Alunos: Luiz Rodrigo Lacé Rodrigues (DRE:11804983)
Livia Barbosa Fonseca (DRE:118039721)

Questão 1 -)

Luiz Rodrigo:
Estou utilizando o sistema operacional Ubuntu 20.04.02. Utilizei o comando “lsb_release -a“ para listar o sistema operacional.

```
luiz@Jarvis:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:        20.04
Codename:       focal
```

Utilizei o comando “lscpu” no terminal do linux para listar as configurações da cpu.
Temos então:

- Cache L1(dados): 128 KiB
- Cache L1(instrução): 256 KiB
- Cache L2: 2 MiB
- Cache L3: 4 MiB

```
luiz@Jarvis:~$ lscpu
Arquitetura:                     x86_64
Modo(s) operacional da CPU:      32-bit, 64-bit
Ordem dos bytes:                 Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          8
Lista de CPU(s) on-line:         0-7
Thread(s) per núcleo:            2
Núcleo(s) por soquete:           4
Soquete(s):                      1
Nó(s) de NUMA:                   1
ID de fornecedor:                AuthenticAMD
Família da CPU:                  23
Modelo:                          24
Nome do modelo:                  AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
Step:                            1
Frequency boost:                 enabled
CPU MHz:                         1396.679
CPU MHz máx.:                    2100,0000
CPU MHz mín.:                    1400,0000
BogoMIPS:                        4191.68
Virtualização:                   AMD-V
cache de L1d:                    128 KiB
cache de L1i:                    256 KiB
cache de L2:                     2 MiB
cache de L3:                     4 MiB
```

Livia Barbosa:
Estou utilizando o sistema operacional Ubuntu 20.04.2. Para descobrir as informações sobre o sistema operacional utilizei o comando “lsb_release -a”

```
livia@livia-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:        20.04
Codename:       focal
```

Foi utilizado o comando “lscpu” no terminal da máquina para listar as informações sobre os tamanhos dos caches. Onde foi encontrado:

- Cache L1 dados: 32 KiB
- Cache L1 instrução: 32 KiB
- Cache L2: 256 KiB
- Cache L3: 12 MiB

```
livia@livia-VirtualBox:~$ lscpu
Arquitetura:                     x86_64
Modo(s) operacional da CPU:      32-bit, 64-bit
Ordem dos bytes:                 Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          1
Lista de CPU(s) on-line:         0
Thread(s) per núcleo:            1
Núcleo(s) por soquete:           1
Soquete(s):                      1
Nó(s) de NUMA:                   1
ID de fornecedor:                GenuineIntel
Família da CPU:                  6
Modelo:                          165
Nome do modelo:                  Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
Step:                            2
CPU MHz:                         2592.002
BogoMIPS:                        5184.00
Fabricante do hipervisor:        KVM
Tipo de virtualização:           completo
cache de L1d:                    32 KiB
cache de L1i:                    32 KiB
cache de L2:                     256 KiB
cache de L3:                     12 MiB
CPU(s) de nó0 NUMA:              0
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
```

Questão 2 -)
Fizemos a correspondência pela tabela ASCII:

70 - p
75 - u
6E - n
C3 - ç
A7 -
C3 - ã
A3 -
6F - o
5F - _
C3 -
AA - ê
74 - t
61 - a
23 - #
64 - d
C3 - ó
B3 -
69 - i
24 - $
0A - “Newline / Line Feed”

Achamos que usuário digitou: punção_êta#dói$

Confirmamos o que ele escreveu utilizando o programa GHex, passando a palavra em arquivo texto e verificando seus caracteres em hexadecimal.

Questão 3 -)

Vamos completar o programa em C que imprime uma máscara (unsigned mask) tendo apenas o MSB (bit mais significativo) ligado e todos os outros bits em zero. Ou seja, o valor $(1000 \ldots 0000)_2$, em hexadecimal vamos ter 0x8000...0000.

