Tecnologias da Informação

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Organização e representação de dados em memória

Dados (texto, número, som, imagem, etc.) são armazenados na memória do computador em sequências de bits (*binary digits*), comumente representados por zeros e uns.

- Como os bits são interpretados, agrupados, e manipulados?
- Como os dados são representados no sistema binário?
- Como converter dados entre representações binária e decimal?

Bits e memória

- O bit é a menor unidade de dados e cada grupo de 8 bits forma uma célula de memória denominada *byte*.

Bits e memória

- O bit é a menor unidade de dados e cada grupo de 8 bits forma uma célula de memória denominada *byte*.

- A memória é organizada em uma sequência de células (uma variável pode ocupar uma ou mais células).

Bits e memória

- O bit é a menor unidade de dados e cada grupo de 8 bits forma uma célula de memória denominada *byte*.
- A memória é organizada em uma sequência de células (uma variável pode ocupar uma ou mais células).
- Um bit pode assumir dois estados, representados por *False* (zero) e *True* (um) em lógica boleana (George Boole, 1815–1864), e as operações boleanas básicas sobre os bits são *and*, *or*, *not*, e *xor*.

Bits e memória

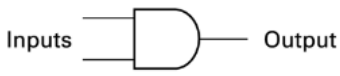
- O bit é a menor unidade de dados e cada grupo de 8 bits forma uma célula de memória denominada *byte*.
- A memória é organizada em uma sequência de células (uma variável pode ocupar uma ou mais células).
- Um bit pode assumir dois estados, representados por *False* (zero) e *True* (um) em lógica boleana (George Boole, 1815–1864), e as operações boleanas básicas sobre os bits são *and*, *or*, *not*, e *xor*.

A	B	A and B	A	B	A or B	A	not A	A	B	A xor B
0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	1	1	0	0	1	1
1	0	0	1	0	1			1	0	1
1	1	1	1	1	1			1	1	0

Bits e memória

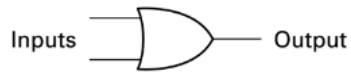
Os dispositivos (eletrônicos/óticos) que implementam as operações boleanas são denominados *gates*.

AND



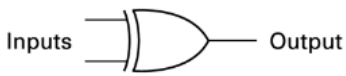
Inputs	Output
0 0	0
0 1	0
1 0	0
1 1	1

OR



Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	1

XOR



Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	0

NOT

Inputs	Output
0	1
1	0

Figura extraída de J. Glenn Brookshear and Dennis Brylow, Computer Science, An overview, 12th edition, Pearson, 2015.

Bits e memória

Os *gates* são agrupados em circuitos básicos denominados *flip-flops*, os quais podem ser usados para armazenar um bit em memória. Um pulso de 1 na parte superior gera 1 na saída, a qual é mantida mesmo quando o pulso retorna a 0. Um pulso de 1 na parte inferior retorna a saída para 0.

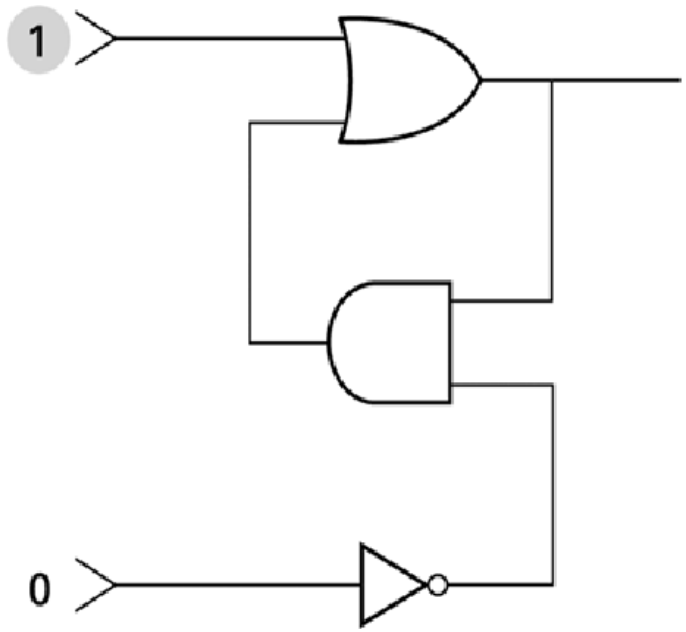


Figura extraída de J. Glenn Brookshear and Dennis Brylow, Computer Science, An overview, 12th edition, Pearson, 2015.

