1 Sistemas de numeração

Utilizar a notação decimal é interessante para nós seres humanos, principalmente pela associação com o número de dedos. Porém para o computador a manipulação de dados através dessa notação é atualmente inviável, e por isso, o computador utiliza a representação de informação através da notação binária (dois estados: ligado ou desligado, 0 ou 1).

1.1 Sistema de numeração decimal

Estamos tão acostumados com a representação de valores no formato decimal que passa despercebido a forma como são compostos, por exemplo, o número 456 é a combinação de:

6 x unidades (peso 1 ou 10^0)
5 x dezenas (peso 10 ou 10^1)
4 x centenas (peso 100 ou 10^2)

Ou outra forma de representação é através do peso das casas:

Valor					4	5	6
Peso	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Resultado					400	50	6

Somando-se os resultados:

$$400 + 50 + 6 = 456$$

Parece uma brincadeira, mas lembre-se que já estamos acostumados com esse sistema de numeração e já sabemos como representar ele.

1.2 Sistema de numeração binário

A codificação utiliza pelo computar possui apenas dois estados possíveis, com isso a tabela de pesos utiliza a base 2, e como podemos ter apenas os valores 0 e 1 nas casas, vamos fazer a demonstração de quanto vale o número 1110010_2 (perceba o 2 subscrito após o número, ele representa em qual base se está apresentado o número)

Valor	1	1	1	0	0	1	0
Peso	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Resultado	64	32	16	0	0	2	0

Somando os resultados:

$$64 + 32 + 16 + 0 + 0 + 2 + 0 = 114_{10}$$

Atividade:

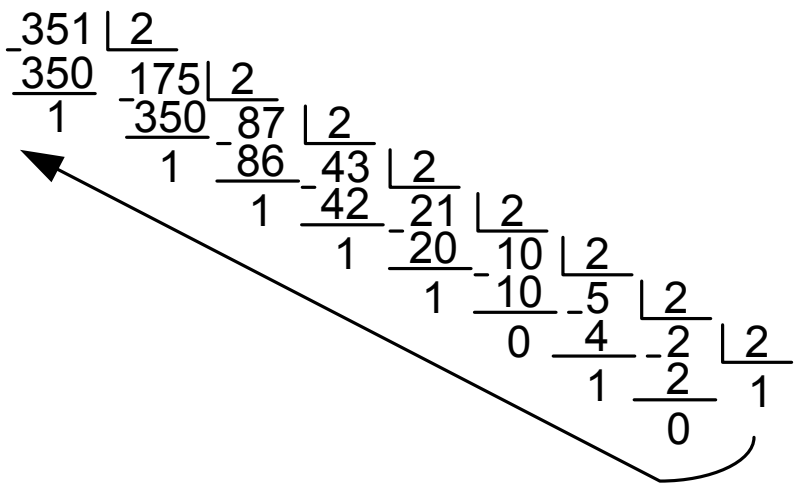
Converta os seguintes números de binário para decimal:

a) 1001001_2
b) 11001111_2
c) 1111001111_2
d) 11111000_2
e) 1000000000_2
f) 1001111000_2
g) 1100011001_2
h) 11000010_2
i) 11111111_2
j) 11000000_2

1.2.1 Convertendo de decimal para binário

O processo de conversão de decimal para binário pode ser feito de duas formas, de forma a exemplificar os processos vamos converter o número 351_{10} para binário:

Método 1 – Dividindo o número por 2 sucessivamente



Utilizando os resultados de trás para frente:

$$351_{10} = 101011111_2$$

Método 2 – Subtração do peso das casas

Nesse método, preciso verificar se posso subtrair o valor do número pelo peso da casa, caso positivo, acrescento 1 (um) a saída e o resto da subtração é enviada para a próxima casa. Se o número é menor que o peso da casa, acrescento 0 (zero) para a saída e continuo o mesmo número no valor da próxima casa.

