实验2

班级：
姓名：
学号：

实验内容：
1 熟悉基本汇编语言程序；
2 不同类型数据在计算机的编码、存储、转换，整型数据加减运算及其计算机底层实现，浮点数据的表示与运算。

实验目标：
1 理解计算机中数据的表示、存储和运算，熟悉程序的机器级表示；
2 学习和掌握程序的调试方法，强化计算机编程实践能力；
3 掌握 C 语言中位操作语句的使用。

实验任务：
1 学习 MOOC 内容
https://www.icourse163.org/learn/NJU-1449521162
第三周 数据的存储与运算
第 1 讲 真值与机器数
第 2 讲 数据的宽度与存储
第 3 讲 数据类型的转换
第 4 讲 整数加减运算
第 5 讲 浮点数的表示和运算
第四周 程序的机器级表示
第 1 讲 传送指令
第 2 讲 加减运算指令
第 3 讲 整数乘法指令

2 完成实验
2.1 C 语言程序如下，利用反汇编程序代码对运行结果进行解释说明。

```
#include “stdio.h”
void 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);
}
```

2.1.1 程序代码和注释说明

```c
#include<stdio.h>
void 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);
}
```

2.1.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ gcc 2_1.c -o 2_1
2_1.c: In function ‘main’:
2_1.c:7:10: warning: ' ' flag used with ‘%u’ gnu_printf format [-Wformat=]
   printf("au = % u, bu = % u, cu = % u\n", au, bu, cu);
          ^
2_1.c:7:10: warning: ' ' flag used with ‘%u’ gnu_printf format [-Wformat=]
2_1.c:7:10: warning: ' ' flag used with ‘%u’ gnu_printf format [-Wformat=]
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ ./2_1
ai =  100, bi = -2147483648, ci = -100
au = 100, bu = 2147483648, cu = 4294967196
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$
```

反汇编代码：

```
0804840b <main>:
 804840b:       8d 4c 24 04             lea    0x4(%esp),%ecx
 804840f:       83 e4 f0                and    $0xfffffff0,%esp
 8048412:       ff 71 fc                pushl  -0x4(%ecx)
 8048415:       55                      push   %ebp
 8048416:       89 e5                   mov    %esp,%ebp
 8048418:       51                      push   %ecx
 8048419:       83 ec 24                sub    $0x24,%esp
 804841c:       c7 45 e0 64 00 00 00    movl   $0x64,-0x20(%ebp)
 8048423:       c7 45 e4 00 00 00 80    movl   $0x80000000,-0x1c(%ebp)
 804842a:       c7 45 e8 9c ff ff ff    movl   $0xffffff9c,-0x18(%ebp)
 8048431:       c7 45 ec 64 00 00 00    movl   $0x64,-0x14(%ebp)
 8048438:       c7 45 f0 00 00 00 80    movl   $0x80000000,-0x10(%ebp)
 804843f:       c7 45 f4 9c ff ff ff    movl   $0xffffff9c,-0xc(%ebp)
 8048446:       ff 75 e8                pushl  -0x18(%ebp)
 8048449:       ff 75 e4                pushl  -0x1c(%ebp)
 804844c:       ff 75 e0                pushl  -0x20(%ebp)
 804844f:       68 00 85 04 08          push   $0x8048500
 8048454:       e8 87 fe ff ff          call   80482e0 <printf@plt>
 8048459:       83 c4 10                add    $0x10,%esp
 804845c:       ff 75 f4                pushl  -0xc(%ebp)
 804845f:       ff 75 f0                pushl  -0x10(%ebp)
 8048462:       ff 75 ec                pushl  -0x14(%ebp)
 8048465:       68 1e 85 04 08          push   $0x804851e
 804846a:       e8 71 fe ff ff          call   80482e0 <printf@plt>
 804846f:       83 c4 10                add    $0x10,%esp
 8048472:       90                      nop
 8048473:       8b 4d fc                mov    -0x4(%ebp),%ecx
 8048476:       c9                      leave
 8048477:       8d 61 fc                lea    -0x4(%ecx),%esp
 804847a:       c3                      ret
 804847b:       66 90                   xchg   %ax,%ax
 804847d:       66 90                   xchg   %ax,%ax
 804847f:       90                      nop
```

2.1.3 结果分析与讨论

在这段代码中，关键点是对 int 和 unsigned int 变量的赋值和打印。int 类型的范围：-2147483648 到 2147483647；unsigned int类型的范围：0 到 4294967295。

对于 bi，其值被设置为 2147483648，这超出了int的范围。在32位系统中，int的最大值为2147483647，因此超出范围的值会导致溢出。当 bi 被设置为 2147483648 时，实际上被解释为负数 -2147483648。这是因为在补码表示中，最高位表示符号，而 2147483648 在32位二进制中表示为 10000000000000000000000000000000，最高位为1，表示负数。

对于 cu， cu 被设置为 4294967196，这是因为在32位无符号整数中， -100 实际上被解释为 4294967196。这是因为在无符号整数的表示中，所有的位都用于表示数值，不区分正负。

2.2 C 语言程序如下，代码运行过程中各变量存储的机器数分别是什么？i1 和 i2 的值是否相同？f1 和 f2 的值是否相同？利用反汇编程序代码对结果进行解释说明。

