Numerical Methods
Exercise Sheet 1

HS 16
M. Chrzaszcz
D. van Dyk

I. Bezshyiko, A. Pattori
`http://www.physik.uzh.ch/en/teaching/PHY233/HS2016.html`

Issued: 21.09.2016
Due: 28.09.2016 16:00

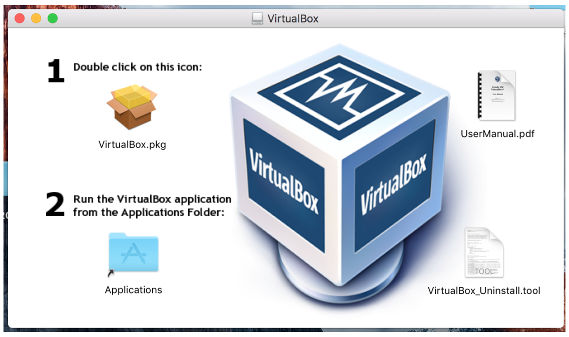
Exercise 0: Install the virtual machine (3 Pts.)

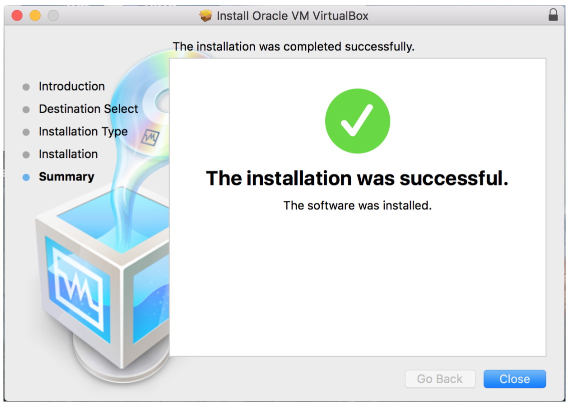
To install a virtual Linux machine on Linux, MacOS and Windows, follow these steps:

a) Follow this link `http://www.physik.uzh.ch/data/mchrzasz/Teaching/MC2016/`, download the zip file “VM_MC2016.zip" and unzip the folder.

b) Follow this link `https://www.virtualbox.org/wiki/Downloads` and download *virtualbox* (select “VirtualBox 5.1.6 for OS X hosts").

c) Install *virtualbox*, following the instructions of the installer.


VirtualBox
1 Double click on this icon:
VirtualBox.pkg
UserManual.pdf
2 Run the VirtualBox application from the Applications Folder:
Applications
VirtualBox_Uninstall.tool

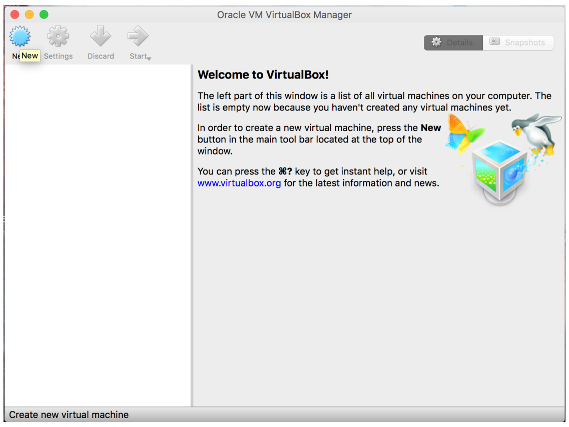

Install Oracle VM VirtualBox

The installation was completed successfully.

The installation was successful.

The software was installed.

d) Open *virtualbox* and press the “New" button to start the installation of a new virtual machine.



Welcome to VirtualBox!

