datalab

曹景琦 2022010796

题目思路

bitxor

ops = 8

刚开始直接使用了(x & ~y) | (~x & y)来表示，测试之后注意到，题目要求只能使用~，&，所以将|直接使用~和&表示即可（DoMorgen）。

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

image.png

tmin

ops = 1

因为常数最大8位，所以一定是使用一个较小的常数，进行操作得到tmin。即将1左移31位就得到了位模式0x80000000，作为补码就表示tmin。

```
int tmin(void) {
  return 0x1 << 31; // 常整数存储为32位，即int大小
}
```

image-1.png

isTmax

ops = 8

如果是Tmax，那么x+1就是Tmin，x+1就是-Tmax，所以2x+2=0.但是如果x=-1也成立，所以需要把x=-1排除。

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

image-2.png

allOBits

ops = 9

使用移位操作构造一个偶数位全是1的int，与x位或，这样x的偶数位无论1还是0结果都是1，则只需要关注x的奇数位即可，如果x的奇数位全是1，那么结果就是全1的int，这样取反就是0；反之，如果x奇数位不全是1，结果并不是全1的int，那么取反之后就不是0。最后再使用！输出结果即可。

```
int allOddBits(int x) {
  int y = (0x55 << 24) + (0x55 << 16) + (0x55 << 8) + 0x55;
  return !(~(x | y));
}
```

image-3.png

negate

ops = 2

x+(~x)为全1的序列0xFFFFFFFF，再+1就是全0的序列0x00000000，那么x的相反数就是~x+1。

```
int negate(int x) {   // 取反加1即可
  return ~x + 1;
}
```

image-4.png

isAsciiDigit

ops = 12

将10种情况简化为两种。

将x右移四位，看是否是3，且提取第3位，第0-2位，如果第三位为0，那一定落在0-7之间，否则，看低三位是否为1或者0，如果是，就是8和9。

```
int isAsciiDigit(int x) {
  int high_four = x >> 4;
  int middle_one = x & 0x08;
  int low_three = x & 0x07;
  return !(high_four ^ 0x03) & (!(middle_one) | !(low_three ^ 0x01) | !low_three);
}
```

image-12.png

conditional

ops = 12

使用0xFFFFFFFF为掩码来进行y和z的选择。如果x为真，那么就构造0xFFFFFFFF与y位与，构造0x00000000与z位与，再将二者的结果或起来，就是最终答案。x为假同理。

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

image-5.png

isLessOrEqual

ops = 18

分两种情况讨论：

- x与y异号，直接判断符号位即可。
- x与y同号，此时判断x-y的符号位即可。如果x < y，那么差的符号位为1；如果x > y，那么差的符号位为0，符合要求。但是当x = y的时候，差的符号位为0，与题目要求相反，于是，我将x = y这一种情况单独讨论，如果x = y就得1；

```
int isLessOrEqual(int x, int y) {
  int x_sign = (x >> 31) & 0x01;
  int y_sign = (y >> 31) & 0x01;
  int not_same_sign = x_sign & !y_sign;   // x和y不同号的情况
  int e = x_sign ^ y_sign;    // 用来检验x和y是否同号
  int same_sign = ((x + (~y + 1)) >> 31) & 0x01;    // x和y同号的情况
  int same = !(x ^ y);    // x和y相同的情况
  return not_same_sign | (!e & same_sign) | same;
}
```

image-6.png

logicalNeg

ops = 5

如果x非0，那么x与-x的符号位相反，考虑x ^ (~x+1)的符号位，为1；如果x=0，那么x与~x+1的符号位都为0，那么x ^ (~x+1)的符号位为0。这样刚好和要求的结果相反，我们需要实现0和1的转换。将结果右移31位，原来的符号位就在最低位，如果是前者，右移之后为0xFFFFFFFF，加1得0；如果是后者，右移之后是0x00000000，加1得1。

但是这里有个坑：不能使用异或^，因为如果x为Tmin，位模式为0x8000000，~x是0x7FFFFFF，~x+1还是Tmin，这样二者符号位都是1，异或之后为0，不符合上面的情况，所以，需要将 ^ 改为 |，这样就全部满足了。

```c
int logicalNeg(int x) {
  return (((~x + 1) | x) >> 31) + 1;    // 这里必须使用 | 或
}
```

image-7.png

howManyBits

ops = 37

- x 为正数，以八位为例：0011 1010，需找到最高位 1，除此以外，还需一位 0 作为符号位；
- x 为负数，以八位为例：1100 1001，需找到最高位 0，除此以外，还需更高一位 1 作为符号位
  为了方便，我们将负数取反，这样就可以统一处理。
  我们使用二分的思想：
  0010 1100 1010 1111 | 1000 0000 0101 0000
  考虑左边16位，如果有1，那么x的位数至少需要16位，并将x右移16位，作为新的x。
  0000 0000 0000 0000| 0010 1100 1010 1111
  接下来考虑右边16位的前8位，同上。
  最后一直到1位。
  再加上剩下的位也就是bit_0，和一个符号位。
  有一个需要注意的就是声明必须要在任何一条非声明的语句之前

```c
int howManyBits(int x) {
  int sign = x >> 31;   // 如果符号位是1，就是全1
  int bit_16, bit_8, bit_4, bit_2, bit_1, bit_0;
  x = (~sign & x) | (sign & ~x);
  bit_16 = !!(x >> 16) << 4;
  x = x >> bit_16;
  bit_8 = !!(x >> 8) << 3;
  x = x >> bit_8;
  bit_4 = !!(x >> 4) << 2;
  x = x >> bit_4;
  bit_2 = !!(x >> 2) << 1;
  x = x >> bit_2;
  bit_1 = !!(x >> 1) << 0;
  x = x >> bit_1;
  bit_0 = x;
  return bit_16 + bit_8 + bit_4 + bit_2 + bit_1 + bit_0 + 1;
}
```

