深圳大学实验报告

课程名称：计算机系统(2)

实验项目名称：数据表示实验

学院：计算机与软件学院

专业：软件工程

指导教师：冯禹洪

报告人：郑杨 学号：2020151002 班级：腾班

实验时间：2022 年 4 月 12 日

实验报告提交时间：2022 年 4 月 13 日

教务处制

实验目的与要求：

1. 了解各种数据类型在计算机中的表示方法
2. 掌握 C 语言数据类型的位级表示及操作

方法、步骤：

1、安装 gcc-multilib：

```
test@szu-VirtualBox:/media/sf_计系2/datalab-handout$ sudo apt-get install gcc-multil
```

或者：

```
test@szu-VirtualBox:/media/sf_计系2/datalab-handout$ su
Password:
root@szu-VirtualBox:/media/sf_计系2/datalab-handout# apt-get install gcc-multilib
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc-multilib is already the newest version (4:7.2.0-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 84 not upgraded.
```

2、根据 bits.c 中的要求补全以下的函数：

intbitXor(int x, int y);
inttmin(void);
intisTmax(int x);
ntallOddBits(int x);
int negate(int x);
intisAsciiDigit(int x);
int conditional(int x, int y, int z);
intisLessOrEqual(int x, int y);
intlogicalNeg(int x);
inthowManyBits(int x);
unsignedfloat_twice(unsigned uf);
unsigned float_i2f(int x);
int float_f2i(unsigned uf);

3、在 Linux 下测试以上函数是否正确，指令如下：
*编译：./dlcbits.c
*测试：makebtest
./btest

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

实验过程及内容：

由于本人在做这个实验之前已经做过 CMU15213 的 datalab 实验了，所以实验环境已经配置好了。这里主要讲解各个函数的设计思路与具体实现过程。

1. $int\ bitXor(int\ x, int\ y)$

题意：

这一题的题意是让我们使用与运算（&）和非运算（~）实现异或运算（^）。

思路：

这一题要求只能使用&和~运算符实现异或运算（^）。

首先我们把x^y看成是x和y的每一位进行异或之后拼起来得到的结果。然后我们观察一下&和^位运算的真值表：

x	y	x&y	x^y	~ (x&y)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

我发现x&y与x^y有三个结果是相反的，所以我把x&y取反之后，~(x&y)与x^y就只有一个位置的结果不同了，如上图。那么我的想法就是把这个位置也变为相同。

我又观察了x|y的真值表，

x	y	x&y	x^y	~ (x&y)	x|y
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	0	0	1

然后我发现，~(x&y)&(x|y)就会等于x^y。但是这道题限定了只能使用&和~运算符。那么现在的问题就转换成了使用&和~运算符表示|运算符。接下来分析一下这个问题的求解思路。由德摩根率可以得到：~(x |y) = (~x)&(~y),那么有：

$$(x|y) = \sim(\ (\sim x)\&(\sim y)\ )$$

于是：

$$x\hat{}y\ =\ \big(\sim(x\&y)\big)\ \&\ (\ \sim(\ (\sim x)\&(\sim y)\ )\ )$$

代码：

代码实现非常的简单，就直接返回结果即可：

```
int bitXor(int x, int y) {
  return (~(x & y)) & (~((~x) & (~y)));
}
```

2. $int\ tmin(void)$

题意：

这一题的题意是让我们返回 32 位二进制补码表示下的最小值。

思路：

这一题可能是最简单的一道题了。

由于返回值是int类型的，且 32 位补码表示法下的最小值就是：最高位为 1，其他为 0：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

$$1\,0\,0\,0\,\cdots\cdots 0\,0$$
$$\underbrace{\qquad\qquad}_{31个}$$

由于可以使用 0~255 的常数，直接把 1 左移 31 位即可得到答案。

代码：

```
int tmin(void) {
        return (0x1 << 31);
}
```

3. *int isTmax(int x)*

题意：

这一题的题意是让我们判断某一个 32 位数字x是否为$Tmax$（0x7FFFFFFF）

思路：

一开始的想法是，把x和$Tmax$做异或运算，若x与$Tmax$相等则运算结果为 0；不相等则运算结果不为 0。但是由于要求只能用 0~255 的常数值，想要表示$Tmax$必须做移位操作（先获得$Tmin$再减 1），但是又由于要求不能使用移位操作符，所以必须换一个思路。

