Arquitetura e Organização de Computadores

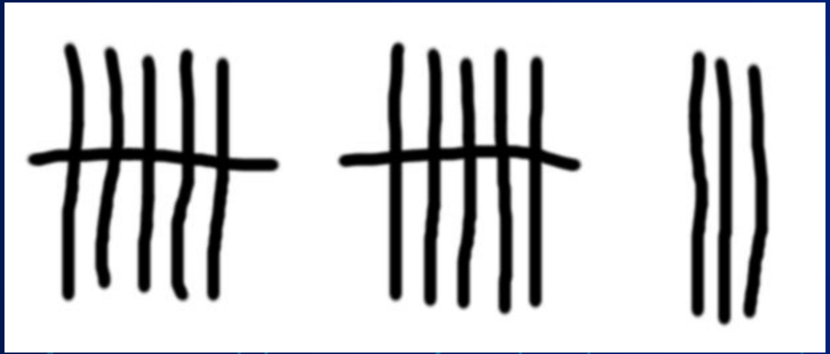
Uma Introdução

Gabriel P. Silva – José Antonio Borges

A informação e sua representação

Capítulo 1

1.1 Sistemas de Numeração



Pré-História

Muitas cavernas pré-históricas registram contagens, provavelmente de animais, na forma de riscos colocados um ao lado do outro, e agrupados por traços diagonais, para melhorar a leitura

Sistema de Contagem Sumério ou Babilônico

A contagem básica estendia-se até 60 (correspondendo a 5 dedos em uma mão e 12 falanges na outra).

Sistema de Contagem Sumério ou Babilônico

- Tanto naqueles povos antigos quanto no mundo de hoje, a contagem de pequenas quantidades poderia facilmente ser feita com uma ou duas mãos.
- Note que o uso da contagem até 60 é muito interessante, esse número é múltiplo de 2, 3, 4, 5 e 6 (além de outros), trazendo simplicidade para as operações aritméticas envolvendo divisão, quando realizadas mentalmente.

Odômetro Mecânico

É fácil perceber que o odômetro tradicional apresenta números formados de maneira idêntica aos que estamos acostumados a usar na nossa civilização, ou seja, são compostos por dígitos de 0 a 9, e os dígitos são colocados lado a lado, indicando as casas de unidades, dezenas, centenas, etc.

Representação Posicional de Números

$$372 = 3 \times 100 + 7 \times 10 + 2 \times 1$$

$$100 = 10^2;\ 10 = 10^1\ e\ 1 = 10^0$$

$$372 = 3 \times 10^2 + 7 \times 10^1 + 2 \times 10^0$$

Base Decimal

Exemplo									
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
...	...	...	...	...	...	...	...	...	...
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109

Representação Binária

$$000_{(2)} = 0_{(10)}$$
$$001_{(2)} = 1_{(10)}$$
$$010_{(2)} = 2_{(10)}$$
$$011_{(2)} = 3_{(10)}$$
$$100_{(2)} = 4_{(10)}$$
$$101_{(2)} = 5_{(10)}$$

Base Octal

Exemplo							
0	1	2	3	4	5	6	7
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
...	...	...	...	...	...	...	...
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77
100	101	102	103	104	105	106	107

Base Hexadecimal

$A_{(16)}$ equivale a $10_{(10)}$

$B_{(16)}$ equivale a $11_{(10)}$

$C_{(16)}$ equivale a $12_{(10)}$

$D_{(16)}$ equivale a $13_{(10)}$

$E_{(16)}$ equivale a $14_{(10)}$

$F_{(16)}$ equivale a $15_{(10)}$

Base Hexadecimal

Exemplo

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F

Conversão para a base 10

$$142_{(8)} = 1 \times 8^2 + 4 \times 8^1 + 2 \times 8^0$$

$$142_{(8)} = 1 \times 64 + 4 \times 8 + 2 \times 1$$
$$142_{(8)} = 64 + 32 + 2 = 98_{(10)}$$