```
#include “stdio.h”
int main()
{
  int i1=0x7fffffff, i2, itemp;
  float f1=0x987654321, f2, ftemp;
```

```
ftemp=i1;
i2=ftemp;
itemp=f1;
itemp=f1;
f2=itemp;
    printf("i1=%d, i2=%d, f1=%f, f2=%f\n", i1, i2, f1, f2);
  }
```

2.2.1 程序代码和注释说明

反汇编代码：

```
0804840b <main>:
 804840b:       8d 4c 24 04             lea    0x4(%esp),%ecx
 804840f:       83 e4 f0                and    $0xfffffff0,%esp
 8048412:       ff 71 fc                pushl  -0x4(%ecx)
 8048415:       55                      push   %ebp
 8048416:       89 e5                   mov    %esp,%ebp
 8048418:       51                      push   %ecx
 8048419:       83 ec 34                sub    $0x34,%esp
 804841c:       c7 45 e0 ff ff ff 7f    movl   $0x7fffffff,-0x20(%ebp)
 8048423:       d9 05 58 85 04 08       flds   0x8048558
 8048429:       d9 5d e4                fstps  -0x1c(%ebp)
 804842c:       db 45 e0                fildl  -0x20(%ebp)
 804842f:       d9 5d e8                fstps  -0x18(%ebp)
 8048432:       d9 45 e8                flds   -0x18(%ebp)
 8048435:       d9 7d d6                fnstcw -0x2a(%ebp)
 8048438:       0f b7 45 d6             movzwl -0x2a(%ebp),%eax
 804843c:       b4 0c                   mov    $0xc,%ah
 804843e:       66 89 45 d4             mov    %ax,-0x2c(%ebp)
 8048442:       d9 6d d4                fldcw  -0x2c(%ebp)
 8048445:       db 5d ec                fistpl -0x14(%ebp)
 8048448:       d9 6d d6                fldcw  -0x2a(%ebp)
 804844b:       d9 45 e4                flds   -0x1c(%ebp)
 804844e:       d9 6d d4                fldcw  -0x2c(%ebp)
 8048451:       db 5d f0                fistpl -0x10(%ebp)
 8048454:       d9 6d d6                fldcw  -0x2a(%ebp)
 8048457:       d9 45 e4                flds   -0x1c(%ebp)
 804845a:       d9 6d d4                fldcw  -0x2c(%ebp)
 804845d:       db 5d f0                fistpl -0x10(%ebp)
 8048460:       d9 6d d6                fldcw  -0x2a(%ebp)
 8048463:       db 45 f0                fildl  -0x10(%ebp)
 8048466:       d9 5d f4                fstps  -0xc(%ebp)
 8048469:       d9 45 f4                flds   -0xc(%ebp)
 804846c:       d9 45 e4                flds   -0x1c(%ebp)
 804846f:       d9 c9                   fxch   %st(1)
 8048471:       83 ec 04                sub    $0x4,%esp
 8048474:       8d 64 24 f8             lea    -0x8(%esp),%esp
 8048478:       dd 1c 24                fstpl  (%esp)
 804847b:       8d 64 24 f8             lea    -0x8(%esp),%esp
 804847f:       dd 1c 24                fstpl  (%esp)
 8048482:       ff 75 ec                pushl  -0x14(%ebp)
 8048485:       ff 75 e0                pushl  -0x20(%ebp)
 8048488:       68 30 85 04 08          push   $0x8048530
 804848d:       e8 4e fe ff ff          call   80482e0 <printf@plt>
 8048492:       83 c4 20                add    $0x20,%esp
 8048495:       b8 00 00 00 00          mov    $0x0,%eax
```

2.2.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ gcc 2_2.c -o 2_2
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ ./2_2
i1 =  2147483647, i2 = -2147483648, f1 =  40926265344.000000, f2 = -2147483648.0
00000
```

