Memória interna

MAC 344 - Arquitetura de Computadores
Prof. Siang Wun Song

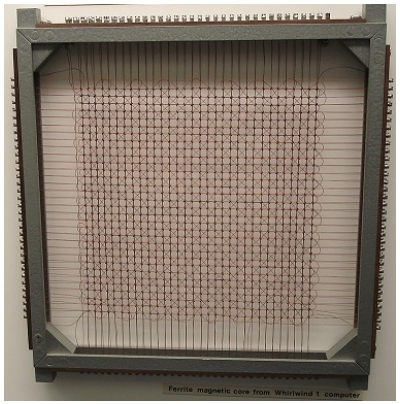
Baseado parcialmente em W. Stallings -

Computer Organization and Architecture

Memória magnética de núcleo de ferrite

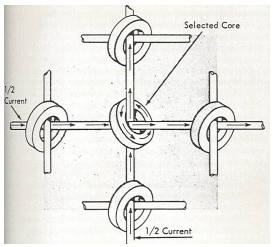
- Nos computadores antigos, a chamada memória interna RAM era feita de núcleos de ferrite (*magnetic-core memory*). (Daí o nome *core memory* usado até hoje para indicar memória interna.)

Source: Science Museum, London



- Dependendo de como o núcleo de ferrite é magnetizado, ele representa 0 ou 1.

Source: IBM Early Computers, MIT Press



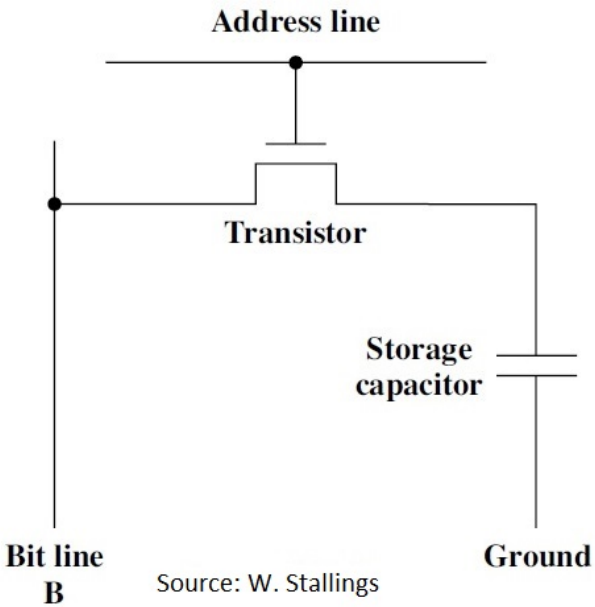
- A memória de núcleo de ferrite é do tipo não-volátil: o valor armazenado não se perde quando a energia é desligada e depois religada.

Memória interna DRAM e SRAM

Hoje a memória RAM é feita de semi-condutor (Silício).

Pode ser de dois tipos: DRAM e SRAM, ambos os tipos voláteis (o seu conteúdo se perde quando o computador é desligado e depois religado).

- DRAM (*Dynamic RAM*): usada na memória principal.
- O capacitor armazena ou não carga elétrica, representando 1 e 0, resp.
- Quando carregado, o capacitor pode perder carga por vazamento.
- Para manter um capacitor que representa 1 sempre carregado, um pulso de refrescamento é aplicado periodicamente. Daí o nome dinâmico.



Source: W. Stallings

Address line controla o transistor para permitir o acesso ao capacitor.

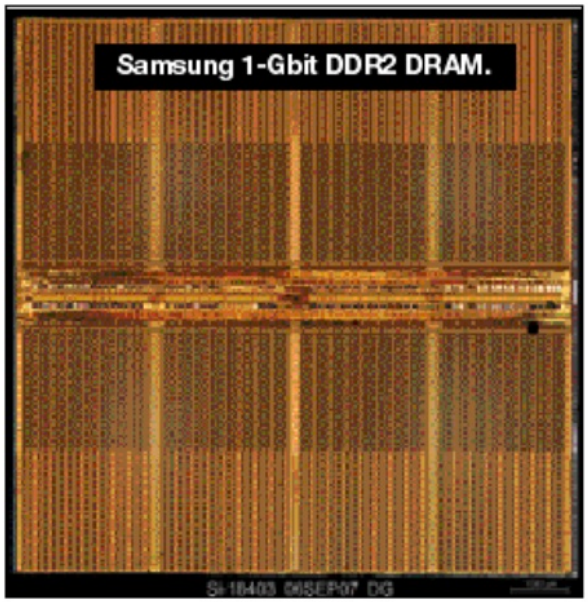
Para escrever 1, usa-se uma voltagem alta no Bit line. Para escrever 0, usa-se uma voltagem baixa.

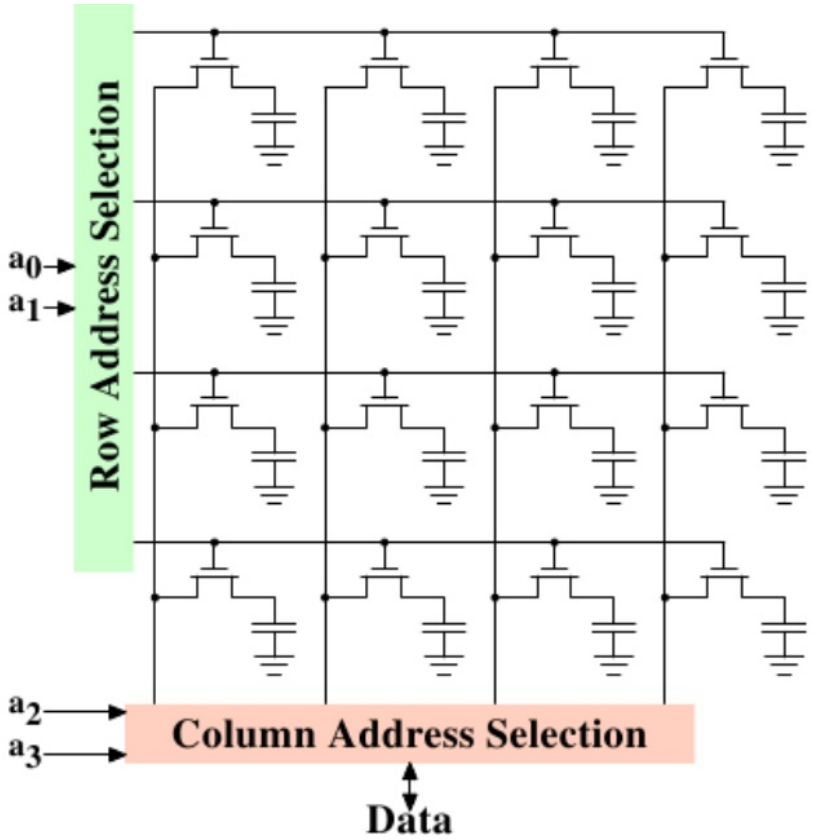
Para ler, Bit line usa a carga do capacitor para ver se é está carregado ou não. A leitura descarrega o capacitor, cuja carga deve ser restaurada.

