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:

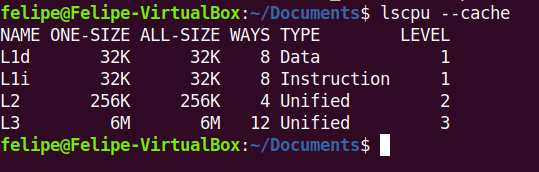
32 KB de cache L1 para dados

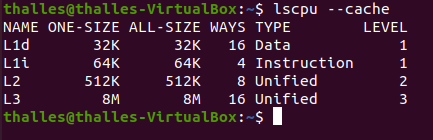
64 KB de cache L1 para instruções

512 KB de cache L2

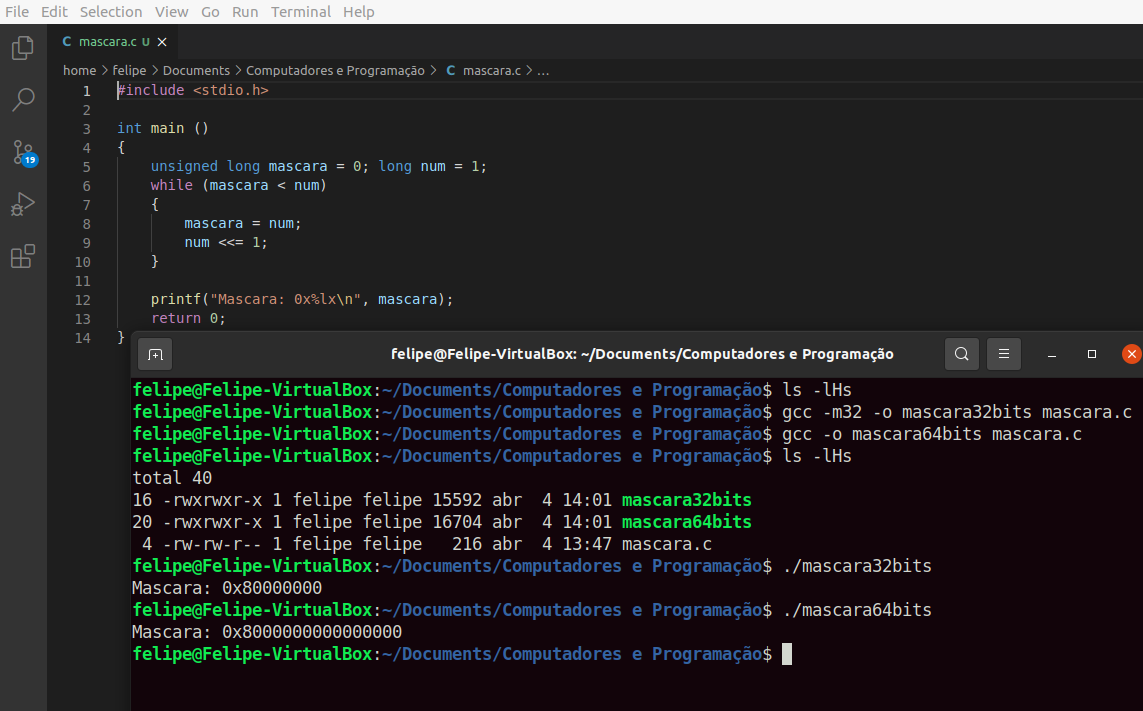
8 MB de cache L3

Sistema Operacional: Ubuntu 20.04 64 bits





**Questão 2)**



**Questão 3) a)**

0.2610 = X2 🡪 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

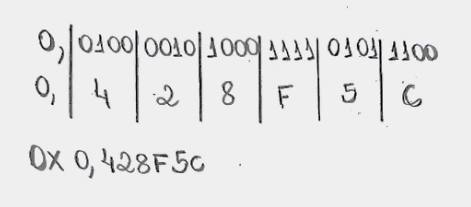
0.84 \* 2 = 1.68 🡪 0.68 \* 2 = 1.36 🡪 0.36 \* 2 = 0.72

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: 0.2610 = 0.0100001010001111010111002



**Questão 3) b)**

5.2610 = 101. 010000101000111101011 (não normalizado)

Normalizado: 1.01010000101000111101011 \* 22

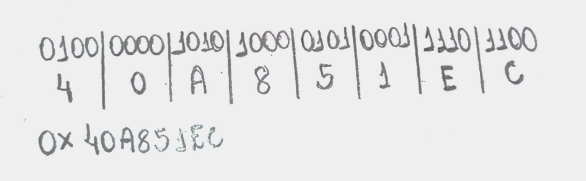
S = 0

Exp = 127 + 2 = 12910 = 100000012

Frac = 01010000101000111101011

Arredondando a parte fracionária: 010100001010001111011002

Resposta: 010000001010100001010001111011002



**Questão 3) c)**

0.110 para binário

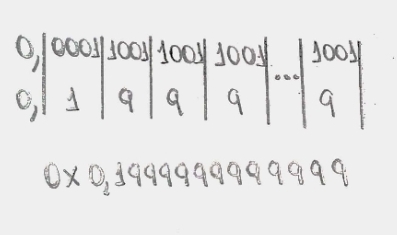
0.1 \* 2 = 0.2 🡪 0.2 \* 2 = 0.4 🡪 0.4 \* 2 = 0.8