由于$Tmax = Tmin - 1$，同时$Tmax + Tmin$ =0xFFFFFFFF，故有：

$$\sim(Tmax + Tmin) = 0$$

即：$\sim\big(Tmax + (Tmax + 1)\big) = 0$

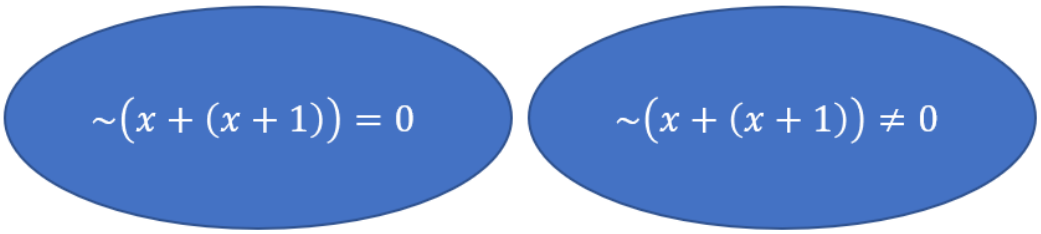
利用这一特点，若$x = Tmax$，那么$\sim\big(x + (x + 1)\big) = 0$。

那么是否存在一个y，满足$y\,! = Tmax$且$\sim\big(y + (y + 1)\big) = 0$呢？

我们发现，当$y = -1$时，也会满足这一个式子，对于y的其他取值，就不满足。

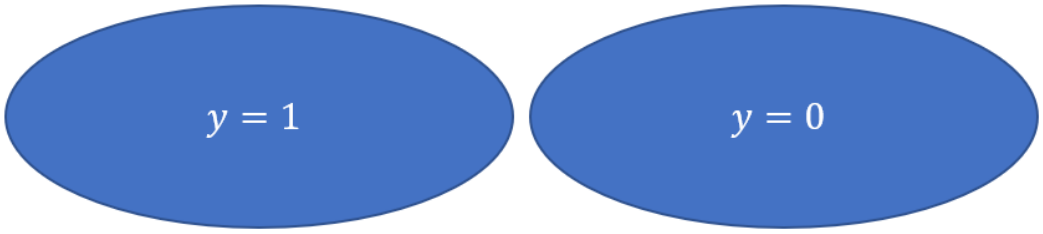
那么现在的问题就是，-1和$Tmax$都满足上述式子，必须把它们区分开才能判断当前值x是否为$Tmax$。相当于我们把所有 32 位数字划分为两类，如下图所示：

$x = -1, x = Tmax$　　$x \neq -1, x \neq Tmax$



我们可以把上面两类用$bool$类型变量区分开来，定义$y = !(\sim\big(x + (x + 1)\big))$，那么上图可以变为下图：

$x = -1, x = Tmax$　　$x \neq -1, x \neq Tmax$



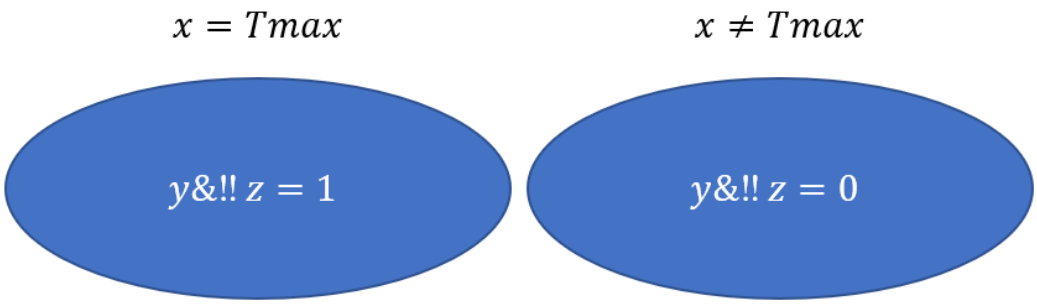
由于-1的特点就是$-1 + 1 = 0$，而对于其他32位数字x而言都有$x + 1 \neq 0$，令$z = x +$

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

1，对于所有的 32 位数字我们可以列出下面这个表格：

x	y	!!z
Tmax	1	1
-1	1	0
else	0	1

如果使$(y\&!!z)$去划分所有数字的话，可以表示为下图：



此时可以将$Tmax$与其他所有数做出划分。

代码：

由上述思路，$y = !(\sim(x + (x + 1)))$，$z = x + 1$，当$x = Tmax$时返回 1，否则返回 0，故代码直接返回$y\&!!z$。

```
int isTmax(int x) {
        int y = !(~(x + (x + 1)));
        int z = x + 1;
        return (y & !!z);
}
```