Conversão para a base 10

$$1001_{(2)} = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$$
$$8 + 0 + 0 + 1 = 9_{(10)}$$

$$1C7A_{(16)} = 1 \times 16^3 + C \times 16^2 + 7 \times 16^1 + A \times 16^0$$

$$1C7A_{(16)} = 7290_{(10)}$$

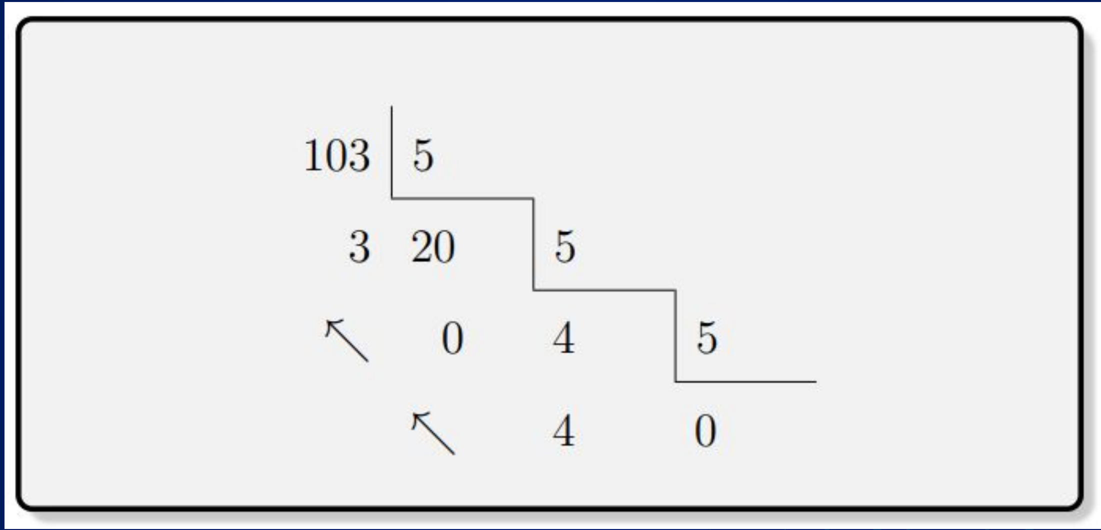
Conversão da base 10 para base 5

$$103/5 = 20, \text{ resto } 3$$
$$20/5 = 4, \text{ resto } 0$$
$$4/5 = 0, \text{ resto } 4$$

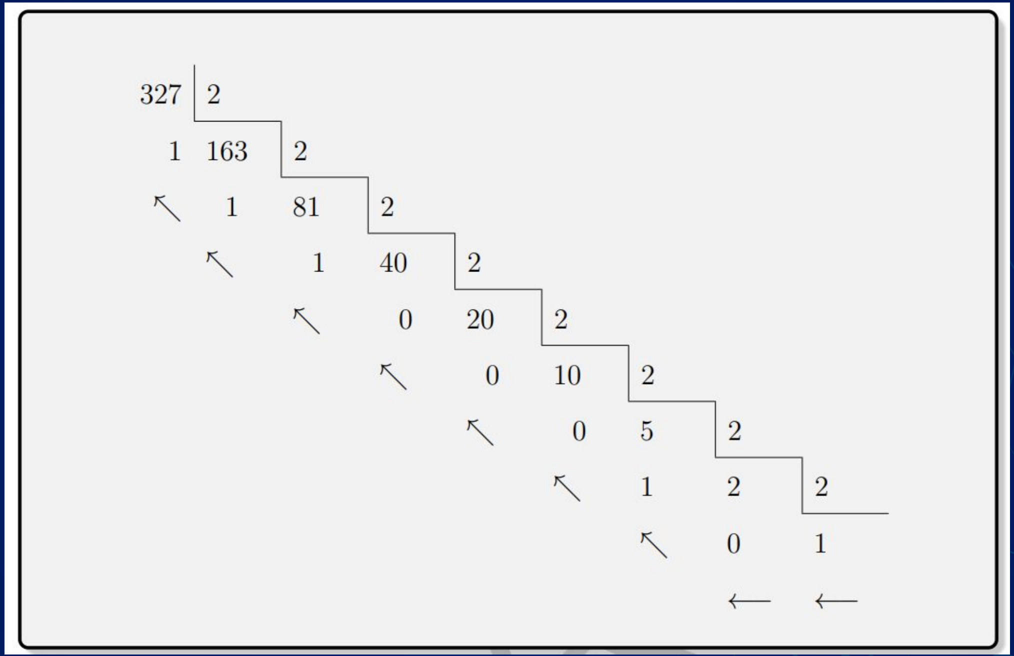
- Agora lemos os restos de trás para diante:

$$103_{(10)} = 403_{(5)}$$

Conversão da base 10 para base 5



Conversão da base 10 para base 2



$$327_{(10)} = 101000111_{(2)}$$

Conversão da base 10 para base 16

$$300/16 = 18, \text{ resto } 12 \rightarrow 12 \text{ vale "C"}$$
$$18/16 = 1, \text{ resto } 2$$
$$1/16 = 0, \text{ resto } 1$$

$$300_{(10)} = 12C_{(16)}$$

Conversão da base 8 para base 2

Exemplo							
00	00000	10	01000	20	10000	30	11000
01	00001	11	01001	21	10001	31	11001
02	00010	12	01010	22	10010	32	11010
03	00011	13	01011	23	10011	33	11011
04	00100	14	01100	24	10100	34	11100
05	00101	15	01101	25	10101	35	11101
06	00110	16	01110	26	10110	36	11110
07	00111	17	01111	27	10111	37	11111

Conversão da base 8 para base 2

$416_{(8)}$ = vale quanto na base 2?

4 escrito em binário com 3 dígitos $\longrightarrow$ 100
1 escrito em binário com 3 dígitos $\longrightarrow$ 001
6 escrito em binário com 3 dígitos $\longrightarrow$ 110

$$416_{(8)} = 100\,001\,110_{(2)}$$

Conversão da base 8 e 16 para base 2

Tabela Auxiliar								
Base 8	0	1	2	3	4	5	6	7
Base 2	000	001	010	011	100	101	110	111

Tabela Auxiliar			
Base 16	Base 2	Base 16	Base 2
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

Conversão da base 16 para base 2

$$3CF1_{(16)} = 0011\ 1100\ 1111\ 0001$$

- Foram deixados espaços em branco no meio do número binário que tem na verdade 16 dígitos em sequência. Isso foi feito para facilitar a leitura, de modo que você perceba claramente que foram usados os números da tabela.

Conversão genérica entre quaisquer bases

1. Converta o número na base B1 para a base 10.

$$317_{(8)} = 3 \times 8^2 + 1 \times 8^1 + 7 \times 8^0 = 207_{(10)}$$

Conversão genérica entre quaisquer bases

2. Depois converta este novo número na base 10 para a base B2.

$207/5 = 41$, resto 2

$41/5 = 8$, resto 1

$8/5 = 1$, resto 3

$1/5 = 0$, resto 1

$$317_{(8)} = 207_{(10)} = 1312_{(5)}$$

1.2 Operações Lógicas

Função NOT

Função NOT	
x	NOT x
0	1
1	0

- Se uma variável *x* tiver o valor ‘1’, NOT *x* valerá ‘0’. Se a variável *x* tiver o valor ‘0’, NOT *x* valerá ‘1’.

Função AND

Função AND		
x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

- Podemos resumir seu funcionamento numa tabela na qual o resultado da função será ‘1’, apenas quando ambas variáveis forem ‘1’.

Função NAND

Função NAND		
x	y	x NAND y
0	0	1
0	1	1
1	0	1
1	1	0

- A função NAND é muito utilizada e é equivalente a um AND seguido de um NOT em sua saída (tabela da verdade do AND com a saída invertida).

Função OR

Função OR		
x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

- Podemos resumir seu funcionamento numa tabela na qual o resultado da função será ‘0’, apenas quando ambas variáveis forem ‘0’.

Função NOR

Função NOR		
x	y	x NOR y
0	0	1
0	1	0
1	0	0
1	1	0

- A função NOR é também comum em eletrônica e é equivalente a um OR com um NOT na sua saída (tabela da verdade do OR com a saída invertida);

Função XOR

Função XOR		
x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

- Na função XOR o resultado é ‘1’ quando as variáveis forem diferentes. O XOR é uma comparação, com a indicação se as variáveis têm valores iguais ou diferentes.

Função XNOR

Função XNOR		
x	y	x XNOR y
0	0	1
0	1	0
1	0	0
1	1	1

- A função XNOR é equivalente a XOR seguido de um NOT em sua saída (tabela da verdade do XOR com a saída invertida).

1.3 Operações Aritméticas

Soma Decimal

$$\begin{array}{r} {\scriptstyle 1} \\ 23 \\ +39 \\ \hline \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 23 \\ +39 \\ \hline 2 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 23 \\ +39 \\ \hline 62 \end{array}$$

- É necessário o uso de uma tabuada, que por eficiência deve ser decorada.
- A conta é feita da direita para a esquerda. Mas poderia ser da esquerda para a direita?

