2.58

```
typedef unsigned char* byte_pointer;
int is_little_endian()
{
    int x = 1;
    return *((byte_pointer)& x);
}
```

2.62

```
int int_shifts_are_arithmetic() {
    return -2 >> 1 == -1;
}
```

2.66

```
int leftmost_one(unsigned x)
{

     x |= (x >> 1);
     x |= (x >> 2);
     x |= (x >> 4);
     x |= (x >> 8);
     x |= (x >> 16);
     return x^(x>>1);
}
```

2.70

```
int fits_bits(int x, int n)
{
    x = x >> (n - 1);
    return !x || !(~x);
}
```

2.74

```
int tsub_ok(int x, int y)
{
    int a = (x > 0) && (y > 0) && (x - y < 0);
    int b = (x < 0) && (y < 0) && (x - y > 0);
    return a | b;
}
```

2.78

```
int divide_power2(int x, int k)
{
    return (x + (1 << k) - 1) >> k;
```

}

2.82

A.不是总为 1。当 x 或 y 为 TMin 时，-x 和-y 也为 TMin，等式左右不相等。

B.总为 1。∵17 * y + 15 * x = 16 * (x + y) + y - x = (x + y) << 4 + y – x. ∴等式总为 1.

C.总为 1。∵~x + ~y + 1 = ~x + 1 + ~y + 1 – 1 = -x + (-y) – 1 = - (x + y) – 1 = ~(x + y) + 1 – 1 = ~(x+y). ∴等式总为 1.

D.总为 1。无符号数与有符号数的位极表示相同。

E.总为 1。算数右移补充最高位，即高位数值不会发生变化。左移的时候会在最右侧补充 0，可能使得数值变小。

2.86

描述	扩展精度	
	值	十进制
最小的非规格化数	$0\ \underbrace{0\cdots0}_{15位}\ 0\ \underbrace{0\cdots01}_{63位}$	$2^{(-61-2^{14})}$
最小的正规格化数	$0\ \underbrace{0\cdots01}_{15位}\ 1\ \underbrace{0\cdots0}_{63位}$	$1*2^{(2-2^{14})}$
最大的规格化数	$0\ \underbrace{1\cdots10}_{15位}\ 1\ \underbrace{1\cdots1}_{63位}$	$(2-2^{-63})*2^{2^{14}-1}$

2.90

```
float fpwr2(int x)
{
    /* Result exponent and fraction */
    unsigned exp, frac;
    unsigned u;
    if (x < -149) {
        /* Too small. Return 0.0 */
        exp = 0;
        frac = 0;
    } else if (x < -126) {
        /* Denormalized result */
        exp = 0;
        frac = 1<<(x+149);
    } else if (x < 128) {
        /* Normalized result. */
        exp = x + 127;
        frac = 0;
    } else {
        /* Too big. Return +oo */
        exp = 0xFF;
        frac = 0;
    }
```

```
    /* Pack exp and frac into 32 bits */
    u = exp << 23 | frac;
    /* Return as float */
    return u2f(u);
}
```

2.94

```
typedef unsigned float_bits;

float_bits float_twice(float_bits f)
{
    unsigned sign = f >> 31;
    unsigned exp = (f >> 23) & 0xff;
    unsigned frac = f & 0x7fffff;

    if (exp == 0xff)
    {
        return f;
    }
    else if (exp == 0)
    {
        frac <<= 1;
    }
    else if (exp == 0xfe)
    {
        exp = 0xff;
        frac = 0;
    }
    else
    {
        exp++;
    }
    return (sign << 31) | ((exp) << 23) | (frac);
}
```