4. $int\ allOddBits(int\ x)$

题意：

这一题是让我们判断某个 32 位数字x的二进制表示中，是否所有的奇数位都为 1（位数计算从 0 开始，即 0~31）

思路：

我们可以通过x&0xAAAAAAAA 的结果来判断，0xAAAAAAAA 的位级表示如下：

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

就是每一个偶数位都为 0，每一个奇数位都为 1。

令y =0xAAAAAAAA，那么如果$x\&y = y$，则说明x中所有奇数位都为 1。这是一个掩码的思想，因为只有当x的奇数位都为 1 时，y的任一个奇数位才不会变为 0，而又因y的偶数位为 0，那么$x\&y$的偶数位必然为 0，所以结果取决于x的奇数位是否都为 1。

由于不能直接使用 0xAAAAAAAA 这个大常数，我们只能使用 0x00~0xFF 之间的常数间接得到0xAAAAAAAA。在这里我的想法是，使用0xAA 进行变换得到0xAAAAAAAA不难想到 0xAAAA=(0xAA << 8) + 0xAA，0xAAAAAAAA=(0xAAAA << 16) + 0xAAAA。

那么如何判断$x\&y == y$呢？由于这里不能使用==运算符，我使用了^运算符，如果$x\&y == y$则有，$(x\&y)\hat{}y = 0$，故返回值为$!((x\&y)\hat{}y)$。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

代码：

令$y = 0xAAAA$，

进行上述两种变换，$y = (y \ll 8) + y, y = (y \ll 16) + y$得到 0xAAAAAAAA。

返回$!((x\&y)\hat{}y)$即可。

```
int allOddBits(int x) {
        int y = 0xAA;
        y = y + (y << 8);
        y = y + (y << 16);
        return !((x & y) ^ y);
}
```

5. *int negate(int x)*

题意：

这一题的题意很简单，就是返回$-x$。

思路：

由补码的知识可以知道$-x = \sim x + 1$，直接返回即可。

代码：

```
int negate(int x) {
        return ~x + 1;
}
```

6. *int isAsciiDigit(int x)*

题意：

题意就是判断某个数x是否是 ASCII 码表示的数字，就是判断 0x30≤ x ≤0x39 是否成立

思路：

首先转换一下问题，判断 0x30≤ x ≤0x39 是否成立等价于判断x −0x30≥ 0与 0x39−x ≥ 0是否同时成立。令$l = 0x30, r = 0x39$，问题变为判断$x - l \geq 0$与$r - x \geq 0$是否同时成立。由上一题可以知道，$-l = \sim l + 1, -x = \sim x + 1$，故$x - l = x + (\sim l + 1), r - x = r + (\sim x + 1)$。现在我们就只需要判断$x + (\sim l + 1)$与$r + (\sim x + 1)$是否同时为非负数即可。

非负数即不是负数，只需取出他们的符号位（最高位），如果符号位为 1 则为负数，为 0 则为非负数。

问题转化为求某个数y的符号位（最高位），可以通过运算$(y \gg 31)\&1$得到。

最后还有一个问题，就是x为负数时可能溢出。由于x为负数时显然不在所要求范围内，故只需取出x的符号位进行判断即可。

令sx为x的符号位，$s1$为$x + (\sim l + 1)$的符号位，$s2$为$r + (\sim x + 1)$的符号位，那么只有当sx、$s1$、$s2$同时为 0 时才返回 1，故返回值为$(!sx)\&(!s1)\&(!s2)$

代码：

按照思路里所述直接实现即可。

```
int isAsciiDigit(int x) {
        int sx = (x >> 31) & 1;
        int l = 0x30, r = 0x39;
        l = ~l + 1;
        int s1 = x + l;
        x = ~x + 1;
        int s2 = x + r;
        s1 = (s1 >> 31) & 1;
        s2 = (s2 >> 31) & 1;
        return (!sx) & (!s1) & (!s2);
}
```

7. *int conditional(int x, int y, int z)*

题意：

这一题的题意为实现三目操作符(x ? y : z)。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

思路：

由三目操作符（?:）的定义可知，当$x=0$时，返回值为z；$x=1$时，返回值为y。

我的思路是，根据x的值让y和z某个值变为 0 同时另一个值保持不变，然后把两个值进行按位或(|)运算输出结果。把某个值变为 0 一种简单的位运算就是&0运算，把某个数跟 0 进行按位与(&)运算。保持某个值不变可以把那个值跟 0xFFFFFFFF 进行按位与(&)由于需要根据x的值，我们可以把x转化为 0 或者 0xFFFFFFFF 的其中一个掩码，则$\sim x$为另一个掩码。

