天津大学

深入理解计算机系统课程实验

题目：实验一

学生姓名 王雨朦

学生学号 2016229082

学院名称 国际工程师学院

专业 计算机

时间 2017/8/5

目录

一、 实验目的

此实验将通过解决一系列编程“难题”来使实验人更加熟悉整数和浮点数的位级表示。许多难题都是人为设计的，但通过实验思考解决这些问题的方法会引起对位级表示更深入的思考。

二、 实验要求和方法

- 实验要求

➢ 整数编码规则

禁止使用以下任意一条：
1.使用任何控制结构，如 if，do，while，for，switch 等
2.定义或使用任何宏。
3.在此文件中定义任何其他功能。
4.调用任何功能。
5.使用任何其他操作，如&&，||， - 或？：等
6.使用任何形式的铸造。
7.使用除 int 之外的任何数据类型。 不能使用数组或结构体。

➢ 浮点数编码规则

对于需要您隐含浮点运算的问题，编码规则不太严格。
1．允许使用循环和条件控制。
2.允许同时使用 int 和 unsign。
3.可以使用任意整数和无符号常量。
您被明确禁止：
1.定义或使用任何宏。
2.在此文件中定义任何其他功能。
3.调用任何功能。
4.使用任何形式的铸造。
5.使用除 int 或 unsigned 之外的任何数据类型。不能使用数组或结构体。
6.使用任何浮点数据类型，操作或常量。

- 实验方法

目标是修改 bits.c 的副本，以便它通过所有在 btest 中的测试，不违反任何编码准则。检验是否通过测试的方法如下：

1.使用 dlc 编译器检查您的解决方案是否符合要求编码规则。
2.使用 BDD 检查器正式验证您的解决方案是否产生正确的答案。

三、 实验环境及设备

- 实验环境

VMware 虚拟机 Ubuntu16.04 (X86)

- 语言

C 语言

四、 实验步骤及函数思路

- 实验步骤

1、 每编码完成 bits.c 文件中一个函数
2、 在控制台进入该文件所在位置
3、 编译指令：./dlc bits.c
4、 测试指令：make btest，再用指令./btest 测试所有功能的正确性并打印出错误消息；
也可以用./btest -f foo 测试函数 foo 的正确性。
注意：每次重新写完函数,需要重新测试 btest:用 make clean; make btest;

- 函数思路

A. 位操作

Name	Description	Rating	Max Ops
`bitAnd(x,y)`	`x & y` using only `|` and `~`	1	8
`getByte(x,n)`	Get byte `n` from `x`.	2	6
`logicalShift(x,n)`	Shift right logical.	3	20
`bitCount(x)`	Count the number of 1's in `x`.	4	40
`bang(x)`	Compute `!n` without using `!` operator.	4	12

Table 1: Bit-Level Manipulation Functions.

1）BitAnd

函数意义	即按位与操作。
思路	运用摩根定律 $\overline{A}\overline{B} = \overline{A} \mid \overline{B}$，非、或操作完成。

2）getByte(int x, int n)

函数意义	从 x 中得到第 n 个字节，注意每个比特是八位。
思路	先把 x 右移(n*8)（即(n<<3)）位数，再取出低八位，即和 0xff 进行&操作即可。

3）logicalShift(int x, int n)

函数意义	将 x 逻辑右移 n 位，高位补 0。
思路	要处理的有负数和非负数两种情况，即对其算术右移后的符号位进行处理，将所有补上的符号位变为补 0。 先用(1<<31)&x，提取其符号位；再用~((t>>n)<<1)，将符号位右移（n-1）位之后按位取反，最后与算术右移之后的结果进行&操作即抵消了符号位。 注意：由于 n=0 时，不可以直接右移 n-1 位，故用(t>>n)<<1。

4）bitCount(int x)

函数意义	计算 32 位二进制 x 中 1 的个数。
思路	如果依次检测 x 的每一位，ops 会超出限制的最大操作符数。故利用 val 累加分别计算 x 四个字节上 1 的个数，val 的每个字节的值为对应 x 每个字节上 1 的个数。具体是用 tmp=0x1111 检测每个字节的 0, 8, 16, 24 位的 1 的个数，即 val=temp&x。然后 x>>i 将 x 每次右移一位（1<=i<=7），再和 tmp 做&操作，记录 0+i, 8+i, 16+i, 24+i 位的 1 的个数，累计入 val。最后将 val 中的每个字节叠加，即最终得到所有位上 1 的个数累计。

5) bang(int x)

函数意义	x 的真假。x=0 时，!x=1；x!=0 时，!x=1。
思路	考虑到 0 的特殊性，设 t=~(x+1)，只有当 x=0 时，t 和 x 的符号位都为 0；而 x!=0 时，x 和 t 的符号位总有一者为 1。 因此对 x 和 t 作|操作并保留至最低位符号位，再取反，即得到!x。 最后&0x01 取出最低位即可。