2.2.3 结果分析与讨论

对main函数进行gdb调试

```
(gdb) ni
0x08048482 in main ()
(gdb) x/16xw $esp
0xbfffee9c:     0x80000000      0x42230eca      0x00000000      0xc1e00000
0xbfffeeac:     0xb7fd51b0      0x00000001      0x00008000      0xb7fbb000
0xbfffeebc:     0x037f0c7f      0x00000001      0x01000000      0x7fffffff
0xbfffeecc:     0x51187654      0x4f000000      0x80000000      0x80000000
```

i1的值为0x7fffffff，i2的值为0x80000000，f1的值为0x51187654，f2的值为0xcf000000，itemp的值为0x80000000，ftemp的值为0x4f000000。

i1=0x7fff ffff; ftemp=i1; i2=ftemp ;

i1: **0x7fff ffff** **补码**

⬇ **= 0111 1111 1111 1111 1111 1111 1111 1111B**

1.11 1111 1111 1111 1111 1111 1111 1111×2^{30} **真值**

+1 **（尾数入操作）**

⬇ **≈10.0×2^{30}=1.0×2^{31}**

ftemp : 0 1001 1110 000 0000 0000 0000 0000 0000B **float**

⬇ **31+127=128+16+8+4+2**

1.0×2^{31} **真值**

⬇ **=1000 0000 0000 0000 0000 0000 0000 0000B**

i2: **1000 0000 0000 0000 0000 0000 0000 0000B** **补码**

=0x8000 0000

补码->float编码->补码，整数与浮点之间的转换不是机器数上的复制，而是编码上的转换。在int->flaot转换中，可能会有精度的损失。

f1=0x987654321；itemp=f1; f2=itemp;

0x987654321=1001 1000 0111 0110 0101 0100 0011 0010 0001B

=1.001 1000 0111 0110 0101 0100 0011 0010 0001×2^{35}

f1: 0 1010 0010 001 1000 0111 0110 0101 0100 float

=0x51187654 35+127=128+32+2

1.001 1000 0111 0110 0101 0100×2^{35} 真值

=1001 1000 0111 0110 0101 0100 0000 0000 0000B

itemp: 1000 0000 0000 0000 0000 0000 0000 0000B 补码

-111 1111 1111 1111 1111 1111 1111 1111B

+1

-1000 0000 0000 0000 0000 0000 0000 0000B 真值

=-1.0×2^{31}

f2: 1 1001 1110 000 0000 0000 0000 0000 0000B float

=0xcf000000 31+127=128+16+8+4+2

float编码->补码-> float编码，整数与浮点之间的转换不是机器数上的复制，而是编码上的转换。在flaot->int转换中，可能会有溢出问题。

2.3 C 语言程序如下，结合反汇编程序代码对运行过程中寄存器 eax，ebx，ecx 中的值进行解释说明。

```
#include “stdio.h”
void main()
{
  int p[2]={0x12345678,0x11223344};
  asm
  (
    “lea -0x14(%ebp),%eax\n\t”
    “mov -0x14(%ebp),%ebx\n\t”
    “mov $1,%ecx\n\t”
    “lea -0x14(%ebp,%ecx,4),%eax\n\t”
    “mov -0x14(%ebp,%ecx,4),%ebx\n\t”
  );
  printf(“understand mov and lea\n”);
}
```

