✍1 Non associativity in the presence of round-off.

**Solution:** This is done in a class demo and the diary should be posted. Here are the commands.

```
n = 10000;
a = randn(n,1);  b = randn(n,1);  c = randn(n,1);
t = ((a+b)+c == a+(b+c));
sum(t)
```

Right-hand side in 3rd line returns 1 for each instance when the two numbers are the same. □

✍2 Find machine epsilon in matlab.

**Solution:**

```
u = 1;
```

```
for i=0:999
    fprintf(1,' i =  %d , u = %e \n',i,u)
    if (1.0 +u == 1.0) break, end
    u = u/2;
end
u = u*2
```

□

✍4 Proof of Lemma: If $|\delta_i| \leq \underline{\mathbf{u}}$ and $n\underline{\mathbf{u}} < 1$ then

$$\Pi_{i=1}^{n}(1+\delta_i) = 1+\theta_n \quad \text{where} \quad |\theta_n| \leq \frac{n\underline{\mathbf{u}}}{1-n\underline{\mathbf{u}}}$$

**Solution:**

The proof is by induction on $n$.

1) Basis of induction. When $n = 1$ then the product reduces to $1 + \delta_i$ and so we can take

$\boldsymbol{\theta_n} = \boldsymbol{\delta_n}$ and we know that $|\boldsymbol{\delta_n}| \le \underline{\mathbf{u}}$ from the assumptions and so

$$|\boldsymbol{\theta_n}| \le \underline{\mathbf{u}} \le \frac{\underline{\mathbf{u}}}{\mathbf{1} - \underline{\mathbf{u}}},$$

as desired.

2) Induction step. Assume now that the result as stated is true for $\boldsymbol{n}$ and consider a product with $\boldsymbol{n+1}$ terms: $\boldsymbol{\Pi_{i=1}^{n+1}(1+\delta_i)}$. We can write this as $\boldsymbol{(1+\delta_{n+1})\Pi_{i=1}^{n}(1+\delta_i)}$ and from the induction hypothesis we get:

$$\boldsymbol{\Pi_{i=1}^{n+1}(1+\delta_i) = (1+\theta_n)(1+\delta_{n+1}) = 1 + \theta_n + \delta_{n+1} + \theta_n\delta_{n+1}}$$

with $\boldsymbol{\theta_n}$ satisfying the inequality $\boldsymbol{\theta_n} \le (\boldsymbol{n}\underline{\mathbf{u}})/(\mathbf{1} - \boldsymbol{n}\underline{\mathbf{u}})$. We call $\boldsymbol{\theta_{n+1}}$ the quantity $\boldsymbol{\theta_{n+1} = \theta_n + \delta_{n+1} + \theta_n\delta_{n+1}}$, and we have

$$\begin{aligned}
|\boldsymbol{\theta_{n+1}}| &= |\boldsymbol{\theta_n + \delta_{n+1} + \theta_n\delta_{n+1}}| \\
&\le \frac{\boldsymbol{n}\underline{\mathbf{u}}}{\mathbf{1} - \boldsymbol{n}\underline{\mathbf{u}}} + \underline{\mathbf{u}} + \frac{\boldsymbol{n}\underline{\mathbf{u}}}{\mathbf{1} - \boldsymbol{n}\underline{\mathbf{u}}} \times \underline{\mathbf{u}} = \frac{\boldsymbol{n}\underline{\mathbf{u}} + \underline{\mathbf{u}}(\mathbf{1} - \boldsymbol{n}\underline{\mathbf{u}}) + \boldsymbol{n}\underline{\mathbf{u}}^{\mathbf{2}}}{\mathbf{1} - \boldsymbol{n}\underline{\mathbf{u}}} = \frac{(\boldsymbol{n}+\mathbf{1})\underline{\mathbf{u}}}{\mathbf{1} - \boldsymbol{n}\underline{\mathbf{u}}} \\
&\le \frac{(\boldsymbol{n}+\mathbf{1})\underline{\mathbf{u}}}{\mathbf{1} - (\boldsymbol{n}+\mathbf{1})\underline{\mathbf{u}}}
\end{aligned}$$

This establishes the result with $\boldsymbol{n}$ replaced by $\boldsymbol{n+1}$ as wanted and completes the proof. $\square$

## *Supplemental notes: Floating Point Arithmetic*

In most computing systems, real numbers are represented in two parts: A mantissa and an exponent. If the representation is in the base $\beta$ then:

$$x = \pm(.d_1 d_2 \cdots d_m)_\beta \beta^e$$

- $.d_1 d_2 \cdots d_m$ is a fraction in the base-$\beta$ representation
- $e$ is an integer - can be negative, positive or zero.
- Generally the form is normalized in that $d_1 \neq 0$.

*Example:* In base 10 (for illustration)

1. 1000.12345 can be written as

$$\mathbf{0.100012345_{10} \times 10^4}$$

2. 0.000812345 can be written as

$$\mathbf{0.812345_{10} \times 10^{-3}}$$

➤ Problem with floating point arithmetic: we have to live with limited precision.