Memória interna DRAM

Exemplo: Samsung 1-Gbit DRAM. Note a regularidade na disposição dos bits da memória DRAM, permitindo uma maior densidade (i.e. mais bits por unidade de área do Silício).

Modern DRAM chip with 8 internal memory banks.

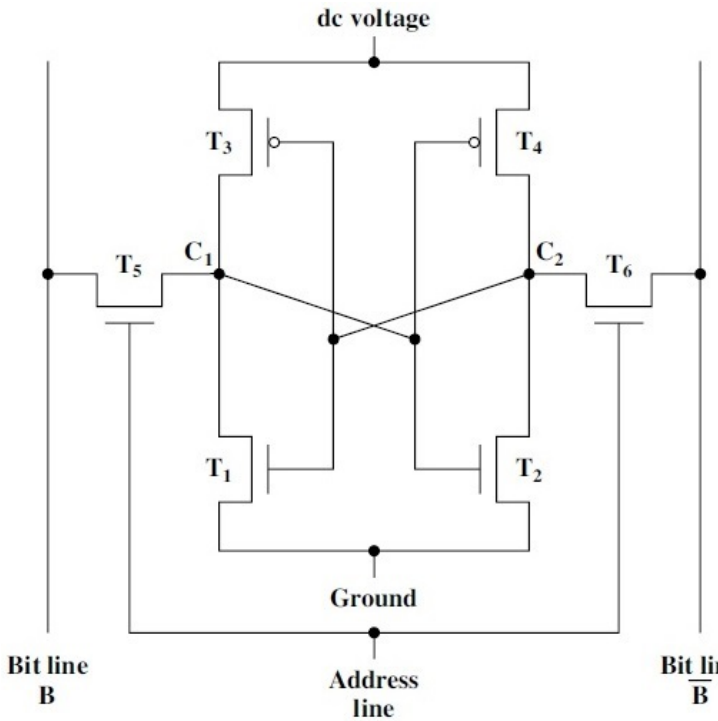




Source: https://www.cl.cam.ac.uk/teaching/1718/SysOnChip/materials.d/kg1-energy/zhp343475fdd.html

Memória interna SRAM

- SRAM (*Static RAM*): usada na memória cache, menos densa, mais rápida e mais custosa do que DRAM.
- A memória estática mantém o dado inalterado, desde que haja energia.
- Uma célula para 1 bit é composta de 4 transistores (T_1, T_2, T_3, T_4) conectados de tal modo que mantêm sempre um de dois estados lógicos estáveis.
- No estado 1, C_1 é alto e C_2 é baixo, com T_1 e T_4 desligados e T_2 e T_3 ligados.
- No estado 0, C_1 é baixo e C_2 é alto, com T_1 e T_4 ligados e T_2 e T_3 desligados.
- Ambos estados são estáveis, desde que haja uma voltagem direta DC aplicada. Assim, SRAM também é volátil.



Address line controla os transistores T5 e T6 para permitir escrita ou leitura da memória.

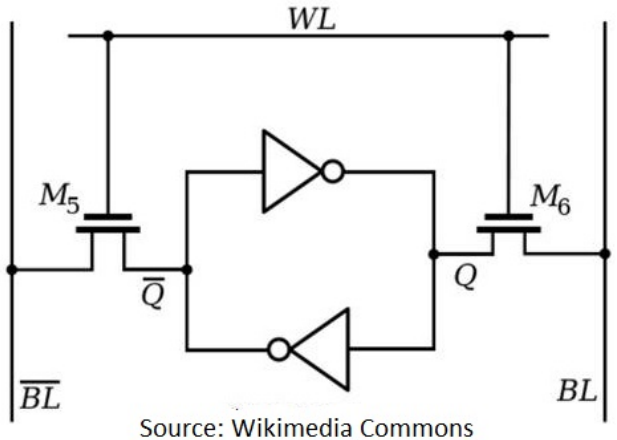
Para escrever, o valor (1 ou 0) é aplicado a Bit line, para definir um dos 2 estados estáveis.

Para ler, o valor do bit é lido na Bit line.

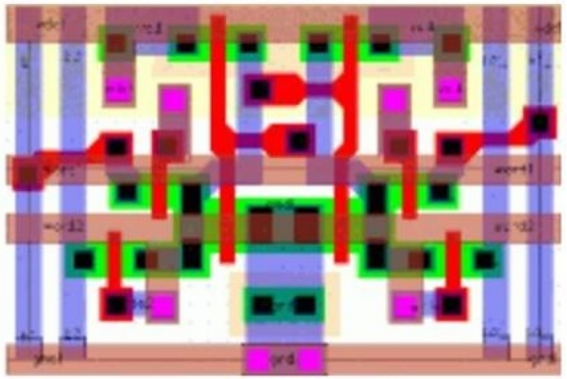
Source: W. Stallings

Memória interna SRAM

Uma outra forma de repressentar uma célula (um bit) SRAM: por meio de duas portas inversoras (portas NÃO). A figura da direita mostra uma célula de um bit em CMOS.



