**武汉大学计算机学院**

**计算机基础课程实验报告（一）**

院（系）名 称：计算机学院

专 业 名 称 ：计算机科学与技术

学 生 姓 名 ：杜沛然

学 生 学 号 ：2019300003090

任 课 教 师 ：蔡朝晖

二〇二〇年十月

1. 实验目的

本次实验是对位操作级别的练习，本次报告是对实验中bits.c文件的再次说明，目的是帮助理解位级别的操作和表达以及培养同学们写代码之前先思考清楚思路的良好习惯，需要同学们认真完成。

1. 报告要求

本次报告需要同学们把对应实验中的函数逐一进行分析说明，写出实现的依据，也就是推理过程，可以是一个简单的数学证明，也可以是代码分析，根据实现中你的想法不同而异。请同学们耐心独立完成。

1. 函数分析
2. **int bitAnd(int x, int y);**

函数实现：

int bitAnd(int x, int y) {

/\* bitAnd is the same as bitOr after bitNot \*/

return ~(~x | ~y);

}

函数分析：

按位与的操作等效于按位取反后按位或在按位与，故直接返回结果即可

1. **int bitNor (int x, int y);**

函数实现：

int bitNor(int x, int y) {

/\* bitNor equals to bitAnd after bitNot \*/

return (~x & ~y);

}

函数分析：

按位异或的操作等效于按位取反后按位与，直接返回结果即可

1. **int copyLSB(int x);**

函数实现：

int copyLSB(int x) {

/\* set the high bit the same as the least bit of x \*/

int tmp = x << 31;

/\*

\* arithmetically right shift adds the sign bit (that is the highest \* bit of tmp) fill all bits of tmp with it's sign bit

\*/

return tmp >> 31;

}

函数分析：

先将输入值左移31位使最高位为输入值的最高位，在算术右移31位，使结果全部补为最高为，得到全部为输入最低位的串

1. **int evenBits(void);**

函数实现：

int evenBits(void) {

/\* init the pattern string with 0x0000 0000;\*/

int pattern = 0;

/\* the following 4 lines set pattern to 0x5555 5555 using bitOr \*and left shift

\*/

pattern |= 0x55 ;

pattern |= 0x55 << 8;

pattern |= 0x55 << 16;

pattern |= 0x55 << 24;

return pattern;

}

函数分析：

0x5等效于0101,即偶数位设置为1（最低位第0位），故左移并按位或即可置位

1. **int logicalShift(int x, int n);**

函数实现：

int logicalShift(int x, int n) {

/\* init the pattern with 0xFFFF FFFF using left shift and right \*shift\*/

int pattern = 0xFF;

pattern = pattern << 24;

pattern = pattern >> 24;

/\* form a pattern string that high n bits set to 0 and low 32-n \*bits set to 1 \*/

pattern = ~(pattern <<(32 + ~n +1));

/\* pattern bitAnd arithmetically shifted x to get the logicalShift \*/

return (x>>n) & pattern;

}

函数分析：

将算术右移转换为逻辑右移只需将高位（右移的位数）置0.故通过左移0xF...F串并取反获得前n位为0后32-n位为1的模式串，按位与即可将x>>n前n位置零。

1. **int bang(int x);**

函数实现：

int bang(int x) {

/\*

\* bitOr to each bit and if result is 1, there must be some bit that is set \* to be 1,

\* just return the reverse result is equal to !x

\*/

int tmp = x;

x = x | (x >>16);

/\* now the lower 2 byte of x is defined \*/

tmp = x;

x = x | (x >>8);

/\* now only lower 1 byte is defined \*/

tmp = x;

x = x | (x >> 4);

/\* 4 bits left \*/

tmp = x;

x = x | (x >> 2);

/\*2bits \*/

tmp = x;

x = x | (x >> 1);

/\* now the last bit is the reverse of result\*/

x += 1;

int pattern = 0x01;

return pattern & x;

}

函数分析：

只要有一位是1,就返回0,否则返回1.故将所有位按位或，若结果为1必有某一或几位非零。通过右移一半长度的位来减少按位或的次数。结果取反并返回即为x的逻辑非。

1. **int leastBitPos(int x)**

函数实现：

int leastBitPos(int x) {

/\*

\* x-1 makes 0 after the least 1 became 1, and the least 1 became 0

\* so ~(x-1) & x only leave the least 1's bit 1.

