1.8 EXERCÍCIOS PROPOSTOS

1. Um número tem dois algarismos, sendo y o algarismo das unidades e x o algarismo das dezenas. Se colocarmos o algarismo 5 à direita desse número, o novo número será:
   a) x + y + 5
   b) 500 + 10•y + x
   c) 100•x + 10•y + 5
   d) 100•y + 10•x + 5

   Resposta: Se o número tem dois algarismos, então no novo número o algarismo das centenas será x, o algarismo das dezenas será y e o algarismo das unidades será 5. Então o novo número será então 100 x + 10 y + 5. A resposta correta é (c).
2. Represente os seguintes valores binários em valores decimais equivalentes (conversão de base 2 para base 10):
   a) $1011_{(2)}$
   b) $01100110_{(2)}$
   c) $100010001_{(2)}$

   Resposta: Para converter de binário para decimal, você pode multiplicar cada dígito binário por 2 elevado à sua posição, começando com 0 à direita e indo para a esquerda. Em seguida, some todos os resultados.
   a) $1011_{(2)} = 2^3 + 2^1 + 2^0 = 8 + 2 + 1 = 11$
   b) $01100110_{(2)} = 2^6 + 2^5 + 2^2 + 2^1 = 64 + 32 + 4 + 2 = 102$
   c) $100010001_{(2)} = 2^8 + 2^4 + 2^0 = 256 + 16 + 1 = 273$
3. Represente os seguintes valores decimais em valores binários equivalentes (conversão de base 10 para base 2):
   a) 6
   b) 24
   c) 132
   d) 512

   Resposta: A conversão pode ser feita dividindo-se sucessivamente o número e os respectivos quociente por 2 até o quociente ser zero e em seguida combinar os restos obtidos na ordem inversa. Assim teremos:
   a) $6 = 110_{(2)}$
   b) $24 = 11000_{(2)}$
   c) $132 = 10000100_{(2)}$
   d) $512 = 1000000000_{(2)}$
4. A partir do valor binário 10110010, escreva os cinco números que se seguem em sequência.
   Resposta: Os valores na sequência são:
   10110011 10110100 10110101 10110110 10110111

5. Exprima os seguintes valores hexadecimais em binário:
   a) $1010_{(16)}$
   b) $2A9B_{(16)}$

c) $ABCD01_{(16)}$

Resposta: A conversão pode ser feita facilmente convertendo-se cada digito hexadecimal pelo seu valor correspondente em binário:

a) $1010_{(16)}$ = 0001 0000 0001 $0000_{(2)}$

b) $2A9B_{(16)}$ = 0010 1010 1001 $1011_{(2)}$

c) $ABCD01_{(16)}$ = 1010 1011 1100 1101 0000 $0001_{(2)}$

6. Exprima os seguintes números decimais em octal e hexadecimal:

a) 157

b) 2048

Resposta: Neste caso a conversão deve ser feita usando o método da divisão, no primeiro caso usando 8 como divisor e no segundo, 16.

a) 157 = 235 $_{(8)}$ = 0x9d $_{(16)}$

b) 2048 = 4000 $_{(8)}$ = 0x800 $_{(16)}$

7. Converta os seguintes números da base 5 para a base 10:

a) $321_{(5)}$

b) $10333_{(5)}$

Resposta:

a) $321_5 = 3 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 = 65 + 10 + 1 = 76$

b) $10333_5 = 1 \times 5^4 + 0 \times 5^3 + 3 \times 5^2 + 3 \times 5^1 + 3 \times 5^0 = 625 + 75 + 15 + 3 = 718$

8. Descreva, baseando-se no funcionamento descrito para um odômetro convencional (base 10), como funcionaria um odômetro octal?

Resposta: O odômetro octal funciona da mesma forma que um odômetro convencional, mas em vez de usar dígitos decimais, usa dígitos octais. Os dígitos octais são 0, 1, 2, 3, 4, 5, 6, 7.