Source: Wikimedia Commons


https://iis-people.ee.ethz.ch/~kgf/aries/5.html

- Há dois estados estáveis:
  - $Q = 1$ e $\overline{Q} = 0$
  - $Q = 0$ e $\overline{Q} = 1$
- Para ler:
  - Linha *WL* alta liga transistores *M*5 e *M*6
  - Valor Q é transmitido para BL (e valor $\overline{Q}$ para $\overline{BL}$)
- Para escrever:
  - Valor desejado (1 ou 0) é armazenado em BL e o complemento em $\overline{BL}$
  - Linha *WL* alta liga transistores *M*5 e *M*6

Memória interna DRAM e SRAM

Comparação entre DRAM e SRAM.

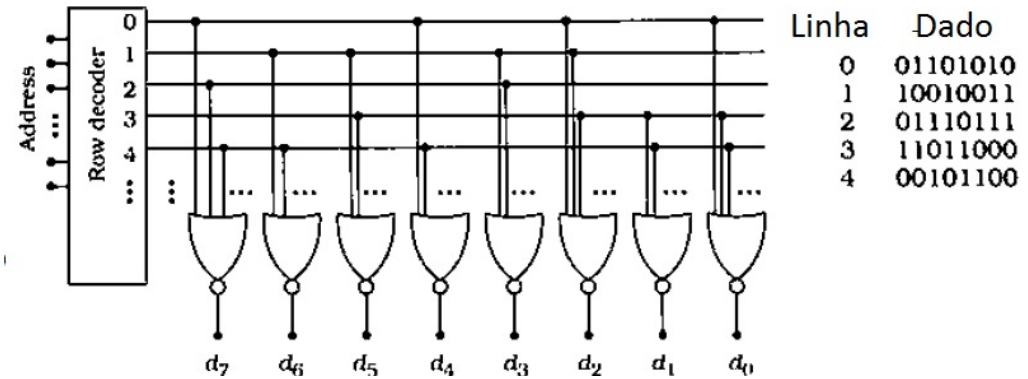
- Ambas são voláteis.
- A célula DRAM é mais simples e ocupa menos espaço que uma célula SRAM.
- Portanto DRAM é mais densa (mais células por unidade de área) e mais barata.
- Por outro lado, DRAM requer uma circuitaria de refrescamento. Para memórias grandes, esse custo fixo é mais que compensado pelo menor custo.
- Daí DRAM é preferida para memórias grandes e SRAM (que é um pouco mais rápida) é mais usada em memória cache.

Tipos de ROM (*Read Only Memory*)

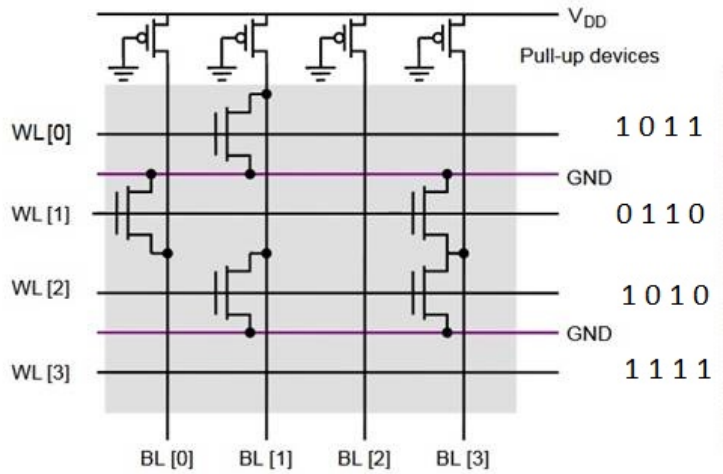
- ROM é uma memória cujo conteúdo é fixo e não pode ser alterado.
- Há vários tipos de ROMs: todos são não-voláteis, i.e. não requerem energia para manter o seu conteúdo.
- Um importante uso de ROM é em processador CISC para armazenar o microprograma.
- ROM pode ser fabricado com portas NOR ou NAND, com um *layout* denso.
- Como ROM não pode ser alterada, erro de um bit pode acarretar em discartar um lote inteiro.

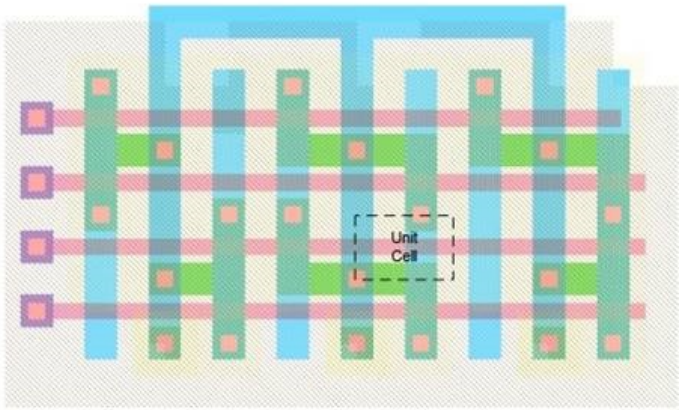
ROM baseado em portas NOR

- Na ROM baseada em NOR, o endereço entra num decodificador e ativa uma das linhas de saída do decodificador: a linha ativada contém 1 e todas as demais 0.
- Se essa linha entra no NOR, a saída é 0, senão é 1.



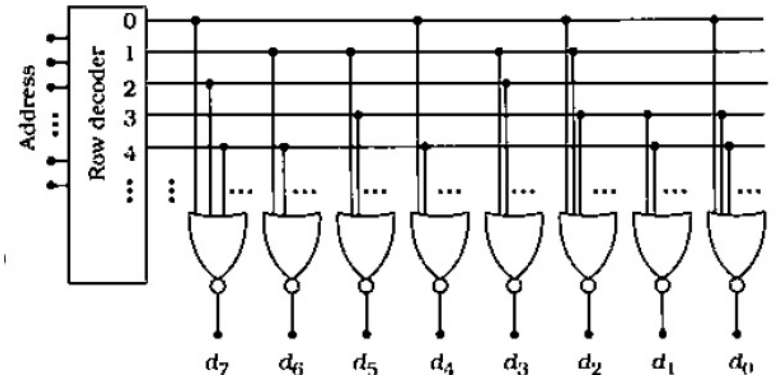
Linha	Dado
0	01101010
1	10010011
2	01110111
3	11011000
4	00101100



