3

Solutions

3.1 5730

3.2 5730

3.3 0101111011010100

The attraction is that each hex digit contains one of 16 different characters (0–9, A–E). Since with 4 binary bits you can represent 16 different patterns, in hex each digit requires exactly 4 binary bits. And bytes are by definition 8 bits long, so two hex digits are all that are required to represent the contents of 1 byte.

3.4 753

3.5 7777 (−3777)

3.6 Neither (63)

3.7 Neither (65)

3.8 Overflow (result = −179, which does not fit into an SM 8-bit format)

3.9 −105 − 42 = −128 (−147)

3.10 −105 + 42 = −63

3.11 151 + 214 = 255 (365)

3.12 62 × 12

Step	Action	Multiplier	Multiplicand	Product
0	Initial Vals	001 010	000 000 110 010	000 000 000 000
1	lsb=0, no op	001 010	000 000 110 010	000 000 000 000
	Lshift Mcand	001 010	000 001 100 100	000 000 000 000
	Rshift Mplier	000 101	000 001 100 100	000 000 000 000
2	Prod=Prod+Mcand	000 101	000 001 100 100	000 001 100 100
	Lshift Mcand	000 101	000 011 001 000	000 001 100 100
	Rshift Mplier	000 010	000 011 001 000	000 001 100 100
3	lsb=0, no op	000 010	000 011 001 000	000 001 100 100
	Lshift Mcand	000 010	000 110 010 000	000 001 100 100
	Rshift Mplier	000 001	000 110 010 000	000 001 100 100
4	Prod=Prod+Mcand	000 001	000 110 010 000	000 111 110 100
	Lshift Mcand	000 001	001 100 100 000	000 111 110 100
	Rshift Mplier	000 000	001 100 100 000	000 111 110 100
5	lsb=0, no op	000 000	001 100 100 000	000 111 110 100
	Lshift Mcand	000 000	011 001 000 000	000 111 110 100
	Rshift Mplier	000 000	011 001 000 000	000 111 110 100
6	lsb=0, no op	000 000	110 010 000 000	000 111 110 100
	Lshift Mcand	000 000	110 010 000 000	000 111 110 100
	Rshift Mplier	000 000	110 010 000 000	000 111 110 100

3.13 62 × 12

Step	Action	Multiplicand	Product/Multiplier
0	Initial Vals	110 010	000 000 001 010
1	lsb=0, no op	110 010	000 000 001 010
	Rshift Product	110 010	000 000 000 101
2	Prod=Prod+Mcand	110 010	110 010 000 101
	Rshift Mplier	110 010	011 001 000 010
3	lsb=0, no op	110 010	011 001 000 010
	Rshift Mplier	110 010	001 100 100 001
4	Prod=Prod+Mcand	110 010	111 110 100 001
	Rshift Mplier	110 010	011 111 010 000
5	lsb=0, no op	110 010	011 111 010 000
	Rshift Mplier	110 010	001 111 101 000
6	lsb=0, no op	110 010	001 111 101 000
	Rshift Mplier	110 010	000 111 110 100

3.14 For hardware, it takes one cycle to do the add, one cycle to do the shift, and one cycle to decide if we are done. So the loop takes (3 × A) cycles, with each cycle being B time units long.

For a software implementation, it takes one cycle to decide what to add, one cycle to do the add, one cycle to do each shift, and one cycle to decide if we are done. So the loop takes (5 × A) cycles, with each cycle being B time units long.

(3 × 8) × 4tu = 96 time units for hardware

(5 × 8) × 4tu = 160 time units for software

3.15 It takes B time units to get through an adder, and there will be A − 1 adders. Word is 8 bits wide, requiring 7 adders. 7 × 4tu = 28 time units.

3.16 It takes B time units to get through an adder, and the adders are arranged in a tree structure. It will require log2(A) levels. An 8 bit wide word requires seven adders in three levels. 3 × 4tu = 12 time units.

3.17 0x33 × 0x55 = 0x10EF. 0x33 = 51, and 51 = 32 + 16 + 2 + 1. We can shift 0x55 left five places (0xAA0), then add 0x55 shifted left four places (0x550), then add 0x55 shifted left once (0xAA), then add 0x55. 0xAA0 + 0x550 + 0xAA + 0x55 = 0x10EF. Three shifts, three adds.

(Could also use 0x55, which is 64 + 16 + 4 + 1, and shift 0x33 left six times, add to it 0x33 shifted left four times, add to that 0x33 shifted left two times, and add to that 0x33. Same number of shifts and adds.)