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:

0.110 = 0.00011001100110011001100110011001100110011001100110012



**Questão 3) d)**

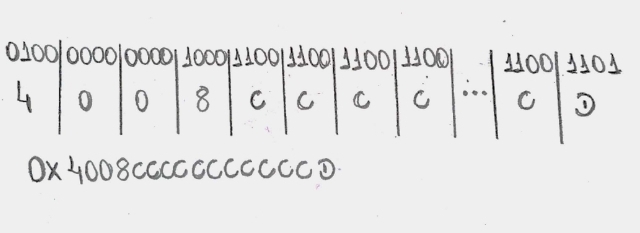
3.110 =11.00011001100110011001100110011001100110011001100110012

3.110 = 1.10001100110011001100110011001100110011001100110011012 \* 210

Exp = 1 + 1023 = 102410 = 100000000002

Frac = 1000110011001100110011001100110011001100110011001101

Resposta: 01000000000010001100110011001100110011001100110011001100110011012



**Questão 3) e)**

1.010100001010001111011002 \* 22

1.10001100110011001100110011001100110011001100110011012 \* 21,

que é igual a 0.110001100110011001100110011001100110011001100110011012 \* 22

Precisamos agora igualar os expoentes:

1.01010000101000111101100000000000000000000000000000002

- 0.11000110011001100110011001100110011001100110011001112

Ao transformar o subtraendo em complemento a dois:

1.01010000101000111101100000000000000000000000000000002

+1.00111001100110011001100110011001100110011001100110012

----------------------------------------------------------------------------------------------

0.10001010001111010111000110011001100110011001100110012 \* 22

Assim: 1.000101000111101011100011001100110011001100110011001 \* 21

S = 1

Exp = 1023 + 1 = 102410 = 100000000002

Frac = 00010100011110101110001100110011001100110011001100102

**Questão 3) f)**

1.00010100011110101110001100110011001100110011001100102 \* 21

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

1.000101000111101011100012, que, arredondado, nos dá 1.000101000111101011100102

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

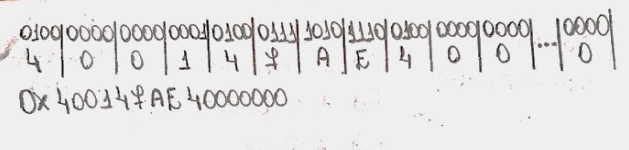
1.00010100011110101110010000000000000000000000000000002 \* 21

Sinal = 0

Exp = 1023 + 1 = 102410 = 100000000002

Frac = 0001010001111010111001000000000000000000000000000000

Resposta: 01000000000000010100011110101110010000000000000000000000000000002



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

Em Little Endian, teremos impressos pelo *printf*:

Parte alta: 0x40000000

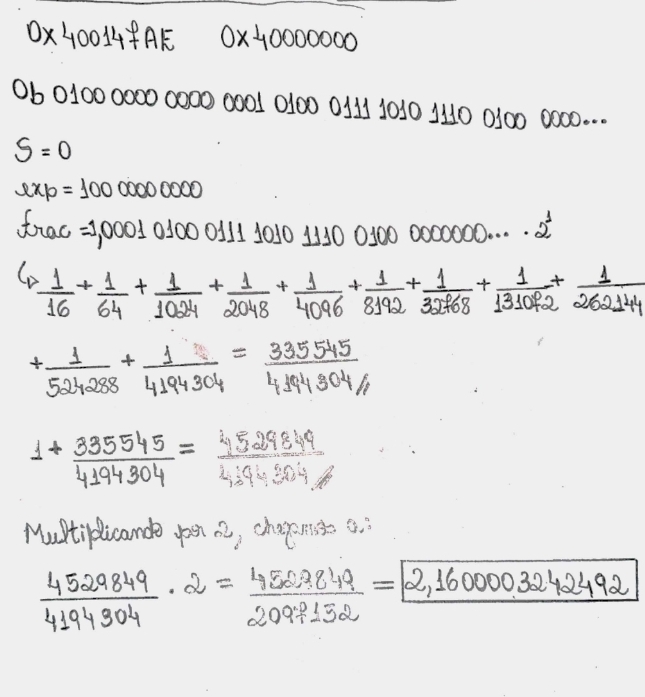
Parte baixa: 0x400147AE

Já em Big Endian, a representação é:

Parte alta: 0x400147AE

Parte baixa: 0x40000000

**Questão 3) g)**



**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, 0.310 = 0.010011001100110011001100110011002 com 32 bits após a vírgula

Então temos:

0.010011001100110011001100110011002 \* 2-136

= 1.0011001100110011001100110011002 \* 2-2 \* 2-136

= 1.0011001100110011001100110011002 \* 2-138

= 1.0011001100110011001100110011002 \* 2-12 \* 2-126

= 1.0000000000000011001100110011001100110011002 \* 2-126

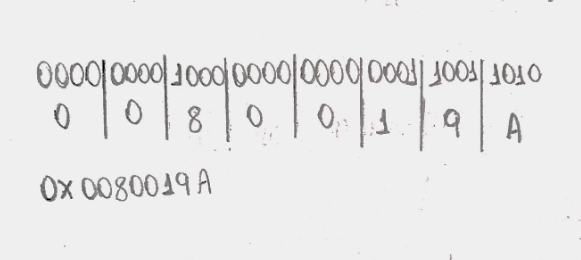
Sinal = 0

Exp = 127 – 126 = 110 = 000000012

Frac = 0000000000000011001100110011001100110011002 (42 bits)

Frac = 000000000000001100110102 (23 bits)

Resposta: 000000001000000000000001100110102



**Questão 4) b)**

0.3 \* 2-136

0.310 = 0.01001100110011001100110011001100110011001100110011001100110011002 (64 bits pós vírgula)

= 1.001100110011001100110011001100110011001100110011001100110011002 \* 2-2 \* 2-136

= 1.001100110011001100110011001100110011001100110011001100110011002 \* 2-138

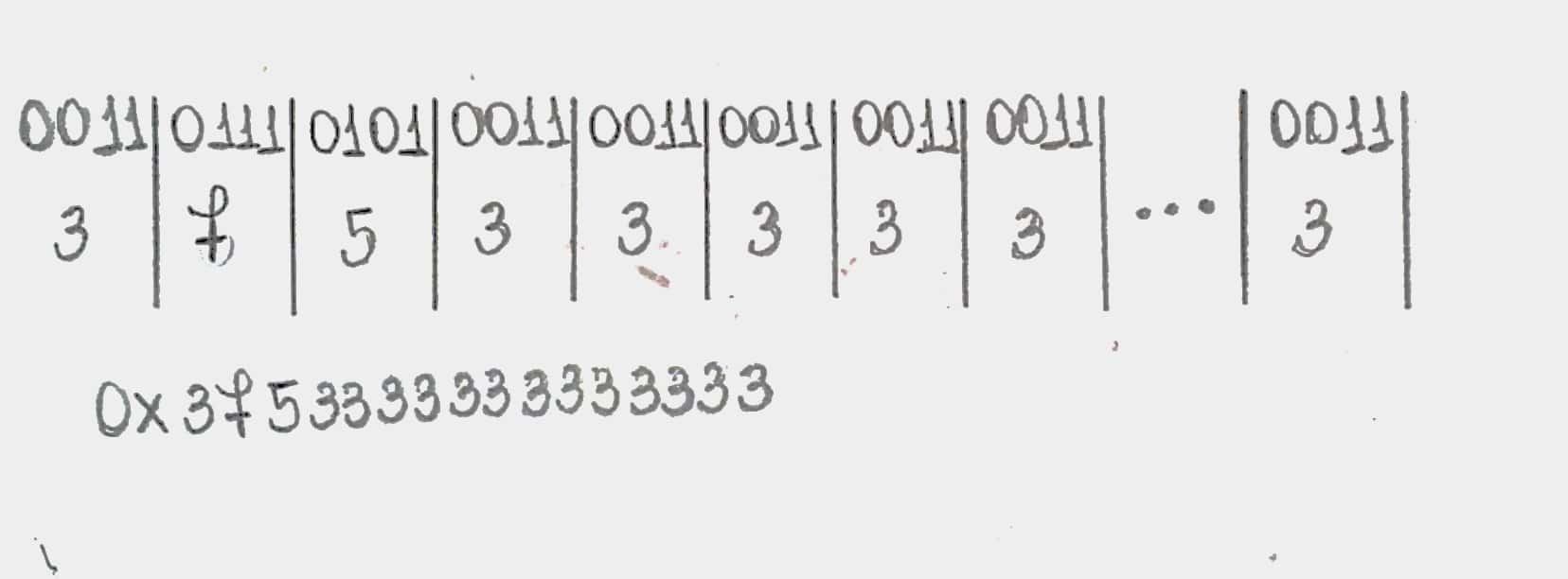
Sinal = 0

Exp = 1023 – 138 = 88510 = 011011101012

Frac = 00110011001100110011001100110011001100110011001100110011001100 (62 bits)

Frac = 0011001100110011001100110011001100110011001100110011 (52 bits)

Resposta: 00110111010100110011001100110011001100110011001100110011001100112



**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:

21023 \* (21 – 2-52), em que 21 é o bit “omitido” após normalizarmos

A partir daí, aplicando a propriedade distributiva, teremos:

21024 – 2971

**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...012  🡪 2-1022 (expoente) \* 2-52 (mantissa) 🡪 2-1074