Para isso, o raciocínio que tivemos, foi pegar o complemento do número 0x0 (tudo 1 em binário) = ~0x0 logo 0xFF...F

Com todos os bits ligados (0xFF...F) deslocamos logicamente em um bit para a direita. (0xFF...F>>1), o que nos resulta 0x7FF.... F

Após, complementamos (0x7FF...F) bit a bit e obtemos ~(0x7FF..F), o que resulta em (0x8000...0).

Visto que em binário temos 0111 = 0x7 e 1111 = 0xF

Assim o complemento de (0x7FF...F) nos retorna $(1000\ 0000 \ldots 0000)_2$

Para completar o programa em C utilizamos a expressão baseada no raciocínio acima. ~((~0x0UL)>>1), a notação “UL” é para referenciar o tipo da máscara que é unsigned long e assim fazer o deslocamento lógico de forma correta.

Finalmente o nosso programa ficou com essa cara:

home > luiz > Área de Trabalho > C mask.c > ...

```c
#include <stdio.h>


int main ( ) {
    unsigned long int mask;
    mask = ~(~(0x0UL) >> 1);
    printf ("máscara = 0x%lx \n", mask);
    return 0;
}

```

Depois compilamos para 32 e 64 bits e rodamos o seus executáveis.

```
luiz@Jarvis:~/Área de Trabalho$ gcc -m32 -o mask-32 mask.c
luiz@Jarvis:~/Área de Trabalho$ gcc  -o mask-64 mask.c
luiz@Jarvis:~/Área de Trabalho$ ./mask-32
máscara = 0x80000000
luiz@Jarvis:~/Área de Trabalho$ ./mask-64
máscara = 0x8000000000000000
```

Questão 4 -)

a) Vamos obter a maior magnitude real que pode ser representada com precisão dupla:

- Sinal: temos um número positivo, logo, s = <0>
- Expoente: o maior valor que pode ser representado em dupla precisão é 2046, então temos em binário: 1111 1111 110
- Parte fracionária: teremos 52 bits para a parte fracionária, logo, o maior número que podemos representar é:

  1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111

Agora vamos representar o número no formato <sinal><exp><frac>:
<0> <1111 1111 110> <1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111>

Vamos representar esse valor em decimal, sabendo que **$V = M \times 2^E$**
Como a mantissa é 1 + a parte fracionária <f>, então **$M = 2 - \epsilon$**, onde ε é o erro.

Temos também que **E = exp - 1023,** como nosso exp = 2046, logo **E = 1023**

Juntando todos esses valores, chegamos em $V = (2-\varepsilon) \times 2^{1023}$
Podemos achar o erro ε para o número no formato 1.111…. pela forma:

$\frac{2^{n+1}-1}{2^{n}} = 2\text{-}2^{-n} = 2 - \varepsilon$, sendo n o número de dígitos depois do ponto.

Logo $\varepsilon = 2^{-52}$ e assim, obtemos **$V = (2 - 2^{-52}) \times 2^{1023}$**

b) Vamos representar o menor número não normalizado e o menor número normalizado:

Para não normalizados:

- Sinal positivo, logo, s = <0>
- Expoente: Como é não normalizado temos exp = 0, significando que E = -1022
- Parte fracionária: Como queremos o menor número não-normalizado possível temos que a parte fracionária será:
  <0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001>

  Agora montando a representação em ponto flutuante de precisão dupla temos:
  $V = (-1)^S \times M \times 2^E$ = $((-1)^0 \times 2^{-1022} \times 2^{-52}) = 2^{-1074}$

Para normalizados:

- Sinal positivo, logo, s = <0>
- Expoente: Para normalizados temos que o mínimo é <0000 0000 001>
- Parte fracionária: Todos os bits são desligados
  <0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000>

  Para a representação em ponto flutuante de precisão dupla, temos:
  $V = (-1)^S \times M \times 2^E$ = $((-1)^0 \times 2^{-1022} \times 1) = 2^{-1022}$

