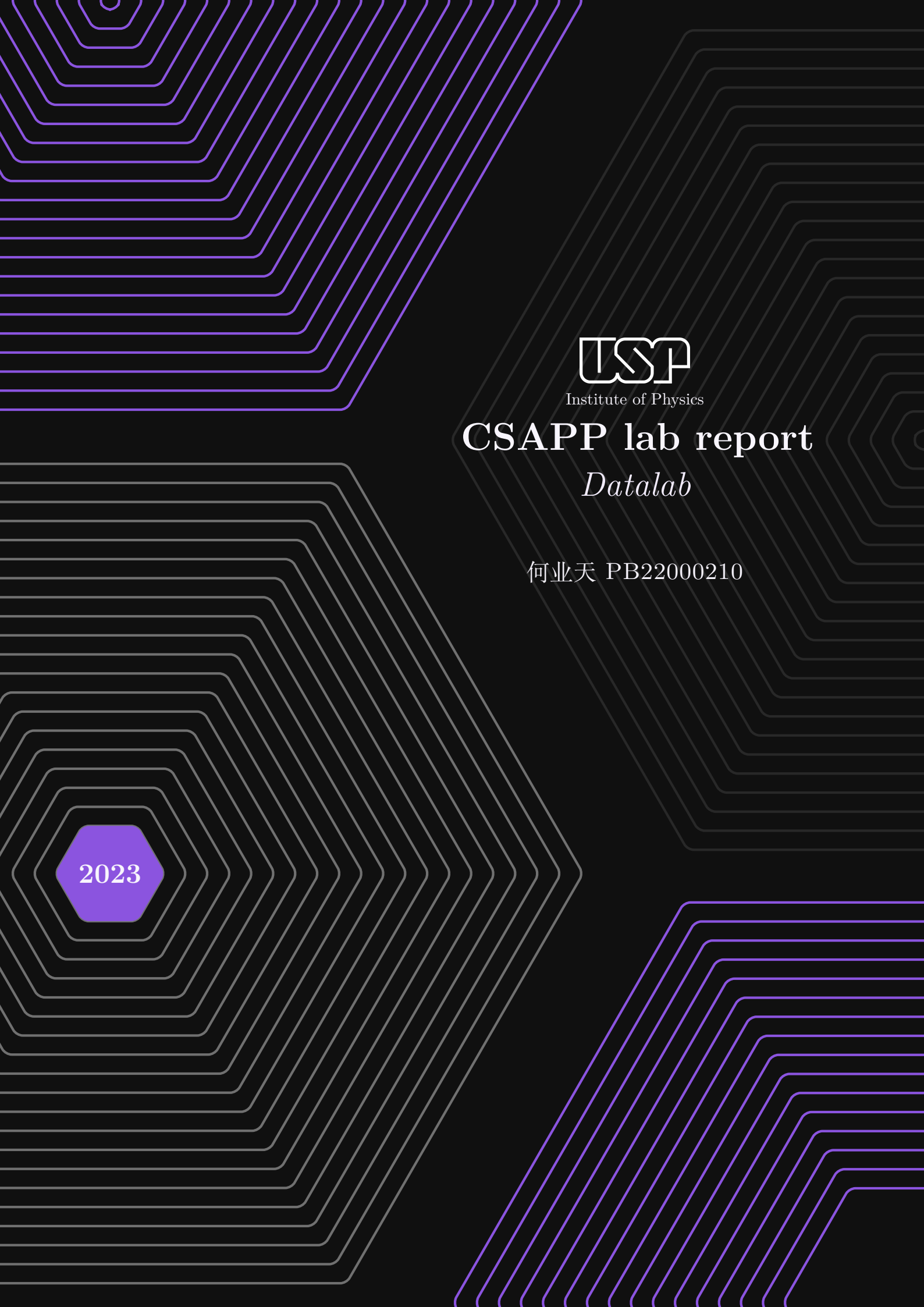


Institute of Physics

CSAPP lab report

Datalab

何业天 PB22000210

2023

HOMEWORK N° 1

实验前的准备

搭建虚拟环境

I. *系统*：在该系列实验中我采用的是 Linux 系统进行操作。使用的 Linux 版本为 Centos 7 64 位。

II. *虚拟机*：我使用的电脑原系统为 Windows，故使用了虚拟机软件 VMware Workstation。为更方便地使用，安装了 xshell 和 xtfp 来与主机进行连接，后续的操作都主要在 xshell 上进行。

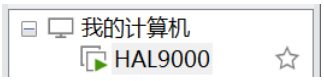


图 1. 本人虚拟机名称；

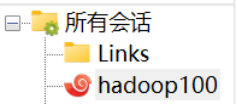


图 2.xshell 会话名称。

III. *获取实验文件:* 进入 CSAPP 官网，获取 Datalab 的 Self-Study Handout, 解压缩之后上传至 github 的 Repositories, 通过 git 中的 gitclone 传输至虚拟机文件夹。

图 3.datalab 文件夹一览

IV. *脚本设计*：为方便于多次测试，编写了脚本 runtest.sh，每次执行脚本便进行 test 即可。现准备工作完毕，下面便开始在 bits.c 中进行

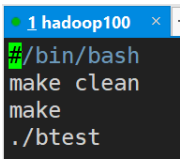


图 4.runtest.sh

bitXor

Puzzle 1:

/*

* bitXor $-x\hat{}y$ using only $\sim$ and &

* Example: bitXor $(4,5) = 1$

* Legal ops: $\sim$ &

* Max ops: 14

* Rating: 1

*/

Answer:

```
int bitXor ( int x, int y){
    return ~((~x)&(~y))&(~(x&y));
}
```
(1.1)

Solution:

本题是要用“与”和“非”来实现异或的功能。不妨对于 x=1,y=0 来分析，后续只需拓展位数即可，易得 $x\hat{}y = 1$。其实也就是判断 x,y 的值是否相等：相等取 0；不相等取 1。如何判断相等呢，首先对于（1，1）与（1，0）的情况，直接用 & 即可判断，但（0，0）却出现了问题。于是我们想到用 $\sim x\& \sim y$ 来处理。现在问题明了了：满足 $x\&y == 1$ 或 $\sim x\& \sim y == 1$ 的即为相等。经过一些条件判断的修正即可得到答案。