Valor		351–256=	95	95-64=	31	31-16=	15-8=	7-4=	3-2=	1-1=0
Peso	$2^9 = 512$	$2^8 = 256$	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Saída		1	0	1	0	1	1	1	1	1

Novamente chegamos ao resultado:

$$351_{10} = 101011111_2$$

Atividade:

Utilizando qualquer um dos métodos estudados, faça a conversão de decimal para binário:

a) 1990_{10}
b) 1024_{10}
c) 1095_{10}
d) 974_{10}
e) 564_{10}
f) 512_{10}
g) 64_{10}
h) 231_{10}
i) 555_{10}
j) 2048_{10}
k) 122_{10}
l) 255_{10}

1.3 Sistema de numeração octal

Para nós humanos, entender uma sequência de 0 e 1 é confuso e fazer a conversão de binário para decimal exige certo cálculos para se obter o valor, de forma a simplificar a visualização de sequências de bits são utilizados outros métodos e sistema de numeração. Um dele é o octal, que utiliza os número de 0 a 7, totalizando 8 possíveis combinação por casa.

1.3.1 Conversão de octal para decimal

Para converter um número em octal para decimal utilizamos o processo de casas e pesos, por exemplo, para converter o número 753_8 para decimal fizemos:

Valor				7	5	3
Peso	$8^5 = 32.768$	$8^4 = 4096$	$8^3 = 512$	$8^2 = 64$	$8^1 = 8$	$8^0 = 1$
Resultado				7x64=448	5x8=40	3x1 = 3

O resultado será:

$$448 + 40 + 3 = 491_{10}$$

Atividade:

Converta os números da base octal para decimal

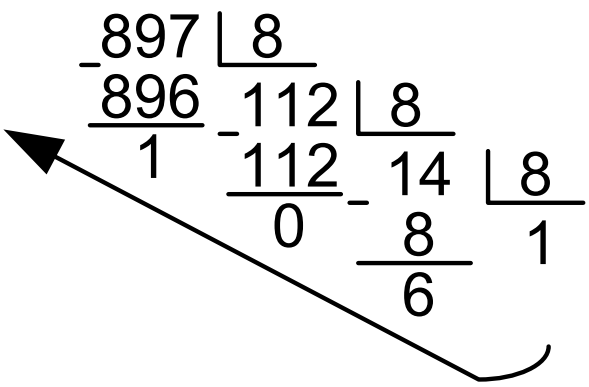
a) 777_8
b) 1000_8
c) 653_8
g) 241_8
h) 231_8
i) 555_8

d) 2730_8	j) 1024_8
e) 512_8	k) 122_8
f) 3276_8	l) 255_8

1.3.2 Conversão de decimal para octal

Para converter um número em decimal para octal, utilizamos o processo de divisão sucessivas por 8.

Veja um exemplo, para converter o número 897_{10} para octal



O resultado é 1601_8

Atividade:

Converta os números em decimal para seu equivalente em octal.

a) 102410	g) 241_{10}
b) 100010	h) 231_{10}
c) 409510	i) 555_{10}
d) 273010	j) 2048_{10}
e) 51210	k) 122_{10}
f) 327610	l) 255_{10}

1.3.3 Convertendo de binário para octal

O real motivo de se utilizar o sistema de numeração octal deve-se a forma de representar os números em binário. Por isso o sistema de conversão é simples, porém antes e interessante conhecer o número em octal e se equivalente em binário.

Tabela 1 - Equivalência octal-binária

Octal	Binário
0	000
1	001
2	010

3	011
4	100
5	101
6	110
7	111

Com base nessa tabela, basta apenas agrupar a cada três bits o número em binário e verificar seu equivalente em octal, caso necessário podemos preencher com zeros a esquerda nosso número em binário.

Veja o exemplo, para converter o número 1100111011_2 para octal:

$$\underbrace{1}_{1}\underbrace{100}_{4}\underbrace{111}_{7}\underbrace{011}_{3}$$

Resultado: $1100111011_2 = 1473_8$

Atividade:

Converta os seguintes números de binário para octal:

a) 111001001_2
b) 1100100111_2
c) 1110011_2
d) 11111000_2
e) 1110000000_2
f) 10110001110_2
g) 11100000111100_2
h) 110101011_2
i) 11111111_2
j) 111000111000111_2

