Base 2^ 15, two’s compliment

Smallest = -16384 to 16383 (4000 – 3FFF) (\*100 0000 0000 0000 \*011 1111 1111 1111)

-1 7FFF (\*111 1111 1111 1111)

1 000001 (\*000 0000 0000 0001)

431 01AF 000 0001 1010 1111

+ -558 7DD2 111 1101 1101 0010

============

-127 7F81 111 1111 1000 0001

1 1000 0001

-256 + 128 + 1

431 01AF 000 0001 1010 1111

-558 7DD2 111 1101 1101 0010

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | 1 |  | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 |  |  |
| \* | **0** | **0** | **0** |  | **0** | **0** | **0** | **1** |  | **1** | **0** | **1** | **0** |  | **1** | **1** | **1** | **1** |
| \* | **1** | **1** | **1** |  | **1** | **1** | **0** | **1** |  | **1** | **1** | **0** | **1** |  | **0** | **0** | **1** | **0** |
|  | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 1 | 0 | 0 | 0 |  | 0 | 0 | 0 | 1 |

23,655 0000 5C67 0 0101 1100 0110 0111

+ -33,544 FFFF 7CF8 1 0111 1100 1111 1000

Broken into words

23,655 0000 5C67 0 0101 1100 0110 0111

0 23,655

+ -33,544 FFFF 7CF8 -1 0111 1100 1111 1000

-1 776

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | 1 |  |  | 110 | 0000 |
|  | 23,655 | \*000 | 0000 | 0000 | 0000 |  | **\*101** | **1100** | **0110** | 0111 |
| + | -33,544 | \*111 | 1111 | 1111 | 1110 |  | **\*111** | **1100** | **1111** | 1000 |
|  | -9889 | \*111 | 1111 | 1111 | 1111 |  | \*101 | 1001 | 0101 | 1111 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | 0 | 0 |  | 1 |
| \* | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  |
| \* | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 0 |  |
|  | **1** | **1** | **1** |  | **1** | **1** | **1** | **1** |  | **1** | **1** | **1** | **1** |  | **1** | **1** | **1** | **1** |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 |  | 1 | 1 | 0 | 0 |  | 1 | 1 | 1 | 0 |  |  | 0 | 0 | 0 |  |
| \* | 1 | 0 | 1 |  | 1 | 1 | 0 | 0 |  | 0 | 1 | 1 | 0 |  | 0 | 1 | 1 | 1 |
| \* | 1 | 1 | 1 |  | 1 | 1 | 0 | 0 |  | 1 | 1 | 1 | 1 |  | 1 | 0 | 0 | 0 |
|  | **1** | **0** | **1** |  | **1** | **0** | **0** | **1** |  | **0** | **1** | **0** | **1** |  | **1** | **1** | **1** | **1** |

\*111 1111 1111 1111 \*101 1001 0101 1111

= -32,768 +22,879

= -9,889

8:55

8:

64-bit result with format 2:16:46 from multiplying two 32-bit values having format 1:8:23

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \*\*11 | 1111 | 1111 | 1111 |  | 1101 | 1001 | 0101 | 1111 |  | 1011 | 1111 | 1111 | 1111 |  | 1101 | 1001 | 0101 | 1111 |

To round this result so that it once more fits into a 1:8:23 format, follow these instructions:

1. Break into a low and a high value.
   1. Low = result AND 0x000000007FFFFFFF (top 33 bits are set to zero, leaving the only bits 30:0 having a value.
   2. High = result SRL by 31 places bits that were in positions 62:31 is now in positions 31:0. (63 -> 32)

var result1\_Low\_L0 = Avx2.And(productVector11, HIGH33\_MASK\_VEC\_L)

var result1\_Low\_L1 = Avx2.ShiftRightLogical(productVector11, EFFECTIVE\_BITS\_PER\_LIMB)

1. Form a result using bits 23:0 from High and bits 30:23, from Low (24 from High and 8 from low)
   1. Top = Shift the High value, left by 8 places, and then AND with 0x000000007FFFFFFF (bits from original positions 54:31 -> 31:8
   2. Bottom = Shift the Low value, right by 23 places (bits from original positions 30:23 -> 7:0

Currently

54:31 -> 31:8

30:23 -> 7:0

Update to:

53:30 -> 31:8

29:22 -> 7:0

var wideResult1Low = Avx2.And(Avx2.ShiftLeftLogical(result1\_Low\_L1, \_shiftAmount), HIGH33\_MASK\_VEC\_L)

wideResult1Low = Avx2.Or(wideResult1Low, Avx2.ShiftRightLogical(Avx2.And(result1\_Low\_L0, HIGH33\_MASK\_VEC\_L), \_inverseShiftAmount))

1. The Top and Bottom values are ORed together form a single 31-bit value.

This is equivalent to shifting the original value, right by 23 places and masking off the top 33 bits. (63 -> 40, 62 -> 39, 54 -> 31, 53 -> 30, 23 -> 0

Update to shift the original by 22, and masking off the top 33. -> Actually, 23 is the correct amount!!

High (Step 1) (Step 2) (Step 3)

63 -> 32 -> 40 Set to 0

62 -> 31 -> 39 Set to 0

61

60

59

58

57

56

55

54 ->23 -> 31 Set to 0

53

52

51

50

49

48

47

46

45

44 (positions 31–53 (total of 23) contribute from the ‘high’ limb.

43

42

41

40

39

38

37

36

35

34

33

32

31 -> 0 ->8

Low

31 -> 8 (which is zero, top 33 bits were masked)

30 -> 7

29

28

27 (positions 23-30 (total of 8) contribute from the ‘low’ limb.

26

25

24

23 -> 0

Then High and Low are ORed together.

22

21

20

19

18

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

0