tmin

Puzzle 2:

/*

* tmin - return minimum two's complement integer

* Legal ops: ! $\sim$ &$\wedge$ | + $\ll\gg$

* Max ops: 4

* Rating: 1

*/

Answer:

```
int tmin(void){
return 1 ≪ 31;
}
```
(1.2)

Solution:

本题要求的是求最小补码整数，即首位为 1 余位为 0 的 32 位二进制数。相较于直接用 16 进制打出了该数，有更好的方式即利用左移运算符即可。

isTmax

Puzzle 3:

/*

* isTmax - returns 1 if x is the maximum, two's complement number,

* and 0 otherwise

* Legal ops: $1 \sim \&^* \mid +$

* Max ops: 10

* Rating: 1

*/

Answer:

```
int isTmax ( int x){
return !∼(x + (x + 1))&!!(∼ x);
}
```
(1.3)

Solution: 第一次作答时未考虑 x 为-1 的特殊情况。

本题是要判断 x 是否是最大的补码整数即首位为 0 余位为 1 的 32 位二进制数。我们可以发现该数的一个性质是 $Tmax+1$ 会发生溢出，生成 $Tmin$ 即 $\sim Tmax$，本题便成为判断 $Tmax+1 == \sim Tmax$ 是否成立。在这题中，我利用 $x+ \sim x+1=0$ 这个等量关系来判断上述条件。同时发现，还有一个数满足上述条件，其便是-1，于是通过$(\sim x)$ 来排除该种情况。之后进行条件修正便得答案。

allOddBits

Puzzle 4:

/*

* alloddBits - return 1 if all odd-numbered bits in word set to 1

* where bits are numbered from 0 (least significant) to 31 (most significant)