1.3.4 Conversão de octal para binário.

Para a conversão de octal para binário o processo é o inverso daquele apresentado anteriormente, utilizando a Tabela 1 utiliza-se a equivalência do número em octal

Para exemplificar, vamos converter o número 4732_8 para binário:

$$\underbrace{4}_{100}\underbrace{7}_{111}\underbrace{3}_{011}\underbrace{2}_{010}$$

O resultado da conversão: 100111011010_2

Atividade:

Converta os seguintes números de octal para binário:

a) 10000_8
b) 551_8
c) 255_8
d) 1024_8
e) 7012_8
f) 444_8
g) 37_8
h) 10_8
i) 177_8
j) 3007_8

1.4 Sistema de numeração hexadecimal

De uma forma parecida ao sistema octal, o hexadecimal é utilizado para melhor visualizar uma sequencia de bits, sendo inclusive muito mais utilizado que o octal. De forma a completar a lista com 16 símbolos, após o 9 se inicia a utilização de letras, começando no A até o F, essa equivalência é vista na Tabela 2

Tabela 2 - Equivalência Decimal - Hexadecimal

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

1.4.1 Conversão de hexadecimal para decimal.

Para a conversão utilizamos o sistema de casas agora com a base 16. Para exemplo, vamos converter o número A5D (perceba a substituição da letra pela seu equivalente decimal)

Valor			A	5	D
Peso	$16^4 = 65536$	$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
Resultado			10x256=2560	5x16=80	13x1 = 13

Resultado = 2560 + 80 + 13 = 2653_{10}

Atividade:

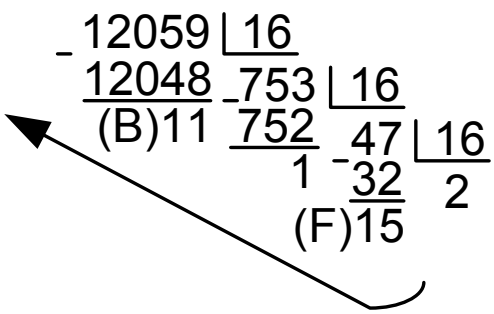
Converta os seguintes números de hexadecimal para decimal:

a) $E5B_{16}$
b) FFF_{16}
c) 1000_{16}
d) $1A_{16}$
e) AFF_{16}
f) 1111_{16}
g) CDE_{16}
h) 999_{16}
i) $11FF_{16}$
j) FAF_{16}

1.4.2 Conversão de decimal para hexadecimal

O processo ocorre por divisão, sempre cuidando para o caso do resto for maior que 10 deve ser substituído pelo sua letra equivalente.

Convertendo o número 12059_{10} para a base hexadecimal:



Resultado = $2F1B_{16}$

Atividade:

Converta os números em decimal para seu equivalente em hexadecimal.

m) 1024_{10}
n) 1000_{10}
o) 4095_{10}
p) 2730_{10}
q) 512_{10}
r) 3276_{10}
s) 241_{10}
t) 231_{10}
u) 555_{10}
v) 2048_{10}
w) 122_{10}
x) 255_{10}

1.4.3 Conversão de hexadecimal para binário

A conversão entre essas bases segue o mesmo raciocínio da conversão entre octal e binário, porém com 4 casas, que é a quantidade de casas necessária para representar os 16 símbolos em hexadecimal.

Hexadecimal	Binário	Hexadecimal	Binário
0	0000	**8**	1000
1	0001	**9**	1001

2	0010	**A**	1010
3	0011	**B**	1011
4	0100	**C**	1100
5	0101	**D**	1101
6	0110	**E**	1110
7	0111	**F**	1111

A conversão acontece agrupando os binário em grupo de 4 e verificando seu equivalente em hexadecimal, o processo inverso também é válido.