Tabuada Soma Binária

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0 \text{ e vai } 1$$

- Explicando a última conta: $1 + 1 = 10_{(2)}$ (claro, o número $10_{(2)}$ vale 2 em decimal), ou em outras palavras, 0 e vai 1.

Soma Binária

$$\begin{array}{r} 01 \\ +11 \\ \hline \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 01 \\ +11 \\ \hline 0 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 01 \\ +11 \\ \hline 100 \end{array}$$

- Dígito mais à direita: 1 + 1 = 0 e vai 1.
- Próximo dígito: 1 + 0 + 1 = $10_{(2)}$ ou 2 em decimal, ou seja, ‘0’ e vai ‘1’.

Tabuada Subtração Binária

$0 - 0 = 0$

$0 - 1 = 1$ e pede emprestado da próxima casa

$1 - 0 = 1$

$1 - 1 = 1$ e pede emprestado da próxima casa

Subtração Binária

$$\begin{array}{r} 100 \\ -011 \\ \hline \end{array}$$

$$\begin{array}{r} {\scriptstyle 1} \\ 100 \\ -011 \\ \hline 1 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1} \\ 100 \\ -011 \\ \hline 001 \end{array}$$

- Da direita para a esquerda: 0 – 1 = 1 e emprestou da próxima casa.
- O resto é trivial e o resultado final é $001_{(2)}$.

Tabuada Multiplicação Binária

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Multiplicação Binária

$$
\begin{array}{rcr}
1010 & & 10 \\
1101 & \times & 13 \\
\hline
1010 & & \\
0000 & & \\
1010 & & \\
1010 & + & \\
\hline
10000010 & & 130
\end{array}
$$

Multiplicação por potência de 2

$$
\begin{array}{rcr}
10011011 & & 155 \\
100 & \times & 4 \\
\hline
00000000 & & \\
00000000 & & \\
10011011 & + & \\
\hline
1001101100 & & 620
\end{array}
$$

Para multiplicar um número binário por uma potência de dois, basta agregar ao final uma quantidade de zeros exatamente igual a esta potência.

Divisão Binária

- Aqui usamos o mesmo algoritmo de divisão usado na base decimal, subtraindo e deslocando o resultado para a direita.
- Os dois primeiros dígitos do dividendo são comparados com o divisor e, se o número for maior ou igual, é escrito 1 no quociente.
- Esse valor é multiplicado pelo divisor e subtraído dos dois primeiros dígitos.
- O processo se repete até chegar ao fim do dividendo, quando o resultado da subtração é o resto da divisão.

Divisão Binária

```
11011 | 11
      +-----
11      1001
000
 00
 001
  00
  011
   11
   00
```

Divisão por potência de 2

$$1001101101_{(2)} \div 10000_{(2)} = 100110_{(2)}$$

Para fazer uma divisão inteira de um número binário por uma potência de dois, basta retirar do final uma quantidade de bits exatamente igual a esta potência..

$$1001101101,_{(2)} \div 10000_{(2)} = 100110,1101_{(2)}$$

No caso da divisão fracionária, basta andar com a vírgula o mesmo número de casas. .

Conversão número binário fracionário

$$
\begin{aligned}
&1 \times 2^5 = 1 \times 100000_{(2)} + \\
&0 \times 2^4 = 0 \times 10000_{(2)} + \\
&0 \times 2^3 = 0 \times 1000_{(2)} + \\
&1 \times 2^2 = 1 \times 100_{(2)} + \\
&1 \times 2^1 = 1 \times 10_{(2)} + \\
&0 \times 2^0 = 0 \times 1_{(2)} + \\
&1 \times 2^{-1} = 1 \times 1/2 = 1 \times 0,1_{(2)} + \\
&1 \times 2^{-2} = 1 \times 1/4 = 1 \times 0,01_{(2)} + \\
&0 \times 2^{-3} = 0 \times 1/8 = 0 \times 0,001_{(2)} + \\
&1 \times 2^{-4} = 1 \times 1/16 = 1 \times 0,0001_{(2)}
\end{aligned}
$$

$$
\begin{aligned}
&1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-1} + 0 \times 2^{-3} + 1 \times 2^{-4} = \\
&32 + 0 + 0 + 4 + 2 + 0 + 0,5 + 0,25 + 0 + 0,0625 = 38,8125
\end{aligned}
$$

Conversão número decimal fracionário

- Quando o número tem uma parte fracionária, temos que fazer a conversão em dois passos:
  - Primeiro convertemos a parte inteira como já mostramos.
  - Depois, temos que usar um método específico para a conversão da parte à direita da vírgula, que é conhecido por “multiplicações sucessivas”.
- Multiplica-se a parte fracionária do número desejado pela base para a qual se deseja converter – neste caso, 2 – até que a mesma seja ZERO.
- O número convertido é igual a concatenação de todas as partes inteiras obtidas nos resultados das multiplicações.