Bits e memória

Os *gates* são agrupados em circuitos básicos denominados *flip-flops*, os quais podem ser usados para armazenar um bit em memória. Um pulso de 1 na parte superior gera 1 na saída, a qual é mantida mesmo quando o pulso retorna a 0. Um pulso de 1 na parte inferior retorna a saída para 0.

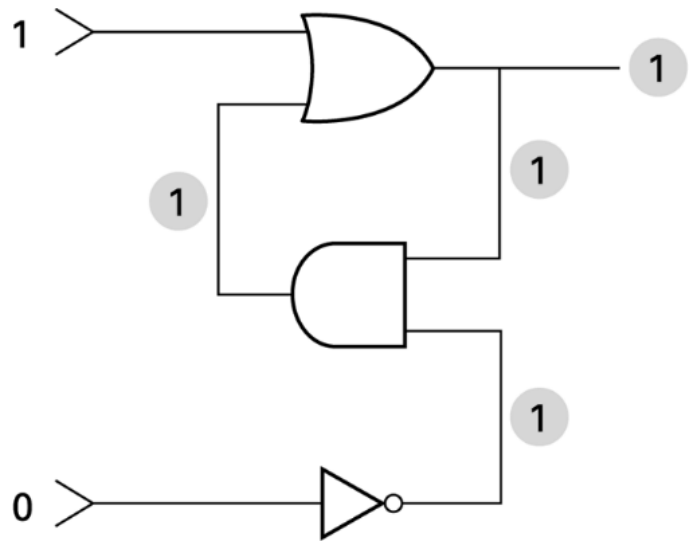


Figura extraída de J. Glenn Brookshear and Dennis Brylow, Computer Science, An overview, 12th edition, Pearson, 2015.

Bits e memória

Os *gates* são agrupados em circuitos básicos denominados *flip-flops*, os quais podem ser usados para armazenar um bit em memória. Um pulso de 1 na parte superior gera 1 na saída, a qual é mantida mesmo quando o pulso retorna a 0. Um pulso de 1 na parte inferior retorna a saída para 0.

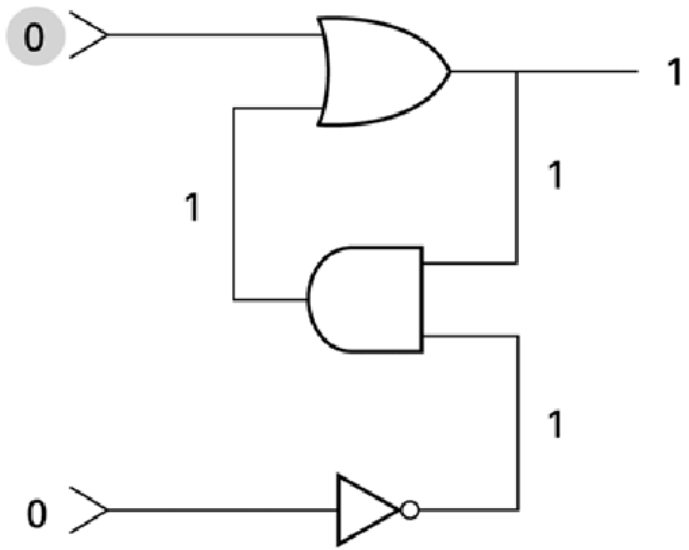


Figura extraída de J. Glenn Brookshear and Dennis Brylow, Computer Science, An overview, 12th edition, Pearson, 2015.

Bits e memória

- Os circuitos de um computador podem conter milhões de flip-flops organizados em um *chip*, usando uma tecnologia conhecida por *Very Large-Scale Integration* (VLSI).

Bits e memória

- Os circuitos de um computador podem conter milhões de flip-flops organizados em um *chip*, usando uma tecnologia conhecida por *Very Large-Scale Integration* (VLSI).

- As células de memória possuem endereços e, portanto, podem ser acessadas diretamente, de forma muito rápida, e reinicializadas, usando as tecnologias de memória DRAM e SDRAM (*Synchronous Dynamic Random Access Memory*).

Bits e memória

- Os circuitos de um computador podem conter milhões de flip-flops organizados em um *chip*, usando uma tecnologia conhecida por *Very Large-Scale Integration* (VLSI).