首先把x转换为$bool$类型，即进行运算$!!x$，令$s=!!x$。然后取$s=-s$，当$s=0$时，会变为 0；$s=1$时，会变为$0xFFFFFFFF$，刚好对应了上述两个掩码！故当x为 0，时最终的$s=0$；当$x!=0$时，最终的$s=0xFFFFFFFF$。由于当$x=0$时，需保持z不变并把y变为 0，此时$s=0xFFFFFFFF$，需让$s\&z$保持z不变，让$\sim s\&y$把y变为 0；当$x!=0$时，需要保持y不变并把z变为 0，此时$s=0$，需让$\sim s\&y$保持y不变，让$s\&z$把z变为 0。故返回值可统一写为：

$$(s\&y)|(\sim s\&z)$$

代码：

```
int conditional(int x, int y, int z) {
        x = !!x;
        x = ~x + 1;
        return (x & y) | (~x & z);
}
```

8. $int\ islessOrEqual(int\ x, int\ y)$

题意：

判断$x \le y$是否成立。

思路：

首先把问题转换为判断$y-x \ge 0$，然后分为以下两种情况：

（1） y与x符号相同

此时直接计算$y-x=y+(\sim x+1)$，判断该值符号位是否为 0 即可。

（2） y与x符号不同

此时就不能直接计算$y-x$，因为有可能出现溢出的情况。当y为负数，x为非负数时，$x \le y$显然不成立；而当y为非负数，x为负数时，$x \le y$显然成立。那么在y与x符号不同时，就只有y的符号位为 0，且x的符号位为 1 时结论成立。

分析完两种情况之后，我们把两种情况得到的$bool$值进行按位或运算(|)即可得到最终答案。

代码：

令sx为x的符号位，sy为y的符号位，s为$y+(\sim x+1)$的符号位

则情况 1 可以表示为$(!(sx\hat{}sy))\&(!s)$

情况 2 可以表示为$(sx\hat{}sy)\&(!sy)\&sx$

```
int isLessOrEqual(int x, int y) {
        int sx = (x >> 31) & 1;
        int sy = (y >> 31) & 1;
        int s = ((y + (~x + 1)) >> 31) & 1;
        int f1 = (!(sx ^ sy)) & (!s);
        int f2 = (sx ^ sy) & (!sy) & sx;
        return f1 | f2;
}
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

9. *int logicalNeg(int x)*

题意：

实现逻辑非运算符(!)。

思路：

考虑x与$y=-x$的符号位，我们发现，当$x=0$与$x=Tmin$时，x与y的符号位相同；当x取其他值时，x与y的符号位不同。令sx为x的符号位，sy为y的符号位，则有下表：

x	sx	sy
Tmin	1	1
0	0	0
>0	0	1
<0 !=Tmin	1	0

由于$sx=(x\gg 31)\&1, sy=(y\gg 31)\&1$,故返回值为$((\sim x\&\sim(\sim x+1))\gg 31)\&1$

代码：

