Take Home Review – Numeric Representation
"Fixed and Floating—What's the Point?"

---

1. Using the "true meaning of 2's complement" method,

using 8-bit word, 2's complement,

a) $0000\ 0011_2 = ?_{10}$

$= -(2^7)*0 + 2^1 + 2^0 = -128*0 + 2 + 1 = 3_{10}$

b) $1000\ 0011_2 = ?_{10}$

$= -(2^7)*1 + 2^1 + 2^0 = -128*1 + 2 + 1 = -125_{10}$

using 8-bit word, 2's complement, 3-bit precision,

c) $00010.101_2 = ?_{10}$

$= -(2^4)*0 + 2^1 + 2^{-1} + 2^{-3} = -16*0 + 2 + 0.5 + 0.125 = 2.625_{10}$

d) $10010.101_2 = ?_{10}$

$= -(2^4)*1 + 2^1 + 2^{-1} + 2^{-2} = -16*1 + 2 + 0.5 + 0.125 = -13.375_{10}$

2. Convert and calculate the following with fixed-point on an 8-bit word and 4-bit precision (all-positive),

a) What is the largest number that can be stored? (Answer in <u>binary</u> and <u>decimal</u>.)

in binary: $\underline{1111.1111}_2$
in decimal: $2^3+2^2+2^1+2^0 + 2^{-1}+2^{-2}+2^{-3}+2^{-4}$
$= 8+4+2+1+0.5+0.25+0.125+0.0625 = \underline{15.9375}_{10}$

b) $10.50_{10} = ?_2$

A) 10 / 2 = 5 + rem 0 : 0 (ls)
5 / 2 = 2 + rem 1 : 1
2 / 2 = 1 + rem 0 : 0
1 / 2 = 0 + rem 1 : 1 (ms) : $10_{10} = \underline{1010}_2$

B) .50 * 2 = 1.0 : 1 (ms)/(ls)
.0 * 2 = 0.0 : $0.5_{10} = .\underline{1}_2$

C) $10.50_{10} = \underline{1010.1000}_2$

c) $6.0625_{10} = ?_2$

A) 6 / 2 = 3 + rem 0 : 0 (ls)
3 / 2 = 1 + rem 1 : 1
1 / 2 = 0 + rem 1 : 1 (ms) : $7_{10} = \underline{0110}_2$

B) .0625 * 2 = 0.125 : 0 (ms)
.125 * 2 = 0.25 : 0
.25 * 2 = .50 : 0
.50 * 2 = 1.0 : 1 (ls)
.0 * 2 -= 0.0 : $0.0625_{10} = .\underline{0001}_2$

C) $7.0625_{10} = \underline{0110.0001}_2$

d) $1010.1010_2 = ?_{10}$

(straight): $2^3+2^1 + 2^{-1}+2^{-3} = 8+2+.50+.125 = \underline{10.625}_{10}$

f) $82_{16} = ?_{10}$

keeping in mind that hexadecimal is a short-cut/compressed form of binary,
= 8 2_{16}
= 1000 0010 => 1000.0010_2 (based on 8-bit word, 4-bit prec, all pos.)

$1000.0010_2 = 2^3 + 2^{-3} = 8+.125 = \underline{8.125}_{10}$

3. Convert and calculate the following with fixed-point on an 8-bit word, 2's complement, and 3-bit precision,

a) What are the *largest positive* and *largest negative* numbers? (Answer in binary and decimal.)

largest positive:
in binary: $\underline{01111.111}_2$
in decimal: $2^3+2^2+2^1+2^0 + 2^{-1}+2^{-2}+2^{-3}$
$= 8+4+2+1+0.5+0.25+0.125 = \underline{15.875}_{10}$

largest negative:
in binary: 10000.000_2 (by rule of numeric cycle; largest positive + .001)
in decimal: (using method in #9) $-(2^4) = \underline{-16.0}_{10}$

b) $-10.50_{10} = ?_2$

(from above: $10.50_{10} = \underline{01010.100}_2$)

-10.50_{10} : 1. flip: 10101.011
2. add 1: + .001
$\underline{10101.100}_2$

c) $13.375_{10} = ?_2$

A) 13 / 2 = 6 + rem 1 : 1 (ls)
6 / 2 = 3 + rem 0 : 0
3 / 2 = 1 + rem 1 : 1
1 / 2 = 0 + rem 1 : 1 (ms) : $13_{10} = \underline{01101}_2$

B) .375 * 2 = 0.75 : 0 (ms)
.75 * 2 = 1.5 : 1
.50 * 2 = 1.0 : 1 (ls)
.0 * 2 = 0.0 : $0.375_{10} = .\underline{011}_2$

C) $13.375_{10} = \underline{01101.011}_2$

d) $-13.375_{10} = ?_2$

(from previous: $13.375_{10} = \underline{01101.011}_2$)

-13.375_{10} : 1. flip: 10010.100
2. add 1: + .001
$\underline{10010.101}_2$

e) $10101.010_2 = ?_{10}$

(straight): $-(2^4)+2^2+2^0 + 2^{-2} = -16+4+1+.25 = \underline{-10.75}_{10}$

(long method): 1. flip: 01010.101
2. add 1: + .001
$01010.110_2 = 2^3+2^1 + 2^{-1}+2^{-3} = 8+2+.5+.25 = 10.75_{10}$
and the number is <u>originally negative</u>, so = $\underline{-10.75}_{10}$

f) $A2_{16} = ?_{10}$

keeping in mind that hexadecimal is a short-cut/compressed form (representation) of binary,

$= \quad A \quad 2_{16}$
$= \; 1010\ 0010 \Rightarrow 10100.010_2$ (based 8-bit word, 3-bit precision)

$10100.010_2 = -(2^4)+2^2 + 2^{-2} = -16+4 +.25 = \underline{-11.75}_{10}$

g) in binary, calculate the result of the value in c) – the value in e)

$13.375_{10} \Rightarrow \underline{01101.011 - 10101.010}_2 = ?_2$

Knowing the subtraction is performed by "adding a negative," the second value must be negated, yet this number is already negative ➔ this implies that the second value becomes positive.

