1. **ex 2.10**

|  |  |  |
| --- | --- | --- |
| 步骤 | \*x | \*y |
| 初始 | a | b |
| 第一步 | a | a^b |
| 第二步 | a^(a^b)=0^b=b | a^b |
| 第三步 | b | b^(a^b)=0^a=a |

1. **ex 2.11**
2. first= last= k;
3. 在最后一次for循环时，left == right == k . inplace\_swap函数参数传递为inplace\_swap(&a[k],&a[k]), 此时在这个函数中，x 和 y 地址是相同的。改变 \*y 也会改变 \*x ，在执行按位异或时，会把 \*x 和 \*y都置为0.
4. 修改函数 inpalce\_swap(int \*x, int \*y) ,在函数汇总判断 x 和 y 是否指向相同地址，若不指向同一地址，函数继续，否则函数返回。
5. **ex 2.14**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 机器数  (十六进制) |  | 机器数  (十六进制) |
| x&y | 0x20 | x&&y | 0x01 |
| x|y | 0x7f | x||y | 0x01 |
| ~x | ~y | 0xdf | !x || !y | 0x00 |
| x & !y | 0x00 | x && ~y | 0x01 |

1. **ex 2.23**

|  |  |  |
| --- | --- | --- |
| **w** | **fun1(w)** | **fun2(w)** |
| **0x00000076** | 0x00000076 | 0x00000076 |
| **0x87654321** | 0x00000021 | 0x00000021 |
| **0x000000C9** | 0x000000C9 | 0xFFFFFFFC9 |
| **0xEDCBA987** | 0x00000087 | 0xFFFFFFF87 |

1. **ex 2.35**

当x=0，容易验证程序的正确性。当x≠0时：

1. 可知w位的补码数x，y相乘最多产生一个有2w位的乘积。将这个乘积写成

其中p表示低w位表示的数值，表示高w位的数值（将低w位置为0）。易知当且仅当t≠0时p的计算溢出。

1. 设P是一个有2w位的数，且P=x\*y。设Q是一个有w位的数，且Q=P/x，有：

当p没有溢出时，也可以写成这个表达式。

1. 由1) 和2) 中的两个式子联立可得

当y≠q时，有，此时r 和 t 不全为0. 故当时，即r = t = 0时，y=q。

1. **ex 2.47**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **位** | **e** | **E** | **2E** | **f** | **M** | **2E\*M** | **V** | **十进制** |
| **00000** | 0 | 0 | 1 | 0/4 | 4/4 | 4/4 | 4/4 | 1.00 |
| **00001** | 0 | 0 | 1 | 1/4 | 5/4 | 5/4 | 5/4 | 1.25 |
| **00010** | 0 | 0 | 1 | 2/4 | 6/4 | 6/4 | 6/4 | 1.50 |
| **00011** | 0 | 0 | 1 | 3/4 | 7/4 | 7/4 | 7/4 | 1.75 |
| **00100** | 1 | 0 | 1 | 0/4 | 4/4 | 4/4 | 4/4 | 1.00 |
| **00101** | **1** | **0** | **1** | **1/4** | **5/4** | **5/4** | **5/4** | **1.25** |
| **00110** | 1 | 0 | 1 | 2/4 | 6/4 | 6/4 | 6/4 | 1.50 |
| **00111** | 1 | 0 | 1 | 3/4 | 7/4 | 7/4 | 7/4 | 1.75 |
| **01000** | 2 | 1 | 2 | 0/4 | 4/4 | 8/4 | 8/4 | 2.00 |
| **01001** | 2 | 1 | 2 | 1/4 | 5/4 | 10/4 | 10/4 | 2.50 |
| **01010** | 2 | 1 | 2 | 2/4 | 6/4 | 12/4 | 12/4 | 3.00 |
| **01011** | 2 | 1 | 2 | 3/4 | 7/4 | 14/4 | 14/4 | 3.50 |
| **01100** | **--** | **--** | **--** | **--** | **--** | **--** | ∞ | **--** |
| **01101** | **--** | **--** | **--** | **--** | **--** | **--** | NaN | **--** |
| **01110** | **--** | **--** | **--** | **--** | **--** | **--** | NaN | **--** |
| **01111** | **--** | **--** | **--** | **--** | **--** | **--** | NaN | **--** |

1. **ex .2.52**

|  |  |  |  |
| --- | --- | --- | --- |
| 格式A | | 格式B | |
| 位 | 值 | 位 | 值 |
| 011 0000 | 1 | 0111 000 | 1 |
| 101 1110 | 15/2 | 1001 111 | 15/2 |
| 010 1001 | 25/32 | 0110 100 | 3/4 |
| 110 1111 | 31/2 | 1100 000 | 32 |
| 000 0001 | 1/64 | 1101 000 | 1/64 |

1. **ex 2.63**
2. unsigned srl(unsigned x, int k)
3. { /\*Perform shift arithmetically\*/
4. unsigned xsra = (int)x >> k;
5. int w = sizeof(x) << 3;
6. int m = (1 << (w - k)) - 1;
7. return m & xsra;
8. }
9. int sra(int x, int k)
10. { /\*Perform shift logically\*/
11. int xsrl = (unsigned)x >> k;
12. int w = sizeof(x) << 3;
13. int m = 1 << w - k - 1;
14. if (m & xsrl) {
15. m << 1, m -= 1;
16. xsrl = xsrl | ~m;
17. }
18. return xsrl;
19. }
20. **ex 2.75**
21. unsigned unsigned\_high\_pprod(unsigned x, unsigned y)
22. {
23. return signed\_high\_prod(x, y) + (x >> (w - 1))\*y + x\*(y >> (w - 1));
24. }
25. **ex 2.81**
26. 否。若y=-（1<<31）。对任意大于y的x值，-x<-y并不满足。
27. 是。
28. 否。x=7，y=5.
29. 是。Int类型转unsigned类型编码没有变化。
30. 是。如果有舍入只会是左边的值变小；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 描述 | Hex | M | E | V |
| -0 | 0x8000 | 0 | -62 | -- |
| 最小的值>1 | 0xfeff | 32 | 6 | -505/2 |
| 256 | 0x4700 | 1 | 8 | -- |
| 最大的非规格化数 | 0x00ff | 255/256 | -62 | 255\* |
| -∞ | 0xff00 | -- | -- | -- |
| 十六进制表示为3AA0的数 | -- | 13/8 | -5 | 13/256 |

1. **ex 2.86**
2. **ex 2.90**
3. 13176759/8388608
4. 11.001001001001001…
5. 第九位