```
int logicalNeg(int x) {
        return ((~x & ~(~x + 1)) >> 31) & 1;
}
```

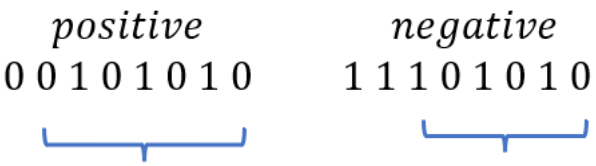
10. *int howManyBits(int x)*

题意：

求出表示x最少所需的二进制位数。

思路：

首先把x分为两种情况，负数和非负数。对于负数的话，我们需要找到位级表示中从高位到低位的第一个 0 的位置；对于非负数的话，我们需要找到位级表示中从高位到低位第一个 1 的位置，然后加上符号位得到最少的位数表示，如下图所示：



因为对于负数而言，高位上连续的 1 只需要保留最后一个，非负数也是类似的道理。由于这样无法统一处理所有的数，我们可以把负数按位取反，统一所有数的处理方法为找到从高位到低位的第一个 1 的位置。所以首先如果x为非负数则保持不变，x为负数则按位取反得到$\sim x$。之前的题目也说过，处理这种二元问题可以使用掩码的思路，把一种情况和 0 做与运算(&)变为 0，另一种情况和$0xFFFFFFFF$做与运算(&)保持不变，再将两个结果做或运算(|)得到最终结果。在这里我们可以获取x的符号位sx，然后做出以下运算把x统一化：

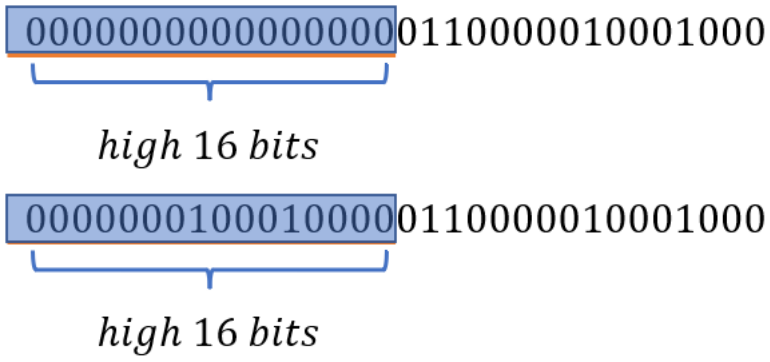
$$x=((sx+\sim 0)\&x)|(\sim(sx+\sim 0)\&\sim x)$$

接下来的问题是，如何找到x的二进制位级表示中，从高位到低位的第一个 1 的位置。由于题目要求不能使用循环，我们可以利用二分的思想解决这个问题。

对于一个32位二进制整数x，若从高位到低位的第一个 1 的位置不在高 16 位，那么高 16 位必然为连续 0 且第一个 1 的位置必然在低 16 位；若从高位到低位第一个 1 的位置

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

在高 16 位，那么我们可以确定所需要的位数至少需要16位（低 16 位）。如下图：



也就是说，我们可以把范围缩小到后 16 位或者前 16 位，只需要记录是否加上了 16 位即可区分这两种情况。那么现在问题转换为找到 16 位数字中从高位到低位第一个 1 的位置，我们也可以使用类似 32 位的解决方案，考虑 16 位数字的高 8 位中是否存在 1（是否高 8 位都为 0），这样就可以继续缩小问题为 8 位数字。依次类推，不断缩小问题规模直到只剩下 1 位。

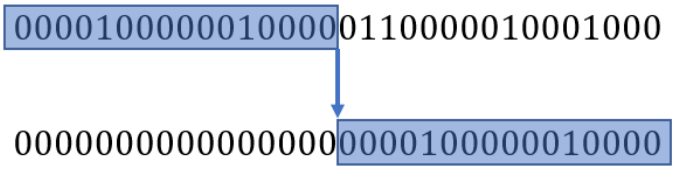
具体而言，以 32 位的问题为例，假设x为 32 位整数，判断x高 16 位是否全为 0 只需要把x右移 16 位，判断$x \gg 16$是否为 0。若$(x \gg 16) == 0$，则x的高 16 位都为 0；反之，x的高 16 位存在 1。令$t16 = !!(x \gg 16)$，则可以表示x的高 16 位是否存在 1。若高 16 位存在 1，则$t16 = 1$；若不存在 1，则$t16 = 0$。此时，$t16 = 1$表示至少需要 16 位来表示x，那么把$t16 *= 16$即可，但是这里没有$*$运算符，可以用$t16 \ll= 4$来表示。最后，我们需要把x缩小为 16 位，若$t16 = 0$，则x可缩小为低 16 位；反之，x可缩小为高 16 位。这里直接把$x \gg= t16$即可。之后按照类似的方法可以算出$t8, t4, t2, t1, t0$。最终的答案就是$t16 + t8 + t4 + t2 + t1 + t0 + 1$，加上 1 是因为我们求出的是从高位到低位第一个 1 的位置，而最终需要加上符号位 0。

接下来以一个例子更加具体地说明这个算法。

假设x的位级表示如下：

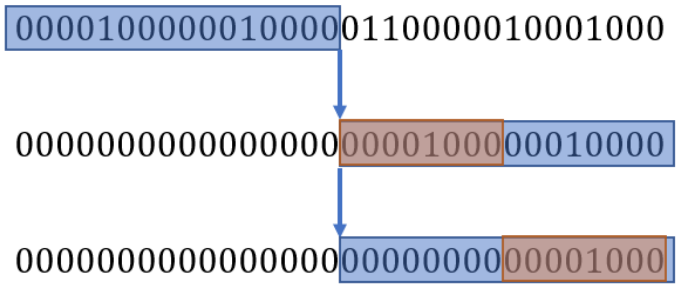
$$0000100000010000011000010001000$$

由于$x \gg 16 = 0000100000010000 \neq 0$，故$t16 = (!!(x \gg 16)) \ll 4 = 16$，那么$x$需要缩小为高 16 位，即$x = x \gg t16 = x \gg 16$，如下：

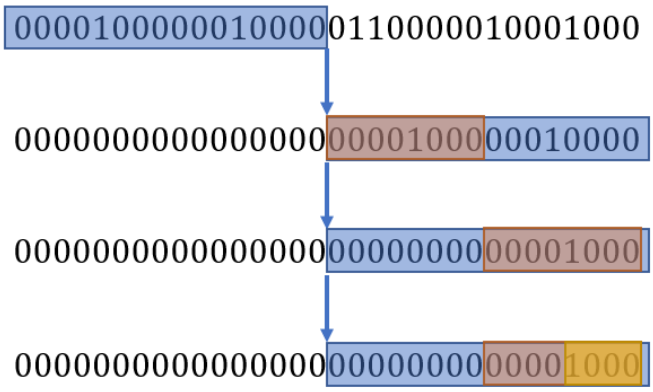


接下来考虑x的高八位，由于$x \gg 8 = 00001000 \neq 0$，故$t8 = (!!(x \gg 8)) \ll 3 = 8$，那么$x$需要缩小为高 8 位，即$x = x \gg t8 = x \gg 8$，如下：

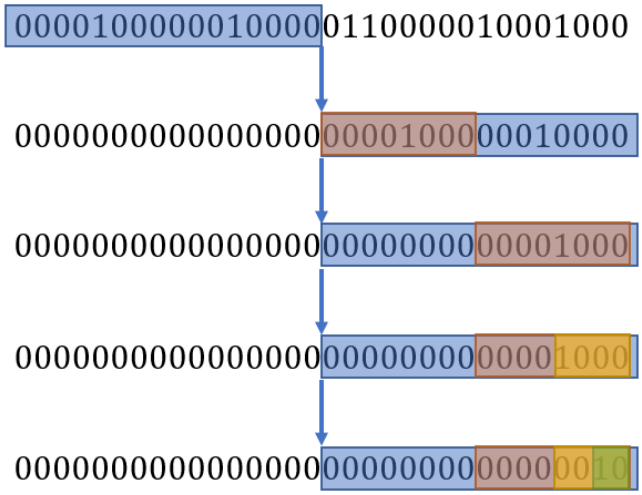
注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。



接下来考虑x的高四位，由于$x \gg 4 = 0000 = 0$，故$t4 = (!!(x \gg 4)) \ll 2 = 0$，那么$x$需要缩小为低 4 位，即$x = x \gg t4 = x \gg 0$，如下：



接下来考虑x的高二位，由于$x \gg 2 = 10 \neq 0$，故$t2 = (!!(x \gg 2)) \ll 1 = 2$，那么$x$需要缩小为高 2 位，即$x = x \gg t2 = x \gg 2$，如下



接下来考虑x的高一位，由于$x \gg 1 = 1 \neq 0$，故$t1 = (!!(x \gg 1)) \ll 0 = 1$，那么$x$需要缩小为高 1 位，即$x = x \gg t1 = x \gg 1$，此时$x = 1$，被缩小为 1 位，则$t0 = x = 1$

故答案为：

$$t16 + t8 + t4 + t2 + t1 + t0 + 1 = 16 + 8 + 0 + 2 + 1 + 1 + 1 = 29$$

代码：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