image-8.png

floatScale2

ops = 17
先提取符号位，阶码和尾码。然后分三种情况讨论：

- uf为非规格化数，exp = 0，这个时候2 *uf相当于给尾数左移1位，此时如果移位之后超出了非规格化数的表示范围，阶数变为1，由于这种精妙的浮点数表示方法，此时实际指数仍旧不变，尾数加了1，而除了1/2，剩下的权重全都变为2倍，仍旧是2* uf。
- uf为NaN，此时exp全1，frac非0，但是这样处理代码不能通过。需要将frac非0的条件去掉（?
- uf为规格化数，将阶数加1即可。
  最后将每个位置的数 | 就合并在了一起。

```c
unsigned floatScale2(unsigned uf) {
  unsigned s = (uf >> 31) & 0x1;
  unsigned exp = (uf >> 23) & 0xFF;
  unsigned frac = uf & 0x7FFFFF;
  if(exp == 0){
    return (s << 31) | (exp << 23) | (frac << 1);
  }
  else if((exp == 0xFF)){   // 这里有一个问题，NaN是exp全1，并且frac非0，为什么在这里不考虑frac。
    return uf;
  }
  return (s << 31) | ((exp + 1) << 23) | frac;
}
```

image-9.png

floatFloat2Int

ops = 17

(int)f就是将float f转换成int，转换的时候，将小数部分直接舍去。

先提取符号位，阶数，尾数。

E = exp - 127表示指数。主要考虑frac，给frac最前面加个1，就是非规格化数的尾数部分。

先考虑正数：

- 如果E < 0，那么结果 < 1，此时返回0；
- 如果E >= 0且E < 23，此时不能将frac部分的分数权重全部变成整数。（或者小数点无法向右移动到数字结尾）。
- 如果E >= 23且E < 31，就可以将分数权重全变成整数，同时需要满足整数的范围。由于int的最高位是符号位，那么就只有31位，由于frac第一位是1，那么就只能左移30位。
- 如果E >=31，那么就返回0x80000000u

  再考虑负数：

  与正数对称，则取求得的正数的相反数就可以了。

```c
int floatFloat2Int(unsigned uf) {
  unsigned s = (uf >> 31) & 0x1;
  unsigned exp = (uf >> 23) & 0xFF;
  unsigned frac = uf & 0x7FFFFF;
  int E = exp - 127;
  frac = frac | (1 << 23);
  if(E < 0){
    return 0;
  }
  else if(E < 23){
    frac = frac >> (23 - E);
  }
  else if(E < 31){
    frac = frac << (E - 23);
  }
  else{
    return 0x80000000u;
  }
  if(s){
    frac = ~frac + 1;
  }
  return frac;
}
```

image-10.png

floatPower2

ops = 11

首先我们要计算单精度浮点数的范围：

格式	min	max
非规格化数	2^{-149}	$2^{-126} \times (1 - 2^{-23})$
规格化数	2^{-126}	$2^{127} \times (2 - 2^{-23})$

- 如果x < -149，那么就无法使用单精度浮点数表示，置为0。
- 如果x >= -149 且x < -126，那么就是非规格化数的范围，此时，指数已经是-126，那么还剩下x+126的指数需要用尾数表示，且尾数应该只有一位是1，则将1左移23+(x+126)位即可。
- 如果x >= -126且x <= 127，那么就是规格化数的范围，只用指数表示即可，尾数全置0。
- 如果x > 127，那么就不是规格化数，返回+inf即可。

```
unsigned floatPower2(int x) {
  if (x < -149)
  {
    return 0;
  }
  else if (x < -126)
  {
    return 1 << (23 + (x + 126));
  }
  else if (x <= 127)
  {
    return (x + 127) << 23;
  }
  else
  {
    return 0xFF << 23;
  }
}
```

image-11.png

结果

```
crazy@DESKTOP-U8TR3MA:~/csapp/datalab/datalab-handout$ ./driver.pl
Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
1       1       0       3       8       bitXor
1       1       0       3       1       tmin
1       1       0       3       8       isTmax
2       2       0       3       9       allOddBits
2       2       0       3       2       negate
3       3       0       3       12      isAsciiDigit
3       3       0       3       12      conditional
3       3       0       3       18      isLessOrEqual
4       4       0       3       5       logicalNeg
4       4       0       3       37      howManyBits
4       4       0       3       17      floatScale2
4       4       0       3       17      floatFloat2Int
4       4       0       3       11      floatPower2
```

image-14.png

感想

自己现在对位级操作才有了比较系统的了解，而且有一些题目有好几种不同的解法，都非常有趣，需要大量的思考。