A2 - Assignment

John Akujobi

---

4) Determine the single-precision and double-precision machine representation of the following decimal numbers:

a. $1.0, -1.0$

Single Precision (32-bit)

For $+1.0$:

- **Normalized form:**
  1.0=1.0×201.0 = 1.0 \times 2^01.0=1.0×20
- **Sign bit:** 0
- **Exponent:** 0+127=1270+127=1270+127=127 →\to→ in 8-bit binary:
  01111111
- **Fraction:** Since the significand is exactly 1.0, the fractional part is all zeros (23 zeros).
- **Combined bit pattern:** $0 \mid 01111111 \mid 00000000000000000000000$
- **Hexadecimal:** `0x3F800000`

For -1.0:

- The only difference is the sign bit.
- **Sign bit:** 1
- **Exponent:** 127127127 (same as above)
- **Fraction:** All zeros
- **Combined bit pattern:** $1 \mid 01111111 \mid 00000000000000000000000$
- **Hexadecimal:** `0xBF800000`

Double Precision (64-bit)

For $+1.0$

- **Normalized form:**
  $1.0 = 1.0 \times 2^0$
- **Sign bit:** 0

- **Exponent:** $0 + 1023 = 1023$ in 11-bit binary: 01111111111
- **Fraction:** 52 zeros
- **Combined bit pattern:** $0 \mid 01111111111 \mid 0000000000000000000000000000000000000000000000000000$
- **Hexadecimal:** `0x3FF0000000000000`

For -1.0:

- **Sign bit:** 1
- **Exponent:** $0 + 1023 = 1023$ in 11-bit binary: 01111111111
- **Fraction:** 52 zeros
- **Combined bit pattern:** $1 \mid 01111111111 \mid 000000000000000000000000000000000000000000000000000$
- **Hexadecimal:** `0xBFF0000000000000`

---

b. $+0.0, -0.0$

I went and read about the IEEE-754 which governs the machine representation of number. I found out that it makes a distinction between $+0$ and -0 even though arithmetically they compare equal.

Single Precision (32-bit)

For $+0.0$:

- **Sign bit:** 0
- **Exponent:** All zeros: `00000000`
- **Fraction:** All zeros (23 bits)
- **Combined bit pattern:** $0 \mid 00000000 \mid 00000000000000000000000$
- **Hexadecimal:** `0x00000000`

For -0.0:

- **Sign bit:** 1
- **Exponent:** `00000000`
- **Fraction:** All zeros
- **Combined bit pattern:** $1 \mid 00000000 \mid 00000000000000000000000$
- **Hexadecimal:** `0x80000000`

Double Precision (64-bit)

For $+0.0$:

- **Sign bit:** 0
- **Exponent:** 11 zeros
- **Fraction:** 52 zeros
- **Combined bit pattern:**
  $0 \mid 00000000000 \mid 0000000000000000000000000000000000000000000000000000$
- **Hexadecimal:** `0x0000000000000000`

For -0.0:

- **Sign bit:** 1
- **Exponent:** 11 zeros
- **Fraction:** 52 zeros
- **Combined bit pattern:**
  $1 \mid 00000000000 \mid 0000000000000000000000000000000000000000000000000000$
- **Hexadecimal:** `0x8000000000000000`

---

c. -9876.54321

Converting the Number to binary

Converting the larger digits (64)

- $9876_{10} = 0010011010010100_2 = 1 * 2^{14}$

Converting the fraction to binary

- $0.54321 * 2^{10}10 \approx 556_{10} \approx 001000101100_2$
- The fraction conversion actually seems to go on forever and does not absolve to a perfect conversion
- So this is just an approximation

Binary

$1.00110100101001000101 10_2 \times 2^{13}$

Determine Exponent Field

	Single Precision	Double Precision
Sign	1	1
Exponent	13 + 127 = 140	14+ 1023 = 1036

	Single Precision	Double Precision
Exp	10001100	10000001100
Fraction	00110100101001000101100	0000 0001 0111 1100 0000 0000 0000 0000 0000 0011 0100 1010 0100 0101 1000 0111 1110 0111 1100 0000 0110 1110 0000 0000 0000 0000
Bin	0 10001100 00110100101001000101100	0 10000001100 0011 0100 1010 0100 0101 1000 0111 1110 0111 1100 0000 0110 1110
Hex	461A522C	40C34A4587E7C06E

Single Precision

Binary = `0 10001100 00110100101001000101100`
Hex = `461A522C`

Double Precision

Binary = `0 10000001100 0011 0100 1010 0100 0101 1000 0111 1110 0111 1100 0000 0110 1110`
Hex = `40C34A4587E7C06E`

---

f. 64.37109375

Converting the Number to binary

Converting the larger digits (64)

- $64_{10} = 01000000_2 = 1 * 2^6$

Converting the fraction to binary

- $0.37109375 = 2^{-8} + 2^{-7} + 2^{-6} + 2^{-5} + 2^{-4} + 2^{-3} + 2^{-3} = 0.01011111$
  or
- $0.37109375 * 2^8 = 95$
- $95_{10} == 01011111$
- So $0.37109375_{10} == 0.01011111_2 == 01011111_2 * 2^{-8}$

Binary

01000000.01011111

Table i used for conversion

0.5	0.1	2^{-1}
0.25	0.01	-2
0.125	0.001	-3
0.0625	0.0001	-4
0.03125	0.00001	-5
0.015 625	0.000001	-6
0.007 812 5	0.0000001	-7
0.003 906 25	0.00000001	-8
0.001 953 125	0.000000001	-9