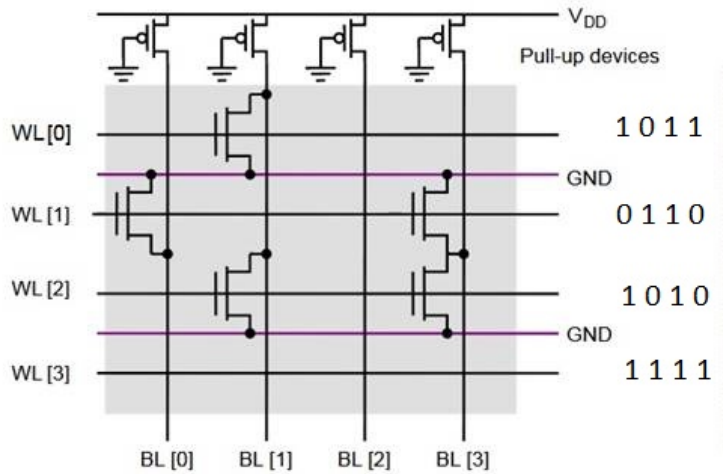


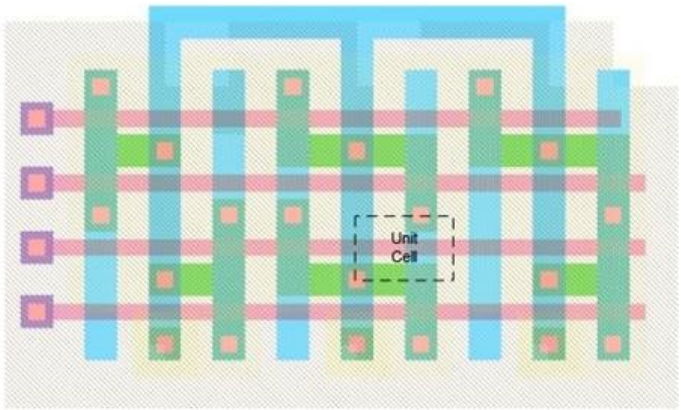
ROM baseado em portas NOR

- A ROM é não-volátil. Mas quando a energia é desligada tudo se apaga.
- Vocês podem explicar então por que a ROM é não-volátil?
- É porque quando a energia é religada os valores estão novamente disponíveis, por estarem codificados na entrada das portas.



Linha	Dado
0	01101010
1	10010011
2	01110111
3	11011000
4	00101100





Como está o meu aprendizado?

- Projete uma ROM com 4 linhas usando portas NAND.

Linha	Dado			
	D_0	D_1	D_2	D_3
l_0	1	0	1	1
l_1	0	1	1	0
l_2	1	0	1	0
l_3	1	1	1	1

A linha endereçada (por exemplo l_2) vale 0, todas as demais valem 1.

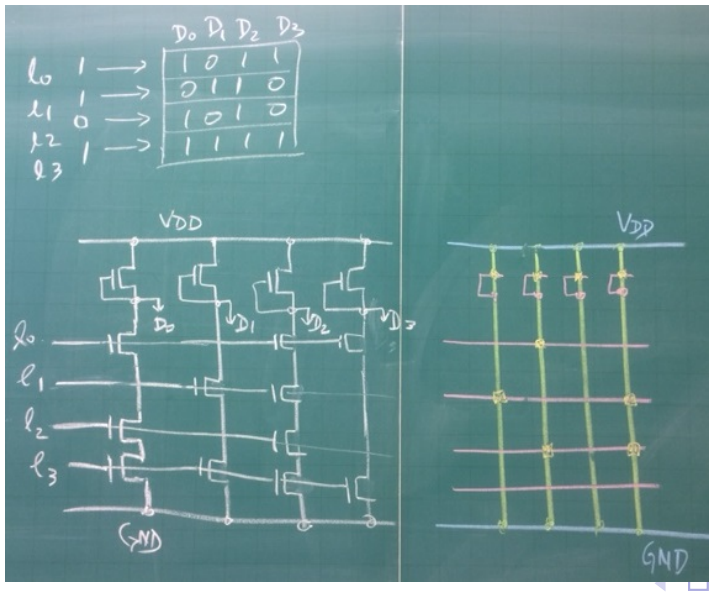
Solução no próximo slide.

Como está o meu aprendizado?

- Projete uma ROM com 4 linhas usando portas NAND.

Linha	Dado			
	D_0	D_1	D_2	D_3
l_0	1	0	1	1
l_1	0	1	1	0
l_2	1	0	1	0
l_3	1	1	1	1

Solução: a linha endereçada (por exemplo l_2) vale 0, todas as demais valem 1. A palavra correspondente a l_2 tem como conteúdo 1 0 1 0.

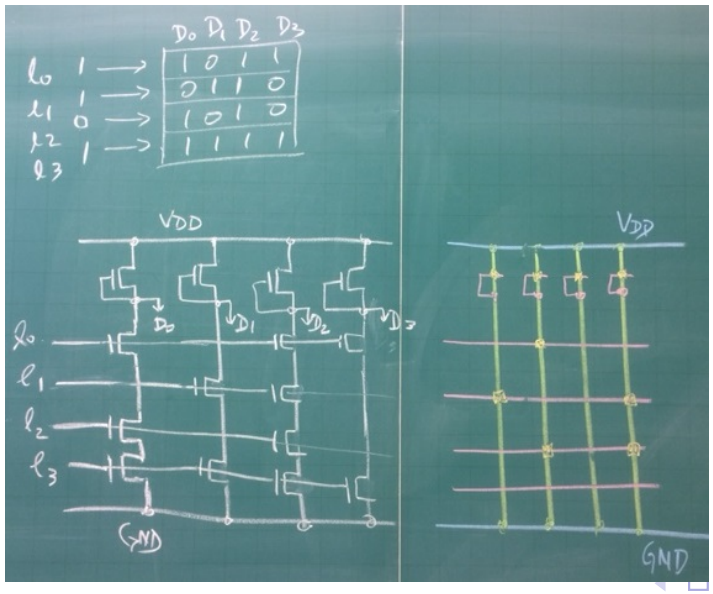


Como está o meu aprendizado?

- Projete uma ROM com 4 linhas usando portas NAND.