9. Converta para a base octal os seguintes números representados na base binária

a) $11101101_{(2)}$

b) $10000111011_{(2)}$

c) $11100001_{0(2)}$

Resposta:

A solução é facilmente obtida convertendo cada grupo de 3 bits do número em binário diretamente para o seu correspondente em octal.

a) $11\ 101\ 101_{(2)} = 355_{(8)}$

b) $10\ 000\ 111\ 011_{(2)} = 2073_{(8)}$

c) $111\ 000\ 010_{(2)} = 702_{(8)}$

10. Converta os seguintes números octais para a base 16:

a) $032_{(8)}$

b) $324217_{(8)}$

c) $11111111_{(8)}$

Resposta: A forma de solução mais fácil é converter o número octal para binário, transformando cada dígito octal em 3 dígitos binários, depois converter cada grupo de 4 dígitos binários em um dígito hexadecimal.

a) $032_{(8)} = 000\ 011\ 010 = 0\ 0001\ 1010 = 1A_{(16)}$

b) $324217_{(8)} = 011\ 010\ 100\ 010\ 001\ 111 = 01\ 1010\ 1000\ 1000\ 1111 = 1A88F_{(16)}$

c) $11111111_{(8)} = 001\ 001\ 001\ 001\ 001\ 001\ 001\ 001 = 0010\ 0100\ 1001\ 0010\ 0100\ 1001 = 249249_{(16)}$

11. Quantos números inteiros positivos podem ser representados em uma base B, cada um com N algarismos significativos?

    Resposta: Existem B^N números inteiros positivos que podem ser representados em uma base B, cada um com N algarismos significativos. Isso ocorre porque existem B opções para cada dígito, e há N dígitos. Por exemplo, se B = 10 e N = 3, então existem $10^3 = 1000$ números inteiros positivos que podem ser representados: 1, 10, 11, 12, ..., 99, 100.

12. Some os seguintes números em binário. Calcule o valor do bit de "carry" (overflow).

    a) $0101_{(2)} + 1011_{(2)}$ =

    b) $10111001_{(2)} + 11010111_{(2)}$ =

    Resposta:

    a) $0101_2 + 1011_2 = 1110_2$ Bit de carry: 1

    b) $10111001_2 + 11010111_2 = 11001110_2$ Bit de carry: 1

13. Subtraia os seguintes números em binário. Calcule o valor do bit de "borrow" (underflow).

    a) $1111_{(2)} - 1111_{(2)}$ =

    b) $10011111_{(2)} - 11111111_{(2)}$ =

    Resposta:

    a) $1111_{(2)} - 1111_{(2)} = 0000_{(2)}$ Bit de carry: 0

    b) $10011111_{(2)} - 11111111_{(2)} = 00000000_{(2)}$ Bit de carry: 1

14. Multiplique os seguintes números em binário, produzindo a saída em 8 bits.

    a) $0101_{(2)} \times 0011_{(2)}$ =

    b) $1111_{(2)} \times 0111_{(2)}$ =

    Resposta:

    a) 0101(2) × 0011(2) = 00001111(2)

    b) 1111(2) × 0111(2) = 01101001(2)

15. Divida os seguintes números binários, obtendo resultados até a 4ª casa fracionária binária:

    a) $1100_{(2)} \div 0011_{(2)}$

    b) $11011011_{(2)} \div 0011_{(2)}$

    Resposta:

    a) 0101(2) ÷ 0011(2) = 0100,0000(2)

    b) 11011010(2) ÷ 0111(2) = 01001000,1010(2)

16. Multiplique os seguintes números binários por 8, 16 e 32

    a) $100110_{(2)}$

    b) $100011,1101_{(2)}$

    Resposta:

    a) $100110_{(2)} \times 8 = 1001100_{(2)}$
    $100110_{(2)} \times 16 = 10011000_{(2)}$
    $100110_{(2)} \times 32 = 100110000_{(2)}$

    b) $100011,1101_{(2)} \times 8 = 100011110,1_{(2)}$
    $100011,1101_{(2)} \times 16 = 1000111101_{(2)}$
    $100011,1101_{(2)} \times 32 = 10001111010_{(2)}$

17. Algumas linguagens implementam as operações de multiplicação de números inteiros por potência de 2 como uma operação alternativa chamada de "deslocamento para a esquerda" (shift left) em que aos números binários são adicionados zeros à direita. Quantos zeros seriam adicionados numa multiplicação por 2048?