- As células de memória possuem endereços e, portanto, podem ser acessadas diretamente, de forma muito rápida, e reinicializadas, usando as tecnologias de memória DRAM e SDRAM (*Synchronous Dynamic Random Access Memory*).

- O número total de células é sempre uma potência de 2, sendo expresso em quantidades inteiras de 1024 (2^{10}) bytes. Por exemplo: 4 kilobytes $= 4 \times 1024 = 4096$ bytes, 4 megabytes $= 4 \times 1024^2$ bytes, 4 gigabytes $= 4 \times 1024^3$ bytes, 4 terabytes $= 4 \times 1024^4$ bytes, e 4 petabytes $= 4 \times 1024^5$ bytes.

Bits e memória

- Os circuitos de um computador podem conter milhões de flip-flops organizados em um *chip*, usando uma tecnologia conhecida por *Very Large-Scale Integration* (VLSI).

- As células de memória possuem endereços e, portanto, podem ser acessadas diretamente, de forma muito rápida, e reinicializadas, usando as tecnologias de memória DRAM e SDRAM (*Synchronous Dynamic Random Access Memory*).

- O número total de células é sempre uma potência de 2, sendo expresso em quantidades inteiras de 1024 (2^{10}) bytes. Por exemplo: 4 kilobytes $= 4 \times 1024 = 4096$ bytes, 4 megabytes $= 4 \times 1024^2$ bytes, 4 gigabytes $= 4 \times 1024^3$ bytes, 4 terabytes $= 4 \times 1024^4$ bytes, e 4 petabytes $= 4 \times 1024^5$ bytes.

- Os bits (dados) também podem ser armazenados de forma sequencial em arquivos em disco rígido, pen-drive, etc, dispositivos mais lentos com capacidade bem maior.

Representação binária dos dados

Como os dados são representados na memória?

- textos e padrão ASCII.
- números inteiros.
- números fracionários.
- sons e imagens.

