Answer 1)

Formula used

i. **Converting binary to decimal**

= 1/0 * (2^N)

Where,

N is the position from right to left starting from decimal.

ii. **Implementing 2's complement**

a. **Invert the bits (0s to 1s and vice versa)**
b. **Add 1 to LSB**
c. **Final value is presented with negative sign in decimal**

2's complement is done when the MSB is set (1) before converting it to decimal from binary.

Unsigned

Sum of 8 bits	R1	R2	Sum	Overflow Test
0110 1110 1001 1111 1 0000 1101(S)	$(01101110)_2$ $= (110)_{16}$	$(10011111)_2$ $= (159)_{16}$	$(00001101)_2$ $= (13)_{16}$	$110 + 159 \neq 13$ **Overflow occurs** (Ignore Carry)
1111 1111 0000 0001 1 0000 0000(S)	$(11111111)_2$ $= (255)_{16}$	$(00000001)_2$ $= (1)_{16}$	$(00000000)_2$ $= (0)_{16}$	$255 + 1 \neq 0$ **Overflow occurs** (Ignore Carry)
1000 0000 0111 1111 0 1111 1111(S)	$(10000000)_2$ $= (128)_{16}$	$(01111111)_2$ $= (127)_{16}$	$(11111111)_2$ $= (255)_{16}$	$128 + 127 = 255$ **No Overflow** No Carry
0111 0001 0000 1111 0 1000 0000(S)	$(01110001)_2$ $= (113)_{16}$	$(00001111)_2$ $= (15)_{16}$	$(10000000)_2$ $= (128)_{16}$	$113 + 15 = 128$ **No Overflow** No Carry

Signed

Sum of 8 bits	R1	R2	Sum	Overflow Test
0110 1110 1001 1111 1 0000 1101(S)	$(01101110)_2$ $= (110)_{16}$	$(10011111)_2$ $= (-97)_{16}$ MSB is 1	$(00001101)_2$ $= (13)_{16}$	$110 +(-97) = 13$ **No Overflow** (Ignore Carry)
1111 1111 0000 0001 1 0000 0000(S)	$(11111111)_2$ $= (-1)_{16}$ MSB is 1	$(00000001)_2$ $= (1)_{16}$	$(00000000)_2$ $= (0)_{16}$	$(-1) + 1 = 0$ **No Overflow** (Ignore Carry)
1000 0000 0111 1111 0 1111 1111(S)	$(10000000)_2$ $= (-128)_{16}$ MSB is 1	$(01111111)_2$ $= (127)_{16}$	$(11111111)_2$ $= (-1)_{16}$ MSB is 1	$(-128)+ 127= -1$ **No Overflow** No Carry
0111 0001 0000 1111 0 1000 0000(S)	$(01110001)_2$ $= (113)_{16}$	$(00001111)_2$ $= (15)_{16}$	$(10000000)_2$ $= (-128)_{16}$ MSB is 1	$113 + 15 \neq -128$ **Overflow occurs** No Carry

Answer 2)
The answer would require 32 bits.

0xABhex × 0xEFhex

= 10101011 × 11101111 (binary)

= 10101011 × (2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0)

Hence, we need to add the following in a variable (final_result) using shift left–

1) Set final_result to 0
2) Shift ABhex by 7 places i.e., 101010110000000
3) Shift ABhex by 6 places i.e., 10101011000000
4) Shift ABhex by 5 places i.e., 1010101100000
5) Shift ABhex by 3 places i.e., 10101011000
6) Shift ABhex by 2 places i.e., 1010101100
7) Shift ABhex by 1 place i.e., 101010110
8) Add ABhex to the final result 10101011

The final result would be 1001111110100101.

Answer 3)
$(\text{DEADBEEF})_{16} = (11011110101011011011111011101111)_2$

According IEEE 754, the MSB is sign bit. The next 8 bits represent exponent value and rest 23bits is significand value (Given)

Here, the sign bit is 1 which indicates negative number.

$(10111101)_2 = (189)_{10}$

The exponent value becomes 189 – 127 i.e., 62

The IEEE decimal value becomes 1.(23 bits) * 2^exp

= 1. 01011011011111011101111 * 2^62

Calculations –

$(0.01011011011111011101111)_2 = (0.35738933086395263672)_{10}$

Final value becomes 1. 35738933086395263672 * 2^{62} in decimal which comes out to be **6.259,853,398,707,798,016 * (10^18) in negative.**

Answer 4)
$(78.75)_{10} = 78 + 0.75$

= 1001110 + 0.11

= 1001110.11 in binary = 1.00111011 * 2^6

1) Single precision format

   The number is positive so, MSB is 0.
   Hence, exponent bits became 127 + 6 i.e., 133
   (133) = (10000101)
   The final representation in binary becomes **0 10000101 00111011000000000000000**.

2) Double precision format (64 bits – 11 bits for exponent and 52 bits for significand value)

   The number is positive so, MSB is 0.
   Hence, exponent bits became 1023 + 6 i.e., 1029
   (1029) = (10000000101)
   The final representation in binary becomes **0 10000000101 0011101100000000000000000000000000000000000000000000**

Answer 5)
$(78.75)_{10} = 78 + 0.75$

= 4E + 0.C
= 4E.C in hexadecimal = 4.EC * 16^1
= 1001110.11 in binary (from answer 4)
= 0.0100111011 * (16^2) to make the LSB 0 (normalization)

The correct bias becomes 64 + 2 from (16^2) i.e., $(1000010)_2$

The MSB will be 0 as the number is positive.