    Resposta: Uma multiplicação por 2048 é equivalente a um deslocamento para a esquerda de 11 posições binárias. Isso ocorre porque 2048 é igual a 2^{11} (2048 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2).

    Em uma operação de deslocamento para a esquerda de 11 posições, 11 zeros seriam adicionados à direita do número binário original, resultando em um número maior. Esse processo é efetivamente equivalente a multiplicar o número original por 2048.

    Por exemplo, considere o número binário 1101. Ao fazer uma multiplicação por 2048 (ou deslocar 11 posições para a esquerda), obtemos:

    1101 « 11 = 1101000000000

    Portanto, 11 zeros foram adicionados à direita (00000000000), e o resultado é o número binário 1101000000000, que é igual ao número decimal 8192 ($2^{11} = 2048$).

18. Divida os seguintes números binários por 8, 16 e 32
    a) $111111_{(2)}$
    b) $1111,11_{(2)}$

    Resposta:
    a) $111111_{(2)} \div 8 = 111,111_{(2)}$ $111111_{(2)} \div 16 = 11,1111_{(2)}$ $111111_{(2)} \div 32 = 1,11111_{(2)}$
    b) $1111,11_{(2)} \div 8 = 1,11111_{(2)}$ $1111,11_{(2)} \div 16 = 0,111111_{(2)}$ $1111,11_{(2)} \div 32 = 0,0111111_{(2)}$

19. Algumas linguagens implementam as operações de divisão de números inteiros por potência de 2 como uma operação alternativa chamada de "deslocamento para a direita" (shift right) em que aos números binários têm seus dígitos à direita removidos. Quantos dígitos seriam removidos numa divisão por 64?

    Resposta:Uma divisão por 64 é equivalente a um deslocamento para a direita de 6 posições binárias. Isso ocorre porque 64 é igual a 2^6 (64 = 2 * 2 * 2 * 2 * 2 * 2).

    Em uma operação de deslocamento para a direita de 6 posições, os 6 dígitos mais à direita do número binário seriam removidos, resultando em um número menor. Esse processo é efetivamente equivalente a dividir o número original por 64.

    Por exemplo, considere o número binário 1011000. Ao fazer uma divisão por 64 (ou deslocar 6 posições para a direita), obtemos:

    1011000 » 6 = 0000010

    Portanto, os 6 dígitos à direita (000010) foram removidos, e o resultado é o número binário 0000010, que é igual ao número decimal 2 ($2^6 = 64$).

20. Represente como uma sequência de números binários em ASCII os caracteres das seguintes palavras (diferencie maiúsculas de minúsculas). Transcreva também para codificação em hexadecimal.
    a) CASA
    b) M72az
    c) Minha vida.

Resposta:

a) CASA

C | 01000011 | 43

A | 01000001 | 41

S | 01001111 | 53

A | 01000001 | 41

b) M72az

M | 01001101 | 4D

7 | 00110111 | 37

2 | 00110010 | 32

a | 01100001 | 61

z | 01111010 | 7A

c) Minha vida.

M | 01001101 | 4D

I | 01001101 | 49

N | 01001110 | 4E

H | 01001110 | 48

A | 01000001 | 41

V | 01001110 | 48

I | 01001101 | 49

D | 01001100 | 44

A | 01000001 | 41

. | 00111110 | 2E

21. Qual foi a motivação ao se criar um código padronizado como o Unicode?

Resposta:

A motivação de mercado para se criar um código padronizado como o Unicode foi a necessidade de permitir que os computadores e aplicativos se comunicassem uns com os outros independentemente do sistema operacional ou do idioma.