Determine Exponent Field

	Single Precision	Double Precision
Sign	0	0
Exponent	6 + 127 = 133	6+ 1023 = 1029
Exp	10000101	10000000101
Fraction	00000001011111000000000	0000 0001 0111 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000
Bin	0 10000101 00000001011111000000000	0 10000000101 0000 0001 0111 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000
Hex	4280BE00	4050BE0000000000

Single Precision

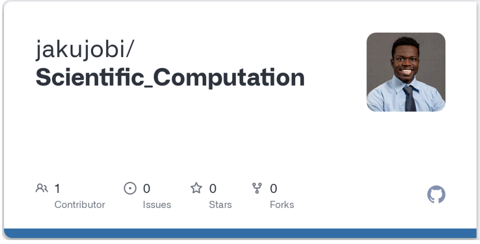
Binary = `0 10000101 00000001011111000000000`
Hex = `4280BE00`

Double Precision

Binary = `0 10000000101 0000 0001 0111 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000`
Hex = `405017C000000000`

Results from python script

Here is the link to my code



[Scientific_Computation/Assignment/A2 - Assignment/A2_...](github.com)

Contribute to jakujobi/Scientific_Computation development by creating an account on GitHub.

github.com

```
Value: +1.0 (1.0)
 Single Precision (32-bit):
  Binary:   00111111100000000000000000000000
  Hex:      0x3f800000
  Fields:   Sign = 0  Exponent = 127  Fraction = 0

 Double Precision (64-bit):
  Binary:   0011111111110000000000000000000000000000000000000000000000000000
  Hex:      0x3ff0000000000000
  Fields:   Sign = 0  Exponent = 1023  Fraction = 0

============================================================

Value: -1.0 (-1.0)
 Single Precision (32-bit):
  Binary:   10111111100000000000000000000000
  Hex:      0xbf800000
  Fields:   Sign = 1  Exponent = 127  Fraction = 0

 Double Precision (64-bit):
  Binary:   1011111111110000000000000000000000000000000000000000000000000000
  Hex:      0xbff0000000000000
  Fields:   Sign = 1  Exponent = 1023  Fraction = 0

============================================================

Value: +0.0 (0.0)
 Single Precision (32-bit):
  Binary:   00000000000000000000000000000000
  Hex:      0x0
  Fields:   Sign = 0  Exponent = 0  Fraction = 0

 Double Precision (64-bit):
  Binary:   0000000000000000000000000000000000000000000000000000000000000000
  Hex:      0x0
  Fields:   Sign = 0  Exponent = 0  Fraction = 0

============================================================

Value: -0.0 (-0.0)
 Single Precision (32-bit):
  Binary:   10000000000000000000000000000000
```

```
  Hex:      0x80000000
  Fields:   Sign = 1  Exponent = 0  Fraction = 0

 Double Precision (64-bit):
  Binary:   1000000000000000000000000000000000000000000000000000000000000000
  Hex:      0x8000000000000000
  Fields:   Sign = 1  Exponent = 0  Fraction = 0

============================================================

Value: -9876.54321 (-9876.54321)
 Single Precision (32-bit):
  Binary:   11000110000110100101001000101100
  Hex:      0xc61a522c
  Fields:   Sign = 1  Exponent = 140  Fraction = 1724972

 Double Precision (64-bit):
  Binary:   1100000011000011010010100100010110000111111001111100000001101110
  Hex:      0xc0c34a4587e7c06e
  Fields:   Sign = 1  Exponent = 1036  Fraction = 926087423443054

============================================================

Value: 64.37109375 (64.37109375)
 Single Precision (32-bit):
  Binary:   01000010100000001011111000000000
  Hex:      0x4280be00
  Fields:   Sign = 0  Exponent = 133  Fraction = 48640

 Double Precision (64-bit):
  Binary:   0100000001010000000101111100000000000000000000000000000000000000
  Hex:      0x405017c000000000
  Fields:   Sign = 0  Exponent = 1029  Fraction = 26113401159680

============================================================
```

---

16. Consider a computer that operates in base β and carries n digits in the mantissa of its floating-point number system. Show that the rounding of a real number x to the nearest machine number $\tilde{x}$ involves a relative error of at most $\frac{1}{2}\beta^{1-n}$

Hint: Imitate the argument in the text.

Floating-Point Representation

- According to our textbook, a normalized number is written as
  $x = d_0.d_1d_2\cdots d_{n-1} \times \beta^e$,
  where $d_{0\neq 0}$ and $1 \leq d0.d1\cdots dn-1 < \beta 1 \leq d_0.d_1\cdots d_{n-1} < \beta.$
- For a fixed exponent e, the machine numbers are evenly spaced by
  $\Delta = \beta^{e-n+1}$.

Rounding and Relative Error

Rounding Process:

Rounding x to the nearest machine number $\tilde{x}$ gives an absolute error of at most
$|x-\tilde{x}| \leq \frac{1}{2}\Delta = \frac{1}{2}\beta^{e-n+1}$.

Relative Error Bound:

Since x is normalized, $x = m \times \beta^e$ with m≥1.

- so, $|x| \geq \beta^e$.
  Dividing the absolute error by |x| yields:

$$\frac{|x-\tilde{x}|}{|x|} \leq \frac{\frac{1}{2}\beta^{e-n+1}}{\beta^e} = \frac{1}{2}\beta^{1-n}$$

This shows that rounding x to the nearest machine number introduces a relative error of at most
$\frac{1}{2}\beta^{1-n}$

Note

I wrote 2 scripts for these question, especially as Dr. Kimn values the skill to use programming to solve computational problems.
They are both in python, and i have saved the results from one of them in a text file.