Box 2.17

Booth's method worked example

Consider -9×11 (signed):

```
 11110111    multiplicand -9
 00001011    multiplier 11
---------
-11110111    (i = 0, subtract multiplicand since bit pair = 10)
 0000000     (i = 1, no action since bit pair = 11)
+110111      (i = 2, add multiplicand << 2 since bit pair = 01)
-10111       (i = 3, subtract multiplicand << 3 since bit pair = 10)
+0111        (i = 4, add multiplicand << 2 since bit pair = 01)
 000         (i = 5 and onwards, no action since all bit pairs = 00)
---------
```

The result is therefore obtained as the summation of the following:

```
-11110111
+11011100
-10111000
+01110000
---------
```

Or by converting the subtractions into additions (see Section 2.4.4):

```
 00001001
+11011100
+01001000
+01110000
---------
=10011101    + Carry
```

Result:

`10011101` $= -128 + 16 + 8 + 4 + 1 = -99$ (correct)

It is important to note that when i = 0, the bits considered are the least significant bit of the multiplier and a hidden zero. Thus, when the least significant bit of the multiplier is a '1', the multiplicand must be subtracted (i.e. treated as a '10' instead). This can be seen in the second worked example (Box 2.17).

There are two points worth mentioning here. First, when dealing with two's complement signed operands, the partial products must be sign extended in the same way as the full partial product multiplier.

Second, when scanning from right to left, the hidden bit at the right-hand side means that the first pair of non-equal bits that is encountered will always be a '10', indicating a subtraction. This regularity may be useful when designing a hardware implementation.

Even for someone who has been doing binary arithmetic for many years, the preparation of this book highlighted how easy it can be to make very trivial binary addition mistakes. If you are required to do this as part of an examination, always