CS330 Architecture and Organization
Assignment Chapter 3

Your Name Here

Problem 1 — (8 points) Convert the following binary values to decimal.

(a) 0.101

(b) 110.101

(c) 1 0000 0000.0000 0000 1

(d) 111 1111.1111 111

Answer:

(a) $2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625$

(b) $2^2 + 2^1 + 2^{-1} + 2^{-3} = 4 + 2 + 0.5 + 0.125 = 6.625$

(c) $2^8 + 2^{-9} = 256.001953125$

(d) This can be done by hand, but a clever trick is to realize that the ones left of the binary point are one less than a power of 2. And the ones to the right of the binary point are the same number but shifted.
$(2^7 - 1) + (2^7 - 1) \cdot 2^{-7} = (2^7 - 1) + (1 - 2^{-7}) = 2^7 - 2^{-7} = 127.9921875$

Problem 2 — (8 points) Convert the following decimal values to binary.

(a) 0.75

(b) 16.0625

(c) 0.1

(d) 12.12

Answer:

(a) In base ten, $75/100 = 3/4 = 2^{-1} + 2^{-2}$, so in binary 0.11.

(b) In base ten, $16 + 625/10000 = 16 + 1/16 = 2^4 + 2^{-4}$, so in binary 1 0000.0001.

(c) This one is best solved by the standard algorithm, multiply by two until you get a number larger than one. Then subtract off the one and keep going.

$$\begin{aligned}
0.1 \cdot 2 = 0.2 &\rightarrow 0 \\
0.2 \cdot 2 = 0.4 &\rightarrow 0 \\
0.4 \cdot 2 = 0.8 &\rightarrow 0 \\
0.8 \cdot 2 = 1.6 &\rightarrow 1 \text{ and subtract one} \\
0.6 \cdot 2 = 1.2 &\rightarrow 1 \text{ and subtract one} \\
0.2 \cdot 2 = 0.4 &\rightarrow 0 \\
&\vdots
\end{aligned}$$

So, the binary value is $0.000110011\ldots$.

(d) The whole part is 12 which is easily seen to be $8 + 4$ or in binary 1100. The fraction part is best computed via the algorithm.

$$\begin{aligned}
0.12 \cdot 2 = 0.24 &\rightarrow 0 \\
0.24 \cdot 2 = 0.48 &\rightarrow 0 \\
0.48 \cdot 2 = 0.96 &\rightarrow 0 \\
0.96 \cdot 2 = 1.92 &\rightarrow 1 \text{ and subtract one} \\
0.92 \cdot 2 = 1.84 &\rightarrow 1 \text{ and subtract one} \\
0.84 \cdot 2 = 1.68 &\rightarrow 1 \text{ and subtract one} \\
0.68 \cdot 2 = 1.36 &\rightarrow 1 \text{ and subtract one} \\
0.36 \cdot 2 = 0.72 &\rightarrow 0 \\
0.72 \cdot 2 = 1.44 &\rightarrow 1 \text{ and subtract one} \\
0.44 \cdot 2 = 0.88 &\rightarrow 0 \\
0.88 \cdot 2 = 1.76 &\rightarrow 1 \text{ and subtract one} \\
0.76 \cdot 2 = 1.52 &\rightarrow 1 \text{ and subtract one} \\
0.52 \cdot 2 = 1.04 &\rightarrow 1 \text{ and subtract one} \\
0.04 \cdot 2 = 0.08 &\rightarrow 0 \\
&\vdots
\end{aligned}$$

This is rational, so we know it will eventually terminate or (more likely) repeat. I haven't calculated that far, but far enough at least to prove that I understand the method. So, $1100.00011110101110\ldots$.

Answer here.

Problem 3 — (8 points) Normalize the following fractional binary numbers. Give your answers in scientific notation, expressing the mantissa in binary and the exponents in decimal.

(a) 100.0

(b) 0.1110 0001

(c) 1100 1010.01

(d) 0.0001 01

Answer:

(a) 1.00×2^{2}

(b) $1.1100\ 001 \times 2^{-1}$

(c) $1.1001\ 0100\ 1 \times 2^{7}$

(d) 1.01×2^{-4}

In class we did all of our examples in 64-bit double precision floating point format, but for the sake of brevity, I would like you to look up the specifications for **single precision** floating point format (which has different sized fields and a different exponent bias). Use single precision for the following exercises.

Problem 4 — (8 points) Show how the following binary numbers would be represented in single precision floating point. In each number, the mantissa is given in binary and the exponent is given in decimal. Present your answers in binary.

