return s|(((uf^s)+lsb)>>1);
    if (exp)
        return (uf-0x800000);
```

}

14. *ilog2*

思路: 以第一步为例，x>>16 并且两次逻辑非，那么若 x>>16 大于零则得到 1，说明对应 log(x) >= 4，于是 1<<4

代码：int ilog2(int x) {

```
  int bitsNumber=0;
      bitsNumber=(!!(x>>16))<<4;
      bitsNumber=bitsNumber+((!!(x>>(bitsNumber+8)))<<3);
      bitsNumber=bitsNumber+((!!(x>>(bitsNumber+4)))<<2);
      bitsNumber=bitsNumber+((!!(x>>(bitsNumber+2)))<<1);
      bitsNumber=bitsNumber+(!!(x>>(bitsNumber+1)));
      return bitsNumber;
```

}

15. *sm2tc*（给出原码表示 x，返回其对应的补码表示）

思路：右移 31 位，将结果存为 t。如果原数为正数，那么 t 此时全为 0。而正数的补码不用改变；否则，t 全为 1（负数右移补 1）。负数补码，取反+1，符号位变为 1。

代码：int sm2tc(int x) {

```
  int sign = x >> 31;
  return (x ^ sign) + (((1 << 31) + 1) & sign);
```

}

Lab1 完成！！！

LAB2

实验概述

Lab2 二进制炸弹需要我们在 32 位 linux 虚拟机上用 AT&T 汇编语言在包含 6 个阶段和一个秘密阶段的可执行程序 Binary Bombs 中完成六个阶段的实验。各阶段要求分别输入一个字符串，若输入符合程序预期则通过，否则爆炸，爆炸会扣除一定分数（若担心爆炸次数过多，可在爆炸函数前设断点，防止爆炸）。通过将可执行程序反汇编并用动态分析调试的方法可以得知汇编代码的具体功能，并得出一个字符串作为答案。得出答案后将答案逐行输入文件中，后可输入特定指令提交评阅。

实验收获总结

在对 lab2 的分析调试中我对机器级程序原理的理解，以及通用调试器和逆向工程技能得到了提升，掌握了基本的 gdb 调试命令和 ddd 的一些使用方法，AT&T 语言的解读能力也有了显著提升。完成了 lab2 的六个部分，计算机基础实验课程也已完成了一半，我对课程的学习也进入了更深层次的阶段，对接下来的部分有了期待和自信。

具体题目分析

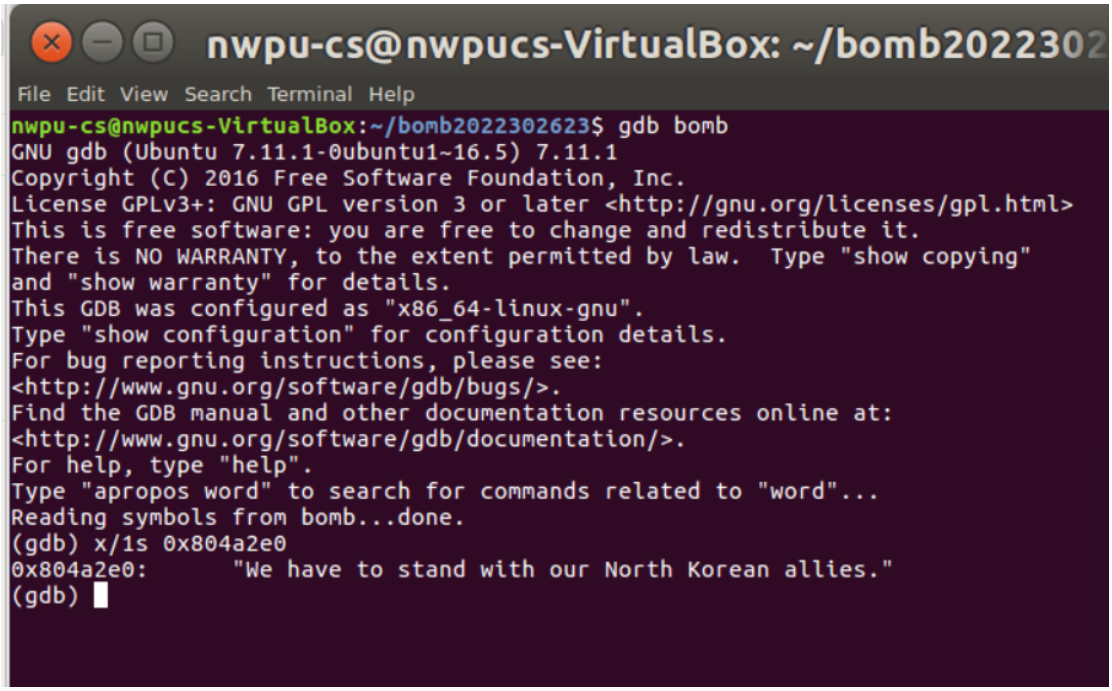
phase_1

本题分析：

首先查看 bomb.c，了解大概的程序。

再输入命令“objdump –d bomb > asm.txt”，对 bomb 进行反汇编并将汇编代码输出到 asm.txt 中

打开反汇编代码，找到 phase_1。根据 strings_not_equal 函数可以看出，该函数为判断字符串是否相等的函数。如果输入的字符串和%eax 里面的字符不相同，则该炸弹爆炸，所以我们通过执行：gdb bomb，查看 0x804a2e0 处存储的数据内容。用 x/1s 0x804a2e0 查看其中的数据，即可得到该阶段的密码。具体操作如图所示



可得出答案为：“We have to stand with our North Korean allies.”

新建一个文本文件，输入第一关的答案

运行 bomb，提交 phase_1 答案，正确。

phase_2

本题分析

首先分析反汇编代码，发现

8048c00: e8 bf 07 00 00 call 80493c4 <read_six_numbers>

可知本题要求输入六个数字组成的字符串。

由 *8048c08: 83 7d dc 00 cmpl $0x0,-0x24(%ebp)*

8048c0c: 79 05 jns 8048c13 <phase_2+0x2c>

可知第一个数要大于零，

由 *8048c13: bb 01 00 00 00 mov $0x1,%ebx*

8048c18: 89 d8 mov %ebx,%eax

8048c1a: 03 44 9d d8 add -0x28(%ebp,%ebx,4),%eax

8048c1e: 39 44 9d dc cmp %eax,-0x24(%ebp,%ebx,4)

可知第二个数要比第一个数大一，否则爆炸，

照此继续分析，可知第三个数比第二个数大二，第四个数比前一个数大三

以此类推，可得六个数。

故答案为："1 2 4 7 11 16"

在文本文件的下一行输入本关的答案

运行 bomb，提交 phase_2 答案，正确。

phase_3

首先分析反汇编代码由 *8048c63: 50 push %eax*

8048c64: 68 36 a3 04 08 push $0x804a336

查看本题要求输入的数据类型