计算机基础实验_lab1（CSAPP datalab）

原创

steer_z 于 2018-12-22 21:54:53 发布 8968 收藏 85

分类专栏： 计算机系统基础实验 文章标签： 计算机系统基础实验 数据表示 lab1 CSAPP

本文链接：https://blog.csdn.net/qq_40931241/article/details/85218488

NPU_CS_DataLab

计算机系统基础实验_数据表示

1. bitAnd
2. upperBits
3. anyEvenBit
4. leastBitPos
5. byteSwap
6. isNotEqual
7. float_neg
8. implication
9. bitMask
10. conditional
11. isLessOrEqual
12. isPositive
13. satMul3
14. float_half
15. float_i2f
16. howManyBits
17. tc2sm

计算机系统基础实验_数据表示

使用限制的操作符，来完成一些任务，主要考察了移位运算等。如果能自己认真完成，一定会非常有收获，会对计算机底层一些东西实现和编码的概念有着一个感性的认识，尤其是浮点数那三道题，会让你对IEEE754标准定义的浮点数有着极为深刻的认识。
这个实验为一个工程，但我们只需要写bits.c里面的一些函数，整个框架是搭好的。然后整个工程智能化程度挺高的，不满足条件都没法make。
这个工程要在ubuntu32位机器上运行，可以安装虚拟机，也可以在ubuntu64位上安装32位的包。
下面时bits.c里面的一些函数。

1. bitAnd

```
/*
 * bitAnd - x&y using only ~ and |
 *   Example: bitAnd(6, 5) = 4
 *   Legal ops: ~ |
 *   Max ops: 8
 *   Rating: 1
 */
int bitAnd(int x, int y) {
  int ans = ~((~x)|(~y));
  return ans;
}
```

有点离散数学基础知识应该就可以完成。

2. upperBits

```
/*
 * upperBits - pads n upper bits with 1's
 *  You may assume 0 <= n <= 32
 *  Example: upperBits(4) = 0xF0000000
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 10
 *  Rating: 1
 */
int upperBits(int n) {
  int a = 1 << 31;
  int b = n + (~0);
  int k = ((!!n) << 31) >> 31;
  return k & (a >> b);
}
```

重点在于处理n为32的情况

3. anyEvenBit

```
/*
 * anyEvenBit - return 1 if any even-numbered bit in word set to 1
 *   Examples anyEvenBit(0xA) = 0, anyEvenBit(0xE) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 2
 */
int anyEvenBit(int x) {
   return !!((x | (x >> 8) | (x >> 16) | (x >> 24)) & 0x55);
}
```

4. leastBitPos

```
/*
 * leastBitPos - return a mask that marks the position of the
 *               least significant 1 bit. If x == 0, return 0
 *   Example: leastBitPos(96) = 0x20
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 *   Rating: 2
 */
int leastBitPos(int x) {
  return (~x + 1) & x;
}
```

题意为用一个1标记出最低有效位的位置。
如：96：0110 0000（int为32位，现为简写）
则返回：0010 0000
精髓在于取反加一。然后进行简单的掩码操作便可。

5. byteSwap

```
/*
 * byteSwap - swaps the nth byte and the mth byte
 *  Examples: byteSwap(0x12345678, 1, 3) = 0x56341278
 *            byteSwap(0xDEADBEEF, 0, 2) = 0xDEEFBEAD
 *  You may assume that 0 <= n <= 3, 0 <= m <= 3
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 25
 *  Rating: 2
 */
int byteSwap(int x, int n, int m) {
  int xn = n << 3;
 int xm = m << 3;
 int x1 = (x >> xn) << xm;
 int x2 = (x >> xm) << xn;
  int ans = (x & (~(0xff << xn)) & (~(0xff << xm))) | (x1 & (0xff << xm)) | (x2 & (0xff << xn));
 return ans;
}
```

高低字节交换，然后进行掩码。

6. isNotEqual

```
/*
 * isNotEqual - return 0 if x == y, and 1 otherwise
 *   Examples: isNotEqual(5,5) = 0, isNotEqual(4,5) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 *   Rating: 2
 */
int isNotEqual(int x, int y) {
  return !!(x^y);
}
```