\*/

int tmp = 0xFF;

tmp = (tmp << 24 ) >> 24;

return ~(x+tmp) & x;

}

函数分析：

当一个串x结尾n位全部为0,n+1位为1时，若令x-1（即x加上-1的补码），则结尾n位的0全部变成1,第n+1位变为0,前32-n-1位全部不变。故先将非（x-1）按位与x，则只有第n+1位为1,其余位全部为0

1. **int tmax(void);**

函数实现：

int tmax(void) {

/\* it should return 0x7FFFFFFF \*/

int result = 0xFF;

result = result << (31);

result = ~result;

return result;

}

函数分析：

用补码表示的最大整型为（b）0111 1111 .... 1111，故直接返回0x7FFF FFFF

1. **int negate(int x);**

函数实现：

int negate(int x) {

/\* in 2's complement substract , minus is the same as add 1 after \* bitNot ,including 0x0000 and 0x8000 \*/

return ~x+1;

}

函数分析：

二进制补码表示情况下取负等效于按位取反后+1,包括0x0000和0x8000,直接返回即可。

1. **int isPositive(int x);**

函数实现：

int isPositive(int x) {

/\*

\* in 2's complement substract, only 0x00 and 0x80 equals to \* itself

\* we need to figure out 0,so we need a 2's complement substract \* of x

\* only x that x has sign bit 0 and -x has sign bit 1 can definitely \* tell that x>0

\*/

/\*

\* (~x+1)>>31 is true only when -x has sign bit 1. but 0 has sign \* bit 0,so there is no 0

\*/

return ((~x+1)>>31 & 0x01) & (~(x>>31)) & 0x01;

}

函数分析：

当x为正数时返回1,即符号位为0且x！=0是返回1.但去除0不方便操作，故取-x，当-x符号位为1时可以判断x>0

1. **int isNonNegative(int x);**

函数实现：

int isNonNegative(int x) {

/\* if x has a sign bit of 0, x is no negative. so just get the minus \* sign bit and return \*/

return ~(x>>31)& 0x01;

}

函数分析：

直接判断x符号位为0即可判断x>=0,故直接返回符号位取反

1. **int sum3(int x, int y, int z);**

函数实现：

int sum3(int x, int y, int z) {

int word1 = 0;

int word2 = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fill in code below that computes values for word1 and word2

without using any '+' operations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

word1 = x^y^z;

word2 = (x & y) | (x & z) | (y & z);

word2 = word2 << 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Don't change anything below here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return sum(word1,word2);

}

函数分析：

Word1 为 不带进位的加法，word2 为进位，进位加上本位加法即为结果

1. **int addOK(int x, int y）;**

函数实现：

int addOK(int x, int y) {

/\* if the sign of x + y is not the same as x and y, it's overflow \*/

int signx = (x>>31) & 0x01;

int nsignx = ~(x>>31) & 0x01;

int signy = (y>>31) & 0x01;

int nsigny = ~(y>>31) & 0x01;

int signr = ((x+y)>>31) & 0x01;

int nsignr = ~((x+y)>>31) & 0x01;

/\* only when (sx ,sy ,sr) = (0 , 0 , 1)or(1 , 1 , 0) , return 0 \*/

return (signx | signy | nsignr) & (nsignx | nsigny | signr);

}

函数分析：

通过取得x，y，result的符号位，若x与y符号位相同，但与result符号位不同，则溢出返回0.结果表达位两个最小项的积。

1. **int logicalAbs(int x);**

函数实现：

int logicalAbs(int x) {

/\* the operation of -x = ~x +1 = x ^ pattern + signbit \*/

int signbit = (x >> 31) & 0x01;

int pattern = x >> 31;

return (x^pattern) + signbit;

}

函数分析：

取绝对值，就是对正数不变，负数取相反数。取负可以看作取反加1,而取反的操作等效于与0xFF异或。同时发现与0x00异或结果不变。这样，可以构建全部为符号位的串，与x异或再加上符号位。这样的话对于正数x不变，对于负数等效于取反加1.

1. **int isNonZero(int x);**

函数实现：

int isNonZero(int x) {

/\* bitor all bits, and the result is 0 only when x =0 \*/

int tmp = x;

x = tmp | (x >> 16);

/\* now last 2 bytes useful\*/

tmp = x;

x = tmp | (x >> 8);

/\* last 1 byte \*/

tmp = x;

x = tmp | (x >> 4);

/\* last 4bits \*/

tmp = x;

x = tmp | (x >> 2);

/\* last 2bits \*/

tmp = x;

x = tmp | (x >> 1);

/\* last bit of x is the result \*/

return x & 0x01;

}

函数分析：

判断x是否等于0只需判断x的所有位是否全0.通过第6题的方法按位与一次减少一半位数，最后所有位按位与结果为0则x等于0.

四、特定题目分析

被选中的题目是：isPositive（）

逐行分析：

int isPositive(int x) {

#if 0

//用二进制补码表示的时候正数和0的符号位为正。

//而题目只在x为正数时返回1,需要剔除0

//这样的剔除操作不便于用位操作实现

//故想到取-x，则若-x的符号位为1,x必为正数或0x8000。

//x按位取反+1等效于-x，将结果右移31位，得到全部为符号位的串

//按位与0x01即可取最后一位的值，再保证x本身非负（按位与x的符号位）。

//即可得出x为正的结论

/\*

\* in 2's complement substract, only 0x00 and 0x80 equals to \* itself

\* we need to figure out 0,so we need a 2's complement substract \* of x

\* only x that x has sign bit 0 and -x has sign bit 1 can definitely \* tell that x>0

\*/

/\*

\* (~x+1)>>31 is true only when -x has sign bit 1. but 0 has sign \* bit 0,so there is no 0

\*/

#endif

return ((~x+1)>>31 & 0x01) & ((~(x>>31))&0x01);

}