3.18 74/21 = 3 remainder 9

Step	Action	Quotient	Divisor	Remainder
0	Initial Vals	000 000	010 001 000 000	000 000 111 100
1	Rem=Rem–Div	000 000	010 001 000 000	101 111 111 100
	Rem<0,R+D,Q<<	000 000	010 001 000 000	000 000 111 100
	Rshift Div	000 000	001 000 100 000	000 000 111 100
2	Rem=Rem–Div	000 000	001 000 100 000	111 000 011 100
	Rem<0,R+D,Q<<	000 000	001 000 100 000	000 000 111 100
	Rshift Div	000 000	000 100 010 000	000 000 111 100
3	Rem=Rem–Div	000 000	000 100 010 000	111 100 101 100
	Rem<0,R+D,Q<<	000 000	000 100 010 000	000 000 111 100
	Rshift Div	000 000	000 010 001 000	000 000 111 100
4	Rem=Rem–Div	000 000	000 010 001 000	111 110 110 100
	Rem<0,R+D,Q<<	000 000	000 010 001 000	000 000 111 100
	Rshift Div	000 000	000 001 000 100	000 000 111 100
5	Rem=Rem–Div	000 000	000 001 000 100	111 111 111 000
	Rem<0,R+D,Q<<	000 000	000 001 000 100	000 000 111 100
	Rshift Div	000 000	000 000 100 010	000 000 111 100
6	Rem=Rem–Div	000 000	000 000 100 010	000 000 011 010
	Rem>0,Q<<1	000 001	000 000 100 010	000 000 011 010
	Rshift Div	000 001	000 000 010 001	000 000 011 010
7	Rem=Rem–Div	000 001	000 000 010 001	000 000 001 001
	Rem>0,Q<<1	000 011	000 000 010 001	000 000 001 001
	Rshift Div	000 011	000 000 001 000	000 000 001 001

3.19 In these solutions a 1 or a 0 was added to the Quotient if the remainder was greater than or equal to 0. However, an equally valid solution is to shift in a 1 or 0, but if you do this you must do a compensating right shift of the remainder (only the remainder, not the entire remainder/quotient combination) after the last step.

74/21 = 3 remainder 11

Step	Action	Divisor	Remainder/Quotient
0	Initial Vals	010 001	000 000 111 100
1	R<<	010 001	000 001 111 000
	Rem=Rem–Div	010 001	111 000 111 000
	Rem<0,R+D	010 001	000 001 111 000
2	R<<	010 001	000 011 110 000
	Rem=Rem–Div	010 001	110 010 110 000
	Rem<0,R+D	010 001	000 011 110 000
3	R<<	010 001	000 111 100 000
	Rem=Rem–Div	010 001	110 110 110 000
	Rem<0,R+D	010 001	000 111 100 000
4	R<<	010 001	001 111 000 000
	Rem=Rem–Div	010 001	111 110 000 000
	Rem<0,R+D	010 001	001 111 000 000

Step	Action	Divisor	Remainder/Quotient
5	R<<	010 001	011 110 000 000
	Rem=Rem–Div	010 001	111 110 000 000
	Rem>0,R0=1	010 001	001 101 000 001
6	R<<	010 001	011 010 000 010
	Rem=Rem–Div	010 001	001 001 000 010
	Rem>0,R0=1	010 001	001 001 000 011

3.20 `201326592` in both cases.

3.21 `jal 0x00000000`

3.22 0×0C000000 = 0000 1100 0000 0000 0000 0000 0000 0000

= 0 0001 1000 0000 0000 0000 0000 0000 000

sign is positive

exp = 0 × 18 = 24 − 127 = −103

there is a hidden 1

mantissa = 0

answer = 1.0×2^{-103}

3.23 $63.25 \times 10^0 = 111111.01 \times 2^0$

normalize, move binary point five to the left

1.1111101×2^5

sign = positive, exp = 127 + 5 = 132

Final bit pattern: 0 1000 0100 1111 1010 0000 0000 0000 000

= 0100 0010 0111 1101 0000 0000 0000 0000 = 0x427D0000

3.24 $63.25 \times 10^0 = 111111.01 \times 2^0$

normalize, move binary point five to the left

1.1111101×2^5

sign = positive, exp = 1023 + 5 = 1028

Final bit pattern:

0 100 0000 0100 1111 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

= 0x404FA00000000000

3.25 $63.25 \times 10^0 = 111111.01 \times 2^0 = 3F.40 \times 16^0$

move hex point two to the left

$0.3F40 \times 16^2$

sign = positive, exp = 64 + 2

Final bit pattern: 01000010001111110100000000000000

3.26 $-1.5625 \times 10^{-1} = -0.15625 \times 10^0$

$= -0.00101 \times 2^0$

move the binary point two to the right

$= -0.101 \times 2^{-2}$

exponent = −2, fraction = −0.10100000000000000000000

answer: 11111111111010110000000000000000000

3.27 $-1.5625 \times 10^{-1} = -0.15625 \times 10^0$

$= -0.00101 \times 2^0$

move the binary point three to the right, $= -1.01 \times 2^{-3}$

exponent = −3 = −3 + 15 = 12, fraction = −0.0100000000

answer: 1011000100000000

3.28 $-1.5625 \times 10^{-1} = -0.15625 \times 10^0$

$= -0.00101 \times 2^0$

move the binary point two to the right

$= -0.101 \times 2^{-2}$

exponent = −2, fraction = −0.10100000000000000000000000

answer: 10110000000000000000000000000101

3.29 $2.6125 \times 10^1 + 4.150390625 \times 10^{-1}$

$2.6125 \times 10^1 = 26.125 = 11010.001 = 1.1010001000 \times 2^4$

$4.150390625 \times 10^{-1} = 0.4150390625 = 0.011010100111 = 1.1010100111 \times 2^{-2}$

Shift binary point six to the left to align exponents,

```
             GR
1.1010001000 00

1.0000011010 10 0111 (Guard 5 1, Round 5 0,
Sticky 5 1)

-------------------

1.1010100010 10
```

In this case the extra bit (G,R,S) is more than half of the least significant bit (0).

Thus, the value is rounded up.

$1.1010100011 \times 2^4 = 11010.100011 \times 2^0 = 26.546875 = 2.6546875 \times 10^1$

3.30 $-8.0546875 \times -1.79931640625 \times 10^{-1}$

$-8.0546875 = -1.0000000111 \times 2^3$

$-1.79931640625 \times 10^{-1} = -1.0111000010 \times 2^{-3}$

Exp: $-3 + 3 = 0$, $0 + 16 = 16$ (10000)

Signs: both negative, result positive

Fraction:

```
            1.0000000111
          × 1.0111000010
             ------------
              00000000000
             10000000111
            00000000000
           00000000000
          00000000000
         00000000000
        10000000111
       10000000111
      10000000111
     00000000000
    10000000111
1.01110011000001001110
```

1.0111001100 00 01001110 Guard = 0, Round = 0, Sticky = 1:NoRnd

$1.0111001100 \times 2^0 = 0100000111001100$ (1.0111001100 = 1.44921875)

$-8.0546875 \times -0.179931640625 = 1.4492931365966796875$

Some information was lost because the result did not fit into the available 10-bit field. Answer (only) off by 0.0000743865966796875

3.31 $8.625 \times 10^1 / -4.875 \times 10^0$

$8.625 \times 10^1 = 1.0101100100 \times 2^6$

$-4.875 = -1.0011100000 \times 2^2$

Exponent = 6 − 2 = 4, 4 + 15 = 19 (10011)

Signs: one positive, one negative, result negative

Fraction:

```
                          1.00011011000100111
10011100000. | 10101100100.0000000000000000
             −10011100000.
               --------------
                10000100.0000
               −1001110.0000
               --------------
                 1100110.00000
                −100111.00000
                --------------
                    1111.0000000
                  −1001.1100000
                  -------------
                     101.01000000
                   −100.11100000
                   -------------
                     000.011000000000
                       −.010011100000
                        --------------
                         .000100100000000
                        −.000010011100000
                         -----------------
                         .0000100001000000
                        −.0000010011100000
                         ------------------
                         .00000011011000000
                        −.00000010011100000
                         -------------------
                         .00000000110000000
```

1.000110110001001111 Guard = 0, Round = 1, Sticky = 1: No Round, fix sign

$-1.0001101100 \times 2^4 = 1101000001101100 = 10001.101100 = -17.6875$

$86.25/-4.875 = -17.692307692307$

Some information was lost because the result did not fit into the available 10-bit field. Answer off by 0.00480769230