利用异或的特性，再用逻辑操作符将结果转化为0或1的逻辑值。

7. float_neg

```c
/*
 * float_neg - Return bit-level equivalent of expression -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 representations of
 *   single-precision floating point values.
 *   When argument is NaN, return argument.
 *   Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
 *   Max ops: 10
 *   Rating: 2
 */
unsigned float_neg(unsigned uf) {
unsigned result= uf ^ 0x80000000;//使用按位异或和掩码对符号位取反其余为不变

unsigned tmp = uf & 0x7fffffff;//为下一步看是不是NaN数准备
  if(tmp > 0x7f800000)//若满足条件则尾数非零，全1阶码
    result = uf;//是NaN数，返回原值，否则返回原数符号位取反对应的数
  return result;
}
```

解释看注释。

8. implication

```c
/*
 * implication - return x -> y in propositional logic - 0 for false, 1
 * for true
 *   Example: implication(1,1) = 1
 *            implication(1,0) = 0
 *   Legal ops: ! ~ ^ |
 *   Max ops: 5
 *   Rating: 2
 */
int implication(int x, int y) {

    return (!x) | (!!y);
}
```

离散数学基础知识。

9. bitMask

```c
/*
 * bitMask - Generate a mask consisting of all 1's
 *   lowbit and highbit
 *   Examples: bitMask(5,3) = 0x38
 *   Assume 0 <= lowbit <= 31, and 0 <= highbit <= 31
 *   If lowbit > highbit, then mask should be all 0's
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int bitMask(int highbit, int lowbit) {
  int i = ~0;
  return (((i << highbit) << 1) ^ (i << lowbit)) & (i << lowbit);
}
```

题意为lowbit道highbit之间全为1。
例bitmask(5,3)应返回第三位到第五位为1其余为为0的一个int型变量。
即0x38 : 0011 1000
没啥说的，自己看代码，巧用移位吧。

10. conditional

```
/*
 * conditional - same as x ? y : z
 *   Example: conditional(2,4,5) = 4
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int conditional(int x, int y, int z) {
  int m = ~!x + 1;

  return (y & ~m) | (z & m);
}
```

类似于数字逻辑里的多路选择器，学了数字逻辑应该就简单了。

11. isLessOrEqual

```
/*
 * isLessOrEqual - if x <= y  then return 1, else return 0
 *   Example: isLessOrEqual(4,5) = 1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 24
 *   Rating: 3
 */
int isLessOrEqual(int x, int y) {
  int signx=x>>31;

  int signy=y>>31;

  int sameSign=(!(signx^signy));
  return (sameSign & ((x+(~y))>>31)) | ((!sameSign) & signx);
}
```

学了数电也应该感觉比较简单。

12. isPositive

```
/*
 * isPositive - return 1 if x > 0, return 0 otherwise
 *   Example: isPositive(-1) = 0.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 8
 *   Rating: 3
 */
int isPositive(int x) {
  return !((x>>31)|(!x));
}
```

比较简单，主要是处理符号位，但对0要特殊处理。

13. satMul3

```c
/*
 * satMul3 - multiplies by 3, saturating to Tmin or Tmax if overflow
 *  Examples: satMul3(0x10000000) = 0x30000000
 *            satMul3(0x30000000) = 0x7FFFFFFF (Saturate to TMax)
 *            satMul3(0x70000000) = 0x7FFFFFFF (Saturate to TMax)
 *            satMul3(0xD0000000) = 0x80000000 (Saturate to TMin)
 *            satMul3(0xA0000000) = 0x80000000 (Saturate to TMin)
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 25
 *  Rating: 3
 */
int satMul3(int x) {
    int two = x << 1;
    int three = two + x;
    int xSign = x & (0x80 << 24);
    int twoSign = two & (0x80 << 24);
    int threeSign = three & (0x80 << 24);

    int mask = ((xSign ^ twoSign) | (xSign ^ threeSign)) >> 31;
    int smask = xSign >> 31;

    return (~mask & three) | (mask &((~smask & ~(0x1 << 31))|(smask & (0x80 << 24))));
}
```