Antes do Unicode, cada sistema operacional e aplicativo tinha seu próprio conjunto de caracteres. Isso significava que era difícil trocar dados entre diferentes sistemas e aplicativos. Isso era um problema para empresas que precisavam compartilhar dados com clientes e parceiros em todo o mundo.

O Unicode foi criado para resolver esse problema. Ele é um padrão universal para a representação de texto em computadores. O Unicode inclui uma ampla gama de caracteres, de todos os principais idiomas do mundo. Isso permite que os computadores e aplicativos se comuniquem uns com os outros independentemente do sistema operacional ou do idioma.

22. Podem os “emoticons” serem codificados em Unicode (pesquise na Internet)?

Resposta:

Os Emoticons ocupam um bloco Unicode que contém emoticons ou emoji. A maioria deles é destinada a representar rostos, embora alguns deles incluam gestos de mão ou personagens não humanos (como um ”diabinho”com chifres, macacos, gatos animados).

O bloco foi proposto pela primeira vez em 2008 e implementado pela primeira vez na versão 6.0

do Unicode (2010). A razão para sua adoção foi principalmente para compatibilidade com um padrão de facto estabelecido pelos operadores de telefonia japoneses no início dos anos 2000, codificado em faixas não utilizadas com bytes iniciais 0xF5 a 0xF9 do padrão Shift JIS. A empresa KDDI foi muito além disso e introduziu centenas de emoticons adicionais no espaço com bytes iniciais 0xF3 e 0xF4.

23. A adoção do UTF-8 como padrão de comunicação de caracteres trouxe que vantagens aos navegadores de Internet? Isso é bom para as páginas na língua portuguesa? E para as páginas chinesas?

    Resposta: O UTF-8 é um codificador mais eficiente do que os codificadores anteriores, como o ASCII e o ISO-8859-1, que eram muito limitados. Isso significa que as páginas codificadas em UTF-8 ocupam menos espaço de armazenamento e são carregadas mais rapidamente. A adoção do UTF-8 como padrão de comunicação de caracteres trouxe uma série de vantagens para os navegadores de Internet, incluindo um suporte a um maior número de caracteres: O UTF-8 é um codificador de caracteres Unicode, o que significa que ele pode representar todos os caracteres do Unicode, incluindo caracteres de idiomas que não são de origem latina. Isso é bom para páginas na língua portuguesa, assim como diversas outras línguas diferentes do inglês, pois permite que essas páginas sejam exibidas corretamente em todos os navegadores de Internet.
    Em termos de representação de caracteres com ideogramas, como chinês, japonês e koreano, além do Unicode (que possui um conjunto de ideogramas unificados CJK), existem sistemas de codificação local. O sistema Guobiao chinês (ou GB, ”padrão nacional”) é usado na China continental e em Cingapura, e o sistema Big5 (principalmente) taiwanês é usado em Taiwan, Hong Kong e Macau, como os dois principais sistemas de codificação local ”legados”. O Guobiao geralmente é exibido usando caracteres simplificados, e o Big5 geralmente é exibido usando caracteres tradicionais. No entanto, não há uma conexão obrigatória entre o sistema de codificação e a fonte usada para exibir os caracteres; a fonte e a codificação geralmente estão vinculadas por razões práticas.
    A questão de qual codificação usar também pode ter implicações políticas, pois o GB é o padrão oficial da República Popular da China e o Big5 é um padrão de facto de Taiwan. Em contraste com a situação no Japão, houve relativamente pouca oposição explícita ao Unicode, que resolve muitos dos problemas envolvidos com o GB e o Big5. O Unicode é amplamente considerado politicamente neutro, tem bom suporte tanto para caracteres simplificados quanto para caracteres tradicionais e pode ser facilmente convertido de e para o GB e o Big5. Além disso, o Unicode tem a vantagem de não estar limitado apenas ao chinês, uma vez que contém códigos de caracteres para (quase) todas as línguas.

