Lista 1 – Felipe Melo – Thalles Nonato

DRE Felipe: 119093752

DRE Thalles: 119058809

Questão 1)

Utilizamos o comando *lscpu --cache* para listar as informações do processador.

Processador de Felipe:

32 KB de cache L1 para dados

32 KB de cache L1 para instruções

256 KB de cache L2

6 MB de cache L3

Sistema Operacional: Ubuntu 20.04 64 bits

Processador de Thalles:

32 KB de cache L1 para dados

64 KB de cache L1 para instruções

512 KB de cache L2

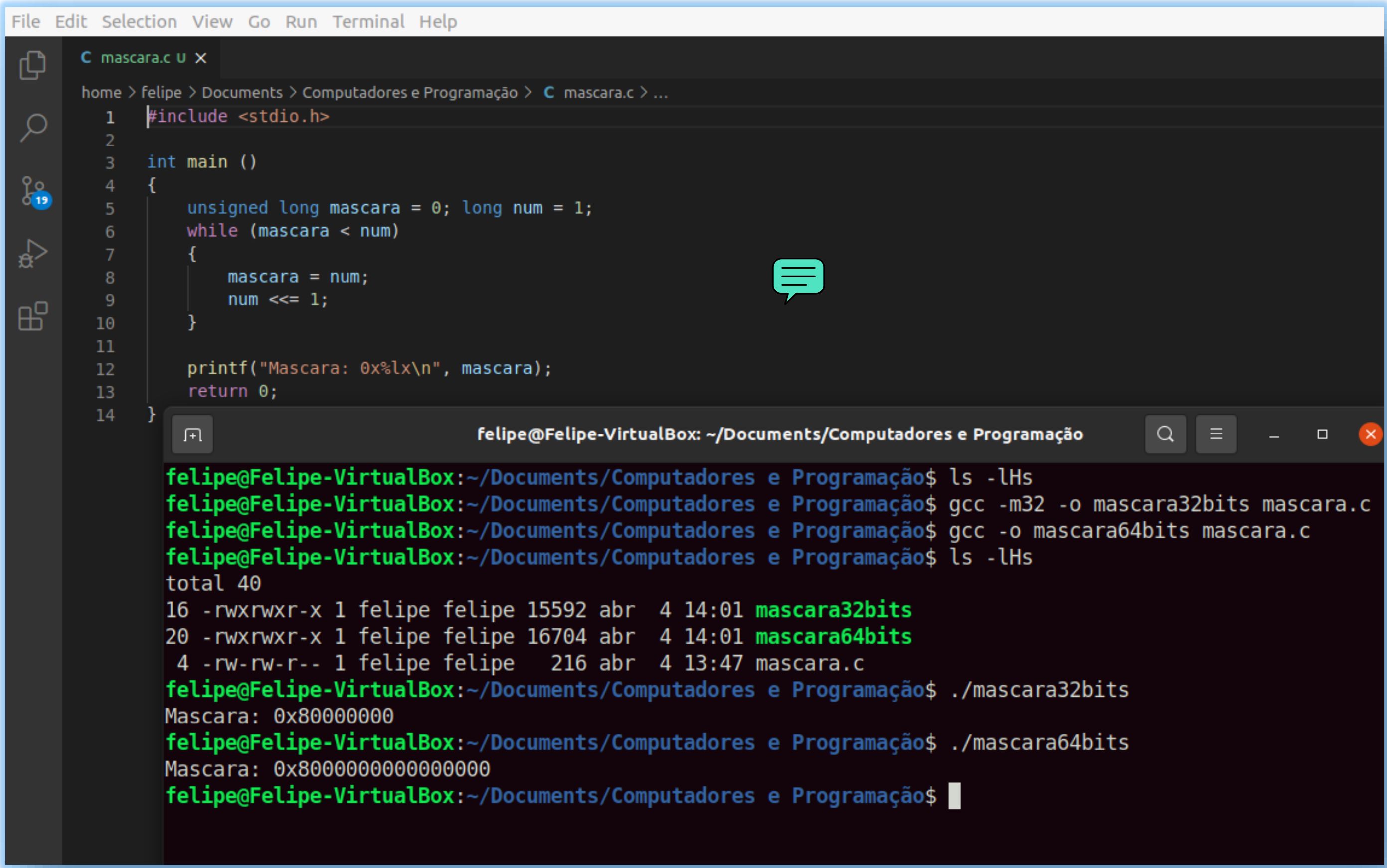
8 MB de cache L3

Sistema Operacional: Ubuntu 20.04 64 bits

```
felipe@Felipe-VirtualBox:~/Documents$ lscpu --cache
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL
L1d       32K      32K    8 Data            1
L1i       32K      32K    8 Instruction     1
L2       256K     256K    4 Unified         2
L3         6M       6M   12 Unified         3
felipe@Felipe-VirtualBox:~/Documents$
```

```
thalles@thalles-VirtualBox:~$ lscpu --cache
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL
L1d       32K      32K   16 Data            1
L1i       64K      64K    4 Instruction     1
L2       512K     512K    8 Unified         2
L3         8M       8M   16 Unified         3
thalles@thalles-VirtualBox:~$
```

Questão 2)



File Edit Selection View Go Run Terminal Help
mascara.c U
home > felipe > Documents > Computadores e Programação > mascara.c > ...
#include <stdio.h>
int main ()
{
unsigned long mascara = 0; long num = 1;
while (mascara < num)
{
mascara = num;
num <<= 1;
}
printf("Mascara: 0x%lx\n", mascara);
