Shift-and-add Multiplication Algorithm

Let's assume that A is multiplicand, B is multiplier, and C is the result (A*B). When the inputs are n bit long, it requires 2n bits to store the result.

Step 1:

- If the inputs are signed numbers, make A and B to 2n bit long by extending the sign bit.
- Clear result variable C.

Step 2:

- Is LSB (least significant bit) of B is 1? (B[0] == 1?)
    - If yes, add A to C.
- Shift A to left by 1 bit
- Shift B to right by 1 bit

Step 3:

- Repeat step 2 for n number of times.

Step 4:

- Is B negative number? (is LSB of B is 1?)
    - If yes, subtract A from the result.

Step 5:

- The final result will be in C.

Examples:

1. Let's multiply two 4-bit signed numbers, 2 and 3. The result should be 2*3 = 6.

4-bit numbers:
$(2)_{10} = (0010)_2$
$(3)_{10} = (0011)_2$
$(6)_{10} = (0110)_2$

8-bit numbers:
A = 0000 0010 (binary)
B = 0000 0011 (binary)
C should be 0000 0110 (binary)

Traditional Multiplication:

```
                 0010 x 0011
-------------------------------------
                 0010
                0010
               0000
              0000
-------------------------------------
Result:       0000110
-------------------------------------
```

Shift-and-add Method:

Counter	Description	A[15:0]	B[15:0]	C[15:0] (result)
		0000 0010	0000 0011	0000 0000
1	B[0] is 1; C = C+A			0000 0010
	Shift A to left	0000 0100		
	Shift B to right		0000 0001	
2	B[0] is 1; C = C+A			0000 0110
	Shift A to left	0000 1000		
	Shift B to right		0000 0000	
3	B[0] is 0			0000 0110
	Shift A to left	0001 0000		
	Shift B to right		0000 0000	
4	B[0] is 0			0000 0110
	Shift A to left	0010 0000		
	Shift B to right		0000 0000	
	B[0] is 0 (it is a positive number. No need to subtract A			0000 0110 (Final result)

2. Let's multiply two 4-bit signed numbers, -2 and 3. The result should be -2*3 = -6.

4-bit numbers:
$(2)_{10} = (0010)_2$ → 2's complement is 1110
$(3)_{10} = (0011)_2$
$(6)_{10} = (0110)_2$ → 2's complement is 1010

8-bit numbers:
A = 1111 1110 (binary)
B = 0000 0011 (binary)
C should be 1111 1010 (binary)

Traditional Multiplication:

```
             1110 x 0011
-------------------------------------
             1110
            1110
           0000
          0000
-------------------------------------
Result:   0101010   ← wrong
-------------------------------------
```

```
    1111 1110 x 0000 0011     (sign extension)
-------------------------------------
              11111110
             11111110
            00000000
           00000000
-------------------------------------
```

Result: 10**11111010** ← correct (8 bits from the right)

Shift-and-add Method:

Counter	Description	A[15:0]	B[15:0]	C[15:0] (result)
		1111 1110	0000 0011	0000 0000
1	B[0] is 1; C = C+A			1111 1110

	Shift A to left	1111 1100		
	Shift B to right		0000 0001	
2	B[0] is 1; C = C+A			1111 1010
	Shift A to left	1111 1000		
	Shift B to right		0000 0000	
3	B[0] is 0			1111 1010
	Shift A to left	1111 0000		
	Shift B to right		0000 0000	
4	B[0] is 0			1111 1010
	Shift A to left	1110 0000		
	Shift B to right		0000 0000	
	B[0] is 0 (it is a positive number. No need to subtract A			1111 1010 (Final result)

3. Let’s multiply two 4-bit signed numbers, 2 and -3. The result should be 2*-3 = -6.

4-bit numbers:
$(2)_{10} = (0010)_2$
$(3)_{10} = (0011)_2$ → 2’s complement is 1101
$(6)_{10} = (0110)_2$ → 2’s complement is 1010

8-bit numbers:
A = 0010 (binary)
B = 1101 (binary)
C should be 1111 1010 (binary)

Traditional Multiplication:

```
       0010 x 1101
-------------------------------------
              0010
             0000
            0010
           0100
-------------------------------------
Result:    0101010  ← wrong
-------------------------------------
```

```
      0000 0010 x 1111 1101      (sign extension)
-------------------------------------
              00000010
             00000000
            00000010
           00000010
          00000010
         00000010
        00000010
       00000010
-------------------------------------
Result:    0011111010 ← correct (8 bits from
the right)
-------------------------------------
```

Shift-and-add Method:

Counter	Description	A[15:0]	B[15:0]	C[15:0] (result)
		0000 0010	1111 1101	0000 0000
1	B[0] is 1; C = C+A			0000 0010
	Shift A to left	0000 0100		

	Shift B to right		0111 1110	
2	B[0] is 0;			0000 0010
	Shift A to left	0000 1000		
	Shift B to right		0011 1111	
3	B[0] is 1; C=C+A			0000 1010
	Shift A to left	0001 0000		
	Shift B to right		0001 1111	
4	B[0] is 1; C=C+A			0001 1010
	Shift A to left	0010 0000		
	Shift B to right		0000 1111	
	B[0] is 1 (it is a negative number. C = C-A C = C+2's complement of A			1111 1010 (Final result)

4. Let's multiply two 4-bit signed numbers, 2 and -3. The result should be -2*-3 = 6.

4-bit numbers:
$(2)_{10} = (0010)_2$ → 2's complement is 1101
$(3)_{10} = (0011)_2$ → 2's complement is 1101
$(6)_{10} = (0110)_2$

8-bit numbers
A = 1110 (binary)
B = 1101 (binary)
C should be 0000 0110 (binary)

Traditional Multiplication:

```
       1110 x 1101
-------------------------------------
              1110
             0000
            1110
           1110
-------------------------------------
Result:     10110110  ← wrong
-------------------------------------
```

```
      1111 1110 x 1111 1101      (sign extension)
   -------------------------------------
                 11111110
                00000000
               11111110
              11111110
             11111110
            11111110
           11111110
          11111110
   -------------------------------------
```

Result: 100011011**00000110** ← correct (8 bits from the right)

Shift-and-add Method:

Counter	Description	A[15:0]	B[15:0]	C[15:0] (result)
		1111 1110	1111 1101	0000 0000
1	B[0] is 1; C = C+A			1111 1110
	Shift A to left	1111 1100		
	Shift B to right		0111 1110	

2	B[0] is 0;			1111 1110
	Shift A to left	1111 1000		
	Shift B to right		0011 1111	
3	B[0] is 1; C=C+A			1111 0110
	Shift A to left	1111 0000		
	Shift B to right		0001 1111	
4	B[0] is 1; C=C+A			1110 0110
	Shift A to left	1110 0000		
	Shift B to right		0000 1111	
	B[0] is 1 (it is a negative number. C = C-A C = C+2’s complement of A			0000 0110 (Final result)