注意溢出的判断与处理，我好像也用了数电里面多路选择器的思想。

14. float_half

```c
/*
 * float_half - Return bit-level equivalent of expression 0.5*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
 */
unsigned float_half(unsigned uf) {
int round, S, E, maskE, maskM, maskS, maskEM, maskSM, tmp;
  round = !((uf & 3) ^ 3);//处理精度问题，进行舍入时需要的参数
  maskS = 0x80000000;
  maskE = 0x7F800000;
  maskM = 0x007FFFFF;
  maskEM= 0x7FFFFFFF;
  maskSM= 0x807FFFFF;
  E = uf & maskE;//用于判断是否为NaN数
  S = uf & maskS;//去符号位

  if (E > 0x7F800000) return uf; //阶码全1，非0尾数，NaN数直接返回

  if (E == 0x00800000) {//阶码只有最后一位为1，除2之后要变为非规格化数，按非//规格化数处理，即处理阶码下溢情况
    return S | (round + ((uf & maskEM)>>1));
  }

  if (E == 0x00000000) { //阶码全0，那要么是0，要么是非规格化数，直接右移一位同时保留符号位
    tmp = (uf & maskM)>>1;
    return S | (tmp + round);
  }
  return (((E>>23)-1)<<23) | (uf & maskSM);//规格化数，正常处理，阶码减一
}
```

这个考察了相当细致了，要对IEEE754标准定义的浮点数有着深刻的理解，对阶码的各种情况对应的不同意义要了解。参考了某位大神的博客才过的这道题。

15. float_i2f

```
/*
 * float_i2f - Return bit-level equivalent of expression (float) x
 *   Result is returned as unsigned int, but
 *   it is to be interpreted as the bit-level representation of a
 *   single-precision floating point values.
 *   Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
 *   Max ops: 30
 *   Rating: 4
 */
unsigned float_i2f(int x) {
  int sign = x>>31 & 1; // 取符号位
   int i;
   int e;  //阶码
   int f = 0; //尾数
   int d;
   int f_mask;

   if(x==0)
       return x; //为0的话，全零阶码，全零尾数，符号位取为0，便可直接返回x。
   else if(x==0x80000000)
       e=158;
   else
   {
       if (sign)
           x = -x;//负数，取其对应正数
       i = 30;
       while ( !(x >> i) )
           i--;      //计算机阶码，看权重最大的不为零的数是第几位

       e = i + 127; //加上偏置常数得到阶码
       x = x << (31 - i);//向左移位至最高
       f_mask = 0x7fffff;
       f = f_mask & (x >> 8);//向右算术移位8位
       x = x & 0xff;//准备判断如何舍入
       d = x > 128 || ((x == 128) && (f & 1));//大于256的一半，或者等于128
//同时差一位就被移出去的那一位为1的话(意为向偶数舍入)，d = 1,
       f += d;  //按IEEE754标准对尾数f进行修正
       if(f >> 23) //如果因为修正（+1）导致f超过23位，应该是将最高的那一位//置0，然后阶码加1
       {
           f &= f_mask;
           e += 1;
       }
   }
   return (sign<<31)|(e<<23)|f; //得到结果
}
```

按照定义做不是非常难，但有一个非常难受的地方，就是尾数只用23位，加上隐藏位最多表示24位有效数字，所以由int型向其转化时就会产生截断误差，就要进行舍入，IEEE754标准规定为小于一半向0舍入，大于一半向1舍入，等于一半像偶数舍入，这一点做好，然后按照定义分别求符号位，阶码，尾数应该就不难了。参考了某位大神的代码才完成此题。

16. howManyBits

```
/* 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(0x80000000) = 32
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 90
 *  Rating: 4
 */
int howManyBits(int x) {
    int t=x^(x>>31);
    int Zero=!t;

    int notZeroMask=(!(!t)<<31)>>31;
    int bit_16,bit_8,bit_4,bit_2,bit_1;
    bit_16=!(!(t>>16))<<4;

    t=t>>bit_16;
    bit_8=!(!(t>>8))<<3;
    t=t>>bit_8;
    bit_4=!(!(t>>4))<<2;
    t=t>>bit_4;
    bit_2=!(!(t>>2))<<1;
    t=t>>bit_2;
    bit_1=!(!(t>>1));
    t=bit_16+bit_8+bit_4+bit_2+bit_1+2;

    return Zero|(t&notZeroMask);

}
```

这个题也挺难的。。。
一开始也没想到用二分法，卡了好久，想到也该就不难了。

17. tc2sm

```
/*
 * tc2sm - Convert from two's complement to sign-magnitude
 *   where the MSB is the sign bit
 *   You can assume that x > TMin
 *   Example: tc2sm(-5) = 0x80000005.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 4
 */
int tc2sm(int x) {
  int sign = x & (0x80 << 24);
  int mask = ~(sign >> 31);


  return sign | ((~mask) & (~x + 1)) | (mask & x);
}
```

编码方式的简单转化，比较简单。

参考文献：https://www.cnblogs.com/tenlee/p/4951639.html