return 0;
}
felipe@Felipe-VirtualBox: ~/Documents/Computadores e Programação
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$ ls -lHs
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$ gcc -m32 -o mascara32bits mascara.c
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$ gcc -o mascara64bits mascara.c
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$ ls -lHs
total 40
16 -rwxrwxr-x 1 felipe felipe 15592 abr 4 14:01 mascara32bits
20 -rwxrwxr-x 1 felipe felipe 16704 abr 4 14:01 mascara64bits
4 -rw-rw-r-- 1 felipe felipe 216 abr 4 13:47 mascara.c
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$ ./mascara32bits
Mascara: 0x80000000
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$ ./mascara64bits
Mascara: 0x8000000000000000
felipe@Felipe-VirtualBox:~/Documents/Computadores e Programação$

Questão 3) a)

$0.26_{10} = X_2$ → $0.26 * 2 = 0.52$ → $0.52 * 2 = 1.04$

$0.04 * 2 = 0.08$ → $0.08 * 2 = 0.16$ → $0.16 * 2 = 0.32$

$0.32 * 2 = 0.64$ → $0.64 * 2 = 1.28$ → $0.28 * 2 = 0.56$

$0.56 * 2 = 1.12$ → $0.12 * 2 = 0.24$ → $0.24 * 2 = 0.48$

$0.48 * 2 = 0.96$ → $0.96 * 2 = 1.92$ → $0.92 * 2 = 1.84$

$0.84 * 2 = 1.68$ → $0.68 * 2 = 1.36$ → $0.36 * 2 = 0.72$

$0.72 * 2 = 1.44$ → $0.44 * 2 = 0.88$ → $0.88 * 2 = 1.76$

$0.76 * 2 = 1.52$ → $0.52 * 2 = 1.04$ → $0.04 * 2 = 0.08$

$0.08 * 2 = 0.16$

Portanto, temos: $0.26_{10} = 0.010000101000111101011100_2$

0,	0100	0010	1000	1111	0101	1100
0,	4	2	8	F	5	C

0X 0,428F5C

Questão 3) b)

5.26_{10} = 101. 010000101000111101011 (não normalizado)