Por exemplo, para converter o número 101111010101_2 para hexadecimal:

$$\underbrace{1011}_{B}\underbrace{1101}_{D}\underbrace{0101}_{5}$$

E de $1F8_{16}$ para binário:

$$\underbrace{1}_{1}\underbrace{F}_{111111}\underbrace{8}_{1000}$$

Atividade:

Converta os números de hexadecimal para binário

a) $F0F_{16}$
b) 1000_{16}
c) $F1_{16}$
d) $B00_{16}$
e) 1024_{16}
f) CD_{16}
g) $FFFF_{16}$
h) 9999_{16}
i) $1A_{16}$
j) 2048_{16}
k) $F22_{16}$
l) 512_{16}

Atividade:

Converta os números de binário para hexadecimal

a) 111110001001_2
b) 11001110011_2
c) 1111111111_2
d) 111110000_2
e) 1111000011110000_2
f) 10110001110_2
g) 11100000111100_2
h) 110101011_2
i) 11111111_2
j) 111000111000111_2

2 Manipulação e organização dos dados

Cada computador possui um tamanho da palavra (word size) que indica o tamanho nominal dos números inteiros e apontadores de dados. Como um endereço virtual é representado pela como uma palavra, o tamanho da palavra determina o tamanho máximo do endereçamento da memória virtual.

Nos computadores pessoais o padrão é 32 bits/64bits, com um sistema que tem 32bits do tamanho da palavra, o tamanho máximo de memória é de 2^{32} = 4GB[1]. Enquanto nos sistemas de 64bits esse limite é igual a 2^{64} = 16 HB (17.179.869.184 GB)

2.1 Tamanho dos dados

Computadores e compiladores suportam vários formatos de dados, utilizando diversas formas para representa-los, por exemplo, inteiros e flutuantes, bem como diferentes larguras. Por exemplo, muitas máquinas tem instruções para manipular bytes simples, como também inteiros representados por 2, 4 ou 8 bytes.

A Tabela 3 apresenta a quantidade de bytes utilizados por alguns tipos de dados em C e a capacidade de armazenar.

Tabela 3 - Tamanho de alguns tipos em C

Declaraçao em C	Bytes	Valores possíveis
char	1	–128 até 127
short int	2	–32,768 até 32,767
int	4	–2,147,483,648 até 2,147,483,647
long int	4	–2,147,483,648 até 2,147,483,647
float	4	3.4E +/- 38 (7 digitos)
double	8	1.7E +/- 308 (15 digitos)

Fonte: CPlusplus[2]

2.2 Endereçamento e ordem de Bytes

Os programas que utilizam vários bytes para armazenar dados, precisar de duas informações: qual o endereço de armazenamento e qual a ordem dos bytes na memória. Para dados armazenados em vários bytes, geralmente uma sequência de bytes contínuos é reservado para esse dado. O endereço dado é o byte inicial, por exemplo, se uma variável *x* é armazenada no endereço *0x100*, o valor da expressão *&x* é *0x100*. Portanto os quatro bytes da variável *x* são armazenados nos endereços *0x100*, *0x101*, *0x102* e *0x103*.

A ordem dos bytes é a forma como os dígitos serão armazenamentos no espaço de memória, podendo iniciar pelo mais significativo (big endian) ou pelo menos significativo (little endian)[3]. Exemplos de arquitetura que são little endian incluem: x86 (inclusive x86-64), 6502

---

[1] Alguns sistemas na prática reconhecem apenas 3GB, para entender isso pesquise por "3 GB barrier"

[2] http://www.cplusplus.com/doc/tutorial/variables/

[3] Protocolos seriais também são classificados quanto a ordem de envio das informações. USB, RS-232, RS-422 e RS-485 são exemplos de padrões seriais do tipo little endian

(inclusive 65802, 65C816), Z80 (inclusive Z180, eZ80 etc.), MCS-48, 8051, DEC Alpha, Altera Nios, Atmel AVR, SuperH, VAX, e PDP-11.

Nossa forma de representar número é equivalente ao big endian, onde começamos a representar um número com o maior peso primeiro e a esquerda. Algumas arquitetura big endian são: Motorola 6800 e 68k, Xilinx Microblaze, IBM POWER, System/360 e seus sucessores System/370, ESA/390, e z/Architecture.

Ainda há um conjunto de arquitetura que a ordem dos bytes pode configurado no momento de inicialização, denominadas bi-endian. São exemplo de arquitetura bi-endian ARM, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC e IA-64.