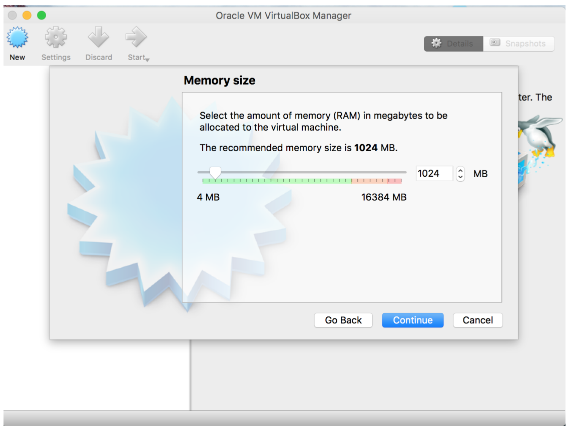
The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the ⌘? key to get instant help, or visit www.virtualbox.org for the latest information and news.

Create new virtual machine

e) Start the installation following the suggestions of the installer, then when required select “use an existing virtual hard disk file" and press "choose a virtual hard disk".

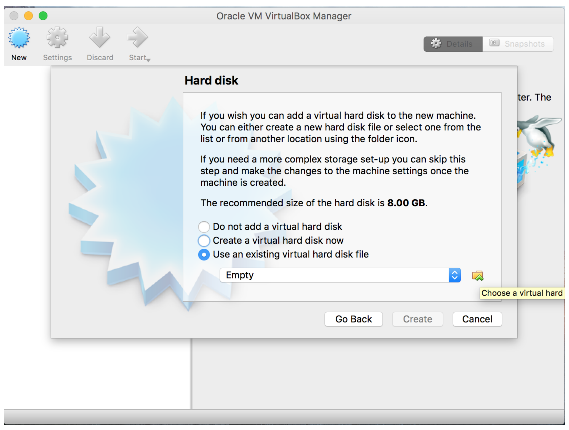

Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

1024 MB

4 MB 16384 MB


Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

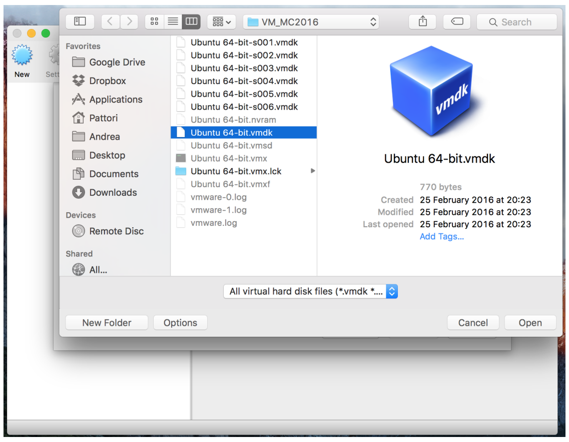
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

Do not add a virtual hard disk
Create a virtual hard disk now
[✓ Use an existing virtual hard disk file]

Empty

f) Browse to the “VM_MC2016" folder and select “Ubuntu 64-bit.vmdk". Then press “Create". Note: please keep nonetheless all the other images present in the folder.

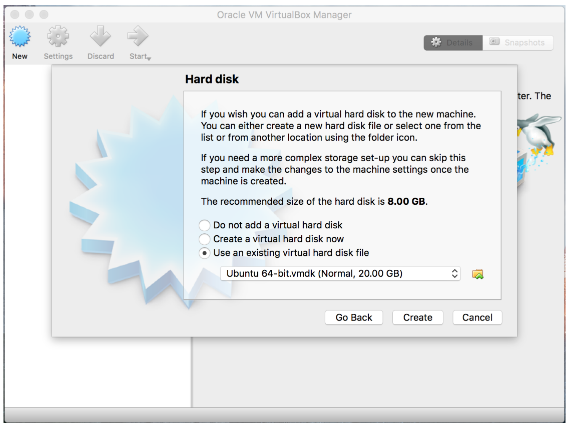

VM_MC2016

Ubuntu 64-bit-s001.vmdk
Ubuntu 64-bit-s002.vmdk
Ubuntu 64-bit-s003.vmdk
Ubuntu 64-bit-s004.vmdk
Ubuntu 64-bit-s005.vmdk
Ubuntu 64-bit-s006.vmdk
Ubuntu 64-bit.nvram
Ubuntu 64-bit.vmdk
Ubuntu 64-bit.vmsd
Ubuntu 64-bit.vmx
Ubuntu 64-bit.vmx.lck
Ubuntu 64-bit.vmxf
vmware-0.log
vmware-1.log
vmware.log

Ubuntu 64-bit.vmdk

770 bytes
Created 25 February 2016 at 20:23
Modified 25 February 2016 at 20:23
Last opened 25 February 2016 at 20:23

All virtual hard disk files (*.vmdk *....


Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

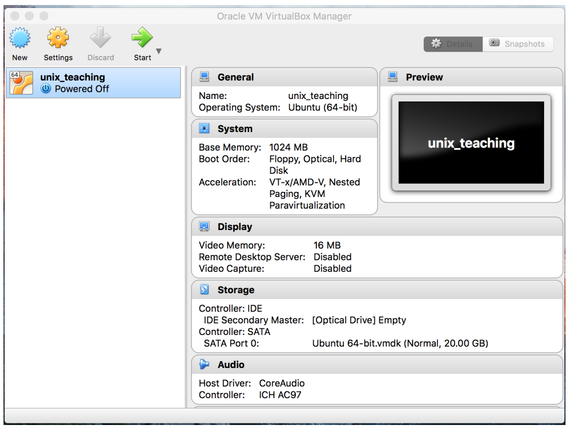
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

Do not add a virtual hard disk
Create a virtual hard disk now
[✓ Use an existing virtual hard disk file]

Ubuntu 64-bit.vmdk (Normal, 20.00 GB)

g) If the installation go smooth, you should end up with a correctly installed virtual machine present in *virtualbox* main menu.


unix_teaching
Powered Off

General
Name: unix_teaching
Operating System: Ubuntu (64-bit)

System
Base Memory: 1024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display
Video Memory: 16 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Ubuntu 64-bit.vmdk (Normal, 20.00 GB)

Audio
Host Driver: CoreAudio
Controller: ICH AC97

h) Once accessed to the virtual machine, these are the needed username and password:

username:	excellent student
password:	student

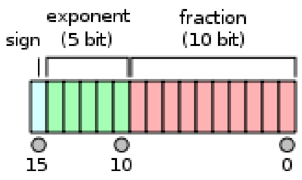
Exercise 1: Floating point representation (3 Pts.)

Let us label the numbers in decimal representation with a subscript '10', in binary representation with a subscript '2', in hexadecimal representation with a subscript '16'.

a) Convert the following integer decimal numbers in hexadecimal and binary representations: 12_{10}, 53_{10}, 123_{10}, 431_{10}. (0.75 Pt.)

b) Convert the following binary and hexadecimal integer numbers in decimal representation: 10011_2, 1101_2, $A2_{16}$, $1AD_{16}$. (0.75 Pt.)

c) In *binary16*, a number is represented using 16 digits in the following way:



In some detail:

- The first bit represent the sign: $0 \leftrightarrow +$ and $1 \leftrightarrow -$.
- The exponent is represented with 5 bits. Thus it can be an integer number from 0 to 31. The values 0 (i.e. 00000) and 31 (i.e. 11111) are reserved for special numbers (NaN, infinity, subnormal numbers). We are thus left with values from 1 to 30. By convention, the exponent of the floating number is the number represented by these 5 digits minus a bias, in this case 15 (example: if we want to store 2^{-3}, the exponent is -3 and thus we should store $-3 + 15 = 12_{10} = 01100_2$ in that slot). In this way we can store exponents from -14 (stored as $00001_2 = 1_{10}$) to $+15$ (stored as $11110_2 = 30_{10}$).
- The remaining 10 bits are left for the mantissa. By convention (with the exception of subnormal numbers), the first significant digit (the one before the dot) is always 1 and is not stored. Thus, if the mantissa stored is 1000000000, one should read it as $1.100_2 = 1.5_{10}$.