Questão 5 -)

a) Vamos representar 0,3 x 2^{-131} em ponto flutuante de precisão simples na arquitetura de 32 bits:

Calculando 0,3 em binário:

$$(0,3)_{10} \times 2 = 0,6 \ \textbf{(0)}$$
$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$
$$(0,2)_{10} \times 2 = 0,4 \ \textbf{(0)}$$
$$(0,4)_{10} \times 2 = 0,8 \ \textbf{(0)}$$
$$(0,8)_{10} \times 2 = 1,6 \ \textbf{(1)}$$
$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$
$$(0,2)_{10} \times 2 = 0,4 \ \textbf{(0)}$$
$$(0,4)_{10} \times 2 = 0,8 \ \textbf{(0)}$$
$$(0,8)_{10} \times 2 = 1,6 \ \textbf{(1)}$$
$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$

.
.
.

Entramos em uma dízima

Logo, $(0,3)_{10}$ = $(0.0100\ 1100\ 1100\ 1100\ 1100\ldots)_2$

Como o expoente -131 é menor que -126, em precisão simples temos que usar a representação não normalizada, ajustando a mantissa:

$(0.0100\ 1100\ 1100\ 1100\ 1100\ldots.\ ) \ x\ 2^{-131}$ =

$(0.0100\ 1100\ 1100\ 1100\ 1100\ldots.\ ) \ x\ 2^{-126} \ x\ 2^{-5}$

$(0.0100\ 1100\ 1100\ 1100\ 1100\ldots.\ ) \ x\ 2^{-131}$ =

$(0.0\ 0000\ 0100\ 1100\ 1100\ 1100\ 1100\ldots.\ ) \ x\ 2^{-126}$

Colocando na representação, temos:

- Sinal: o número é positivo então s = 0
- Expoente: Como se trata de uma representação não normalizada, temos que exp = 0000 0000
- Parte fracionária (até 23 bits de mantissa) =

0.0 0000 0100 1100 1100 1100 1100....
= 000 0001 0011 0011 0011 0011

O número resultante será <sinal><exp><frac>:

<0> <0000 0000> <000 0001 0011 0011 0011 0011>
0000 0000 0000 0001 0011 0011 0011 0011

Em **hexadecimal**, obtemos : **0x 0 0 0 1 3 3 3 3**

b) Agora vamos representar o valor 0,3 x 2^{-131} de forma normalizada em precisão dupla.

Do exercício anterior obtemos que:
$(0,3)_{10}$ = $(0.0100\ 1100\ 1100\ 1100\ 1100\ldots)_2$
0,3 x 2^{-131} = $(0.0100\ 1100\ 1100\ 1100\ 1100\ldots.\ )\ x\ 2^{-131}$

$(0.0100\ 1100\ 1100\ 1100\ 1100\ldots.\ )\ x\ 2^{-131}$ = $(1.00\ 1100\ 1100\ 1100\ 1100\ldots.\ )\ x\ 2^{-133}$

Representando o valor temos:

- Sinal: o valor é positivo então s = 0
- Expoente: 1023 - E = 1023 - 133 = 890
  $(890)_{10} = (01101111010)_2$
- Parte fracionária (até 52 bits) = mantissa - 1=
-

$1.00\ 1100\ 1100\ 1100\ 1100110011001100110011001100\ 1100\ 1100 - 1 =$

0.00 1100 1100 1100 1100110011001100110011001100 1100 1100

Truncando o valor em 52 bits temos (removemos os dígitos em vermelho):
0.0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011

Representando o número, obtemos:

<0><01101111010><0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011>

0011 0111 1010 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011

Em **hexadecimal**, obtemos: **0x 3 7 A 3 3 3 3 3 3 3 3 3 3 3**

A dupla precisão permite fazer a representação normalizada do número, evitando a perda de precisão associada ao processo não normalizado.
Logo, a dupla precisão é importante para evitar erros que poderiam ocorrer na precisão simples que é limitada.