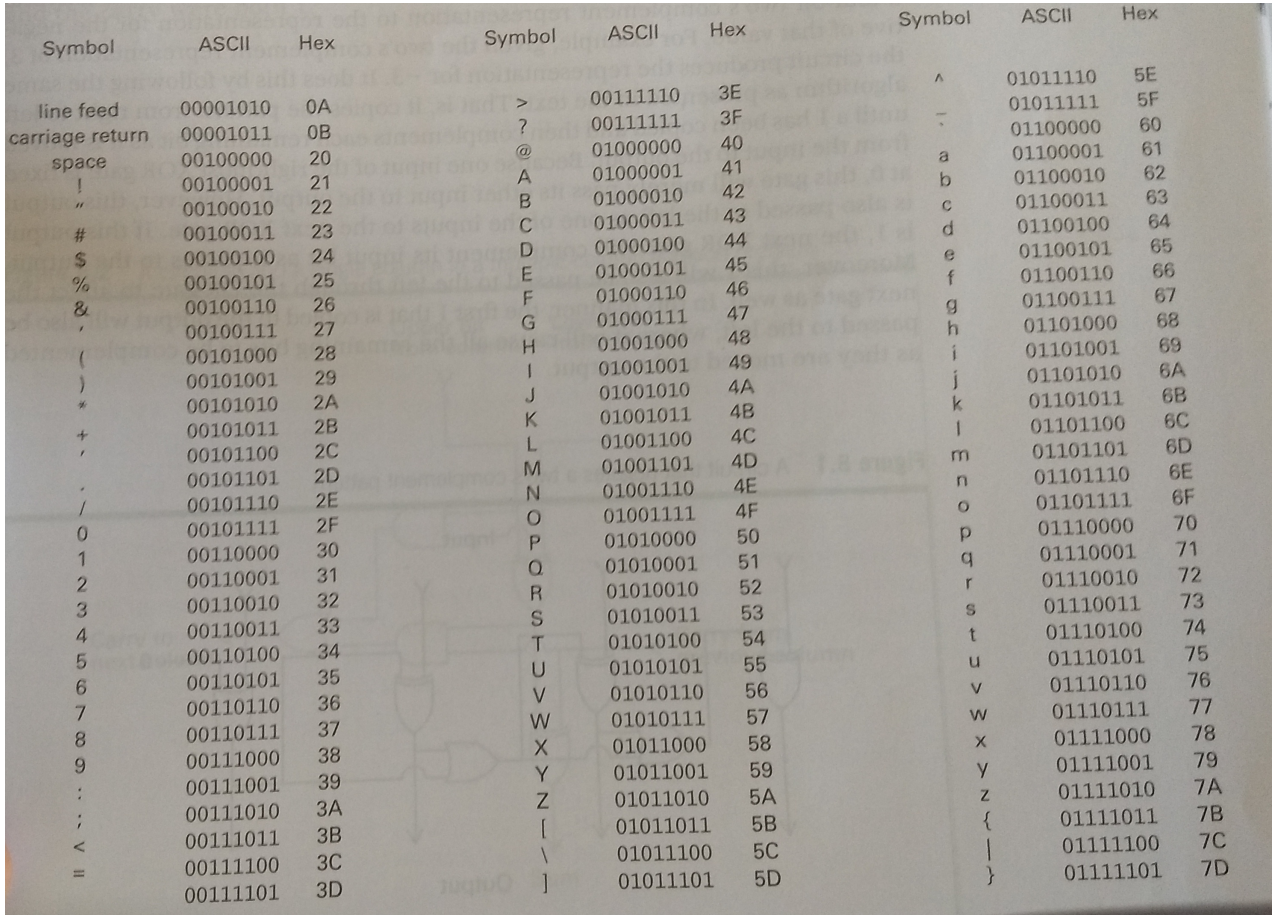
Representação binária de texto

O padrão ASCII (American Standard Code for Information Interchange) utiliza padrões de 7 bits para representar símbolos de controle via teclado e do alfabeto Inglês e o bit adicional para estender o padrão para 8 bits com símbolos de outras línguas.

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Representação binária de texto

O padrão ASCII (American Standard Code for Information Interchange) utiliza padrões de 7 bits para representar símbolos de controle via teclado e do alfabeto Inglês e o bit adicional para estender o padrão para 8 bits com símbolos de outras línguas.



Symbol	ASCII	Hex
line feed	00001010	0A
carriage return	00001011	0B
space	00100000	20
!	00100001	21
"	00100010	22
#	00100011	23
$	00100100	24
%	00100101	25
&	00100110	26
'	00100111	27
(	00101000	28
)	00101001	29
*	00101010	2A
+	00101011	2B
,	00101100	2C
-	00101101	2D
.	00101110	2E
/	00101111	2F
0	00110000	30
1	00110001	31
2	00110010	32
3	00110011	33
4	00110100	34
5	00110101	35
6	00110110	36
7	00110111	37
8	00111000	38
9	00111001	39
:	00111010	3A
;	00111011	3B
<	00111100	3C
=	00111101	3D
>	00111110	3E
?	00111111	3F
@	01000000	40
A	01000001	41
B	01000010	42
C	01000011	43
D	01000100	44
E	01000101	45
F	01000110	46
G	01000111	47
H	01001000	48
I	01001001	49
J	01001010	4A
K	01001011	4B
L	01001100	4C
M	01001101	4D
N	01001110	4E
O	01001111	4F
P	01010000	50
Q	01010001	51
R	01010010	52
S	01010011	53
T	01010100	54
U	01010101	55
V	01010110	56
W	01010111	57
X	01011000	58
Y	01011001	59
Z	01011010	5A
[	01011011	5B
\	01011100	5C
]	01011101	5D
^	01011110	5E
_	01011111	5F
`	01100000	60
a	01100001	61
b	01100010	62
c	01100011	63
d	01100100	64
e	01100101	65
f	01100110	66
g	01100111	67
h	01101000	68
i	01101001	69
j	01101010	6A
k	01101011	6B
l	01101100	6C
m	01101101	6D
n	01101110	6E
o	01101111	6F
p	01110000	70
q	01110001	71
r	01110010	72
s	01110011	73
t	01110100	74
u	01110101	75
v	01110110	76
w	01110111	77
x	01111000	78
y	01111001	79
z	01111010	7A
{	01111011	7B
|	01111100	7C
}	01111101	7D

Representação binária de texto

- O texto “casa” será codificado em 4 bytes como 01100011 01100001 01110011 01100001.

Representação binária de texto

- O texto “casa” será codificado em 4 bytes como 01100011 01100001 01110011 01100001.
- Em Python, por exemplo, o comando ord('#') converte o símbolo para o decimal correspondente, 35, e chr(35) faz o contrário.