Try to convert into *binary16* the following decimal floating numbers: 0.3125_{10}, -431_{10}. (0.5 Pt.)

d) With respect to the results of the problem (*c*), convert to hexadecimal representation the two bytes representing your floating number and discuss the difference in the storage of your float between big endian and little endian. (0.5 Pt.)

e) Discuss in some detail why, using a machine working with *half precision* (i.e. *binary16*), one would get the following results:

```
INPUT A:
float a ,b, c;
a = 2050;
b = 1;
c = a+b;

OUTPUT A:
c = 2050;
```

```
INPUT B:
float a;
a = 2050;
for (int i=0; i<10; i++)
    a = a+1;

OUTPUT B:
a = 2050;
```

```
INPUT C:
float a, b, c;
a = 2050;
b = 0;
for (int i=0; i<10; i++)
    b = b+1;
c = a+b;

OUTPUT C:
c = 2060;
```

Write down the binary representation of relevant intermediary results. (0.5 Pts.)

SOLUTION:

a) The answers are:

Decimal:	Hexadecimal:	Binary:
12	C	1100
53	35	110101
123	7B	1111011
431	1AF	110101111

A simple algorithm to convert an integer N from decimal to hexadecimal is the following:

1. set $i = 1$.
2. Compute $h_i = N_{\text{mod}\,16}$, i.e. the remainder of the integer (Euclidean) division $N \div 16$.
3. Use the hexadecimal “dictionary" to convert h_i:

h:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
hex:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

4. Replace $N \to (N - h_i)/16$, set $i = i + 1$ and go back to step 2 until $N = 0$.
5. The number in hexadecimal representation is: $h_n h_{n-1} \cdots h_2 h_1$

This algorithm is actually valid for every base change.

Let us apply the algorithm above.

$N = 12$:

$$i = 1;$$
$$h_1 = 12_{\text{mod}\,16} = 12 \to \boxed{C};$$
$$N \to (N - 12)/16 = 0;$$
$$\text{end} : 12_{10} = \boxed{C_{16}}$$

$N = 53$:

$$i = 1$$
$$h_1 = 53_{\text{mod}\,16} = 5 \to \boxed{5};$$
$$N \to (N - 5)/16 = 3;$$

$$i = 2$$
$$h_2 = 3_{\text{mod}\,16} = 3 \to \boxed{3}$$
$$N \to (N - 3)/16 = 0;$$
$$\text{end} : 53_{10} = \boxed{35_{16}}$$

$N = 123$:

$$i = 1$$
$$h_1 = 123_{\text{mod}\,16} = 11 \to \boxed{B};$$
$$N \to (N - 11)/16 = 7;$$

$$i = 2$$
$$h_2 = 7_{\text{mod}\,16} = 7 \to \boxed{7}$$
$$N \to (N - 7)/16 = 0;$$
$$\text{end} : 123_{10} = \boxed{7B_{16}}$$

$N = 431$:

$i = 1$

$h_1 = 431_{\text{mod}\,16} = 15 \rightarrow \boxed{F}$;

$N \rightarrow (N-11)/16 = 26$;

$i = 2$

$h_2 = 26_{\text{mod}\,16} = 10 \rightarrow \boxed{A}$

$N \rightarrow (N-10)/16 = 1$;

$i = 3$

$h_3 = 1_{\text{mod}\,16} = 1 \rightarrow \boxed{1}$

$N \rightarrow (N-1)/16 = 0$;

end : $431_{10} = \boxed{1AF_{16}}$

The former algorithm can be applied also for the conversion in binary. One just need to change mod 16 → mod 2 in step 2 and to divide by 2 (not by 16) in step 4.

$N = 12$:

$i = 1$

$h_1 = 12_{\text{mod}\,2} = \boxed{0}$

$N \rightarrow (N-0)/2 = 6$;

$i = 2$

$h_2 = 6_{\text{mod}\,2} = \boxed{0}$

$N \rightarrow (N-0)/2 = 3$;

$i = 3$

$h_3 = 3_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 1$;

$i = 4$

$h_4 = 1_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 0$;

end : $12_{10} = \boxed{1100_2}$

$N = 53$:

$i = 1$

$h_1 = 53_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 26$;

$i = 2$

$h_2 = 26_{\text{mod}\,2} = \boxed{0}$

$N \rightarrow (N-0)/2 = 13$;

$i = 3$

$h_3 = 13_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 6$;