Therefore,

$$\begin{aligned} 01101.011 - 10101.010_2 &= 01101.011 + (-10101.010)_2 \\ &= 01101.011 + \quad 01010.110\ _2 \\ &= 11000.001_2 \end{aligned}$$

The result is a negative value instead of a positive because of overflow.

4. Express the following decimal values in floating-point form with a 16-bit word, 7-bit exponent, and 8-bit mantissa,

a) 0.0_{10}

A) $0.0_{10} = 0.0_2$
B) normalise: $0.0_2 = 0.0_2 * 2^0$ (exponent = 0_{10})
C) exponent: $0_{10} = 0_2$
D) sign bit = 0 (positive), store: <u>0 0000000 00000000</u>

or just conclude: no sign bit, no exponent, no mantissa = <u>0 0000000 00000000</u>
(zero is a <u>special floating-point value</u> that is commonly just stored as: 0)

b) 1.0_{10}

A) $1.0_{10} = 1.0_2$
B) normalise: $1.0_2 = 0.1_2 * 2^1$ (exponent = 1_{10})
C) exponent: $1_{10} = 0000001_2$
D) sign bit = 0 (positive), store: <u>0 0000001 10000000</u>

c) -0.5_{10}

A) $0.5_{10} = 0.1_2$
B) normalise: $0.1_2 = 0.1_2 * 2^0$ (exponent = 0_{10})
C) exponent: $0_{10} = 0000000_2$
D) sign bit = 1 (negative), store: <u>1 0000000 10000000</u>

d) -5.62_{10}

A) $5.62_{10} = 101.1001111101_2$ (10 bits precision is good enough!)
B) normalise: $101.1001111101_2 = .1011001111101_2 * 2^3$ (exponent = 3_{10})
C) exponent: $3_{10} = 0000011_2$
D) sign bit = 1 (negative), store: <u>1 0000011 10110011</u>
(the stored floating-point value suffers from <u>underflow</u>.)

e) 1/64 $_{10}$

A) $1/64 = 2^{-6} = 0.000001_2$
B) normalise: $0.000001_2 = 0.1_2 * 2^{-5}$ (exponent = -5_{10})
C) exponent: -5_{10} = ($5_{10} = 0000101_2$; $-5_{10} = 1111011_2$) = 1111011
D) sign bit = 0 (positive), store: <u>0 1111011 10000000</u>

5. Express the following floating-point numbers in decimal (__ $_{10}$), 16-bit word, 7-bit exponent, and 8-bit mantissa,

a) 0|000 0000|1000 0000
A) sign bit: 0 – positive
B) exponent: $000\ 0000_2 = 0_{10}$
C) mantissa: $.1_2 * 2^0 = .1_2$
D) decimal : $(2^{-1}) = 0.5_{10}$

b) 0|000 0010|1111 0000
A) sign bit: 0 – positive
B) exponent: $000\ 0010_2 = 2_{10}$
C) mantissa: $.1111_2 * 2^2 = 11.11_2$
D) decimal : $(2^1+2^0+2^{-1}+2^{-2}) = 3.75_{10}$

c) 1|111 1111|1010 0000
A) sign bit: 1 – negative
B) exponent: $111\ 1111_2 = -1_{10}$
C) mantissa: $.101_2 * 2^{-1} = .0101_2$
D) decimal : $-(2^{-2}+2^{-4}) = -0.3125_{10}$

d) 1|000 0000|1001 0100
A) sign bit: 1 – negative
B) exponent: $000\ 0000_2 = 0_{10}$
C) mantissa: $.100101_2 * 2^0 = .100101_2$
D) decimal : $-(2^{-1}+2^{-4}+2^{-6}) = -0.578125_{10}$

e) 0|111 1111|1100 0000
A) sign bit: 0 – positive
B) exponent: $111\ 1111_2 = -1_{10}$
C) mantissa: $.11_2 * 2^{-1} = .011_2$
D) decimal : $(2^{-2}+2^{-3}) = 0.375_{10}$

f) What is the *largest number* and *smallest number* that can be stored in this FP format?

<u>largest</u>
A) sign bit: *0 or 1 (irrelevant when referring only to magnitude or prec.)*
B) exponent: $011\ 1111_2 = 63_{10}$
C) mantissa: $.11111111_2 * 2^{63}$
D) decimal : *left for discussion*

<u>smallest</u>
A) sign bit: *0 or 1 (irrelevant when referring only to magnitude or prec.)*
B) exponent: $100\ 0000_2 = -64_{10}$
C) mantissa: $.1_2 * 2^{-64}$
D) decimal : *left for discussion*

6. The following floating-point numbers are invalid. Indicate why.

a) `0|000 0010|0101 0100`
   - mantissa does not begin with 0.1; indicating the mantissa was not normalised correctly, or an error occurred in storing the bits

b) `1|000 0000|0000 0000`
   - exponent and mantissa describe zero, but sign bit indicates a negative; negative zero is not a valid FP number.

c) `0|000 0100|0000 0100`
   - same problem as a), mantissa is invalid.

d) `1|100 0000|0000 0001`
   - same problem as a) and c), mantissa is invalid
   (note: this FP number does not represent the smallest number; although the exponent is the most negative value possible exponent and the mantissa seems the smallest value, the mantissa is not normalised and the FP is invalid)