Normalizado: 1.01010000101000111101011   * 2^2

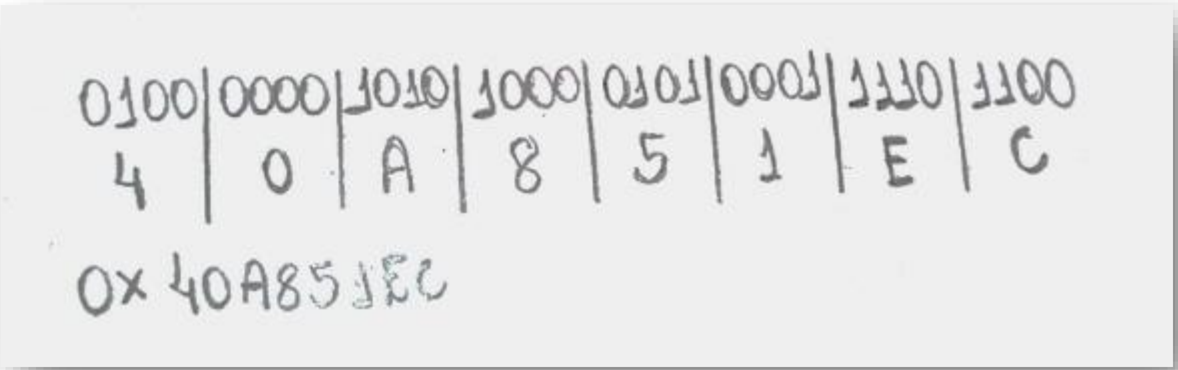
S = 0

Exp = 127 + 2 = 129_{10} = 10000001_2

Frac = 01010000101000111101011

Arredondando a parte fracionária: $01010000101000111101100_2$

Resposta: $01000000101010000101000111101100_2$



0100	0000	1010	1000	0101	0001	1110	1100
4	0	A	8	5	1	E	C

0x40A851EC

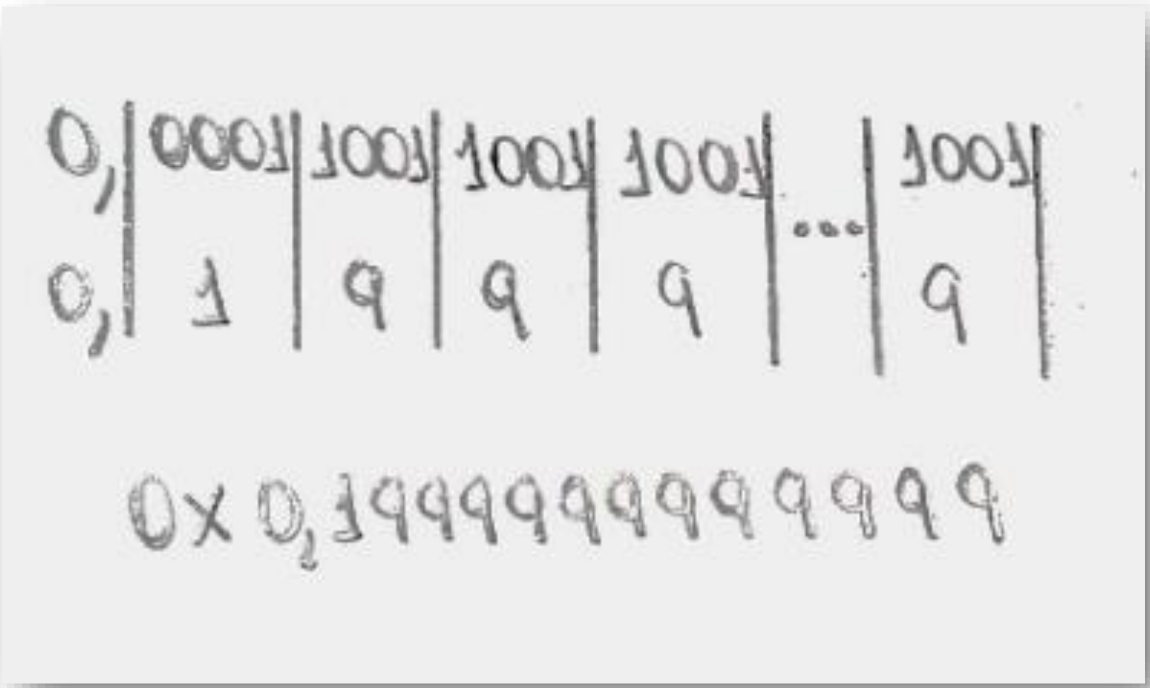
Questão 3) c)