```
int howManyBits(int x) {
        int sx = (x >> 31) & 1;
        x = ((sx + ~0) & x) | (~(sx + ~0) & ~x);
        int t16 = (!!(x >> 16)) << 4;
        x = x >> t16;
        int t8 = (!!(x >> 8)) << 3;
        x = x >> t8;
        int t4 = (!!(x >> 4)) << 2;
        x = x >> t4;
        int t2 = (!!(x >> 2)) << 1;
        x = x >> t2;
        int t1 = (!!(x >> 1));
        x = x >> t1;
        int t0 = x;
        return t16 + t8 + t4 + t2 + t1 + t0 + 1;
}
```

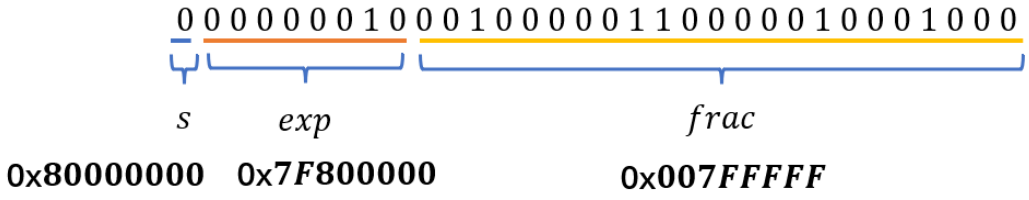
11. *unsigned float_twice(unsigned uf)*

题意：

把二进制位级表示的$float$浮点数乘以 2 之后返回，若该值为 NaN，则直接返回该值。即实现$2 * uf$。

思路：

首先我们可以使用掩码的思想取出$float$类型数字uf的二进制数表示下的符号位、指数位和尾数位，分别表示为$s = uf \ \& \ 0x80000000, exp = uf \ \& \ 0x7F800000, frac = uf \ \& \ 0x007FFFFF$。如下图：



然后按照exp把所有$float$类型浮点数分为三类：非规格化$(exp = 0)$，规格化$(0 < exp < 255)$和特殊数字$(exp = 255)$。

（1） $exp = 0$

当$exp = 0$时，为非规格化浮点数，此时如果尾数部分的第一位为 1，那么乘以 2 之后，会变为规格化浮点数；若尾数部分的第一位为 0，则乘以 2 相当于把尾数部分向左移动 1 位。于是取出尾数部分的第一位进行判断，也是利用掩码的思想，取出尾数部分第一位的掩码为$0x00400000$。

（2） $0 < exp < 255$

此时为规格化浮点数，规格化浮点数进行乘二运算，则直接把$exp + 1$即可。注意这里加 1 指的是加上$0x00800000$。若$exp + 1 = 255$，则溢出为无穷大，此时需要把$frac$部分清空。

（3） $exp = 255$

此时为特殊数字，直接返回参数即可。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

代码