Linha	Dado			
	D_0	D_1	D_2	D_3
l_0	1	0	1	1
l_1	0	1	1	0
l_2	1	0	1	0
l_3	1	1	1	1

Solução: a linha endereçada (por exemplo l_2) vale 0, todas as demais valem 1. A palavra correspondente a l_2 tem como conteúdo 1 0 1 0.

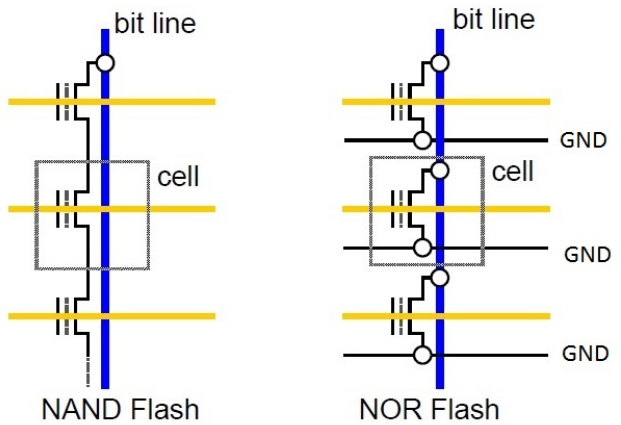


Tipos de ROM (*Read Only Memory*)

- PROM (*Programable ROM*): como ROM, também pode ser escrita uma só vez. A escrita é por meio elétrico e pode ser feita depois de fabricada a pastilha.
- PROM oferece mais flexibilidade, mas ROM ainda é preferível para grandes quantidades.
- EPROM (*Erasable Programable ROM*): leitura e esrita é como numa PROM. Porém, antes de uma operação de escrita, toda a memória é apagada por meio de radiação ultra-violeta. Esse processo pode ser repetido para gravar um novo conteúdo.
- EPROM é mais custosa.
- EEPROM (*Electrically Erasable ROM*): não é necessário apagar todo o conteúdo para atualização, apenas bytes selecionados são alterados. A escrita de uma EEPROM é demorada: centenas de micro-segundos por byte.
- EEPROM é mais custosa e menos densa.

Memória flash ou *flash memory*

- Flash memory, introduzida nos anos 80, é uma memória intermediária entre EPROM e EEPROM, em custo e funcionalidade.
- Recebe o nome *flash* devido à velocidade com que pode ser alterada: uma memória flash por ser apagada em poucos segundos.
- É possível apagar blocos de memória, mas não no nível de byte.
- Dois tipos: NOR e NAND.
- Como EPROM, flash memory usa um transistor por bit, portanto é bastante densa.



Memória flash ou *flash memory*

- Há um limite no número de ciclos de escrita de uma memória flash.
- Esse limite é entre 10.000 a 100.000 para memória flash do tipo NOR e de 100.000 a 1.000.000 para o tipo NAND.

  https://focus.ti.com/pdfs/omap/diskonchipvsnor.pdf
- Em 2012, usando uma técnica de *auto-cura*, Macronix relata a invenção de uma memória flash que sobrevive 100 milhões de ciclos de escrita.

  https://spectrum.ieee.org/semiconductors/memory/flash-memory-survives-100-million-cycles

Deteção e correção de erros de memória

- Erros de leitura e escrita de memória podem ocorrer, e.g. por problemas de voltagem nas linhas ou radiações.
- Códigos de deteção e de correção são usados para detectar ou corrigir erros de memória.
- De modo geral, para uma palavra original de M bits que queremos gravar na memória, K bits adicionais, obtidos como uma função dos M bits, são acrescentados, formando um código de $M + K$ bits.
- Esse código é gravado na memória.
- Após a leitura do código ($M + K$ bits) da memória, usando os M bits lidos, calculamos com a mesma função os K bits que são então comparados com os K bits lidos.
- Se a comparação der igualdade, então consideramos o código lido correto.
- Se a comparação der desigualdade, detectamos um erro de leitura. Dependendo do código usado, podemos corrigir o erro.

Código de deteção de erro

Um bit paridade ($K = 1$) é acrescentado a cada palavra original da memória de M bits.

O bit paridade é escolhido de tal modo que o número de 1's do código resultante (palavra+paridade) é par (ou ímpar).

Seja o código	00110 (último bit é paridade)
se lido como	01110 (erro de 1 bit: erro detectado)
se lido como	00111 (erro de 1 bit: erro detectado)
se lido como	01111 (erro de 2 bits: nao detectado)
Em geral:	erro detectado se há um no. ímpar de bits errados
	erro não detectado se há um no. par de bits errados

Código de deteção de erro

O bit paridade (paridade par) pode ser obtido fazendo o ou-exclusivo dos bits da palavra original.

Palavra = $x_1 x_2 x_3 x_4$

o bit paridade $x_5 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$

onde $\oplus$ representa a operação *ou exclusivo*.

Código de correção de erro - código de Hamming

Source: Wikipedia

Richard Hamming

Hamming queria fazer Engenharia, mas não tinha recursos. Acabou fazendo Matemática pois conseguiu uma bolsa na Universiade de Chicado, onde não havia curso de engenharia. Fez depois mestrado e doutorado em Matemática. Trabalhou na Bell Labs e inventou o famoso código de Hamming. Não se arrendeu de ter feito Matemática, pois o profundo conhecimento teórico o ajudou a resolver um problema de pesquisa de vanguarda: se o computador sabe detectar um erro de memória, por que não pode corrigi-lo?. Hamming recebeu o Turing Award em 1968.