Questão 6 -)

a) Vamos converter 0,3 de decimal para binário fracionário, mostrando o passo a passo e obtendo uma quantidade de pelo menos 24 bits após a vírgula. Para isso, iremos multiplicando por 2 e encontrando os bits desejados (os bits encontrados estão ao lado das operações):

$$(0,3)_{10} \times 2 = 0,6 \ \textbf{(0)}$$
$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$
$$(0,2)_{10} \times 2 = 0,4 \ \textbf{(0)}$$
$$(0,4)_{10} \times 2 = 0,8 \ \textbf{(0)}$$
$$(0,8)_{10} \times 2 = 1,6 \ \textbf{(1)}$$
$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$
$$(0,2)_{10} \times 2 = 0,4 \ \textbf{(0)}$$
$$(0,4)_{10} \times 2 = 0,8 \ \textbf{(0)}$$
$$(0,8)_{10} \times 2 = 1,6 \ \textbf{(1)}$$
$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$
$$.$$
$$.$$
$$.$$

E assim entra em uma dízima periódica.
Logo 0,3 em binário é 0,0100 1100 1100 1100 1100 1100…

Sabendo que:

$(0100)_2$ = $(4)_{16}$

$(1100)_2$ = $(C)_{16}$

Representando $(0,0100\ 1100\ 1100\ 1100\ 1100\ 1100\ldots)_2$ em hexadecimal, chegamos em:

$(0,4CCCCC...)_{16}$

b) Vamos obter a representação normalizada de 4,3 em precisão simples.
Primeiramente, vamos transformar 4,3 de decimal para binário:

$$4,3 = 4 + 0,3$$
$$(4)_{10} = (0100)_2$$

E, do resultado da questão anterior temos que:

$$(0,3)_{10} = (0,0100\ 1100\ 1100\ 1100\ 1100\ 1100\ldots)_2$$

Assim:

$$(4,3)_{10} = (0100,0100\ 1100\ 1100\ 1100\ 1100\ 1100\ldots)_2$$

Sabendo que as representações serão sempre normalizadas no formato <s><exp><f>. Vamos inicialmente encontrar <f>.

Como a mantissa M deve ser 1 <= M < 2, temos que:

$$\mathbf{M} = ((2^{\ 2}) \times (1,000100\ 1100\ 1100\ 1100\ 1100\ 1100\ldots)_2$$

Logo, <f> = $(000100\ 1100\ 1100\ 1100\ 1100\ 1100\ldots)_2$

Fazendo o truncamento para 23 bits e arredondando, chegamos em:

$$\mathbf{<f> =} (0001\ 0011\ 0011\ 0011\ 0011\ 010\ )_2$$

Expoente: Temos que E = 2 e <exp> = 127 + E , logo <exp> = 129

$$(129)_{10} = (\mathbf{1000\ 0001})_2$$

Sinal: O número é positivo então temos que s = 0.

O número resultante vai ser:

<0**><**1000 0001**><**0001 0011 0011 0011 0011 010**>**

A representação desse número em hexadecimal fica :

$$(0100\ 0000\ 1000\ 1001\ 1001\ 1001\ 1001\ 1010)_2$$

=

(0 x 4 0 8 9 9 9 9 A)

c) Convertendo 0,2 para binário fracionário, mostrando o passo a passo e obtendo uma quantidade de bits de pelo menos 52 bits após a vírgula. Vamos multiplicando por 2 para obter os bits desejados (os bits encontrados em cada operação se encontram ao lado dentro dos parênteses):

$$(0,2)_{10} \times 2 = 0,4 \ \textbf{(0)}$$

$$(0,4)_{10} \times 2 = 0,8 \ \textbf{(0)}$$

$$(0,8)_{10} \times 2 = 1,6 \ \textbf{(1)}$$

$$(0,6)_{10} \times 2 = 1,2 \ \textbf{(1)}$$

$$(0,2)_{10} \times 2 = 0,4 \ \textbf{(0)}$$

.
.
.

