Bit position 1 requires a borrow from bit position 2. Because of this borrow, minuend bit 2 is a 0. The subtraction continue· through the MSB.

Example 1.18

```
  |7|6|5|4|3|2|1|0|   bit position

    0 0     0
    + + θ θ + 0 1 1   minuend
        1 1
  − 0 1 1 0 1 1 1 0   subtrahend
  -----------------
    0 1 0 1 1 1 0 1   difference
```

Bit 2 requires a borrow from bit 3; after this borrow, minuend bit 3 is 0. Then, bit 3 requires a borrow. Because bits 4 and 5 of the minuend are zeros, borrowing is from bit 6. In this process, the intermediate minuend bits 4 and 5 each attain a value of 1 (compare this with the decimal subtraction). The subtraction continues through the MSB.

Multiplication Binary multiplication is similar to decimal multiplication. From Table 1.3(c), we can see that $0 \times 0 = 0$, $0 \times 1 = 0$, $1 \times 0 = 0$, and $1 \times 1 = 1$. An example follows.

Example 1.19

```
                 1011  multiplicand
               × 1100  multiplier
               ------                  ┌─multiplier bits
                 0000         (1011) × 0
Partial         0000          (1011) × 0
products       1011           (1011) × 1
              1011            (1011) × 1
             --------
             10000100  product
```

In general, the product of two n-bit numbers is $2n$ bits long. In Example 1.19, there are two nonzero bits in the multiplier, one in position 2

corresponding to 2^2 and the other in position 3 corresponding to 2^3. These two bits yield partial products whose values are simply that of the multiplicand shifted left two and three bits, respectively. The 0 bits in the multiplier contribute partial products with 0 values. Thus, the following shift-and-add algorithm can be adopted to multiply two n-bit numbers A and B, where $B = (b_{n-1}\, b_{n-2} \cdots b_1 b_0)$.

1. Start with a $2n$-bit product with a value of 0.
2. For each $b_i\ (0 \le i \le n-1) \ne 0$ shift A i positions to the left and add to the product.

This procedure reduces the multiplication to repeated shift and addition of the multiplicand.

Division The longhand (trial-and-error) procedure of decimal division can also be used in binary, as shown in Example 1.20.

Example 1.20

$$110101 \div 111 = ?$$

```
        0111   Quotient
divisor            X      Y
  111 | 110,101    110  < 111    q1 = 0   do not subtract
       −111
        1101       1101 > 111    q2 = 1   subtract
       −111
         1100      1100 > 111    q3 = 1   subtract
        −111
          1011     1011 > 111    q4 = 1   subtract
         −111
           100     remainder
```

In this procedure, the divisor is compared with the dividend at each step. If the divisor is greater than the dividend, the corresponding quotient bit is 0; otherwise, the quotient bit is 1, and the divisor is subtracted from the dividend. The compare-and-subtract process is continued until the LSB of the dividend. The procedure is formalized in the following steps.

1. Align the divisor (Y) with the most significant end of the dividend. Let the portion of the dividend from its MSB to its bit aligned with