Código de correção de erro - código de Hamming

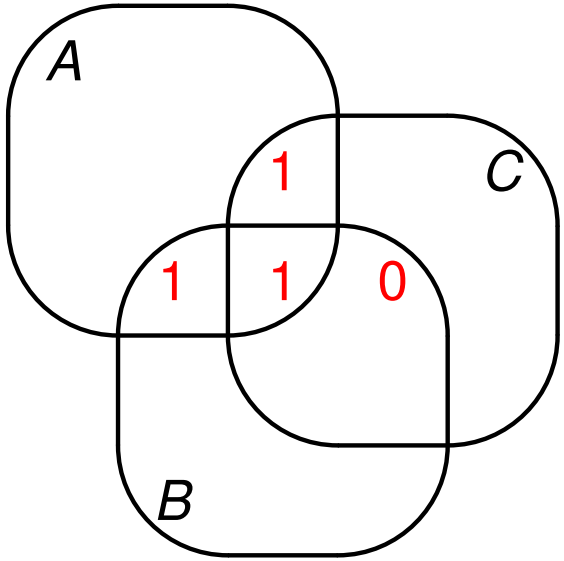
- O código de deteção pode detectar erro, mas não se sabe qual bit está erro. O dado precisa ser lido de novo da memória ou retransmitido no caso de transmissão de dados.
- O código de Hamming é um código de correção que sabe qual bit errado, quando há apenas 1 bit errado. Assim é possível corrigi-lo. Pode corrigir erro de memória, de disco RAID (que usa código de Hamming estendido capaz de corrigir erro de 1 bit e detecção de erros em 2 bits), e em comunicação de dados entre computadores.

Código de correção de erro - código de Hamming

Seja uma palavra original de $M = 4$ bits. Vamos acrescentar nesse caso mais $K = 3$ bits adicionais.

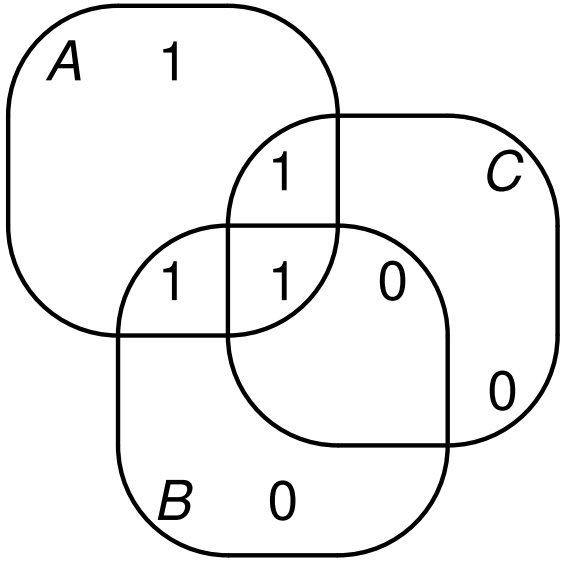
Usamos o diagrama abaixo apenas para fim didático, no caso específco de palavra de 4 bits. O método com diagrama não serve para o caso geral em que a palavra possui mais bits. Mostramos como tratar do caso geral mais tarde.

Palavra de $M = 4$ bits e $K = 3$ bits adicionais



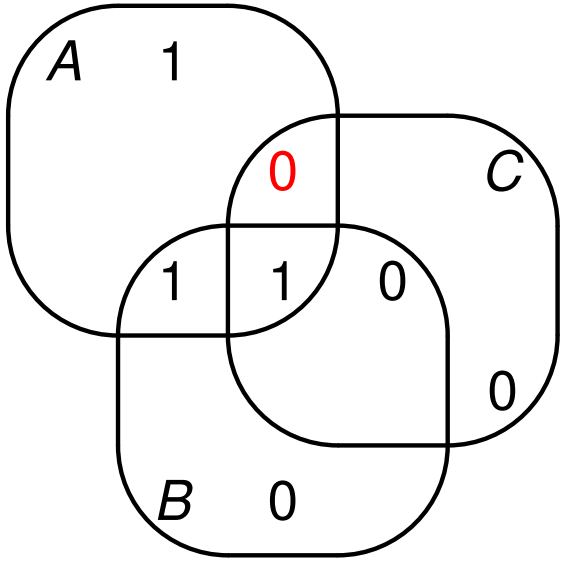
- Seja a palavra dada 1101.
- Para obter os $K = 3$ bits extras, vamos considerar 1101 como os bits nas regiões de interseção AB, AC, BC, ABC, onde A, B, C são diagramas de Venn.

Como obter os $K = 3$ bits adicionais



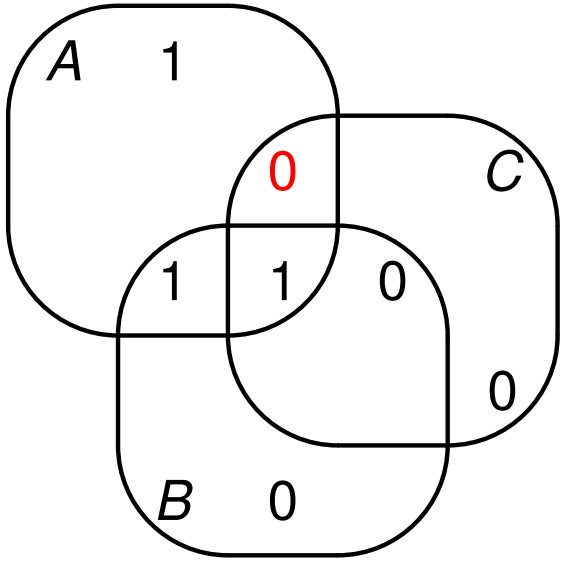
- Acrescentamos um bit de paridade em cada uma das 3 regiões vazias acima para dar paridade par em A, B, e C.
- Os 7 bits (4 da palavra original e 3 adicionais) formam o código de Hamming.

Erro em 1 bit da palavra original



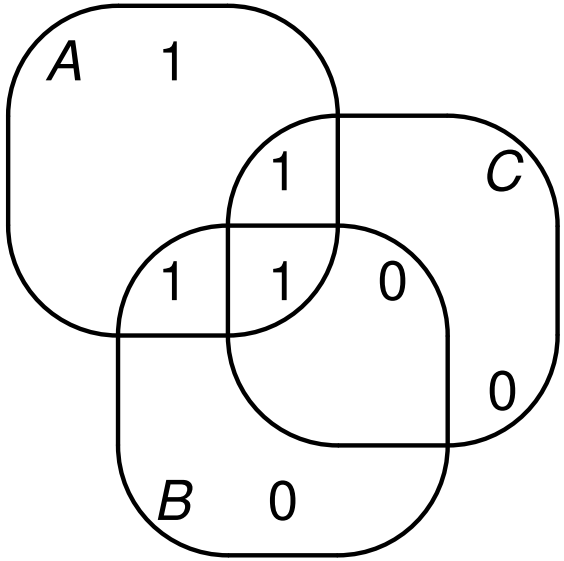
- Erro de 1 bit na palavra original pode ser localizado e corrigido.
- Tal erro pode ser detectado de modo simples, como se segue.