Entramos em uma dízima

Logo,

$(0,2)_{10}$ =

$(0,0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011)_2$

Agora vamos representar este número em **hexadecimal**: 0x3333333333333

d) Agora vamos obter a representação normalizada de 3,2 em precisão dupla e arredondando ao truncar em 52 bits. Primeiramente, vamos transformar 3,2 de decimal para binário:

$$(3)_{10} = (0011)_2$$

Do exercício anterior achamos que:

$(0,2)_{10}$ = $(0,0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011...)_2$

Logo

$(3,2)_{10}$ = $(0011,0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011...)_2$

Vamos agora achar <f>:
Como a mantissa M deve ser 1 <= M < 2, temos que:

M = $((2^{1}) \times (1,10011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011...)_2)$

Fazendo o truncamento para 52 bits e arredondando, chegamos em:
<f> = 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001

Sinal:
O número em questão é positivo então temos que s = 0

Expoente:
Temos que E = 1 e <exp> = 1023 + E , logo <exp> = 1024

$$(1024)_{10} = (100\ 0000\ 0000)_2$$

O número resultante vai ser:
<0**><**100 0000 0000**><**1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001**>**

O número representado em hexadecimal fica:

$$(0100\ 0000\ 0000\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001)_2$$

=

(0x 4 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9)

e) Vamos obter a diferença x – z, estendo a mantissa de x para 52 bits e realizando a operação como se estivéssemos subtraindo dois valores representados em double.

- Mantissa de 4,3 (float) =

$$((2^{2}) \times (1,00\ 0100\ 1100\ 1100\ 1100\ 1100\ 1100\ldots)_2$$

Truncando para 23 bits temos: $((2^{2})\ x\ (1,000\ 1001\ 1001\ 1001\ 1001\ 1010)$

Completando com 0 para 52 bits:

$((2^{2})\ x\ (1,0001\ 0011\ 0011\ 0011\ 0011\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)$

Mantissa 3,2 (z) double:

$(2^{1})\ x\ (1,1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001)$

$(2^{2})\ x\ (0\ ,1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1101\ )$

- Fazendo então a subtração de 4,3 - 3,2 =

$1,0001\ 0011\ 0011\ 0011\ 0011\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$ **-**
$0,1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1101$

Passamos os dois números para hexadecimal para facilitar a subtração

1, 1 3 3 3 3 4 0 0 0 0 0 0 0
- 0, C C C C C C C C C C C D

0,4 6 6 6 6 7 3 3 3 3 3 3 3 ------> hexa

=

$(0,0100\ 0110\ 0110\ 0110\ 0110\ 0111\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011)_2$

Assim, o valor resultante da subtração das mantissas foi de:
$(0,0100\ 0110\ 0110\ 0110\ 0110\ 0111\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011)_2 \times 2^2$

Normalizando esse valor, temos que andar com a vírgula:
$(1,00\ 0110\ 0110\ 0110\ 0110\ 0111\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011) \times 2^0$

Vamos achar <s>, <f> e <exp> e representar o valor em precisão dupla:
- Sinal: O número é positivo então temos s = 0
- Expoente: exp = E + 1023 -> 0 + 1023 = 1023 = $(011\ 1111\ 1111)_2$
- Parte fracionária: f: 0001 1001 1001 1001 1001 1100 1100 1100 1100 1100 1100 1100 1100

Assim, temos a representação em double:
<0> <01111111111> <0001 1001 1001 1001 1001 1100 1100 1100 1100 1100 1100 1100 1100>

$(0011\ 1111\ 1111\ 0001\ 1001\ 1001\ 1001\ 1001\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100)_2$