3.32 $(3.984375 \times 10^{-1} + 3.4375 \times 10^{-1}) + 1.771 \times 10^3)$

$3.984375 \times 10^{-1} = 1.1001100000 \times 2^{-2}$

$3.4375 \times 10^{-1} = 1.0110000000 \times 2^{-2}$

$1.771 \times 10^3 = 1771 = 1.1011101011 \times 2^{10}$

shift binary point of smaller left 12 so exponents match

```
(A)         1.1001100000
(B)        +1.0110000000
           -------------
           10.1111100000 Normalize,
(A+B)       1.0111110000 × 2^-1
(C)        +1.1011101011
(A+B)         .0000000000 10 111110000 Guard = 1,
                          Round = 0, Sticky = 1
          ---------------
(A+B)+C  +1.1011101011 10 1 Round up
```

$(A+B)+C = 1.1011101100 \times 2^{10}$ = 0110101011101100 = 1772

3.33 $3.984375 \times 10^{-1} + (3.4375 \times 10^{-1} + 1.771 \times 10^3)$

$3.984375 \times 10^{-1} = 1.1001100000 \times 2^{-2}$

$3.4375 \times 10^{-1} = 1.0110000000 \times 2^{-2}$

$1.771 \times 10^3 = 1771 = 1.1011101011 \times 2^{10}$

shift binary point of smaller left 12 so exponents match

```
(B)          .0000000000 01 0110000000 Guard = 0,
                            Round = 1, Sticky = 1
(C)        +1.1011101011
           -------------
(B+C)      +1.1011101011
```

```
(A)       .0000000000  011001100000
        --------------
A + (B + C) + 1.1011101011 No round
```

A + (B + C) + $1.1011101011 \times 2^{10}$ = 0110101011101011 = 1771

3.34 No, they are not equal: (A+B)+C = 1772, A+(B+C) = 1771 (steps shown above).

Exact: 0.398437 + 0.34375 + 1771 = 1771.742187

3.35 $(3.41796875 \times 10^{-3} \times 6.34765625 \times 10^{-3}) \times 1.05625 \times 10^{2}$

(A) $3.41796875 \times 10^{-3} = 1.1100000000 \times 2^{-9}$

(B) $4.150390625 \times 10^{-3} = 1.0001000000 \times 2^{-8}$

(C) $1.05625 \times 10^{2} = 1.1010011010 \times 2^{6}$

Exp: −9−8 = −17

Signs: both positive, result positive

Fraction:

```
(A)                  1.1100000000
(B)                × 1.0001000000
                     --------------
                 11100000000
              11100000000
            ----------------------
          1.11011100000000000000
A×B       1.1101110000 00 00000000
Guard = 0, Round = 0, Sticky = 0: No Round
```

A×B $1.1101110000 \times 2^{-17}$ UNDERFLOW: Cannot represent number

3.36 $3.41796875 \times 10^{-3} \times (6.34765625 \times 10^{-3} \times 1.05625 \times 10^{2})$

(A) $3.41796875 \times 10^{-3} = 1.1100000000 \times 2^{-9}$

(B) $4.150390625 \times 10^{-3} = 1.0001000000 \times 2^{-8}$

(C) $1.05625 \times 10^{2} = 1.1010011010 \times 2^{6}$

Exp: − 8 + 6 = −2

Signs: both positive, result positive

Fraction:

```
(B)    1.0001000000
(C)  × 1.1010011010
          ------------
           10001000000
         10001000000
        10001000000
      10001000000
   10001000000
  10001000000
 -----------------------
1.11000000111010000000 0
1.1100000011 10 100000000 Guard 5 1, Round 5 0, Sticky
5 1: Round
```

$B \times C \quad 1.1100000100 \times 2^{-2}$

Exp: $-9-2 = -11$

Signs: both positive, result positive

Fraction:

```
(A)     1.1100000000
(B × C)× 1.1100000100
          ------------
           11100000000
     11100000000
    11100000000
   11100000000
  ---------------------
11.000100011100000000000    Normalize, add 1 to exponent
 1.1000100011 10 0000000000 Guard=1, Round=0, Sticky=0:
Round to even
```

$A \times (B \times C)\ 1.1000100100 \times 2^{-10}$

3.37 b) No:

$A \times B = 1.1101110000 \times 2^{-17}$ UNDERFLOW: Cannot represent

$A \times (B \times C) = 1.1000100100 \times 2^{-10}$

A and B are both small, so their product does not fit into the 16-bit floating point format being used.