* Examples alloddBits (0xFFFFFFFD) = 0, alloddBits(0xAAAAAAAA) = 1

* Legal ops: :∼ &∧ | + ≪≫

* Max ops: 12

* Rating: 2

*/

Answer:

```
int alloddBits ( int x)  {
   int t = 0xAA;
  t = t + (t << 8) + (t << 16) + (t << 24)
   return !((x&t)^t)
}
```
(1.4)

Solution:

本题是要对 2 进制的各个奇数数位进行判断，均为 1 则返回 1。我们不妨来先构造一个各奇数位为 1 偶数位为 0 的标准数。构造方式是代码前两行内容，主要是通过左移来实现的。接着将 x 与标准数比较，我们想要的是判断奇数位是否均相等，偶数位则可全置为 0，于是先采用 & 操作，奇数位若为 1 则置为 1（0 则为 0），偶数位置为 0，再用 ^ 则奇数位 1 变为 0 而 0 变为 1，偶数位仍为 0，通过判断结果是否为 0 即可。

negate

Puzzle 5:

/*

* negate - return $-x$

* Example: negate $(1) = -1$.

* Legal ops: ! $\sim \wedge \mid + \ll \gg$

* Max ops: 5

* Rating: 2

*/

Answer:

```
int negate ( int x){
return ~x + 1;
}
```
(1.5)

Solution:

本题即求负数，较易便不再赘述。

isAsciiDigit

Puzzle 6:

/*

* isAsciiDigit - return 1 if $0x30 <= x <= 0x39$ (ASCII codes for characters '0' to '9')

* Example: isAsciiDigit $(0x35) = 1$.

* isAsciiDigit $(0 \times 3a) = 0$

* isAsciiDigit $(0 \times 05) = 0$

* Legal ops: ! $\sim$ &$\wedge \mid + \ll \gg$

* Max ops: 15

* Rating: 3

*/

Answer:

```
int isAsciiDigit ( int x){
    return (!((x + (~ 0x30 + 1)) >> 31))&((x + (~ 0x3A + 1)) >> 31);
}
```
(1.6)

Solution:

本题是要判断数是否在范围内。思路就是分别与上界、下界作差，通过正负分析来判断。我们知道数的首位是 1 或 0 即可判断数的正负，故采用算术右移 31 位来判断首位数字，最后结合一下两次差值比较即可。需要注意的是考虑到 $0x39$ 作差均大于等于 0 的特殊情况，采用 $0x3A$ 为上界以规避。

conditional

Puzzle 7:

/* * conditional - same as x ? y: z

* Example: conditional $(2, 4, 5) = 4$

* Legal ops: $! \sim \& \wedge | + \ll \gg$

* Max ops: 16

* Rating: 3

* /

Answer:

```
int conditional(int x, int y, int z){
return ((!x + (~ 1 + 1))&y) | (~ (!x + (~ 1 + 1))&z);
}
```
(1.7)

Solution:

此题是要模拟编程语言中的三元运算符。我们无法使用判断语句，于是采用的思路则是 $y \mid z$, 例如当 $x == 0$ 时 $y = 0$,$z = z$ 即可。于是我们想到的是用 $0xFFFFFFFF$ 与 0 来和 y、z 进行 & 操作，如何达成呢，我们发现 $0xFFFFFFFF + 1 = 0$，于是加上一项 $!x$ 即可。

isLessOrEqual

Puzzle 8:

```
/*
 * isLessOrEqual - if x <= y then return 1, else return 0
 *   Example: isLessOrEqual (4,5) = 1.
 *   Legal ops: ! &^ | + << >>
 *   Max ops: 24
 *   Rating: 3
 */
```

Answer:

```
int isLess0rEqual (int x, int y){
int a = ((x^y) >> 31) &0x1;
int b = ((y + (~ x + 1)) >> 31)&0x1;
return (!a&&!b) | (a&&((x >> 31)&0x1));
}
```

(1.8)

Solution:

本题是要比较两个数的大小。首先可以通过符号位进行初步判断，若 x 与 y 不同号即可直接得到答案，通过 a 来储存符号信息。其次若同号，我们可以通过 x 与 y 作差的符号位来判断，用 b 来储存大小信息。接着情况讨论，不妨来先讨论 $x \leq y$ 的几种情况：$1.x < 0, y \geq 0$，则 $a = 1$ 且 $b = 0$;$2.x$ 与 y 同号，$a = 0$ 且 $b = 0$. 针对这两种情况，故采用 | 来合并。

Logicalneg

Puzzle 9:

/*

* logicalNeg - implement the ! operator, using all of

* * the legal operators except !