Representação binária de texto

- O texto “casa” será codificado em 4 bytes como 01100011 01100001 01110011 01100001.
- Em Python, por exemplo, o comando ord('#') converte o símbolo para o decimal correspondente, 35, e chr(35) faz o contrário.
- Isso significa que ord('a')+5 é 102, onde chr(102) é 'f'.

Representação binária de texto

- O texto “casa” será codificado em 4 bytes como 01100011 01100001 01110011 01100001.
- Em Python, por exemplo, o comando ord('#') converte o símbolo para o decimal correspondente, 35, e chr(35) faz o contrário.
- Isso significa que ord('a')+5 é 102, onde chr(102) é 'f'.
- Uma extensão do ASCII que pode variar de 1 a 4 bytes é a codificação UTF-8, presente em várias linguagens tais como XML e HTML.

Representação binária de inteiros

- Em decimal, após contar de 0 a 9 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10.

Representação binária de inteiros

- Em decimal, após contar de 0 a 9 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10.

- Em binário, após contar de 0 a 1 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10 (o número 2).

Representação binária de inteiros

- Em decimal, após contar de 0 a 9 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10.
- Em binário, após contar de 0 a 1 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10 (o número 2).
- Desta forma, uma contagem 0,1,2,3,4,5,6,7,8,9, e 10 em decimal é representada em binário como 0,1,10,11,100,101,110,111,1000,1001, e 1010.

Representação binária de inteiros

- Em decimal, após contar de 0 a 9 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10.
- Em binário, após contar de 0 a 1 retornamos ao dígito 0 e acrescentamos o próximo dígito 1 na frente dele, ficando 10 (o número 2).
- Desta forma, uma contagem 0,1,2,3,4,5,6,7,8,9, e 10 em decimal é representada em binário como 0,1,10,11,100,101,110,111,1000,1001, e 1010.
- Então necessitamos de até b bits para representar números decimais de 0 a $2^b - 1$ (e.g., $b = 8$ bits representa de 0 a 255).

Representação binária de inteiros

- Para representar números negativos, um dos bits deve ser usado para o sinal (o bit mais à esquerda).

Representação binária de inteiros

- Para representar números negativos, um dos bits deve ser usado para o sinal (o bit mais à esquerda).
- Duas notações são complemento de 2, onde 0 indica sinal positivo e 1 indica sinal negativo, e excesso, onde 0 indica sinal negativo e 1 indica sinal positivo.

Representação binária de inteiros

- Para representar números negativos, um dos bits deve ser usado para o sinal (o bit mais à esquerda).
- Duas notações são complemento de 2, onde 0 indica sinal positivo e 1 indica sinal negativo, e excesso, onde 0 indica sinal negativo e 1 indica sinal positivo.
- Para 3 bits teremos:
  - Complemento de 2: 011=3, 010=2, 001=1, 000=0, 111=-1, 110=-2, 101=-3, e 100=-4.
  - Excesso: 111=3, 110=2, 101=1, 100=0, 011=-1, 010=-2, 001=-3, e 000=-4.

Representação binária de inteiros

- Para representar números negativos, um dos bits deve ser usado para o sinal (o bit mais à esquerda).
- Duas notações são complemento de 2, onde 0 indica sinal positivo e 1 indica sinal negativo, e excesso, onde 0 indica sinal negativo e 1 indica sinal positivo.
- Para 3 bits teremos:
  - Complemento de 2: 011=3, 010=2, 001=1, 000=0, 111=-1, 110=-2, 101=-3, e 100=-4.
  - Excesso: 111=3, 110=2, 101=1, 100=0, 011=-1, 010=-2, 001=-3, e 000=-4.
- Em Python, esses detalhes não aparecem. O comando bin(-3) mostra a representação binária de -3 como $-0b11$ e bin(3) mostra $0b11$. Desta forma, bin(255) mostrará 0b11111111 e bin(-255) mostrará -0b11111111, porém 9 bits são necessários para armazenar números de -255 a 255.

Conversão entre binário e decimal

- Em python, o comando int(-0b101) converte de binário para decimal, gerando o resultado -5 neste caso.

Conversão entre binário e decimal

- Em python, o comando int(-0b101) converte de binário para decimal, gerando o resultado -5 neste caso.

- Para um número binário $0bb_{n-1} \ldots b_1 b_0$, esta conversão é $b_0 2^0 + b_1 2^1 + \ldots + b_{n-1} 2^{n-1}$ com sinal positivo no caso. Para o caso acima, $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$, sendo que o sinal negativo gera -5.