Conversão número decimal fracionário

$$0,75 \times 2 = 1,50$$
$$0,50 \times 2 = 1,00$$

$$0,75_{(10)} = 0,11_{(2)}$$

1.4 Representação de Caracteres

Tabela ASCII

	Valores em Hexadecimal							
	0	1	2	3	4	5	6	7
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	‘ 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	” 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	’ 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

Tabela ISO/IEC 8859-1

Valores em Hexadecimal

	8	9	A	B	C	D	E	F
0	0080	0090	NBSP 00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
1	0081	0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	0082	0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	0083	0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	0084	0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	0085	0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	0086	0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	0087	0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	0088	0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	0089	0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	008A	009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	008B	009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	008C	009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	008D	009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	008E	009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	008F	009F	/ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Padrão Unicode

- O Unicode é um padrão que permite a codificação, representação e manipulação de textos de uma forma consistente na maioria dos sistemas de escrita do mundo.
- Esse padrão é mantido pelo Unicode Consortium, sendo que na versão de março de 2020 (Unicode 13.0) tinha um total de 143.859 caracteres, cobrindo 154 sistemas de escrita (scripts em inglês) modernos e históricos, além de vários conjuntos de símbolos e também emojis.
- O repertório de caracteres do padrão Unicode é sincronizado com a norma ISO/IEC 10646 (UCS - Universal coded character set) e ambos possuem códigos idênticos.

Codificação UTF-8

Comparaçao UTF			
Nome	UTF-8	UTF-16	UTF-32
Menor ponto de código	0000	0000	0000
Maior ponto de código	10FFFF	10FFFF	10FFFF
Tamanho da unidade do código	8 bits	16 bits	32 bits
Mínimo de bytes por caractere	1	2	4
Máximo de bytes por caractere	4	4	4

- Se a mensagem contiver apenas pontos de código da tabela ASCII, ela deve ter o mesmo tamanho da codificação em 8 bits - na verdade, por simplicidade, o código deve ser o mesmo;
- Os pontos de código do conjunto Unicode que não pertencessem à tabela ASCII seriam transformadas em um conjunto de 2, 3 ou 4 bytes.

Codificação UTF-8

UTF-8	
intervalo do código (hexadecimal)	sequência de bytes UTF-8 (binário)
0000 0000 –> 0000 007F	0xxxxxxx (7 bits)
0000 0080 –> 0000 07FF	110xxxxx 10xxxxxx (11 bits)
0000 0800 –> 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx (16 bits)
0001 0000 –> 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx (21 bits)

Exemplo					
caractere	Unicode	binário		UTF-8 (binário)	UTF-8 (hexa)
a	61	01100001	–>	01100001	61
ç	E7	11100111	–>	11000011 10100111	C3 A7
õ	F5	11110101	–>	11000011 10110101	C3 B5
e	6F	01101111	–>	01101111	6F
s	73	01110011	–>	01110011	7F

Padrão UCS

- O Universal coded character set (UCS) é um conjunto padrão de caracteres definido pela norma internacional ISO/IEC 10646 que é a base de muitas codificações de caracteres.
- A versão mais recente, de 2020, contém mais de 136.000 caracteres abstratos, cada um identificado por um nome não ambíguo e um número inteiro chamado ponto de código.
- Este padrão é mantido em conjunto com o padrão Unicode e ambos possuem códigos idênticos.

Padrão GB18030

- O GB18030 é um padrão do governo da República Popular da China que define o suporte de idioma e caracteres necessários para os softwares comercializados na China, que veio substituir o padrão GB2312/GBK.
- Como um formato de transformação Unicode (ou seja, uma codificação de todos os pontos de código Unicode), o GB18030 suporta caracteres chineses simplificados e tradicionais, isso inclui o padrão pré-existente GB2312/GBK, mais 6.582 caracteres do padrão Unicode Extension-A e 1.948 caracteres adicionais não incluídos no sistema de escrita Han (tais como Mongol, Uigur, Tibetano e Yi).