$i = 4$

$h_4 = 6_{\text{mod}\,2} = \boxed{0}$

$N \rightarrow (N-0)/2 = 3$;

$i = 5$

$h_5 = 3_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 1$;

$i = 6$

$h_6 = 1_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 0$;

end : $53_{10} = \boxed{110101_2}$

$N = 123$:

$i = 1$;

$h_1 = 123_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 61$;

$i = 2$

$h_2 = 61_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 30$;

$i = 3$

$h_3 = 30_{\text{mod}\,2} = \boxed{0}$

$N \rightarrow (N)/2 = 15$;

$i = 4$

$h_4 = 15_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 7$;

$i = 5$

$h_5 = 7_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 3$;

$i = 6$

$h_6 = 3_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 1$;

end : $53_{10} = \boxed{1111011_2}$

$i = 7$

$h_7 = 1_{\text{mod}\,2} = \boxed{1}$

$N \rightarrow (N-1)/2 = 0$;

$N = 431$:

$i = 1$

$h_1 = 431_{\mathrm{mod}\,2} = \boxed{1}$

$N \to (N-1)/2 = 215$;

$i = 2$

$h_2 = 215_{\mathrm{mod}\,2} = \boxed{1}$

$N \to (N-1)/2 = 107$;

$i = 3$

$h_3 = 107_{\mathrm{mod}\,2} = \boxed{1}$

$N \to (N)/2 = 53$;

$i = 4, 5, 6, 7, 8, 9$

$\ldots$ (see $N = 53$)

$\ldots$

end : $431_{10} = \boxed{110101111_2}$

b) One needs to understand that the decimal representation of an integer is just a short-cut notation. With 124_{10} one really means:

$$124_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 \ .$$

This is valid in <u>every</u> basis. Thus is easy to find:

$$\begin{aligned}
10011_2 &= (1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0)_{10} = (16 + 2 + 1)_{10} = 19_{10} \\
1101_2 &= (1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)_{10} = (8 + 4 + 1)_{10} = 13_{10} \\
A2_{16} &= (A \cdot 16^1 + 2 \cdot 16^0)_{10} = (10 \cdot 16 + 2)_{10} = 162_{10} \\
1AD_{16} &= (1 \cdot 16^2 + A \cdot 16^1 + D \cdot 16^0)_{10} = (1 \cdot 256 + 10 \cdot 16 + 13)_{10} = 429_{10}
\end{aligned}$$

c) Before converting a value into a floating number, one should convert it into a binary expression. We already know how to convert -431_{10}. To convert a non-integer number N into binary, one can use an extension of the algorithm used in exercise (1a).

1. i=0.
2. Take $\lfloor N \rfloor$ (i.e. integer part of N) and convert it into a binary expression (call it h_i).
3. Substitute $N \to 2(N - \lfloor N \rfloor)$.
4. Set $i = i+1$ and go back to step 2 until $N = 0$ or you have reached the desired precision.
5. Your number is $h_0.h_1 h_2 \ldots h_n$

Let us convert 0.3125 with the mentioned algorithm:

$i = 0$

$h_0 = \lfloor 0.3125 \rfloor = \boxed{0}$

$N \to 2(N - \lfloor N \rfloor) = 0.625$;

$i = 1$

$h_1 = \lfloor 0.625 \rfloor = \boxed{0}$

$N \to 2(N - \lfloor N) \rfloor) = 1.25$;

$i = 2$

$h_2 = \lfloor 1.25 \rfloor = \boxed{1}$

$N \to 2(N - \lfloor N) \rfloor) = 0.5$;

$i = 3$

$h_3 = \lfloor 0.5 \rfloor = \boxed{0}$

$N \to 2(N - \lfloor N \rfloor) = 1$;

$i = 4$

$h_1 = \lfloor 1 \rfloor = \boxed{1}$

$N \to 2(N - \lfloor N) \rfloor = 0$;

end : $0.3125_{10} = \boxed{0.0101_2}$

Thus $0.3125_{10} = 0.0101_2 = (1.01 \cdot 2^{-2})_2$. Thus:

- The sign is $+ \to 0$

- The exponent is -2. Exponent+bias: $-2+15=13=01101_2$.
- The mantissa is 1.010. The first digit (1.) is not stored, and we store 0100000000.

