Two's Complement Multiplication

Here are a couple of ways of doing two's complement multiplication by hand. Direct implementations of these algorithms into the circuitry would result in very slow multiplication! Actual implementations are far more complex, and use algorithms that generate more than one bit of product each clock cycle. ECE 352 should cover these faster algorithms and implementations.

Remember that the result can require 2 times as many bits as the original operands. We can assume that both operands contain the same number of bits.

"No Thinking Method" for Two's Complement Multiplication

In 2's complement, to always get the right answer without thinking about the problem, *sign extend* both integers to twice as many bits. Then take the correct number of result bits from the least significant portion of the result.

A 4-bit, 2's complement example:

```
            1111 1111        -1
          x 1111 1001     x  -7
     ----------------    ------
             11111111         7
            00000000
           00000000
          11111111
         11111111
        11111111
       11111111
 +    11111111
      ----------------
       1  00000000111
             --------  (correct answer underlined)
```

Another 4-bit, 2's complement example, showing both the incorrect result (when sign extension is not done), and the correct result (with sign extension):

```
     WRONG !                Sign extended:
       0011 (3)               0000 0011 (3)
     x 1011 (-5)            x 1111 1011 (-5)
     ------                 -----------
       0011                    00000011
      0011                    00000011
     0000                    00000000
  + 0011                    00000011
  ---------                00000011
    0100001               00000011
 not -15 in any          00000011
  representation!     + 00000011
                     ------------------
                             1011110001
                               --------
                         take the least significant 8 bits 11110001 = -15
```

"Slightly Less Work Method" for Two's Complement Multiplication

```
  multiplicand
x   multiplier
----------------
   product
```

If we do *not* sign extend the operands (multiplier and multiplicand), before doing the multiplication, then the wrong answer sometimes results. To make this work, *sign extend the partial products* to the correct number of bits.

To result in the least amount of work, classify which do work, and which do not.

```
   +           -            +            -      (muliplicands)
 x +         x +          x -          x -      (mulipiers)
 ----        ----         ----         ----
  OK         sign          |            |
            extend         | take       |
           partial         | additive   |
           products        | inverses   |

                            -            +
                          x +          x +
                          ----          ----
                           sign          OK
                         extend
                        partial
                        products
```

Example:

```
 without                          with correct
 sign extension                   sign extension

     11100 (-4)                         11100
   x 00011 (3)                        x 00011
   ------------                      ---------
     11100                         1111111100
    11100                         1111111100
   ------------                 ---------------
   1010100 (-36)                   1111110100  (-12)
    WRONG!                            RIGHT!
```

Another example:

```
   without adjustment              with correct adjustment

     11101 (-3)                       11101 (-3)
   x 11101 (-3)                     x 11101 (-3)
  -------------                    -------------
     11101                           (get additive inverse of both)
    11101
   11101                              00011 (+3)
 +11101                             x 00011 (+3)
 -----------                       -------------
 1101001001 (wrong!)                  00011
                                   + 00011
                                  ---------
                                     001001 (+9) (right!)
```