(a) 1.01×2^0

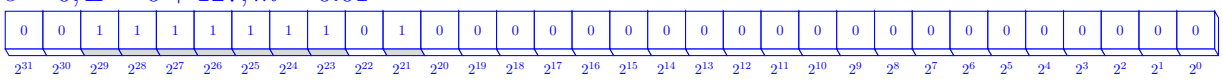
(b) $-1.1000\ 1100\ 0010 \times 2^{10}$

(c) $1.0100\ 1100\ 0010 \times 2^{-10}$

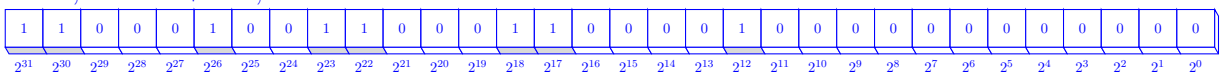
(d) $1.1101\ 0001\ 1 \times 2^{-127}$

Answer: All of these use 1 sign bit, 8 bits of exponent with a bias of 127, and 23 bits of mantissa (with an understood one, except in unnormalized numbers).

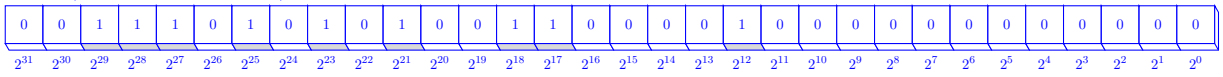
(a) $s = 0, E = 0 + 127, m = 0.01$



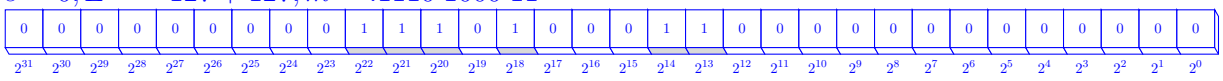
(b) $s = 1, E = 10 + 127, m = 0.1000\ 1100\ 0010$



(c) $s = 0, E = -10 + 127, m = 0.0100\ 1100\ 0010$



(d) In this case, the value of E is 0, the non-normalized exponent value where we no longer assume a leading digit of 1. So we will have to record all of the digits of the mantissa (including the digit left of the binary point).
$s = 0, E = -127 + 127, m = .1110\ 1000\ 11$



Problem 5 — (8 points) Convert the following IEE754 single precision bit patterns to decimal values, using regular or scientific notation as you believe appropriate.

(a) `0x BAC0 0000`

(b) `0x C0D0 0000`

(c) `0x F020 1000`

(d) `0x 7FFF FFFF`

(a) In binary this is 1011 1010 1100 0000 0000 0000 0000 0000, or separated into the floating point fields, 1 01110101 10000000000000000. The sign is negative, the exponent is $e = 117 - 127$ and the mantissa is 1.1. Thus, the value in decimal is $-1.5 \times 2^{-10} = -0.00146484375$.

(b) In binary this is 1100 0000 1101 0000 0000 0000 0000 0000, or separated into the floating point fields, 1 10000001 10100000000000000000000. The sign is negative, the exponent is $e = 129 - 127$ and the mantissa is 1.101. Thus, the value in decimal is $-1.625 \times 2^2 = -6.5$.

(c) In binary this is 1111 0000 0010 0000 0001 0000 0000 0000, or separated into the floating point fields, 0 11100000 01000000001000000000000. The sign is negative, the exponent is

$e = 224 - 127$ and the mantissa is 1.01000000001. Thus, the value in decimal is $-(1 + 2^{-2} + 2^{-11}) \times 2^{97} = -2^{97} - 2^{95} - 2^{86}$.

(d) +NaN, although technically the sign bit is not meaningful for not-a-number, so NaN is equally (or arguably more) correct.

Problem 6 — (5 points) The Python programming language has arbitrary integer arithmetic and double precision floating point arithmetic. Consider the following calculations:

```
>>> 2**53
9007199254740992

>>> float(2**53)
9007199254740992.0

>>> 1+2**53
9007199254740993

>>> float(1+2**53)
9007199254740992.0

>>> 3+2**53
9007199254740995

>>> float(1+2**53)
9007199254740996.0
```

Use your knowledge of floating point to explain what's going on. In your explanation, explain what happens for $5 + 2^{53}$ and $6 + 2^{53}$.

Answer: The operations that are not wrapped in `float()` are arbitrary precision integer calculations. They are all exactly correct. When we convert them to float, they are converted to double precision floating point numbers, which we know to have 52 stored mantissa bits.

These integers would all require 53 bits of mantissa to store. So, the conversion to floating point requires rounding the mantissa. And we learned in class that a common rounding method when there is a single extra bit (as there is here) is to round toward "even mantissa" values that end with a 0 bit.

So $1 + 2^{53}$ is rounded down to $0 + 2^{53}$ which is the closest mantissa ending in a 0 bit, while $3 + 2^{53}$ is rounded up to $4 + 2^{53}$ for the same reason.

Note: The value $2 + 2^{53}$ can be expressed exactly as a double precision floating point number, but its mantissa ends in a 1 bit, so it won't be the target of a rounding operation for the numbers above.

Similarly $4 + 2^{53}$ and $6 + 2^{53}$ can be expressed exactly as floating point numbers, the first ending in binary 0 and the latter ending in a 1 (so nothing will round to it). Thus, the value $5 + 2^{53}$ will be rounded down to $4 + 2^{53}$, and $6 + 2^{53}$ is represented exactly.