Erro em 1 bit da palavra original



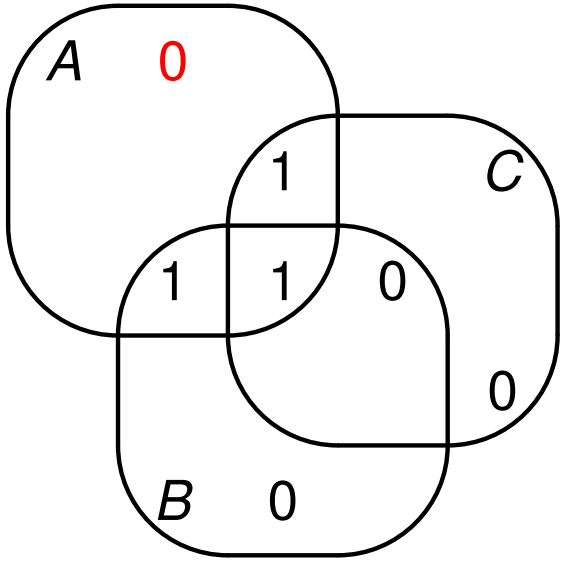
- Calculamos os bits de paridade:
  Região *A*: paridade errada. Região *B*: paridade OK.
- Região *C*: paridade errada. Temos 2 paridades erradas.
  Logo regiao *AC* errada e o bit de *AC* deve ser 1.

Erro em 1 bit da palavra original



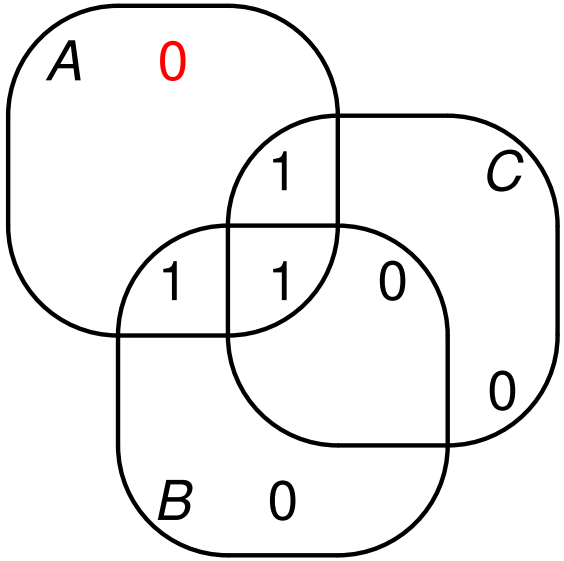
- Calculamos os bits de paridade:
  Região *A*: paridade errada. Região *B*: paridade OK.
- Região *C*: paridade errada. Temos 2 paridades erradas.
  Logo regiao *AC* errada e o bit de *AC* deve ser 1.

Erro em 1 dos K bits adicionais



- Erro de 1 dos K bits adicionais não causa problema, pois não afeta a palavra original.
- Tal erro (apesar de não importante) pode ser detectado de modo simples, como se segue.

Erro em 1 dos K bits adicionais



- Calculamos os bits de paridade:
  Região A: paridade errada. Região B: paridade OK.
- Região C: paridade OK. Temos 1 paridade errada. Logo regiao A errada e o bit de A deve ser 1.

Uma palavra de M bits precisa de K bits adicionais

- Para o caso geral de uma palavra de M bits, quantos bits adicionais são necessários? Isto é: Dado M, quanto vale K?
- Suponha que o código de Hamming lido (de $M + K$ bits) pode ou estar correto ou errado em no máximo 1 bit. Não consideramos erros em mais de um bit. O código de Hamming não funciona para este caso.
- Depois de lido o código de Hamming, calculamos K paridades.
  - Se 0 paridade está errada: a palvra está correta.
  - Se 1 ou mais paridades erradas: um dos $M + K$ bits foi lido erradamente.

Uma palavra de M bits precisa de K bits adicionais

- Calculadas K paridades, cada uma podendo estar correta ou errada: temos assim 2^K possibilidades.
- Se zero paridade está errada, então o código de Hamming foi lido corretamente.
- Se 1 ou mais paridades erradas, então isso indica algum dos $M + K$ bits do código está errado.
  - Temos $2^K - 1$ possibilidades cada uma indicando um dos $M + K$ bits errado.
  - Devemos ter portanto: $2^K - 1 \geq M + K$.
  - Portanto K deve ser tal que $2^K - 1 - K \geq M$.
    - Exemplo: para $M = 4$ e $K = 3$ temos $2^3 - 1 - 3 = 4 \geq 4$.
    - Para $M = 8$ e $K = 4$ temos $2^4 - 1 - 4 = 11 \geq 8$.

Caso geral: palavra de M bits

Seja uma palavra dada de M bits. O código de Hamming precisa de K bits adicionais, com a condição: $2^K - 1 - K \geq M$.

Palavra de M bits	Exemplo $M = 2^s$	K bits adicionais
4	4	3
5 até 11	8	4
12 até 26	16	5
27 até 57	32	6
58 até 120	64	7
121 até 247	128	8

- Temos $2^K \geq M + 1 + K > M$.
  Para $M = 2^s$, $2^K > M$ ou $2^K > 2^s$.
  Assim temos $K > s$.
  Podemos fazer $K = s + 1 = \log M + 1$.
- Para M grande, o *overhead* é menor. Mas lembre-se que o código só funciona para erro de um só bit no código.