0.1_{10} para binário

0.1 * 2 = 0.2	→	0.2 * 2 = 0.4	→	0.4 * 2 = 0.8
0.8 * 2 = 1.6	→	0.6 * 2 = 1.2	→	0.2 * 2 = 0.4
0.4 * 2 = 0.8	→	0.8 * 2 = 1.6	→	0.6 * 2 = 1.2

A partir daí, o número começa a se repetir em dízima periódica [0011]. Sendo assim:

$0.1_{10} = 0.0001100110011001100110011001100110011001100110011001_2$



Questão 3) d)

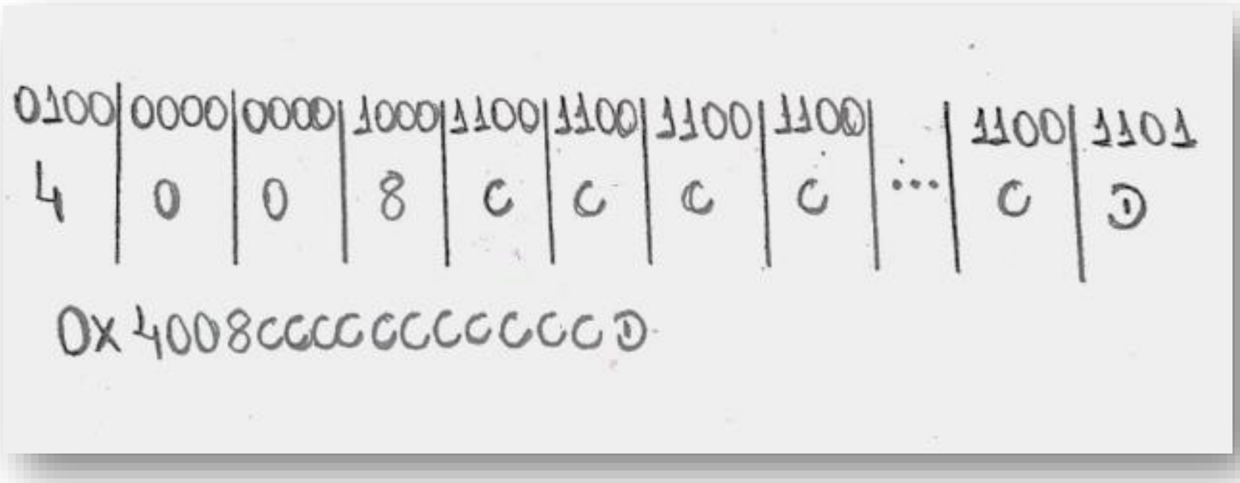
$3.1_{10} = 11.0001100110011001100110011001100110011001100110011001_2$

$3.1_{10} = 1.1000110011001100110011001100110011001100110011001101_2 * 2_{10}$

$Exp = 1 + 1023 = 1024_{10} = 10000000000_2$

Frac = 1000110011001100110011001100110011001100110011001101

Resposta: $0100000000001000110011001100110011001100110011001100110011001101_2$



Questão 3) e)

$1.01010000101000111101100_2 * 2^2$

$1.1000110011001100110011001100110011001100110011001101_2 * 2^1$,

que é igual a $0.11000110011001100110011001100110011001100110011001101_2 * 2^2$

Precisamos agora igualar os expoentes:

$$
\begin{array}{rl}
 & 1.0101000010100011110110000000000000000000000000000000_2 \\
- & 0.1100011001100110011001100110011001100110011001100111_2
\end{array}
$$

Ao transformar o subtraendo em complemento a dois:

$$
\begin{array}{rl}
 & 1.0101000010100011110110000000000000000000000000000000_2 \\
+ & 1.0011100110011001100110011001100110011001100110011001_2 \\
\hline
 & 0.1000101000111101011100011001100110011001100110011001_2 * 2^2
\end{array}
$$

Assim: $1.000101000111101011100011001100110011001100110011001 * 2^1$

S = 1

Exp = 1023 + 1 = 1024_{10} = 10000000000_2

Frac = $0001010001111010111000110011001100110011001100110010_2$

Questão 3) f)

