1 Codificação

Representação em base B é igual a $N = \sum_i a_i * B^i$, onde a base B designa a quantidade de algarismos distintos utilizados, por exemplo:

- base 10: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- base 2: { 0, 1 }
- base 8: { 0, 1, 2, 3, 4, 5, 6, 7 }

Na notação posicional: o peso do algarismo depende da posição dentro do número, o algarismo na posição zero tem peso $B^0 = 1$, os algarismos à esquerda da vírgula na posição P tem pesos $= B^P$.

Os algarismos a direita da vírgula tem pesos iguais a potências negativas de B:na base 10: $0,1 = 10^{-1}; 0,01 = 10^{-2}$

O valor gerado por um algarismo é dado pela multiplicação do algarismo pelo seu peso e o valor do número N é igual a soma dos produtos algarismo x peso. Por exemplo:

Base10 $123,45 = 1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 4 * 10^{-1} + 5 * 10^{-2}$

Base2 $101,01 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2}$

1.1 Bases Importantes em computação

A Base binária é utilizada em computadores por ser mais simples de armazenar e manipular.

- dois dígitos: { 0, 1 }
- dois valores de tensão: 0 = GND, 1 = VCC

A base Octal:

- oito algarismos: { 0, 1, 2, 3, 4, 5, 6, 7 }
- cada algarismo octal corresponde a um **número binário de 3 bits**, o que facilita a conversão binário para octal e vice-versa. Entretanto, a base octal não é muito utilizada pois:
  - É menor que a base decimal;
  - Para compor um byte (8 bits), dois dígitos octais não são suficientes e três dígitos excede o valor.

OCTAL	Binário
0	000
1	001

2	010
3	011
4	100
5	101
6	110
7	111

A Base Hexadecimal (base 16), é a mais popular em computação após a binária e usa exatamente dois dígitos para representar um byte:

- 16 algarismos: { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E }
- cada algarismo corresponde a um **número binário de 4 bits**, dois dígitos formam 1 byte ! A conversão binário/hexadecimal também é direta.

HEXA	Binário
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

1.2 Conversão entre bases

Diversos métodos permitem transformar uma representação de um número em uma base origem em uma base destino.

Método Polinomial: interpreta-se o número como um polinômio na aritmética da base destino. Por exemplo de binário para decimal:

- $100101_2 = 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 37_{10}$

Método das Divisões: determinam-ne os algarismos de menor peso (mais a direita) tomando-se o resto da divisão do número pela base destino.

Suponha $x = a_{n-1} * B_{n-1} + \ldots + a_1 B_1 + a_0 B_0$ na base B. Portanto se dividirmos x pela base B temos $x/B = a_{n-1} * B_{n-2} + \ldots + a_1 B_0$ com $resto = a_0$, ou seja, o resto calcula o algarismo mais a direita, que é o a_0. Repete-se o método sobre o quociente obtido até que este seja zero. Por exemplo, a conversão de 46_{10} na base 10 para a base binária:

$46/2 = 23$, resto: 0

$23/2 = 11$, resto: 1

$11/2 = 5$, resto: 1

$5/2 = 2$, resto: 1

$2/2 = 1$, resto: 0

$1/2 = 0$, resto: 1.

$46_{10} = 101110_2$

Método da substituição direta: nas bases que são potência de 2, como a octal e hexadecimal, a conversão para a base binária pode ser feita diretamente pela substituição de algarismos por grupos de dígitos binários. Por exemplo:

- Em binário 110011_2 é igual à $(110)(011) = 63_8$
- $001110101111 = (0011)(1010)(1111) = 3AF_{16}$

2 Códigos Binários

Qualquer informação no computador é representada por códigos binários: caracteres, números, símbolos, etc. A melhor forma de representar depende dos critérios de avaliação: menor tamanho, maior facilidade para manipulação, compatibilidade com padrões já estabelecidos, etc. Além disso, explorar redundâncias, apesar da perda em armazenamento, é muito útil para deteção e correção de erros e redução do consumo de potência.

Códigos Alfanuméricos

- A representação de informações textuais foi uma das primeiras formas de codificação utilizadas em computadores. Os primeiros códigos utilizavam 6 bits para representar as letras maiúsculas (26), os dígitos decimais (10) e uma série de caracteres especiais (ponto, vírgula, dois pontos, etc). Um destes códigos é o de Hollerith, utilizado pela IBM em cartões perfurados.