24. Descreva algumas regras simples para transcrição de um conjunto de letras a partir de sua codificação em Unicode para UTF-8 de palavras na língua portuguesa. Use estas regras para transcrever a palavra “Atenção” para uma sequência em UTF-8 em representação hexadecimal.
    Resposta:
    Para transcrever um conjunto de letras em Unicode para UTF-8, é necessário seguir algumas

regras simples:

1. Se o código Unicode da letra for menor ou igual a 127 (ou seja, pode ser representado em 7 bits), a letra será representada em UTF-8 utilizando um único byte, igual ao código Unicode da letra.

2. Se o código Unicode da letra for maior que 127, a letra será representada em UTF-8 utilizando múltiplos bytes.

- Para caracteres com código Unicode entre 128 e 2047 (ou seja, que podem ser representados em 11 bits), a letra será representada em UTF-8 utilizando dois bytes. O primeiro byte começará com '110' seguido dos 5 bits mais significativos do código Unicode da letra. O segundo byte começará com '10' seguido dos 6 bits menos significativos do código Unicode da letra.
- Para caracteres com código Unicode entre 2048 e 65535 (ou seja, que podem ser representados em 16 bits), a letra será representada em UTF-8 utilizando três bytes. O primeiro byte começará com '1110' seguido dos 4 bits mais significativos do código Unicode da letra. O segundo e terceiro bytes começarão com '10' seguido dos 6 bits do meio e 6 bits menos significativos do código Unicode da letra, respectivamente.
- Para caracteres com código Unicode entre 65536 e 1114111 (ou seja, que podem ser representados em 21 bits), a letra será representada em UTF-8 utilizando quatro bytes. O primeiro byte começará com '11110' seguido dos 3 bits mais significativos do código Unicode da letra. Os três bytes seguintes começarão com '10' seguido dos 6 bits de cada terço do código Unicode da letra.

Agora, vamos transcrever a palavra "Atenção"para UTF-8 em representação hexadecimal:

A: 41 (em hexadecimal).

t: 74 (em hexadecimal).

e: 65 (em hexadecimal).

n: 6E (em hexadecimal).

ç: C3 A7 (em hexadecimal).

ã: C3 A3 (em hexadecimal).

o: 6F (em hexadecimal).

Portanto, a representação UTF-8 correta da palavra "Atenção"é: 41 74 65 6E C3 A7 C3 A3 6F.

25. Represente em binário de 4 bits os seguintes números em BCD:
    a) 1234
    b) 24019

    Resposta:
    a) Para representar 1234 em BCD, precisamos converter cada dígito (1, 2, 3, 4) para binário de 4 bits:

       1 = 0001 2 = 0010 3 = 0011 4 = 0100

       Agora, juntamos esses dígitos em BCD para formar o número completo:

       1234 em BCD = 0001 0010 0011 0100
    b) Para representar 24019 em BCD, precisamos converter cada dígito (2, 4, 0, 1, 9) para binário de 4 bits:

       2 = 0010 4 = 0100 0 = 0000 1 = 0001 9 = 1001

       Agora, juntamos esses dígitos em BCD para formar o número completo:

       24019 em BCD = 0010 0100 0000 0001 1001

26. Represente em sinal magnitude (em binário de 8 bits) os seguintes números decimais:
    a) 3
    b) -3
    c) -127
    d) 128

    Resposta: Para representar os números decimais em sinal-magnitude em binário de 8 bits, usamos o bit mais significativo para o sinal (0 para positivo e 1 para negativo) e os 7 bits restantes para representar o valor absoluto do número em binário. Aqui estão as representações para cada número:

    a) 3 em binário de 8 bits é: 00000011
    b) Para representar -3 em sinal-magnitude, usamos o bit de sinal 1 e o valor absoluto de 3 em binário, que é 00000011. Portanto, -3 em binário de 8 bits é: 10000011
    c) -127:

       Para representar -127 em sinal-magnitude, usamos o bit de sinal 1 e o valor absoluto de 127 em binário, que é 01111111. Portanto, -127 em binário de 8 bits é: 11111111
    d) 128:

       Observe que para o número 128, ele não pode ser representado em sinal-magnitude com apenas 8 bits, pois o maior valor absoluto representável em 8 bits é 127 (01111111 em binário).