B. 二进制补码运算

Name	Description	Rating	Max Ops
`tmin()`	Most negative two's complement integer	1	4
`fitsBits(x,n)`	Does `x` fit in `n` bits?	2	15
`divpwr2(x,n)`	Compute $x/2^n$	2	15
`negate(x)`	`-x` without negation	2	5
`isPositive(x)`	`x > 0`?	3	8
`isLessOrEqual(x,y)`	`x <= y`?	3	24
`ilog2(x)`	Compute $\lfloor \log_2(x) \rfloor$	4	90

Table 2: Arithmetic Functions

6) Tmin()

函数意义	返回二进制补码最小整数。
思路	由补码的表示方式：$B2Tw(\vec{x}) = -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$ 。 可知，$x_{w-1}=0$时，即 0x80000000 为最小的 32 位整数。

7) fitsBits(int x, int n)

函数意义	判断 x 是否可以用 n 位补码来表示。
思路	分 x 为负数和非负数两种情况。 当 x 为负数，最高位符号位，即第（n-1）位为 1，x>>n 结果 0； 当 x 为非负数，最高位为 0，x>>n 结果为 0。 取 x 的符号位 s，当 x=0 时，x>>n-1 的结果为 0；反之(~x)>>n-1 的结果为 0。

8）divpwr2(int x, int n)

函数意义	求 x/(2^n)的值。
思路	分 x 为负数和非负数两种情况。 当 x 为非负数，直接将 x 右移 n 位即可； 当 x 为负数，如果 x 移出的位不是全部为 0，则将其右移 n 位之后再加一。

9）negate(int x)

函数意义	计算负 x。
思路	当 x 为非负数，返回~x+1 即为-x; 当 x 为负数，同样即可。

10)isPositive(int x)

函数意义	判断 x 是否为正数。
思路	符号位为 0 且值不为 0，即为整数。 符号位和数值判断的时候，取对应的 bool 值即可。

11)isLessOrEqual(int x, int y)

函数意义	判断 x<=y 是否成立。
思路	将 x 移到等号右边，即判断 y-x>=0 是否成立，转化成判断 y-x 的正负。 当 x 和 y 同号，y-x 不会溢出，判断 y-x 是否为非负数即可； 当 x 和 y 异号，y-x 可能溢出，结果同 y 的符号一致。

12)ilog2(int x)

函数意义	求 x 的最高位 1 对应的 log base 2。
思路	由 x>0,ilog 的结果不会超过 31，可以用 5 位二进制来表示，依次折半累加得出最高位的权即可。 设结果 ret=ijklm(2),首先判断 x 的高 16 位是否全为 0，ret+=2^4*i，x 相应右移 16 位；再判断右移后 x 的高 8 位是否全为 0，ret+=2^3*j，x 相应右移 8 位；依次累加得出结果 ret。

C. 浮点数操作

Name	Description	Rating	Max Ops
float_neg(uf)	Compute -f	2	10
float_i2f(x)	Compute (float) x	4	30
float_twice(uf)	Computer 2*f	4	30

13)Float_neg(unsigned uf)

函数意义	计算负 uf。
思路	符号位取反可以用抑或。 当 x=NaN 时，阶码全为 1，小数位不为 0，返回 NaN; 当 x=! NaN 时，直接将符号位取反，返回-x 即可。 由 32 位浮点数的符号位 s 占 1 位，阶码占 8 位，尾数 23 位。 判断阶码是否不全为 1 可利用：(uf>>32)^0xff; 判断尾数不为 0 可利用：uf&((1<<23) - 1); 如果满足以上条件任意一个，说明该数不是 NaN; 否则返回 NaN.

14)Float_i2f(int x)

函数意义	将 int 型 x 强制转换成 float。
思路	浮点数操作考虑符号，阶码和尾数三部分。 取符号位：s= x& (1<<31)。 初始化阶码：exp= (x>>31)?158:0。当 x=0,初始化 exp=0,返回 0; 当 x=1,初始化 exp=127+31(e+Bias)，最终返回-1。 尾数部分：可以先去除前缀 0，然后取高 23 位和剩余位数四舍五入的精度位：用(1<<23)-1 和低 23 位进行&运算，取 23 位尾数; 对于精度部分，判断 x& 0xff，遵循四舍五入原则： 由四舍五入，即判断的部分不小于 2^7 即可产生进位; 当剩余精度部分超过 2^7,直接进位，frac++; 当剩余精度部分等于 2^7，判断 frac 的最后一位为 1 或 0： 　frac 最后一位为 0，不进位，直接截断; 　frac 最后一位为 1，产生进位，frac++。

15)float_twice(unsigned uf)

函数意义	计算 uf*2 的值。
思路	浮点数操作考虑符号，阶码和尾数三部分。 计算 uf*2 有两种情况： 当阶码部分为 0，只需对尾数部分左移一位； 当阶码部分为 1，则将阶码加一。 取符号位：s= uf& (1<<31)。 取 8 位阶码：exp=(uf>>23)& 0xff。 取尾数: frac= uf& ((1<<23)- 1)。 判断 uf 是否为 NaN,如果是，不处理，最后返回 NaN。 最后将符号，阶码和尾数三部分整合到返回值，即： ret= s|(exp<<23)|frac。

五、 实验结果截图