1.5 Representação de Inteiros

Representação em BCD

- Nesta forma de representação, cada dígito do número decimal é representado por um conjunto separado de 4 bits
- É comum que dois dígitos sejam agrupados por byte (8 bits), no que é conhecido como representação BCD compactada.

Tabela Decimal — BCD

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Representação em sinal magnitude

- Reserva o bit mais significativo (mais à esquerda) para representar se o número é positivo (igual a ‘0’) ou negativo (igual a ‘1’) e os demais bits para a magnitude.

$$+6 \rightarrow 00000110$$

$$-6 \rightarrow 10000110$$

$$+100 \rightarrow 01100100$$

$$-100 \rightarrow 11100100$$

Representação em excesso-K

- Deve-se subtrair um valor K da representação binária para se obter o valor real do número.
- Não existe um padrão para o valor de K, mas normalmente para um número com n bits é $K = 2^{n-1}$, por exemplo, o valor de K para um número binário de oito dígitos seria $2^7 = 128$.
- Como consequência um valor negativo mínimo representado por todos os bits em ‘0’, o valor “zero” é representado por um ‘1’ no bit mais significativo e ‘0’ em todos os outros bits, e o valor positivo máximo é representado por todos os bits em ‘1’.

Representação em complemento a um

- Na representação em complemento a um, todos os números positivos têm o bit mais significativo igual a ‘0’, e os números negativos são obtidos complementando-se, isto é, invertendo-se de ‘0’ para ‘1’ e de ‘1’ para ‘0’ todos os bits do respectivo número positivo.

$$+6 \rightarrow 00000110$$
$$-6 \rightarrow 11111001$$
$$+100 \rightarrow 01100100$$
$$-100 \rightarrow 10011011$$

Representação em complemento a dois

- Na representação em complemento a dois, os números positivos têm o bit mais significativo igual a ‘0’ e os números negativos tem o bit mais significativo igual a ‘1’.
- Contudo, os números negativos são obtidos complementando-se todos os bits do número positivo correspondente e somando-se 1.

$$+6 \rightarrow 00000110$$
$$-6 \rightarrow 11111010$$
$$+100 \rightarrow 01100100$$
$$-100 \rightarrow 10011100$$

Operações aritméticas em complemento a 2

- Soma
  - No caso da soma, dados dois números representados em complemento a dois, fazemos a soma normal em base 2.
  - Se o resultado ocupar mais de N bits, utilizamos apenas os N bits menos significativos, descartando o ‘1’ mais à esquerda.

- Subtração
  - As subtrações podem ser transformadas em somas.
  - O procedimento é alterar o sinal do subtraendo (calculando o seu complemento a dois) e somar esse valor com o minuendo.

Operações aritméticas em complemento a 2

- Soma

$$\begin{array}{r} 00001010 \\ 00000101 \\ \hline 00001111 \end{array} + \begin{array}{r} 10 \\ 5 \\ \hline 15 \end{array} \qquad \begin{array}{r} 00001010 \\ 00010100 \\ \hline 00011110 \end{array} + \begin{array}{r} 10 \\ 20 \\ \hline 30 \end{array}$$

$$\begin{array}{r} 01001010 \\ 11011010 \\ \hline (1)00100100 \end{array} + \begin{array}{r} 74 \\ -38 \\ \hline 36 \end{array} \qquad \begin{array}{r} 10100101 \\ 11100110 \\ \hline (1)10001011 \end{array} + \begin{array}{r} -91 \\ -26 \\ \hline -117 \end{array}$$

Operações aritméticas em complemento a 2

- Subtração

$$
\begin{array}{r}
00001010 \\
00000101 \\ \hline
\end{array}
\;-\;
\begin{array}{r}
10 \\
5 \\ \hline
\end{array}
\qquad\qquad
\begin{array}{r}
01001010 \\
11010110 \\ \hline
\end{array}
\;-\;
\begin{array}{r}
74 \\
-42 \\ \hline
\end{array}
$$

$$
\begin{array}{r}
00001010 \\
11111011 \\ \hline
(1)00000101
\end{array}
\;+\;
\begin{array}{r}
10 \\
-5 \\ \hline
5
\end{array}
\qquad\qquad
\begin{array}{r}
01001010 \\
00101010 \\ \hline
01110100
\end{array}
\;+\;
\begin{array}{r}
74 \\
42 \\ \hline
116
\end{array}
$$

