Assignment 5

Alex Clemmer
Student number: u0458675

Problem 1:

3.10.3: **a)** `0x456AE400 = 0100 0101 0110 1010 1101 0100 0000 0000`$_{two}$

Sign	Exponent	Fraction
0	`10001010`$_{two}$	`110101011010100000000000`$_{two}$
$(-1)^0$	`127 + 11 = 138`	`(1+) 0.835083`

$$\text{Value as a 32-bit float:}(-1)^0(1+0.8350830)2^11 \approx 3758.25 \tag{1}$$

b) `0xBE4CCCD = 1011 1110 0100 1100 1100 1100 1100 1101`$_{two}$

Sign	Exponent	Fraction
1	`01111100`$_{two}$	`10011001100110011001101`$_{two}$
$(-1)^1$	`127 - 3 = 124`	`(1+) 0.6`

$$\text{Value as a 32-bit float:}(-1)^1(1+0.6)2^{-3} = -0.2 \tag{2}$$

3.10.4: **a)** Convert 98.6 to 32-bit float

$$98.6 = 1100010.10011001100110011001101_{two} \tag{3}$$
$$= 1.10001010011001100110011001101_{two} * 2^6 \tag{4}$$

Sign	Exponent	Fraction
$(-1)^0$	`127 + 6 = 133`	
0	`10000101`$_{two}$	`1000101001100110011001`$_{two}$

In hex, our number is: `0x21629999`.
b) Convert -40 to 32-bit float

$$-40 = -101000_{two} \tag{5}$$
$$= -1.01_{two} * 2^6 \tag{6}$$

Sign	Exponent	Fraction
$(-1)^1$	`127 + 5 = 132`	
1	`10000100`$_{two}$	`01000000000000000000000`$_{two}$

In hex, our number is: `42200000`.

3.10.5: Transformation to 64-bit double is trivial:

a) Convert 98.6 to 64-bit float

Sign	Exponent	Fraction
$(-1)^0$	`1023 + 9 = 1029`	
0	$\texttt{10000000101}_{two}$	$\texttt{10001010011001100110011...}_{two}$

b) Convert -40 to 64-bit float

Sign	Exponent	Fraction
$(-1)^1$	`1023 + 5 = 1029`	
1	$\texttt{10000000100}_{two}$	$\texttt{0100000...}_{two}$

Problem 2:

Sign	Exponent	Fraction
$(-1)^0$	`124-127 = -3`	
0	$\texttt{011111100}_{two}$	$\texttt{1 .01010101010101010101010}_{two}$

First add the exponents:

$$-3 + 1 = -2 \tag{7}$$

Then multiply the numbers:

$$\begin{aligned} & 1.01010101010101010101010_{two} * 110 && (8)\\ = {} & 101010101010101010101010 + 1010101010101010101010100 && (9)\\ = {} & .1111111111111111111111111 && (10)\\ & && (11) \end{aligned}$$

The result is not 1 because the floating point does is not rounding, it's truncating. The ULP, or magnitude of the error is in the last position in the float, or a 2^{-23}.

Problem 3:

The guard bit is used to improve accuracy. It's an extra bit inside the float ALU or FPU (or whatever) that just keeps track of extra bits that get carried out.

The round bit is similar to the guard bit. The reason we need two of these is because of multiplication.

The sticky bit is used to navigate the nature of the round – i.e., should we be rounding up or down? It records if numbers were shifted off the end, and by knowing that, we can give the illusion of calculating to infinite precision and simply rounding off.

$$\begin{aligned} & 1.01010101010101010101010_{two} * 110 && (12)\\ = {} & 101010101010101010101010 + 1010101010101010101010100 && (13)\\ = {} & 1.0 && (14)\\ & && (15) \end{aligned}$$

Really the only thing that's different here is that the sticky bit will let us know to round up, and rounding up creates a chain reaction that rounds the whole thing up to 1. This time there are no ULPs and error.

Problem 4:

SEE ALSO LOGISIM SCREENSHOT.

```
a b | EO
T T | F
T F | T
F T | T
F F | F

(~A & B) + (A & ~B)
```

Problem 5:

```
a b c d | ODD
F F F F | T
F F F T | F
F F T F | F
F F T T | T
F T F F | F
F T F T | T
F T T F | T
F T T T | F
T F F F | F
T F F T | F
T F T F | T
T F T T | F
T T F F | T
T T F T | F
T T T F | F
T T T T | T

(~A & B) + (A & ~B)
```

Problem 6:

D = (A & B & C & D) + (A & B & C & D) + (A & B & C & D) + (A & B & C & D) + (A & B & C & D) + (A & B & C & D)

Problem 7:

SEE LOGISIM SCREENSHOT.