```
unsigned float_twice(unsigned uf) {
        int s = uf & 0x80000000;
        int exp = uf & 0x7F800000;
        int frac = uf & 0x007FFFFF;
        if (!exp)
        {
                if (frac & 0x00400000)
                {
                        exp += 0x00800000;
                        frac ^= 0x00400000;
                        frac <<= 1;
                }
                else
                        frac <<= 1;
        }
        else if (exp == 0x7F800000)
                return uf;
        else
        {
                exp += 0x00800000;
                if (exp == 0x7F800000)
                        frac = 0;
        }
        return s | exp | frac;
}
```

12. $unsigned\ float\_i2f(int\ x)$

题意：

实现把int类型转换为$float$类型，即实现$(float)x$

思路：

由于int类型的范围为$[-2^{31}, 2^{31}-1]$，使用$float$类型表示不会产生溢出。由于$float$的正负只由符号位决定，我们可以先取出x的符号位s，然后把负数转换为其相反数进行计算（为了和正数统一）。但是由于0和$Tmin$的相反数（补码）都为他们本身，所以需要先特判以下，分为以下三种情况：

（1） $x = 0$

直接返回 0 即可。

（2） $x = Tmin$

由于$Tmin$为-2^{31}，则$s = 0x80000000$，$exp = (31 + 127) \ll 23$，$frac = 0$。直接返回$s|\text{exp}\ |frac$即可。

（3） $else$

由于前面提到已经将所有负数都转换为正数，这里就直接展开正整数转换为浮点数的解决方案。考虑当前正整数从高位到低位的第一个 1 的位置i，

若$i \le 23$，则$float$的精度足以精确表示这个整数，直接计算$frac = \big(x \ll (23 - i)\big) - (1 \ll 23)$即可。

若$i > 23$，则$float$的精度不足以表示这个整数，这时候需要进行舍入。此时我的做法是获取超过23位的那些数的大小，判断是否超过一半，超过一半则向上舍入，未超过一半则向上舍入，若恰好为一半则向偶数舍入。若舍入之后$frac = (1 \ll 23)$，则需把$exp + 1$并把$frac$清零。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

代码：