According to single precision format **0 1000010 0100 1110 1100 0000 0000 0000**.

Answer 6)
a)
According IEEE 754, the MSB is sign bit. The next 5 bits represent exponent value and rest 10bits is significand value for half precision (Given)

Bias for 5 exponent bits = 2 (5-1) – 1 = 15

The binary format of $-1.3625 * 10^{-1}$ will be:
$= - (0.001000101110)_2$
$= (1.000101110)_2 * 2^{-3}$

Value of Exponent = E – base
E = Value of Exponent + base
= -3 + 15 = 12
$= (01100)_2$

According to half precision format **1 01100 0001011100**.
The single precision IEEE 754 format for range and accuracy of the above 16-bit point format will be explained in Answer 7 (Given)

b)
Sum of $1.6125 * 10^1$ and $3.150390625 * 10^{-1}$

I. 16.125 = 10000.001

Remainder of Division by 2	
	16
0	8
0	4
0	2
0	1
1	0
Int of Multiplication by 2	
	0.125
0	0.25
0	0.5
1	1

Similarly for operand 2,

II. 0.3150390625 = 0.01010000101001

Int of Multiplication by 2	
	0.3150390625
0	0.630078125
1	1.26015625
0	0.5203125
1	1.040625
0	0.08125
0	0.1625
0	0.325
0	0.65
1	1.3
0	0.6
1	1.2
0	0.4
0	0.8
1	1.6

Sum of 2 operands will be done by
$= (1.0000001000 + 0.0000010100001010) * 2^4$
$= 1.0000011100\ 001010 * 2^4$

Therefore, Gourd bit becomes 0, Round bit also becomes 0 and Sticky bit is 1.
Thus,
$(10000.011100001010)_2$
$= (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) + (1 \times 2^{-4}) + (0 \times 2^{-5}) + (0 \times 2^{-6}) + (0 \times 2^{-7}) + (0 \times 2^{-8}) + (1 \times 2^{-9}) + (0 \times 2^{-10}) + (1 \times 2^{-11}) + (0 \times 2^{-12})$
$= (16.43994140625)_{10}$

Answer 7)
a) Single Precision
Range of exponent will lie between 1 and (2^8) – 2
The value of exponent will hence be between
1 – 127 i.e., -126 and 254 - 127 = 127
The mantissa would be between all 23 bits 0s and 1s.
Hence, the range becomes

$= (1.0)2 * 2\text{-}^{126}$ and $(1.11111111111111111111111)2 * 2^{127}$
$= 1.0 * 2\text{-}^{126}$ and $1.9999998807907104 * 2^{127}$
= 1.1754943508222875e-38 and 3.4028234663852886e+38

Value of $2^{-(23-1)} = 2^{-22} = 2.384185791015625\text{e-}07$.
This tells us that the above one (single precision) is accurate only to 7 places post decimal.

b) Half Precision
Range of exponent will lie between 1 and (2^4) – 2
The value of exponent will hence be between
1 – 15 i.e., -14 and 30 - 15 = 15
The mantissa would be between all 10 bits 0s and 1s.
Hence, the range becomes

= (1.0)2 * 2^{-14} and (1. 1.1111111111)2 * 2^{15}
= 1.0 * 2^{-14} and 1.9990234375* 2^{15}
= 6.103515625e-05 and 6.5504e+04

Value of $2^{-(10-1)} = 2^{-9} =$ 1.953125e-03
This tells us that the above one (half precision) is accurate only to 3 places post decimal.

c) Floating Point
Range of exponent will lie between 1 and (2^8) – 2
The value of exponent will hence be between
1 – 127 i.e., -126 and 254 - 127 = 127
The mantissa would be between all 7 bits 0s and 1s.
Hence, the range becomes

= (1.0)2 * 2^{-126} and (1. 1111111)2 * 2^{127}
= 1.0 * 2^{-126} and 1.9999998807907104 * 2^{127}
= 1.1754943508222875e-38 and 3.4028234663852886e+38

Value of $2^{-(7-1)} = 2^{-6} =$ 1.5625e-02. This tells us that the above one (bfloat) is accurate only to 2 places post decimal.

Answer 8)
Bias = $2^{(3-1)} - 1 = 3$

Number	Binary	Decimal
0	0 000 000	0.0
-0.125	1 000 000	-0.125
Smallest positive normalized number	0 001 000 (+ive no) (min value is 1) (all 0s)	0.25 = (1 * $2^{(1-3)}$)
largest positive normalized number	0 110 111 (+ive no) ($2^{(3)} - 2$) (all 1s)	15.0 = (1.111 * $2^{(6-3)}$)
Smallest positive denormalized number > 0	0 000 001 (+ive no) (denormalized=0) (1)	0.03125 = (0.001 * $2^{(-2)}$)
largest positive denormalized number > 0	0 000 111 (+ive no) (denormalized=0) (all 1s)	0.21875 = (0.111 * $2^{(-2)}$)

b) As stated above, smallest +ve no is 0 001 000 and largest +ve no is 0 110 111.

Assumption: c = -4.0 as 1 100 010 which negative FP no.

Case 1: a + (b + c)
The calculation would first take place for b + c to fetch 11.0. Subsequently, a would be added to this resulting in 11.25.

Case 2: (a + b) + c
The calculation would first take place for a + b to fetch 15 only given it is being represented by the 7 bit IEEE system and is the largest normalized no. which results in a get overlooked during calculation.
Subsequently, a would be added to this resulting in 11.00

Therefore, the two above stated methods resulted in **different** outputs and does not hold true for the 7 bit machine.