A seguir segue um exemplo de armazenamento do valor 0x0A0B0C0D nas formas Big e Little Endian:

0x103	0D
0x102	0C
0x101	0B
0x100	0A

Figura 1 - Big Endian

0x103	0A
0x102	0B
0x101	0C
0x100	0D

Figura 2 - Little Endian

2.3 Representando string

String em C é codificado como uma sequência de caracteres terminada pelo caracter nulo (valor 0). Cada caracter é um número inteiro convertido através de uma definição para um caracter. A tabela mais utilizada é a ASCII, que apresenta a equivalência de vários símbolos para o binário, por exemplo a string “arquitetura” será representada conforme mostra a Figura 3, perceba que o computador irá armazenar o valor em binário.

Caracter	a	r	q	u	i	t	e	t	u	r	a
Decimal	97	114	113	117	105	116	101	116	117	114	97
Binário	1100001	1110010	1110001	1110101	1101001	1110100	1100101	1110100	1110101	1110010	1100001
Hexadecimal	61	72	71	75	69	74	65	74	75	72	61

Figura 3 - Representação de caracteres

Desafio:

Programe um programa em C que receba uma string e converta os caracteres para binário, decimal e hexadecimal.

2.4 Representação de inteiros

Existem diversas formas de representar um número inteiro, a princípio o computador é capaz de armazenar inteiros positivos apenas, porém, como proceder se precisar armazenar um número negativo? Esse capítulo irá tratar dessas formas de representação

2.4.1 Codificação de inteiros sem sinal

Assumimos que armazenamos um inteiro com a largura ω, o vetor que contêm os valores chamaremos de ň, uma sequência de bits que será convertido para inteiro sem sinal. A equação que transcreve a forma de conversão é apresentado a seguir:

$$I_{ss} = \sum_{i=0}^{\omega-1} \check{n}[i] \times 2^i$$

2.4.2 Codificação de inteiros com sinal[4]

Para representar um número com sinal o processo mais comum é o complemento de 2. Ele utiliza o bit mais significativo da palavra como sinal. A expressão que define a forma de conversão é:

$$I_{cs} = -(\check{n}[\omega - 1] \times 2^{\omega-1}) + \sum_{i=0}^{\omega-2} \check{n}[i] \times 2^i$$

Por exemplo, qual o número em decimal representado pelo número 11011010_2, perceba que a largura do vetor é 8:

Valor	1	1	0	1	1	0	1	0
Peso	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Resultado	128	64	0	16	8	0	2	0

- (128) + 64 + 16 + 8 + 2 = -38

Atividade:

Converta os seguintes números binários para decimal com sinal e sem sinal, para todos o tamanho de armazenamento é 8.

a) 10000001_2
b) 11111111_2
c) 10000000_2
d) 11110000_2
e) 1010101_2
f) 10000001_2
g) 11001000_2
h) 11111110_2
i) 1111111_2
j) 10011101_2

Atividade:

[4] Fim do mundo para o Linux e similares, um problema de limitação na variável de armazenamento da data nos sistemas com padrão POSIX podem causar problemas, o fim será as 03hrs14min07seg de 19 de Janeiro de 2038. Para maiores detalhes procure por "Bug 2038"

Converta os seguintes números negativos para binário, o limite de armazenamento é 8 bits.

a) -1_{10}
b) -100_{10}
c) -2_{10}
d) -64_{10}
e) -16_{10}
f) 127_{10}
g) -99_{10}
h) -22_{10}
i) -120_{10}
j) -128_{10}

Atividade:

Identifique o valor da variável `ux` e explique o resultado.

```
int x = -1;
unsigned ux = (unsigned) x;
```

A linguagem C considera os números inteiros como com sinal, com isso algumas conversões produzem resultados inesperados.