Conversão entre binário e decimal

- Em python, o comando int(-0b101) converte de binário para decimal, gerando o resultado -5 neste caso.

- Para um número binário $0bb_{n-1}\ldots b_1 b_0$, esta conversão é $b_0 2^0 + b_1 2^1 + \ldots + b_{n-1} 2^{n-1}$ com sinal positivo no caso. Para o caso acima, $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$, sendo que o sinal negativo gera -5.

- Na conversão decimal para binário, os bits b_0, b_1, $\ldots$, b_{n-2} são os restos de sucessivas divisões inteiras por 2, do número (e depois dos quocientes), até o quociente $b_{n-1} = 1$. Por exemplo: $5\%2 = 1 = b_0$, o quociente é $5//2 = 2$, depois $2\%2 = 0 = b_1$, o quociente é $2//2 = 1 = b_2$, formando o número $0bb_2 b_1 b_0 = 0b101$.

Representação binária de números fracionários

- Para representar números fracionários, o bit mais significativo também é usado para o sinal e os demais bits são divididos em campos denominados expoente e mantisa.

Representação binária de números fracionários

- Para representar números fracionários, o bit mais significativo também é usado para o sinal e os demais bits são divididos em campos denominados expoente e mantisa.

- Os números podem ser representados em padrões de 4 a 8 bytes. Para entender a conversão para fracionário, vamos considerar o exemplo de um byte tal que para $b_7 b_6 \ldots b_1 b_0$, $b_7 \in \{0, 1\}$ é o bit de sinal, $b_6 b_5 b_4$ é o expoente, e $b_3 b_2 b_1 b_0$ é a mantissa.

Representação binária de números fracionários

- Para representar números fracionários, o bit mais significativo também é usado para o sinal e os demais bits são divididos em campos denominados expoente e mantisa.

- Os números podem ser representados em padrões de 4 a 8 bytes. Para entender a conversão para fracionário, vamos considerar o exemplo de um byte tal que para $b_7 b_6 \ldots b_1 b_0$, $b_7 \in \{0, 1\}$ é o bit de sinal, $b_6 b_5 b_4$ é o expoente, e $b_3 b_2 b_1 b_0$ é a mantissa.

- A mantissa é primeiro expressa como $.b_3 b_2 b_1 b_0$ e o sinal do expoente indica se o ponto será deslocado para a direita (positivo) ou para a esquerda (negativo). Por exemplo, em notação de excesso, para 11101011, a mantissa 1011 fica .1011 e o expoente é 110 é $+2$, deslocando o ponto para a direita, 10.11.

Representação binária de números fracionários

- Para 10111100, a mantissa 1100 fica .1100 e o expoente 011 é -1 em notação de excesso, deslocando o ponto para a esquerda, .01100.

Representação binária de números fracionários

- Para 10111100, a mantissa 1100 fica .1100 e o expoente 011 é -1 em notação de excesso, deslocando o ponto para a esquerda, .01100.

- Após expressar a mantissa com o ponto deslocado, os bits $b'_1 b'_2 b'_3 \ldots$ após o ponto são multiplicados por frações $2^{-1}, 2^{-2}, 2^{-3}, \ldots$, os bits $\ldots b''_2 b''_1, b''_0$ antes do ponto são multiplicados por $\ldots, 2^2, 2^1, 2^0$, e o total é a representação fracionária do número.

Representação binária de números fracionários

- Para 10111100, a mantissa 1100 fica .1100 e o expoente 011 é -1 em notação de excesso, deslocando o ponto para a esquerda, .01100.

- Após expressar a mantissa com o ponto deslocado, os bits $b'_1 b'_2 b'_3 \ldots$ após o ponto são multiplicados por frações $2^{-1}, 2^{-2}, 2^{-3}, \ldots$, os bits $\ldots b''_2 b''_1, b''_0$ antes do ponto são multiplicados por $\ldots, 2^2, 2^1, 2^0$, e o total é a representação fracionária do número.

- Em 10.11 teremos $1 \times 2^1 + 0 \times 2^0 = 2$ e $1 \times 2^{-1} + 1 \times 2^{-2} = \frac{3}{4}$, que é 2.75. Em 0.01100 teremos $0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5}$, que é $\frac{3}{8}$.