```
unsigned float_i2f(int x) {
        if (x == 0) return 0;
        else if (x == (1 << 31)) return ((31 + 127) << 23) + 0x80000000;
        int s = x & 0x80000000;
        if (s) x = ~x + 1;
        int i;
        for (i = 31; i >= 0; i -- ) if (x & (1 << i)) break;
        int exp = (i + 127) << 23;
        int frac;
        if (i <= 23)
                frac = (x << (23 - i)) - (1 << 23);
        else
        {
                frac = (x >> (i - 23)) - (1 << 23);
                int t = 0;
                for (int j = (i - 23 - 1); j >= 0; j -- )
                {
                        t <<= 1;
                        if (x & (1 << j)) t ++;
                }
                int half = (1 << (i - 23 - 1));
                if (t > half) frac ++;
                else if(t == half && (frac & 1)) frac ++;
                if (frac == (1 << 23)) frac = 0, exp += (1 << 23);
        }
        return s | exp | frac;
}
```

13. *int float_f2i(unsigned uf)*

题意：

实现把$float$类型转换为int类型，即实现$(int)uf$

思路：

由于 C 语言对于这一个强制类型转换的操作是把浮点数直接下取整得到整数，那么对于指数小于 0 的那一部分浮点数，返回值都为 0。

我们需要得到浮点数uf的指数部分，由于$s = uf \ \& \ 0x80000000, exp = uf \ \& \ 0x7F800000, frac = uf \ \& \ 0x007FFFFF$，故实际的指数$E = (exp \gg 23) - 127$。

由于 32 位整数int所表示的范围为$[-2^{31}, 2^{31} - 1]$，我把E分为了几个情况：

（1） $E < 0$

由于此时表示的浮点数绝对值都小于 1，转换为整形时为 0，直接返回 0 即可。

（2） $0 \le E < 31$

此时表示的浮点数都在int类型所表示的范围内，由于此时的浮点数都为规格化浮点数，尾数部分大于等于 1，我们可以先把$frac += (1 \ll 23)$。然后如果$E \le 23$，那么说明$frac$里的数无法完全转化为整数，则把$frac$往右移一些位来进行下取整，即$frac \gg= (23 - E)$；若$E > 23$，则说明$frac$里面的所有位都可以转化为整数，此时需要进行$frac \ll= (E - 23)$。

（3） $E = 31$

此时若浮点数uf是正数$(s = 0)$，则超出整形表示范围，返回$0x80000000$。若此时uf为负数，并且$frac$部分不为 0，那么也超出表示范围，返回$0x80000000$；若$frac$为 0，那么返回$(1 \ll 31)$。

（4） $E > 31$

由于 32 位整数int所表示的范围为$[-2^{31}, 2^{31} - 1]$，故当$E > 31$时，超出表示范围，返回$0x80000000$。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

代码：

```
int float_f2i(unsigned uf) {
        int s = uf & 0x80000000;
        int exp = uf & 0x7F800000;
        int frac = uf & 0x007FFFFF;
        int E = (exp >> 23) - 127;
        if (E < 0) return 0;
        else if (E < 31)
        {
                frac = frac + (1 << 23);
                if (E <= 23) frac = frac >> (23 - E);
                else frac = frac << (E - 23);
                return s ? -frac : frac;
        }
        else if (E == 31)
                return 0x80000000;
        else
        {
                if (!s) return 0x80000000;
                else if (frac) return 0x80000000;
                else return (1 << 31);
        }
}
```

实验结论：

编译并运行上述所有代码，均通过测试！

```
zhengyang_2020151002@zy-virtual-machine:~/csapp/datalab/datalab-handout$ ./btest
Score   Rating  Errors  Function
 1      1       0       bitXor
 1      1       0       tmin
 2      2       0       isTmax
 2      2       0       allOddBits
 2      2       0       negate
 3      3       0       isAsciiDigit
 3      3       0       conditional
 3      3       0       isLessOrEqual
 4      4       0       logicalNeg
 4      4       0       howManyBits
 4      4       0       float_twice
 4      4       0       float_i2f
 4      4       0       float_f2i
Total points: 37/37
```

心得体会：

通过本次数据表示实验，深刻理解了计算机的位级运算，掌握了一些位级运算的巧妙运用方法。对实验过程和思路进行了比较详细的阐述，加强了自身的逻辑表达能力。对于其中的一些题目使用了常用的算法技巧（如二分），也使用了集合论进行题目分析。特判也是很重要的一部分，在最后三道关于浮点数的题目中，使用了一些特判特判一些特殊情况，使得大部分的代码在处理大部分情况。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

指导教师批阅意见：   成绩评定： 指导教师签字： 冯禹洪 2022 年 月 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。