Atividade

Verifique as comparações abaixo verificando quais não condizem com o esperado e justifique a diferença (são três os casos que não condizem com o esperado):

Expressão	Saída esperada	Inesperado?
0 == 0U	Verdadeiro	
-1 < 0	Verdadeiro	
-1 < 0U	Verdadeiro	
2147483647 > -2147483648	Verdadeiro	
2147483647U > -2147483648	Verdadeiro	
2147483647 > (int) 2147483648U	Verdadeiro	

2.5 Ponto flutuante

O ponto flutuante é uma forma de representar números na forma racional (envolvendo frações). A padronização dos números em ponto flutuante em 1985 garantiu a homogeneidade da representação e cálculos desse tipo de número. Antes disso, cada fabricante adotava uma forma própria de representar e também de operar, muitas vezes o fator precisão era deixado em segundo plano em favor de um melhor desempenho dos cálculos.

2.5.1 Números fracionários binários

Na nossa forma norma de representação de números fracionários decimais, utilizamos o sistema de peso também no lado direito da vírgula que separa a parte inteira da fracionária. Por exemplo, o numero 12,345 seria representado assim:

Valor		1	2	3	4	5	
Peso	...	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	...

Resultado		10	2	0,3	0,04	0,005	

O resultado seria a soma dos resultados, ou na forma matemática:

$$D = \sum_{i=-n}^{m} 10^{i} \times d[i]$$

Onde *D* seria o número decimal final, *m* a quantidade de casas inteiras, *n* o número de casas fracionárias, e *d[i]* a parte do número decimal na casa *i*.

Por analogia, considere agora essa forma para representar o sistema binário, com o mesmo princípio de casas e pesos. Por exemplo, vamos representar o número 101.11_2

Valor		1	0	1	1	1	
Peso	...	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 1/2$	$2^{-2} = 1/4$	...
Resultado		4	0	1	1/2	1/4	

$$\text{Resultado } = 5\frac{3}{4}$$

A representação matemática é:

$$B = \sum_{i=-n}^{m} 2^{i} \times b[i]$$

Porém essa forma de representar números fracionários tem problemas de precisão[5][6], por exemplo, para representar o número 0,2 (1/5), não há formas precisão de representação, conforme mostrado na Tabela 4.

Tabela 4 - aproximação de 1/5

Representação	Valor	Decimal
$0{,}01_2$	1/4	$0{,}25_{10}$
$0{,}0011_2$	3/16	$0{,}1875_{10}$
$0{,}001101_2$	13/64	$0{,}203125_{10}$
$0{,}00110011_2$	51/256	$0{,}19921875_{10}$

2.5.2 Representação de ponto flutuante de acordo com a IEEE

Com o padrão IEEE 754 de 1985, baseada num pequeno e consistente conjunto de princípios tornou a representação em número em ponto flutuante elegante e entendível.

A IEEE 754 representa um número no formato $V = (-1)^S \times M \times 2^e$ onde:

- S: sinal do número, negativo é definido por S = 1 ou positivo S = 0

[5] Um fato curioso sobre essa problema de representação foi na guerra do Golfo, para saber mais detalhes pesquise num site de busca por "erro arredondamento míssil Patriot".

[6] Ainda podemos verificar problemas de arredondamento no Excel, para isso numa célula qualquer insira a equação = (0,5-0,4-0,1). O resultado deveria ser 0.

- M: Mantissa, é a representação da parte fracionária
- e: expoente da base.

A IEEE 754 padroniza o formato em dois tamanhos, o ponto flutuante de **Precisão simples** e o de **Precisão dupla**. Cada um com diferentes tamanhos de campos conforme apresentado respectivamente nas figuras Figura 4 Figura 5

bits	1	8	23
	+-	Expoente	Mantissa

Figura 4 - Ponto flutuante de precisão simples

bits	1	11	52
	+-	Expoente	Mantissa

Figura 5 - Ponto flutuante de precisão dupla

O expoente é definido através do calculo com o *bias*, esse número é obtido através do número de bits no expoente com a fórmula $2^{k-1} - 1$. Para o caso de precisão simples, o bias é $2^{8-1} - 1 = 127$. Para o ponto flutuante de precisão dupla, $2^{11-1} - 1 = 1023$.