Representação de sons e imagens

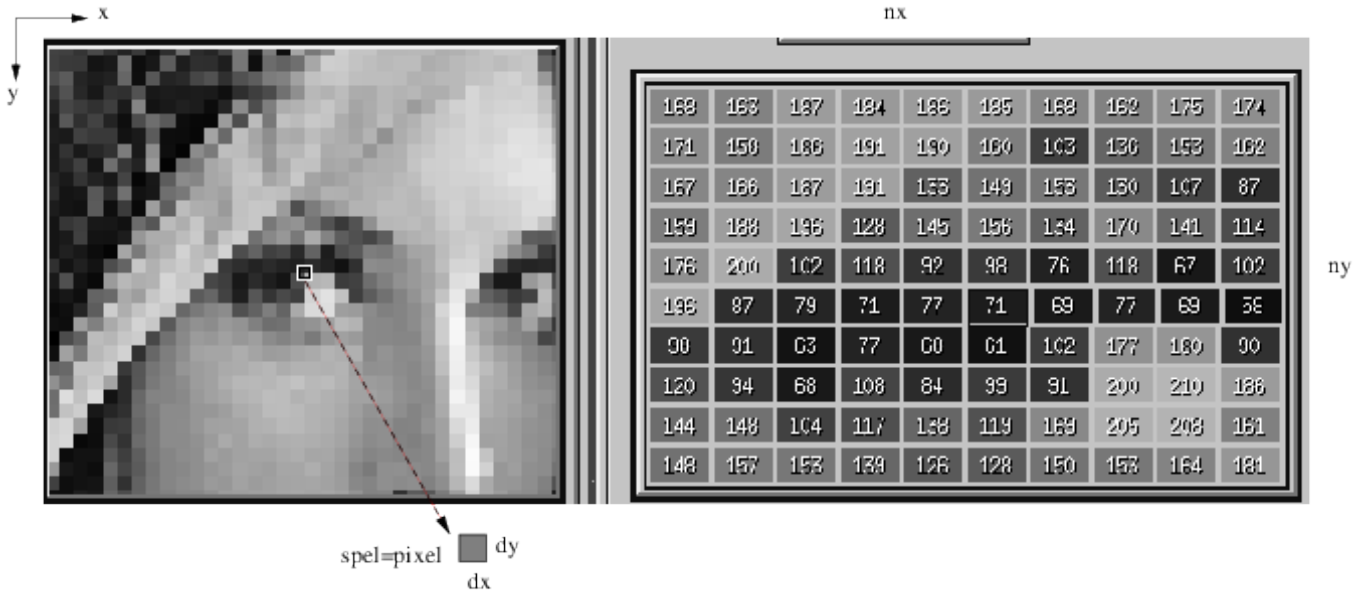
- No caso de sons, a amplitude do som $f(t)$, convertida em sinal elétrico, é amostrada em n pontos separados por intervalos de tempo δ_t, gerando uma sequência $f(t_0)$, $f(t_1)$, $\ldots$, $f(t_{n-1})$ de valores, que podem ser armazenados como números fracionários ou, mais comumente, como números inteiros em $[0, 2^b - 1]$, onde b é o número de bits por ponto.

Representação de sons e imagens

- No caso de sons, a amplitude do som $f(t)$, convertida em sinal elétrico, é amostrada em n pontos separados por intervalos de tempo δ_t, gerando uma sequência $f(t_0)$, $f(t_1)$, $\ldots$, $f(t_{n-1})$ de valores, que podem ser armazenados como números fracionários ou, mais comumente, como números inteiros em $[0, 2^b - 1]$, onde b é o número de bits por ponto.

- No caso de imagens, a refletância de uma cena $f(x, y)$ pode ser amostrada em $n_x \times n_y$ pixels (*picture elements*), separados em intervalos δ_x e δ_y ao longo dos eixos espaciais x e y, gerando uma matriz de valores $f(x_0, y_0)$, $f(x_1, y_0)$, $\ldots$, $f(x_{n_x-1}, y_0)$, $f(x_0, y_1)$, $f(x_1, y_1)$, $\ldots$, $f(x_{n_x-1}, y_1)$, $f(x_0, y_{n_y-1})$, $f(x_1, y_{n_y-1})$, $\ldots$, $f(x_{n_x-1}, y_{n_y-1})$, comumente armazenados como inteiros em $[0, 2^b - 1]$, onde b é o número de bits por pixel (profundidade da imagem).

Representação de sons e imagens



No caso de imagens coloridas, cada componente de cor (vermelho, verde, e azul) é representado por uma matriz como esta.