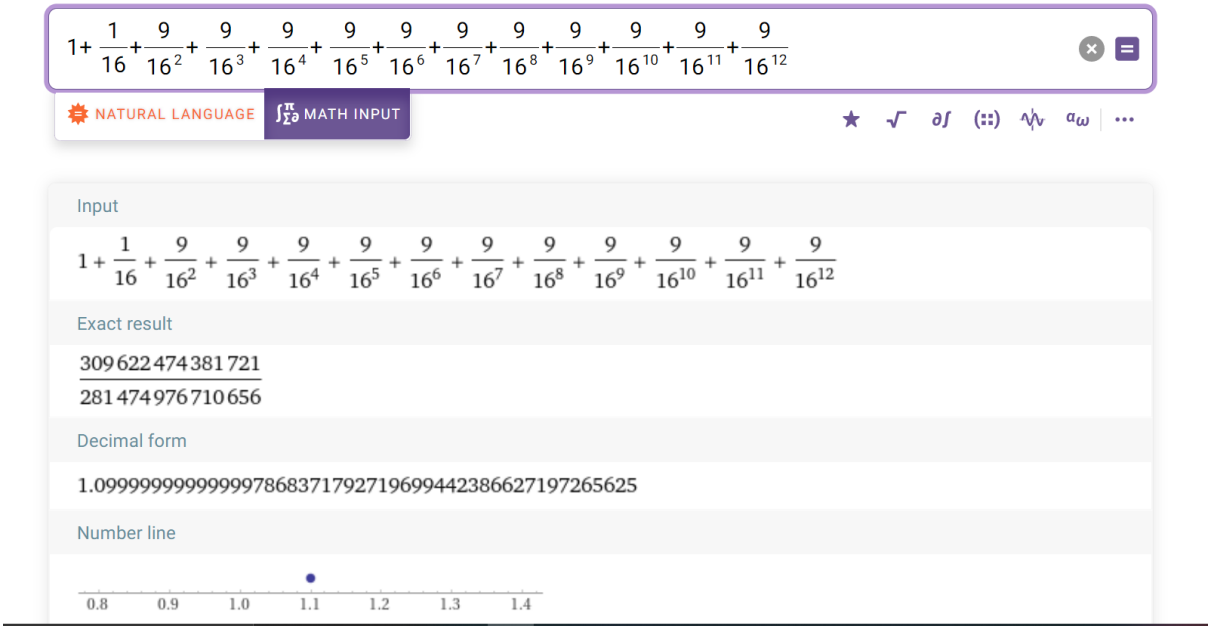
Obtendo a representação desse valor em hexadecimal temos:
0x3FF19999CCCCCCCC

f) Vamos calcular matematicamente o valor que será impresso pelo programa como uma soma de frações.

Valor em binário da subtração:
1. 0001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001

Primeiramente, para montar a soma de frações, agrupamos os dígitos do número em binário em quatro, onde o numerador de cada parcela da fração equivale ao valor número do grupo em decimal e o denominador equivale a sua casa decimal. Assim, obtemos a seguinte soma de PG:

$$1 + \frac{1}{16} + \frac{9}{16^2} + \frac{9}{16^3} + \frac{9}{16^4} + \frac{9}{16^5} + \frac{9}{16^6} \ldots$$



Temos então que a soma resulta em 1,0999999999999, o que converge para 1,1 dependendo da quantidade de frações que somarmos

```
luiz@Jarvis:~/Área de Trabalho$ gcc exee.c -Wall -o exee
luiz@Jarvis:~/Área de Trabalho$ ./exee
y = 4,3 - 3.2 = 1.1000001907349
```

Questão 7 -)

Fazendo as modificações para arredondar para $\frac{x}{2^k}$

Vamos deslocar 1 k vezes para a esquerda e depois diminuímos 1. Após somarmos nosso número em 1, vamos deslocá-lo novamente, mas dessa vez k vezes para a direita.

$$(x + (1 << k) - 1) >> k$$

Como exemplo, temos x = -4 e k =1:

$$(-4)_{10} = (1100)_2$$

$$(1100+(1<<1)-1) >> 1 = 1110$$

Podemos observar que a fórmula acima atende, visto que o valor obtido é igual ao valor calculado pela fórmula dada:

$$\frac{-4}{2^1} = -2$$