***Example:*** Assume that we have only 5 digits of accuray in the mantissa and 2 digits for the exponent (excluding sign).

| | | | | | | |
|---|---|---|---|---|---|---|
| $.d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $e_1$ | $e_2$ |

Try to add 1000.2 = .10002e+03 and 1.07 = .10700e+01:

$$\mathbf{1000.2} = \boxed{.1\,|\,0\,|\,0\,|\,0\,|\,2\,\|\,0\,|\,4}\ ; \qquad \mathbf{1.07} = \boxed{.1\,|\,0\,|\,7\,|\,0\,|\,0\,\|\,0\,|\,1}$$

*First task:* align decimal points. The one with smallest exponent will be (internally) rewritten so its exponent matches the largest one:

$$\mathbf{1.07 = 0.000107\ \times 10^4}$$

*Second task:* add mantissas:

$$\begin{array}{lcccccc} & 0. & 1 & 0 & 0 & 0 & 2 & \\ + & 0. & 0 & 0 & 0 & 1 & 0 & 7 \\ \hline = & 0. & 1 & 0 & 0 & 1 & 2 & 7 \end{array}$$

## *Third task:*

round result. Result has 6 digits - can use only 5 so we can

- Chop result: | .1 | 0 | 0 | 1 | 2 | ;
- Round result: | .1 | 0 | 0 | 1 | 3 | ;

## *Fourth task:*

Normalize result if needed (not needed here)

result with rounding: | .1 | 0 | 0 | 1 | 3 || 0 | 4 | ;

✍5 Redo the same thing with $7000.2 + 4000.3$ or $6999.2 + 4000.3$.

## *Some More Examples*

➤ Each operation $fl(x \odot y)$ proceeds in 4 steps:

1. Line up exponents (for addition & subtraction).
2. Compute temporary exact answer.
3. Normalize temporary result.
4. Round to nearest representable number (round-to-even in case of a tie).

| | | | |
|---|---|---|---|
| | .40015 e+02 | .40010 e+02 | .41015 e−98 |
| + | .60010 e+02 | .50001 e−04 | −.41010 e−98 |
| temporary | 1.00025 e+02 | .4001050001e+02 | .00005 e−98 |
| normalize | .100025e+03 | .400105⊕ e+02 | .00050 e−99 |
| round | .10002 e+03 | .40011 e+02 | .00050 e−99 |
| note: | round to even | round to nearest ⊕=not all 0's | too small: unnormalized |
| | exactly halfway between values | closer to upper value | exponent is at minimum |

# *The IEEE standard*

*32 bit* (Single precision) :

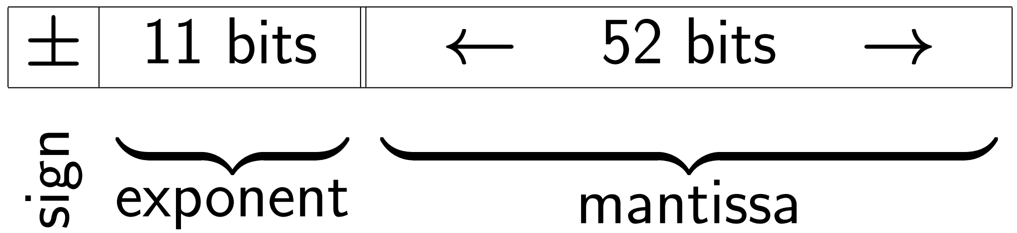
| $\pm$ | 8 bits | $\leftarrow$ 23 bits $\rightarrow$ |
|---|---|---|
| sign | exponent | mantissa |

➤ Number is scaled so it is in the form $1.d_1d_2...d_{23} \times 2^e$ - but leading one is not represented.

➤ $e$ is between -126 and 127.

➤ [Here is why: Internally, exponent $e$ is represented in "biased" form: what is stored is actually $c = e + 127$ – so the value $c$ of exponent field is between 1 and 254. The values $c = 0$ and $c = 255$ are for special cases (0 and $\infty$)]

## *64 bit* (Double precision) :




➤ Bias of 1023 so if $e$ is the actual exponent the content of the exponent field is $c = e + \mathbf{1023}$

➤ Largest exponent: $\mathbf{1023}$; Smallest = -1022.

➤ $c = \mathbf{0}$ and $c = \mathbf{2047}$ (all ones) are again for 0 and $\infty$

➤ Including the hidden bit, mantissa has total of 53 bits (52 bits represented, one hidden).

➤ In single precision, mantissa has total of 24 bits (23 bits represented, one hidden).

6 Take the number 1.0 and see what will happen if you add $1/2, 1/4, ...., 2^{-i}$. Do not forget the hidden bit!

Hidden bit (Not represented)

Expon. ↓ ← 52 bits →

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| e | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

.......

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| e | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| e | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(Note: The 'e' part has 12 bits and includes the sign)

- Conclusion

$fl(1 + 2^{-52}) \neq 1$ but: $fl(1 + 2^{-53}) == 1$ n+1

## *Special Values*

- Exponent field = 00000000000 (smallest possible value)
  No hidden bit. All bits == 0 means exactly zero.

- Allow for unnormalized numbers,
  leading to gradual underflow.

- Exponent field = 11111111111 (largest possible value)
  Number represented is "Inf" "-Inf" or "NaN".