Thus we have:

$$0.3125_{10} \to \underbrace{0}_{\text{sign}}\ \underbrace{01101}_{\text{exp+bias}}\ \underbrace{0100000000}_{\text{mantissa}} \to \underbrace{00110101}_{\text{byte 1}}\ \underbrace{00000000}_{\text{byte 2}} \to \underbrace{35\ \ 00}_{\text{hexadecimal}}$$

For -431_{10} the exercise is similar. We have already computed $431_{10} = 110101111 = 1.10101111 \cdot 2^8$. Thus:

- The sign is $- \to 1$
- The exponent is 8. Exponent+bias: $8+15=23=10111_2$.
- The mantissa is 1.10101111. The first digit (1.) is not stored, and we store 1010111100.

Thus we have:

$$-431_{10} \to \underbrace{1}_{\text{sign}}\ \underbrace{10111}_{\text{exp+bias}}\ \underbrace{1010111100}_{\text{mantissa}} \to \underbrace{11011110}_{\text{byte 1}}\ \underbrace{10111100}_{\text{byte 2}} \to \underbrace{\text{DE}\ \ \text{BC}}_{\text{hexadecimal}}$$

d) We have already found the hexadecimal bytes representation for our floating numbers: 0.3125_{10} = 35 00 and -431_{10} = DE BC. In big endian, these bytes would be sequentially stored in the memory “from left to right", while in little endian the sequential storage would go in the other direction. Explicitly:

$$0.3125_{10} \to \text{big endian}: \underbrace{\boxed{35}}_{\text{slot i}}\ \underbrace{\boxed{00}}_{\text{slot i+1}} \qquad -431_{10} \to \text{big endian}: \underbrace{\boxed{\text{DE}}}_{\text{slot i}}\ \underbrace{\boxed{\text{BC}}}_{\text{slot i+1}}$$

$$\to \text{little endian}: \underbrace{\boxed{00}}_{\text{slot i}}\ \underbrace{\boxed{35}}_{\text{slot i+1}} \qquad \to \text{little endian}: \underbrace{\boxed{\text{BC}}}_{\text{slot i}}\ \underbrace{\boxed{\text{DE}}}_{\text{slot i+1}}$$

e) The number 2050 expressed in binary is 100000000010.

In *binary16* its representation is:

$$\underbrace{0}_{\text{sign}}\ \underbrace{11010}_{\text{exp+bias}}\ \underbrace{0000000001}_{\text{mantissa}} \quad \Rightarrow \quad +\,2^{26-15} \times 1.0000000001 = 1.0000000001 \times 2^{11} \tag{1}$$

From this representation is already clear that the *binary16* precision doesn't allow to store numbers like 2049 or 2051 at all, since this would require a 12 digit precision.

Thus, both in INPUT A and in INPUT B, the code is requiring the machine a precision it cannot reach. As it should be clear from how the number 2050 is stored, the machine has reached it maximal precision and it cannot store the number 2051 whatsoever. This means that at each step in which the number 2051 is met, the machine truncate it consistently with its maximum working precision, returning 2050.

INPUT C is well written and works fine. Differently from INPUT B, this time the machine works <u>at first</u> with numbers of order 1, which are manipulated without difficulties. For example, the *binary16* representation of 5 would be:

$$\underbrace{0}_{\text{sign}}\ \underbrace{10001}_{\text{exp+bias}}\ \underbrace{0100000000}_{\text{mantissa}-1} \quad \Rightarrow \quad +\,2^{17-15} \times 1.0100000000 = 1.01 \times 2^2 = 101_2 \tag{2}$$

where indeed $101_2 = 5_{10}$. After having dealt with all order-1 numbers, it finally sums them to 2050, getting 2060 which is a perfectly storable number.

In conclusion, one should be aware of the bad consequences of requiring the machine to work with more significant digits than its maximum.