$1.0001010001111010111000110011001100110011001100110010_2 * 2^1$

Truncando a parte após a vírgula para 23 bits, teremos:

$1.00010100011110101110001_2$, que, arredondado, nos dá $1.00010100011110101110010_2$

Agora, incrementando os 29 bits para chegar a 52 bits, temos:

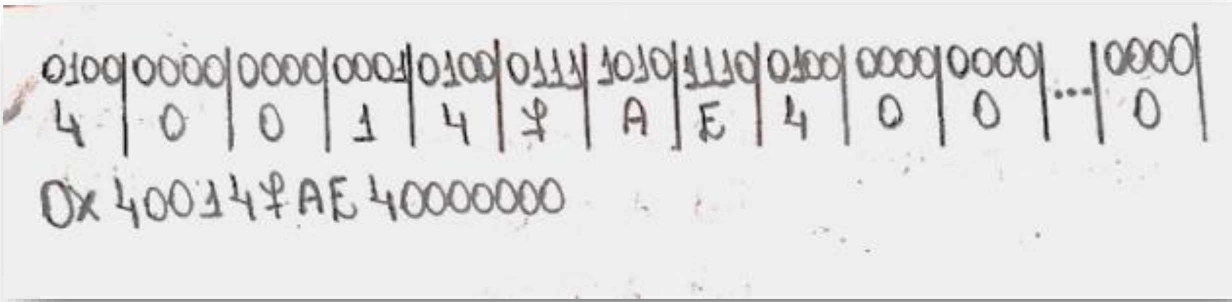
$1.0001010001111010111001000000000000000000000000000000_2 * 2^1$

Sinal = 0

Exp = 1023 + 1 = 1024_{10} = 10000000000_2

Frac = 0001010001111010111001000000000000000000000000000000

Resposta: $0100000000000001010001111010111001000000000000000000000000000000_2$



Além disso, para representar a parte alta, devemos considerar o byte order do processador.

Em Little Endian, teremos impressos pelo *printf*:

Parte alta: 0x40000000

Parte baixa: 0x400147AE

Já em Big Endian, a representação é:

Parte alta: 0x400147AE

Parte baixa: 0x40000000

Questão 3) g)

0x40014fAE   0x40000000

0b 0100 0000 0000 0000 0001 0100 0111 1010 1110 0100 0000...

$S = 0$

$exp = 100\ 0000\ 0000$

$frac = 1{,}0001\ 0100\ 0111\ 1010\ 1110\ 0100\ 0000000\ldots \cdot 2^1$

$$\hookrightarrow \frac{1}{16} + \frac{1}{64} + \frac{1}{1024} + \frac{1}{2048} + \frac{1}{4096} + \frac{1}{8192} + \frac{1}{32768} + \frac{1}{131072} + \frac{1}{262144} + \frac{1}{524288} + \frac{1}{4194304} = \frac{335545}{4194304}$$

$$1 + \frac{335545}{4194304} = \frac{4529849}{4194304}$$

Multiplicando por 2, chegamos a:

$$\frac{4529849}{4194304} \cdot 2 = \frac{4529849}{2097152} = \boxed{2{,}16000032424492}$$

Questão 4) a)

$0.3 * 2^{-136}$

$0.3 * 2 = 0.6$ → $0.6 * 2 = 1.2$ → $0.2 * 2 = 0.4$ → $0.4 * 2 = 0.8$

$0.8 * 2 = 1.6$ → $0.6 * 2 = 1.2$ → $0.2 * 2 = 0.4$ → $0.4 * 2 = 0.8$

Portanto, $0.3_{10} = 0.01001100110011001100110011001100_2$ com 32 bits após a vírgula

Então temos:

$0.01001100110011001100110011001100_2 * 2^{-136}$

$= 1.001100110011001100110011001100_2 * 2^{-2} * 2^{-136}$

$= 1.001100110011001100110011001100_2 * 2^{-138}$

$= 1.001100110011001100110011001100_2 * 2^{-12} * 2^{-126}$

$= 1.000000000000001100110011001100110011001100_2 * 2^{-126}$

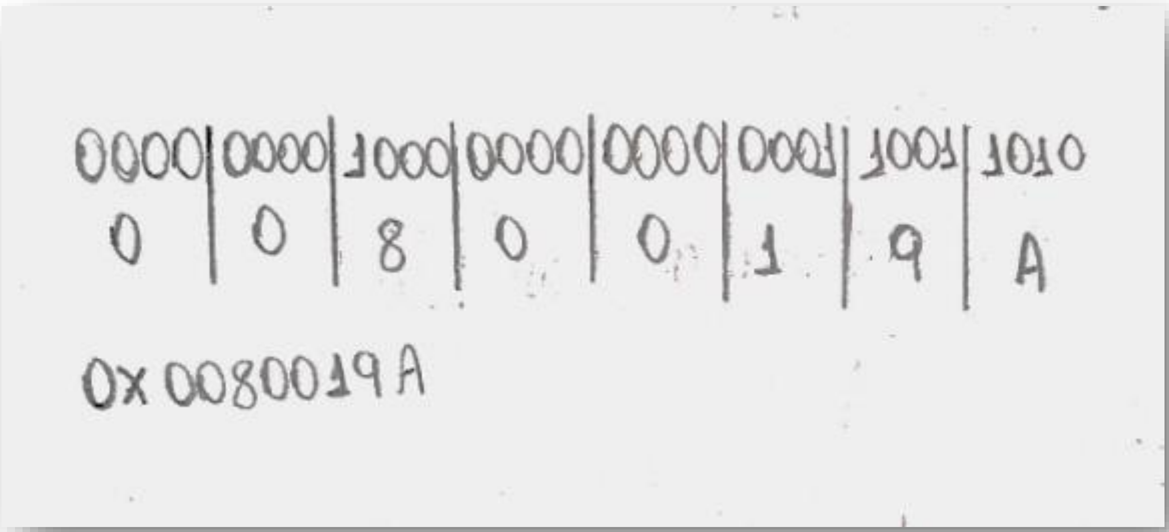
Sinal = 0

Exp = 127 – 126 = 1_{10} = 00000001_2

Frac = $000000000000001100110011001100110011001100_2$ (42 bits)

Frac = $00000000000000110011010_2$ (23 bits)

Resposta: $00000000100000000000000110011010_2$


0000 | 0000 | 1000 | 0000 | 0000 | 0001 | 1001 | 1010
0 | 0 | 8 | 0 | 0 | 1 | 9 | A

0x 0080019A

Questão 4) b)

$0.3 * 2^{-136}$

$0.3_{10} = 0.0100110011001100110011001100110011001100110011001100110011001100_2$ (64 bits pós vírgula)

$= 1.00110011001100110011001100110011001100110011001100110011001100_2 * 2^{-2} * 2^{-136}$

$= 1.00110011001100110011001100110011001100110011001100110011001100_2 * 2^{-138}$

Sinal = 0

Exp = 1023 – 138 = $885_{10} = 01101110101_2$

Frac = 00110011001100110011001100110011001100110011001100110011001100 (62 bits)

Frac = 0011001100110011001100110011001100110011001100110011 (52 bits)

Resposta: $0011011101010011001100110011001100110011001100110011001100110011_2$

0011	0111	0101	0011	0011	0011	0011	0011		0011
3	7	5	3	3	3	3	3	...	3

0x3753333333333333

Questão 5) a)

Para encontrarmos a **maior** magnitude real que pode ser representada em precisão dupla, devemos utilizar todos os campos do expoente, assim como todos os campos da mantissa. Dessa forma, teremos:

$2^{1023} * (2^{1} – 2^{-52})$, em que 2^{1} é o bit “omitido” após normalizarmos

A partir daí, aplicando a propriedade distributiva, teremos:

$2^{1024} – 2^{971}$

Questão 5) b)

Agora, para encontrarmos a **menor** magnitude real, devemos utilizar o mínimo de campos de expoente e de parte fracionária. Assim:

$1.00000000...01_2$ → 2^{-1022} (expoente) * 2^{-52} (mantissa) → 2^{-1074}