3.38 $1.666015625 \times 10^0 \times (1.9760 \times 10^4 - 1.9744 \times 10^4)$

(A) $1.666015625 \times 10^0 = 1.1010101010 \times 2^0$

(B) $1.9760 \times 10^4 = 1.0011010011 \times 2^{14}$

(C) $-1.9744 \times 10^4 = -1.0011010010 \times 2^{14}$

Exponents match, no shifting necessary

```
(B)      1.0011010011
(C)     −1.0011010010
        ----------------
(B + C)  0.0000000001 × 2^14
(B + C)  1.0000000000 × 2^4
```

Exp: 0 + 4 = 4

Signs: both positive, result positive

Fraction:

```
(A)                    1.1010101010
(B + C)              × 1.0000000000
                       ------------
            11010101010
           -----------------------
           1.10101010100000000000
A × (B + C) 1.1010101010 0000000000 Guard = 0, Round =
            0, sticky = 0: No round
```

$A \times (B + C)$ 1.1010101010×2^4

3.39 $1.666015625 \times 10^0 \times (1.9760 \times 10^4 - 1.9744 \times 10^4)$

(A) $1.666015625 \times 10^0 = 1.1010101010 \times 2^0$

(B) $1.9760 \times 10^4 = 1.0011010011 \times 2^{14}$

(C) $-1.9744 \times 10^4 = -1.0011010010 \times 2^{14}$

Exp: 0 + 14 = 14

Signs: both positive, result positive

Fraction:

```
(A)              1.1010101010
(B)            × 1.0011010011
                  ------------
                   11010101010
                  11010101010
                 11010101010
                11010101010
               11010101010
              11010101010
             ------------------------------
             10.0000001001100001111 Normalize, add 1 to
                                    exponent
A × B    1.0000000100 11 00001111 Guard = 1, Round = 1,
                                     Sticky = 1: Round
A × B    1.0000000101 × 2^15
```

Exp: 0 + 14 = 14

Signs: one negative, one positive, result negative

Fraction:

```
(A)                              1.1010101010
(C)                            × 1.0011010010
                                 ------------
                                 11010101010
                                11010101010
                               11010101010
                              11010101010
                             11010101010
                            -------------------------
                            10.0000000111110111010
Normalize, add 1 to exponent
A × C                    1.0000000011 11 101110100
Guard = 1, Round = 1, Sticky = 1: Round
A × C       −1.0000000100 × 2^15
A × B              1.0000000101 × 2^15
A × C             −1.0000000100 × 2^15
                    --------------
A × B+A × C          .0000000001 × 2^15
A × B+A × C         1.0000000000 × 2^5
```

3.40 b) No:

A×(B+C) = 1.1010101010 × 2^4 = 26.65625, and (A×B)+(A×C) = 1.0000000000 × 2^5 = 32

Exact: 1.666015625 × (19,760 − 19,744) = 26.65625

3.41

Answer	sign	exp	Exact?
1 01111101 00000000000000000000000	–	−2	Yes

3.42 b+b+b+b = −1

b × 4 = −1

They are the same

3.43 0101 0101 0101 0101 0101 0101

No

3.44 0011 0011 0011 0011 0011 0011

No

3.45 0101 0000 0000 0000 0000 0000

0.5

Yes

3.46 01010 00000 00000 00000

0.A

Yes

3.47 Instruction assumptions:

(1) 8-lane 16-bit multiplies

(2) sum reductions of the four most significant 16-bit values

(3) shift and bitwise operations

(4) 128-, 64-, and 32-bit loads and stores of most significant bits

Outline of solution:

```
load register F[bits 127:0] = f[3..0] & f[3..0] (64-bit
load)
load register A[bits 127:0] = sig_in[7..0] (128-bit load)
```

```
 for i = 0 to 15 do
      load register B[bits 127:0] = sig_in[(i*8+7..i*8]
      (128-bit load)

      for j = 0 to7 do
      (1) eight-lane multiply C[bits 127:0] = A*F
      (eight 16-bit multiplies)
      (2) set D[bits 15:0] = sum of the four 16-bit values
      in C[bits 63:0] (reduction of four 16-bit values)
         (3) set D[bits 31:16] = sum of the four 16-bit
         values in C[bits 127:64] (reduction of four 16-
         bit values)
         (4) store D[bits 31:0] to sig_out (32-bit store)
         (5) set A = A shifted 16 bits to the left
         (6) set E = B shifted 112 shifts to the right
         (7) set A = A OR E
         (8) set B = B shifted 16 bits to the left
 end for
end for
```