- EBCDIC (Extended Binary Coded Decimal Interchange Code): código utilizado pela IBM incluindo letras maiúsculas, minúsculas, caracteres especiais, etc.
- ASCII (American Standard Code for Information Interchange): código inicialmente de 7 bits, posteriormente extendido para 8 bits, é o mais utilizado correntemente. Inclui caracteres de controle (0 a 31), como tabulação, retorno de linha, ejeção de página, bip, e outros; dígitos decimais; caracteres maúsculos e minúsculos (65 - A, 66 - B; etc);
- Código BCD (Binary Coded Decimal) utilizado em sistemas computacionais para codificar algarismos decimais utilizando 4 bits. Usado para facilitar a entrada e saída para conversão de binário para decimal e exibição em display de segmentos.

Tamanho Fixo da codificação

Uma questão importante: Qual a quantidade de informação que podemos armazenar em uma palavra de n bits ?

$$2^n$$

Por exemplo, 1 bit gera $2^1 = 2$ códigos: 0 e 1; 2 bits geram $2^2 = 4$ códigos: 00,01,10,11.

Nos computadores, a unidade de representação da informação normalmente é múltiplo de 8 bits, o que equivale a $2^8 = 256$ códigos diferentes. Uma palavra de 32 bits podem representar 4.294.967.296 códigos, ou seja, aproximadamente 4 bilhões de códigos diferentes. Um dica: para número com muitos bits, você pode fazer rápidamente a conversão substituindo $2^{10} \simeq 10^3$. Portanto $2^{32} = 2^2 * 2^{10} * 2^{10} * 2^{10} = 4 * 10^9$.

Outra questão importante, dado uma certa quantidade x, quantos b bits são necessários ? Basta realizar a operação inversa:

$$b = \lceil log_2 X \rceil$$

Por exemplo, 84 elementos equivale a $\lceil log_2 84 \rceil = \lceil 6,392 \rceil = 7 bits$

3 Aritmética Binária

3.1 Adição sem sinal

A aritmética binária é bem mais simples que a aritmética decimal que ensina no primeiro grau. Qualquer soma maior que 1, irá propagar o bit de vai-um (carry em inglês) para o dígito a esquerda. Por exemplo, $11_{10} + 10_{10} = 1011_2 + 1010_2 = 10101_2 = 1 * 2^4 + 1 * 2^2 + 1 = 21_{10}$

$$\begin{array}{cccccc} & 1 & 0 & 1 & 0 & \leftarrow \\ & & 1 & 0 & 1 & 1 \\ + & & 1 & 0 & 1 & 0 \\ - & - & - & - & - & - \\ & 1 & 0 & 1 & 0 & 1 \end{array}$$

Podemos observar que é possível definir recursivamente a adição de duas palavras A e B de n bits cada. Seja S = A + B; então $s_i = (a_i + b_i + c_{i-1}) mod 2$ e $c_i = (a_i + b_i + c_{i-1}) div 2$, onde mod e div são os operadores de divisão inteira e resto, a_i representa o dígito i de A. A soma de 2 números de n bits pode gerar um número de n+1 bits.

Uma das limitações da implementação de circuitos digitais para aritmética é a escolha de um padrão fixo para representar os números. Suponha 8 bits seja escolhido. O maior inteiro será 255 ($11111111_2 = FF_{hexa}$). Ao somarmos A+B, onde A e B são maiores de 128, a soma gerada será maior que 255 e não poderá ser representada com 8 bits. Esta situação é chamada de OVERFLOW.

Suponha a representação de 4 bits. Como vimos no exemplo anterior, $A = 11_{10} = 1011_2$ e $B = 10_{10} = 1010_2$ podem ser representados com 4 bits. Porém A+B irá gerar um número maior que 15 (maior número de 4 bits). Se calculamos S=A+B em 4 bits, o resultado será 0101 como mostrado abaixo:

$$\begin{array}{cccccc} & 1 & 0 & 1 & 0 & \leftarrow \\ & & 1 & 0 & 1 & 1 \\ + & & 1 & 0 & 1 & 0 \\ - & - & - & - & - & - \\ & 1 & 0 & 1 & 0 & 1 \end{array}$$

Mas $0101_2 = 5_{10}$ será um código inválido, diferente do número 5 gerado por exemplo

3.2 Representação com sinal

3.2.1 Número com 1 bit de sinal com n bits

$$X = -1^{b_{n-1}} * \sum_{i=0}^{n-2} 2^i * b_i$$

onde b_{n-1}representa o sinal: 0 se o número é positivo e 1 se o número é negativo. Por exemplo, se temos $n = 4$, a faixa de números será:

$$\begin{bmatrix} -7 & \dots & -0 & +0 & \dots & +7 \\ 1111 & \dots & 1000 & 0000 & \dots & 0111 \end{bmatrix}$$

A vantagem desta representação é a facilidade de compreensão, pois foi assim que aprendemos aritmética. Porém esta representação possui uma grave desvantagem: diferentes regras para somar e subtrair, o que dificulta a implementação de um circuito. Além disso, são usados dois códigos para representar o número 0.