2.3.1 程序代码和注释说明

程序反汇编代码

```
0804846b <main>:
 804846b:       8d 4c 24 04             lea    0x4(%esp),%ecx
 804846f:       83 e4 f0                and    $0xfffffff0,%esp
 8048472:       ff 71 fc                pushl  -0x4(%ecx)
 8048475:       55                      push   %ebp
 8048476:       89 e5                   mov    %esp,%ebp
 8048478:       51                      push   %ecx
 8048479:       83 ec 14                sub    $0x14,%esp
 804847c:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048482:       89 45 f4                mov    %eax,-0xc(%ebp)
 8048485:       31 c0                   xor    %eax,%eax
 8048487:       c7 45 ec 78 56 34 12    movl   $0x12345678,-0x14(%ebp)
 804848e:       c7 45 f0 44 33 22 11    movl   $0x11223344,-0x10(%ebp)
 8048495:       8d 45 ec                lea    -0x14(%ebp),%eax
 8048498:       8b 5d ec                mov    -0x14(%ebp),%ebx
 804849b:       b9 01 00 00 00          mov    $0x1,%ecx
 80484a0:       8d 44 8d ec             lea    -0x14(%ebp,%ecx,4),%eax
 80484a4:       8b 5c 8d ec             mov    -0x14(%ebp,%ecx,4),%ebx
 80484a8:       83 ec 0c                sub    $0xc,%esp
 80484ab:       68 60 85 04 08          push   $0x8048560
 80484b0:       e8 8b fe ff ff          call   8048340 <puts@plt>
 80484b5:       83 c4 10                add    $0x10,%esp
 80484b8:       90                      nop
 80484b9:       8b 45 f4                mov    -0xc(%ebp),%eax
 80484bc:       65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 80484c3:       74 05                   je     80484ca <main+0x5f>
 80484c5:       e8 66 fe ff ff          call   8048330 <__stack_chk_fail@plt>
 80484ca:       8b 4d fc                mov    -0x4(%ebp),%ecx
 80484cd:       c9                      leave
 80484ce:       8d 61 fc                lea    -0x4(%ecx),%esp
 80484d1:       c3                      ret
```

2.3.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ gcc 2_3.c -o 2_3
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ ./2_3
understand mov and lea
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ objdump -S 2_3 > 2_3.s
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$
```

对程序进行gdb调试，在0x8048495处查看寄存器esp的内容

```
0x08048495 in main ()
(gdb) x/7xw $esp
0xbfffeed0:     0x00000001      0x12345678      0x11223344      0x2107a900
0xbfffeee0:     0xb7fbb3dc      0xbfffef00      0x00000000
(gdb)
```

0x12345678就是p0的值，0x11223344就是p1的值。

执行一个mov和lea指令后，查看寄存器eax和ebx。

```
0x0804849b in main ()
(gdb) i r eax ebx
eax            0xbfffeed4       -1073746220
ebx            0x12345678       305419896
(gdb)
```

可以发现，寄存器eax存放p0的地址，寄存器ebx存放p0的值。

执行一个mov和lea指令后，再次查看寄存器eax和ebx。

```
0x080484a8 in main ()
(gdb) i r eax ebx
eax            0xbfffeed8       -1073746216
ebx            0x11223344       287454020
(gdb)
```

可以发现，寄存器eax存放p1的地址，寄存器ebx存放p1的值。

2.3.3 结果分析与讨论

lea指令实现的是地址传送，mov实现的是数据传送。

2.4 只用运算符～和|来实现位的与操作函数：

int bitAnd(int x, int y)

例如：bitAnd(6, 5) = 4

2.4.1 程序代码和注释说明

```c
# include<stdio.h>
int bitAnd(int x, int y) {
 return ~(~x | ~y);
}

void main() {
 int ans = bitAnd(6, 5);
 printf("%d\n", ans);
}
```

2.4.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ gcc 2_4.c  -o 2_4
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ ./2_4
4
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$
```

2.4.3 结果分析与讨论

要利用运算符~和|来实现bitand的功能，只需要将两个操作数先分别取反，然后做或操作，结果再取反即可。

2.5 只用运算符!～&^|+<<>>实现比较 x 和 y 的大小的函数：

int isLessOrEqual(int x, int y)

例如：isLessOrEqual(4, 5) = 1

2.5.1 程序代码和注释说明

```c
# include<stdio.h>
int isLessOrEqual(int a, int b) {
 int diff = a ^ b;
 if (!diff) return 1; //a = b

 // 001xxxxx -> 00100000
 diff |= diff >> 1;
 diff |= diff >> 2;
 diff |= diff >> 4;
```

```
10.  diff |= diff >> 8;
11.  diff |= diff >> 16;
12.  diff ^= diff >> 1;
13.
14.  if (!(a & diff))
15.   return 1; //a < b
16.  else
17.   return 0; //a > b
18. }
19. void main() {
20.  int ans = isLessOrEqual(4, 5);
21.  printf("%d\n", ans);
22. }
```

2.5.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ gcc 2_5.c -o 2_5
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$ ./2_5
1
sanfenbai@ubuntu:~/Desktop/计算机系统/实验2$
```

2.5.3 结果分析与讨论

只需要找出第一个从最高位开始找出第一个不同的bit，这一位是1的数是较大者。