27. O que é o problema da dupla representação do zero na representação em sinal-magnitude?
    Resposta:

    O problema da dupla representação do zero na representação em sinal-magnitude está relacionado com a forma como os números são representados usando esse sistema numérico. Na representação em sinal-magnitude, o bit mais significativo (o bit mais à esquerda) é usado para indicar o sinal do número, sendo 0 para números positivos e 1 para números negativos. Os bits restantes representam o valor absoluto do número em questão. O valor zero tem duas representações diferentes, uma para zero positivo e outra para zero negativo. Na prática, temos:

    - Zero positivo: 00000000
    - Zero negativo: 10000000

    Essa ambiguidade na representação do zero pode levar a alguns problemas indesejados

    - Dificuldades de comparação: Se você estiver trabalhando com números em sinal-magnitude e realizar comparações, pode haver inconsistências, pois os zeros positivo e negativo são considerados distintos.
    - Complexidade no hardware: Implementar operações aritméticas e lógicas usando a representação em sinal-magnitude pode ser mais complexo devido à necessidade de tratar os dois zeros diferentes.

    Por causa dessas limitações, a representação em sinal-magnitude não é amplamente utilizada em sistemas computacionais modernos. Em vez disso, outros sistemas numéricos, como o complemento de dois, são preferidos por causa de suas propriedades mais convenientes para a aritmética e a representação única do zero.

28. Quantos números diferentes se consegue representar em sinal magnitude para números de 8 bits?

Resposta: Com 8 bits, há 256 combinações possíveis de bits. No entanto, existe a dupla representação para o número zero. Portanto, há 255 números diferentes que podem ser representados em sinal magnitude com 8 bits.

29. Que vantagem pode ser atribuída à representação em Excesso-K, quando comparada a outras formas de representação?

    Resposta: A representação em Excesso-K apresenta como vantagem em relação à representação sinal magnitude o fato de não apresentar a duplicidade de representação do zero. As operações de subtração também podem ser feitas invertendo-se o sinal do subtraendo e realizando a soma dos valores. No entanto, a obtenção do número negativo é feita subtraindo 2*K do seu valor positivo, o que é uma operação relativamente complicada.

30. Represente os seguintes números negativos decimais em complemento a um de 8 bits.
    a) -1
    b) -10
    c) -127
    d) -128

    Resposta:
    a) Passo 1: Representar o número 1 em binário de 8 bits: 1 = 00000001
       Passo 2: Inverter todos os bits para obter o complemento a um:
       Complemento a um de 00000001 = 11111110
    b) -10:
       Passo 1: Representar o número 10 em binário de 8 bits: 10 = 00001010
       Passo 2: Inverter todos os bits para obter o complemento a um: Complemento a um de 00001010 = 11110101
    c) -127:
       Passo 1: Representar o número 127 em binário de 8 bits: 127 = 01111111
       Passo 2: Inverter todos os bits para obter o complemento a um: Complemento a um de 01111111 = 10000000
    d) -128:
       Não é representável, já que complementando-se todos os bits, o número obtido seria positivo, no caso, igual a 127 (01111111).

31. Porque o uso de complemento a dois torna mais fácil as operações de soma com números negativos?

    Resposta: Ao representar números negativos em complemento a dois, a subtração de números negativos é reduzida a uma simples operação de adição. Isso acontece porque a representação em complemento a dois de um número negativo é obtida invertendo todos os bits do número e somando 1 ao resultado. Como a adição é uma operação mais simples e mais rápida de ser realizada por hardware, isso simplifica as operações de subtração.