* * Examples: logicalNeg (3) = 0, logicalNeg (0) = 1

* Legal ops: $\sim \wedge \mid + \ll \gg$

* Max ops: 12

* Rating: 4

*/

Answer:

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

(1.9)

Solution:

本题是要实现！的功能。很显然，我们判断的重点便是 0 这个数，那么如何判断 0 呢？其实 0 具有的一条重要性质就是它等于自己的相反数。由此，我们利用 x 相反数的符号位进行判断即可。最终可巧妙利用 +1 来使不成立情况返回值置为 1。

howManyBits

Puzzle 10:

/*

howManyBits - return the minimum number of bits required to represent x in two's complement

* Examples:
* howManyBits (12) = 5
* howManyBits (298) = 10
* howManyBits (−5) = 4
* howManyBits (0) = 1
* howManyBits (−1) = 1
* howManyBits (0 × 80000000) = 32
* Legal ops: ! ∼ &∧ | + ≪≫
* Max ops: 90
* Rating: 4

Answer:

```
int howManyBits ( int x){
int sign  = x >> 31;
x = (~ sign &x) | ( sign & ~ x);
int b16=!! (x >> 16) << 4;
x = x >> b16;
int b8=!! (x >> 8) << 3;
x = x >> b;
int b4=!! (x >> 4) << 2;
x = x >> 4;
int b2=!! (x >> 2) << 1;
x = x >> b;
int b1=!! (x >> 1);
x = x >> b1;
int b0=x;
return b16+b8+b4+b2+b1+b0+1;
}
```

(1.10)

Solution: 本题参考过网上资料进行改良

本题是要求出储存该数的最小二进制数位。本题的查找方式为二分查找法。首先我们要解决的问题是负数，负数的首位为 1，对我们的判断有干扰，故代码的前两行为取 x 的绝对值。接着我们进行二分查找。首先分半对前 16 位判断，有数则将 $b16$ 赋值为 16，若 $b16! = 0$ 再来分析 x 的前 8 位，反之分析后 8 位；如此循坏直至 x 剩一位，最后将各个 b 相加即可。

floatScale2

Puzzle 11:

/*

* floatScale2−Return bit − level equivalent of expression $2 * f$ for

* floating point argument f.

* Both the argument and result are passed as unsigned int's, but

* they are to be interpreted as the bit-level representation of

* single-precision floating point values.

* When argument is NaN, return argument

* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

* Max ops: 30

* Rating: 4

* /

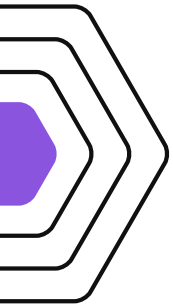
Answer:

```
unsigned floatScale2(unsigned uf) {
int sign=uf&0x80000000;
int frac=uf&0x007fffff;
int exp=uf&0x7f800000;
if (exp == 0x7f800000) return uf;
if(exp==0) return (uf«1)|sign;
exp+=0x00800000;
if(exp==0x7f80000) return uf|sign;
return sign | exp |frac;
}
```

(1.11)

Solution:

本题是要求对浮点数进行乘法运算。首先根据我们对浮点数的了解，我们可以将其拆解为三个部分：sign,frac,exp（用 & 实现）。接着我们要来讨论一下浮点数的特殊情况。首先是 $exp == 255$, 为无穷大，则直接返回原数即可。第二是 $exp == 0$ 的情况，此时为非规格化的值，通过左移一位实现翻倍，要注意补上符号位。对于规格化的值，我们只需将 exp 的末位加一即可实现翻倍。最后一定要小心，对于符号位受到影响的情况，我们补上符号位即可。



floatFloat2Int

Puzzle 12:

```
/*
* floatFloat2Int - Return bit-level equivalent of expression (int) f
*     for floating point argument f.
*     Argument is passed as unsigned int, but
*     it is to be interpreted as the bit-level representation of a
*     single-precision floating point value.
*     Anything out of range (including NaN and infinity) should return
*     0x80000000u.
*     Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
*     Max ops: 30
*     Rating: 4
*/
```

Answer:

```
int floatFloat2Int(unsigned uf) {
int sign=uf&0x80000000;
int frac  = ( uf &0x007fffff) | 0x00800000;
int exp=uf&0x7f800000;
if (exp==0) return 0;
if (exp==0x7f800000) return 0x80000000u;
int E = (exp >> 23) − 127;
if (E < 0) return 0;
if (E > 31) return 0x 80000000;
if (E < 23) frac=frac»(23-E);
else frac=frac «(E-23);
return sign? (∼  frac+1): frac;                    }
```

(1.12)

Solution: 左移过大的情况一开始做时未考虑而报错

本题是要将浮点数转化为相应的整数。首先同上题一样，分解浮点数，分析 exp 非规格化的值 0 与 255，按题目要求输出即可。再来讨论普通情况，取出真实指数 E，对 E 进行分类讨论:$E < 0$ 时无整数部分，直接返回 0；$E > 31$ 时左移超过了最大位数，编译器会取余来得到新的左移数；最后分析 E 和 23 的大小关系，即取差值绝对值进行左移。最终注意一下正负即可。

floatPower2

```
/*
 * floatPower 2 - Return bit-level equivalent of the expression 2.0^x
 *   (2.0 raised to the power x) for any 32-bit integer x.
 *
 *    The unsigned value that is returned should have the identical bit
 *    representation as the single-precision floating-point number 2.0^x.
 *    If the result is too small to be represented as a denorm, return
 *    0. If too large, return + INF.
 *
 *    Legal ops: Any integer/unsigned operations incl. ||, &&. Also if, while
 *    Max ops: 30
 *    Rating: 4
 */
```

Answer:

```
unsigned floatPower2(int x){
if (x < -149)
return 0;
if (x > 127)
return 0x7f800000;
unsigned exp = x + 127;
unsigned pattern = exp << 23;
return pattern;
```

(1.13)

Solution:

本题是要求 2 的指数函数的浮点数表达。继续我们的分类讨论：首先是考虑浮点数的最小值，此时即为 $x < -149$ 的情况，返回 0 即可。其次是浮点数的最大值，即 $x > 127$，返回 NAN。最后讨论正常情况，我们将 $x + 127$ 置为指数阶码，然后左移 exp 即可得到答案。

实验测试与结果

```
[root@hadoop100 datalab-handout]# ./runtest.sh
rm -f *.o btest fshow ishow *~
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
gcc -O -Wall -m32 -o fshow fshow.c
gcc -O -Wall -m32 -o ishow ishow.c
Score   Rating  Errors  Function
 1      1       0       bitXor
 1      1       0       tmin
 1      1       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       floatScale2
 4      4       0       floatFloat2Int
 4      4       0       floatPower2
Total points: 36/36
```

实验总结

在本实验中主要在以下几个方面的知识掌握中获得了提升：

1. 各项数字间的操作，比如三种移位，比如与或，比如取反。datalab 其中不少题都是巧用这些看似基本的计算方法来实现现实的计算问题的，举个例子：利用取反加 1 来得到负数之类。

2. 考虑一些极端情况。做 datalab 不少题的时候都会遇到这样的问题：设计好的运行思路看似合理却得不了分。这是因为 test 时的极端情况设计时没有考虑到，在改正之后也对这些错误有了更深刻的印象。

3. 浮点数的结构与运算思路。最后三道浮点数的题是较难的，但在了解清楚浮点数的三块区域各自的作用后，分开进行处理，分情况讨论的思路还是很清晰的。