2.5.3 Convertendo um número fracionário decimal para binário[7]

De forma a exemplificar a conversão, vamos considerar uma definição própria do tamanho do ponto flutuante, o sinal será de 1 bit, o expoente de 3 bits (bias igual a 3) e mantissa de 4 bits, totalizando 8 bits para armazenar o número que será nosso exemplo: 2,625:

1. Para a parte inteira, realiza-se a conversão normal de decimal para binário: $2_{10} = 10_2$
2. Para a parte fracionária, faz-se sucessivas multiplicações da parte fracionária por 2:

   $0{,}625 \times 2 = 1{,}25$ $\boxed{1}$ *Retira o número inteiro e continua a conta*
   $0{,}25 \times 2 = 0{,}5$ $\boxed{0}$ *Gerou 0, continuando...*
   $0{,}5 \times 2 = 1{,}0$ $\boxed{1}$ *Gerou 1 e restou 0, para aqui.*
   Portanto, $0{,}625_{10} = 0{,}101_2$
3. Juntando a parte inteira e a parte fracionária, temos 10.101_2
4. Adicionando o expoente: $10.101_2 \times 2^0$
5. Normalizando: $10.101_2 \times 2^0 = 1.0101_2 \times 2^1$ (verifique que foi deslocado o ponto decimal uma casa para a direita, por isso, o expoente aumentou uma casa).
6. Mantissa: 0101
7. Expoente: $1 + 3 = 4_{10} = 100_2$
8. Bit de sinal positivo: 0
9. Resultado é 01000101, em hexadecimal 45_{16}

[7] Forma de conversão retirado de http://sandbox.mc.edu/~bennet/cs110/flt/dtof.html

Vamos para outro exemplo, converter o número -39887.5625 para o formato IEEE 754 de precisão simples:

1. A parte inteira: $39887_{10} = 1001101111001111_2$
2. A fração $0{,}5625_{10} = 0{,}1001_2$

0,5625 × 2 = 1,125	$\boxed{1}$	*Generate 1 and continue with the rest.*
0,125 × 2 = 0,25	$\boxed{0}$	*Generate 0 and continue.*
0,25 × 2 = 0,5	$\boxed{0}$	*Generate 0 and continue.*
0,5 × 2 = 1,0	$\boxed{1}$	*Generate 1 and nothing remains.*

3. A conversão completa: $39887{,}5625_{10} = 1001101111001111{,}1001_2$
4. Normalizando: $1001101111001111{,}1001_2 = 1{,}0011011110011111001_2 \times 2^{15}$.
5. Mantissa é 00110111100111110010000, o expoente é 15 + 127 = 142 = 100011102, bit de sinal é 1.
6. Portanto 39887,5625 é 11000111000110111100111110010000 = $C71BCF90_{16}$

2.5.4 Convertendo um número fracionário binário para decimal.

O processo inverso, para converter $C4A42A00_{16}$ para decimal, através da normativa IEEE de precisão simples:

1. Convertendo: $C4A42A00_{16} = 11000100101001000010101000000000_2$
2. Sinal: 1 = negativo
3. Expoente: 10001001_2 = 137 – 127 (bias) = 10
4. Mantissa: $01001000010101000000000_2 = 0{,}28253173828125_{10}$
5. Aplicando a mantissa e o expoente = $1{,}28253173828125 \times 2^{10} = -1313{,}3125_{10}$

Atividade:

Converta os seguintes números fracionários em decimal para binário, utilizando a formatação de 8 bits de ponto flutuante (1 bit de sinal, 3 de expoente e 4 de mantissa)

a) $1{,}8_{10}$
b) 5/32
c) $-1{,}25_{10}$
d) $0{,}75_{10}$

Atividade:

Converta os seguintes números fracionários em decimal para binário, utilizando a formatação de 32 bits de ponto flutuante da IEEE

a) $1{,}8_{10}$
b) $5/32_{10}$
c) $-1{,}25_{10}$
d) $0{,}75_{10}$
e) $1/9_{10}$
f) $-1/5_{10}$
g) $0{,}1_{10}$
h) $1{,}99_{10}$

Atividade:

Converta os seguintes números fracionários em hexadecimal para decimal, utilizando a formatação de 32 bits de ponto flutuante da IEEE

a) $42E48000_{16}$
b) $3F880000_{16}$
c) $40490FDB_{16}$
d) 00800000_{16}
e) $C7F00000_{16}$
f) $C02DF855_{16}$