3.2.2 Complemento de 1 e de 2

Dado um N número de n-bits, os complementos são definidos por:

complemento de 1 $\overline{N} = -N + (2^n - 1)$

complemento de 2 $\widetilde{N} = -N + (2^n - 1) + 1 = -N + 2^n$

Uma dica, para calcular o complemento de 1 basta inverter os bits. Para calcular o complemento de 2 basta inverter os bits e somar 1.

Por exemplo, suponha n=4. Se N=7 então $\overline{N} = \overline{7_{10}} = \overline{0111} = 1000$ e $\widetilde{N} = \overline{N} + 1 = 1000 + 1 = 1001$.

As vantagens do complemento de 2 são: um único zero (0) e mesmas regras para soma e subtracao. A desvantagem é o fato de ser um código assimétrico.

A tabela abaixo sumariza as diferentes representações.

decimal	binario	dec. c/sinal	bin c/sinal	Compl1	Compl2
0	000	0	0 00	000	000
1	001	1	0 01	001	001
2	010	2	0 10	010	010
3	011	3	0 11	011	011
4	100	-0	1 00	111	
5	101	-1	1 01	110	111
6	110	-2	1 10	101	110
7	111	-3	1 11	100	101
		-4			100

Como podemos notar, o código complemento de 2 segue uma sequência em torno do 0:

$$\begin{bmatrix} -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ 100 & 101 & 110 & 111 & 000 & 001 & 010 & 011 \end{bmatrix}$$

Por exemplo, -2 + (+2) $=-2+(+2) = 110+(010) = \begin{array}{ccccc} & & 1 & 1 & 0 \\ + & & 0 & 1 & 0 \\ - & - & - & - & - \\ & 1 & 0 & 0 & 0 \end{array} =$
0

Soma em complemento de 2

Para somar X+Y, basta executar a operação de soma. Por exemplo 3+(-2) na representação de 3 bits:

$3 = 011$

$-2 = 110$

Portanto $3 + (-2) = \begin{array}{ccccc} & & 0 & 1 & 1 \\ + & & 1 & 1 & 0 \\ - & - & - & - & - \\ & 1 & 0 & 0 & 1 \end{array} = 001_2 = 1_{10}$

Subtração em complemento de 2

Para subtrair Y de X, ou seja, X-Y, basta calcular o complemento de 2 de Y e executar $X - Y = X + \widetilde{Y}$. Por exemplo 2-1=2+(-1) na representação de 3 bits:

$2 = 010$

$1 = 001e\ \widetilde{1} = \widetilde{001} = \overline{001} + 1 = 110 + 1 = 111 = -1_{10}$

Portanto $2 - 1 = \begin{array}{ccccc} & & 0 & 1 & 0 \\ + & & 1 & 1 & 1 \\ - & - & - & - & - \\ & 1 & 0 & 0 & 1 \end{array} = 001_2 = 1_{10}$

4 Representação de Ponto Flutuante

Uma representação usada é o formato IEEE 754 onde

sinal S	expoente E	mantissa M
1 bit	8 bits	23 bits

onde são usados 32 bits no total. A codificação tem o seguinte significado:

$$N = (-1)^s 2^{(e-127)} * 1.M$$

Por exemplo:

0	00000000	00000000000000000000000	$+0.0$
1	00000000	00000000000000000000000	-0.0
0	01111111	00000000000000000000000	$+1.0$
1	01111111	00000000000000000000000	-1.0
0	10000000	00000000000000000000000	$+2.0$
0	10000001	11100000000000000000000	$+7.5$
0	11111111	01001010100010000000000	NaN

Onde NaN =Not a Number, ou infinito.

5 Bibliografia

1. Notas de Aula do Professor Ricardo Jacobi
2. Notas de aula do curso 6004
3. Notas do curso ICS155 do Prof. Gupta