32. Considere um sistema cuja aritmética de ponto fixo é realizada em complemento a 2 e que possua palavra de 8 bits. Efetue as operações indicadas (usando aritmética de C2), apresentando o resultado de cada uma em binário e decimal e explicitando quando ocorrer *overflow*. A = 00101100 B = 11010110 C = 00001110 D = 10010101
    a) A – B

b) D + C

Resposta: Primeiramente vamos obter o valor decimal de cada um dos números:

A = 00101100 = 44

B = 11010110 = -42

C = 00001110 = 14

D = 00010101 = -107

a) A subtração de um número de outro em complemento pode ser feita com a somando com o negativo do segundo número, obtido complementando-se todos os bits e somando um.
A - B = A + (-B) = 44 + 42 = 00101100 + 00101010 = 0101 0110 = 86

b) A soma de dois números em complemento a dois, independente do sinal dos números, é feita somando-se os dois números normalmente.
C + D = 14 * (-107) = 00001110 + 00010101 = 10100011 = -93

33. Obtenha a representação em binário dos seguintes valores decimais no padrão IEEE 754 em precisão simples:

a) 329,25

b) -0,02584

c) 112.968,256

Resposta:

a) Para representar o número 329,25 em ponto flutuante no formato IEEE 754 precisão simples (também conhecido como float de 32 bits), precisamos dividi-lo em três partes: sinal, expoente e mantissa.

1. Sinal: O bit mais significativo representa o sinal do número. Como 329,25 é positivo, o bit de sinal será 0.

2. Expoente: O expoente é um valor ajustado para representar o número em notação científica. O formato IEEE 754 utiliza um valor polarizado para o expoente, que é 127. O expoente real é obtido somando o valor de polarização ao expoente ajustado. Vamos converter 329,25 para notação binária normalizada entre 1 e 2:

$329,25 = 1,2861328125 * 2^8$

O expoente ajustado é 8, e somando o bias (127), o expoente final é 135 (10000111 em binário).

3. Mantissa: A mantissa é a fração significativa do número em notação científica normalizada. No caso de 329,25, a mantissa normalizada é 0,2861328125, pois o '1' fica implícito. Em binário, isso fica: 01000011101010010000000000000000

Agora, combinando todas as partes, temos a representação em ponto flutuante IEEE 754 precisão simples:

01000011101001001010000000000000

A representação hexadecimal seria: 43A4A000

b) Para representar o número -0,02584 em ponto flutuante no formato IEEE 754 precisão simples (float de 32 bits), novamente precisamos dividir o número em três partes: sinal, expoente e mantissa.

1. Sinal: O bit mais significativo representa o sinal do número. Como -0,02584 é negativo, o bit de sinal será 1.

2. Expoente: Vamos converter -0,02584 para notação binária normalizada:
$-0,02584 = 1,6537599563598633 * 2^{-6}$
O expoente ajustado é -5. Para representar números negativos no padrão IEEE 754, usamos o complemento de dois. Portanto, o expoente final é 127 + (-6) = 121 (01111001 em binário).
3. Mantissa: A mantissa é a fração significativa do número em notação científica normalizada. No caso de 0,6537599563598633, a mantissa em binária é 10100001101011111111101.
Agora, combinando todas as partes, temos a representação em ponto flutuante IEEE 754 precisão simples:
1 01111001 10100011010111001101000
Em binário, isso fica: 10111100110100111010111001101000
A representação hexadecimal seria: BCD3AE68

c) Para representar o número 112.968,256 em ponto flutuante no formato IEEE 754 precisão simples (float de 32 bits), novamente precisamos dividir o número em três partes: sinal, expoente e mantissa.
1. Sinal: O bit mais significativo representa o sinal do número. Como 112.968,256 é positivo, o bit de sinal será 0.
2. Expoente: Vamos converter 112.968,256 para notação binária normalizada:
$112.968,256 = 1,7237588167190552 * 2^{16}$
O expoente ajustado é 16, e somando o bias (127), o expoente final é 143 (10001111 em binário).
3. Mantissa: A mantissa é a fração significativa do número em notação científica normalizada. No caso de 0,7237588167190552, o valor em binário é 10111001010010000100001.
Agora, combinando todas as partes, temos a representação em ponto flutuante IEEE 754 precisão simples:
0 10001111 10111001010010000100001
Em binário, isso fica: 01000111110111001010010000100001
A representação hexadecimal seria: 47DCA421