1.6 Representação de Números Fracionários

Representação em ponto fixo

- Uma determinada quantia de bits é utilizada para representar os números, variando entre 8, 16, 32 ou 64 bits.
- Alguns desses bits são reservados para a parte inteira, e o restante para a parte fracionária.
- O número de bits utilizados para cada parte é arbitrado pelo programador.
- Ao longo deste texto chamaremos de N o número de bits reservados para representar cada número e de F o número de bits reservados para a parte fracionária (obviamente, N-F é o número de bits alocados para a parte inteira).

Representação em ponto fixo

- Um programador poderia arbitrar que, para um número representado com 32 bits, seriam utilizados 12 bits para a parte fracionária.

$$11001,01_{(2)} = 27,25_{(10)}$$

Parte fracionária f=12	
00000000000000000000000000011001	010000000000
20 bits	12 bits

Representação em ponto flutuante

- O formato mais utilizado atualmente é aquele estabelecido no padrão IEEE-754.
- A precisão do número a ser representado varia de acordo com o número de bits empregados na representação do número real podendo, basicamente, ser de 32 bits (precisão simples), 64 bits (precisão dupla) ou 128 bits (precisão quádrupla).
- A precisão simples equivale a um número com 7 dígitos decimais, a precisão dupla a um número com 15 dígitos decimais e a precisão quádrupla, 34 dígitos decimais.

Representação em ponto flutuante

$$N = s \times m \times 2^e$$

IEEE-754					
Precisão	Sinal (bits)	Expoente (bits)	Mantissa (bits)	Total (bits)	Excesso Expoente
Meia	1	5	10	16	15
Simples	1	8	23	32	127
Dupla	1	11	52	64	1023
Quádrupla	1	15	112	128	16383

Representação em ponto flutuante

- Os números positivos tem o bit de sinal s = 0 e os números negativos tem s = 1.
- O expoente é polarizado, isto é, é somado um valor fixo para sua representação: 127 no caso da precisão simples, 1023 no caso da precisão dupla e 16383 no caso de precisão quádrupla.
- A mantissa é sempre normalizada para um valor entre 1 e 2, sendo que somente a parte fracionária é representada, ficando o ‘1’ inicial implícito, ganhando-se assim um bit a mais de precisão.

Representação em ponto flutuante

Valores Especiais Ponto Flutuante			
Valor	Sinal	Expoente	Mantissa
Zero	0	0	0
+ Infinito	0	11...11	0
- Infinito	1	11...11	0
NaN	0	11...11	Diferente de 0

Conversão de ponto flutuante para decimal

Valor Inicial		
s	e	m
1	10000010	00110000000000000000000

1. Convertendo o expoente **100000010** $\longrightarrow$ **130**;
2. Despolarizando o expoente $\longrightarrow \mathbf{130 - 127 = 3}$;
3. Somando 1 à mantissa $\longrightarrow \mathbf{1,0011}$;
4. Desnormalizando de acordo com o expoente $\longrightarrow \mathbf{1001,1}$;
5. Convertendo para decimal $\longrightarrow \mathbf{9,5}$;
6. Adicionando o sinal $\longrightarrow \mathbf{-9,5}$.

Operações de soma e subtração

- Equalização dos expoentes dos operandos
- Soma ou subtração das mantissas dos operandos
- Normalização da mantissa do resultado

Soma de ponto flutuante			
	s	expoente	mantissa
	0	10000101	01110000000000000000000
+	0	10000011	00010010000000000000000

Operações de soma e subtração

- Ajuste do expoente

Passo a passo	
expoente	mantissa
10000011	00010010000000000000000
↓ +1	Desloca para a direita ↓
10000100	10001001000000000000000
↓ +1	Desloca para a direita ↓
10000101	01000100100000000000000

Operações de soma e subtração

- Soma das mantissas e resultado final

Soma das mantissas	
1,	01110000000000000000000
+ 0,	01000100100000000000000
1,	10110100100000000000000

Resultado		
s	expoente	mantissa
0	100000101	10110100100000000000000

Obrigado !

Arquitetura e Organização de Computadores: Uma Introdução

Mais recursos em:
https://simulador-simus.github.io