Código de Hamming para palavra de $M = 8$ bits

Vamos mostrar a obtenção do código de Hamming para uma palavra de $M = 8$ bits. Numere os bits de

$$m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8$$

A essa palavra de 8 bits vamos acrescentar 4 bits adicionais, formando o código de Hamming de 12 bits.
Numere os bits do código de Hamming como sendo:

$$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}$$

Inserir os M bits originais no código de Hamming

$$
\begin{aligned}
x_1 &= \text{a determinar} \\
x_2 &= \text{a determinar} \\
x_3 &= m_1 \\
x_4 &= \text{a determinar} \\
x_5 &= m_2 \\
x_6 &= m_3 \\
x_7 &= m_4 \\
x_8 &= \text{a determinar} \\
x_9 &= m_5 \\
x_{10} &= m_6 \\
x_{11} &= m_7 \\
x_{12} &= m_8
\end{aligned}
$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
?	?	m_1	?	m_2	m_3	m_4	?	m_5	m_6	m_7	m_8

Falta obter x_1, x_2, x_4, x_8: note índices todos potências de 2.

Obtenção dos bits adicionais

Os 4 bits adicionais x_1, x_2, x_4 e x_8 são assim calculados, onde $\oplus$ representa a operação *ou exclusivo*:

$$
\begin{aligned}
x_1 &= x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11} \\
x_2 &= x_3 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11} \\
x_4 &= x_5 \oplus x_6 \oplus x_7 \oplus x_{12} \\
x_8 &= x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12}
\end{aligned}
$$

Observe que a operação ou-exclusivo é equivalente à paridade par.

Uma regra simples para chegar às fórmulas

$$
\begin{aligned}
x_1 &= x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11} \\
x_2 &= x_3 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11} \\
x_4 &= x_5 \oplus x_6 \oplus x_7 \oplus x_{12} \\
x_8 &= x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12}
\end{aligned}
$$

1 a 12 em binário	8	4	2	1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Uma regra simples para chegar às fórmulas

$$
\begin{aligned}
x_1 &= x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11} \\
x_2 &= x_3 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11} \\
x_4 &= x_5 \oplus x_6 \oplus x_7 \oplus x_{12} \\
x_8 &= x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12}
\end{aligned}
$$

1 a 12 em binário	8	4	2	1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Uma regra simples para chegar às fórmulas

$$
\begin{aligned}
x_1 &= x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11} \\
x_2 &= x_3 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11} \\
x_4 &= x_5 \oplus x_6 \oplus x_7 \oplus x_{12} \\
x_8 &= x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12}
\end{aligned}
$$

1 a 12 em binário	8	4	2	1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Correção de erro

Agora suponha que esses 12 bits são lidos como sendo:

$$y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12}$$

Se não houver erro, então cada y_i é igual seu respectivo x_i.

Se houver erro em um bit apenas, é possível detectar esse erro e corrigi-lo.

Para isso fazemos o seguinte cálculo de 4 bits de paridade, denominados k_1, k_2, k_3 e k_4.

Correção de erro

$$
\begin{aligned}
k_1 &= y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11} \\
k_2 &= y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11} \\
k_3 &= y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12} \\
k_4 &= y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}
\end{aligned}
$$

Se $k_1 = k_2 = k_3 = k_4 = 0$, então não há erro.

Senão o número binário codificado pelos 4 bits $k_4 k_3 k_2 k_1$ determina a posição do bit errado.

Correção de erro

$$
\begin{aligned}
k_1 &= y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11} \\
k_2 &= y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11} \\
k_3 &= y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12} \\
k_4 &= y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}
\end{aligned}
$$

Exemplo, se $k_4 k_3 k_2 k_1 = 0111$ entao o bit y_7 está errado.

Uma referênciaa:

Vera Pless. Introduction to the theory of error-correcting codes. New York : Wiley, 1982, ISBN 0471086843

Erro em mais de 1 bit

- O código de Hamming não funciona para erro em mais de um bit no código.
- Há uma extensão do método que permitea corrigir erros de 1 bit e detectar erros de 2 bits (mas sem corrigi-los). Esse método usa um bit a mais, i.e. $K + 1$ bits adicionais. (Não vamos ver esse método aqui.)
- Em comunicação da dados, onde uma sequência longa de bits é transmitida de um local a outro, é comum uma série consecutiva de bits ser danificada.
- Veremos um truque que permite detectar e corrigir erros em uma sequência de bits.
- (Uau, que legal!, não posso perder essa dica! :-)

Como foi o meu aprendizado?

- Escolha um número M entre 5 a 11. Invente um número de M bits e escreva o código de Hamming.
- Agora erre um bit nesse código e use a técnica para corrigir o erro.

Solução no próximo slide.

Como foi o meu aprendizado?

- Escolha um número M entre 5 a 11. Invente um número de M bits e escreva o código de Hamming.

Resposta: por exemplo escolhi $M = 7$ e o dado de 7 bits como sendo 1001110.

Temos então um código de 7 + 4 = 11 bits.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
1	1	1	1	0	0	1	0	1	1	0

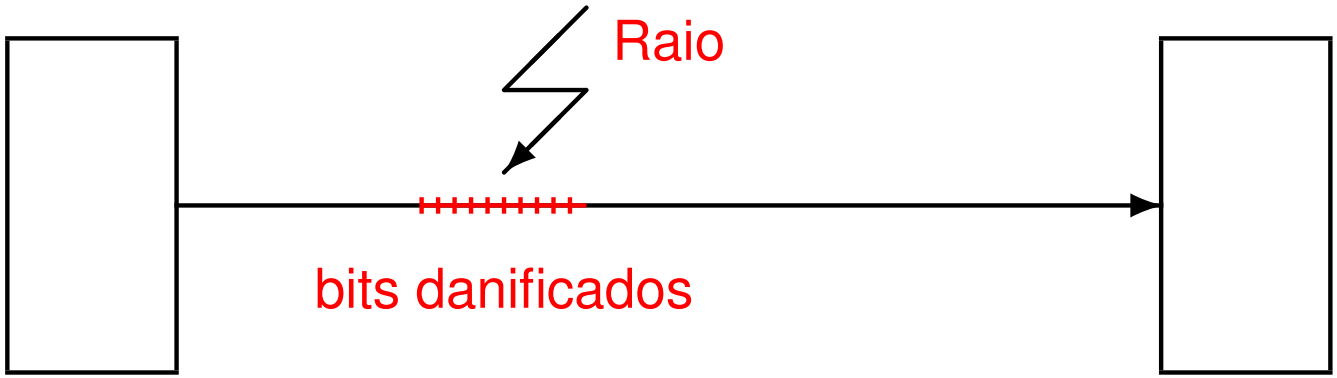
1 a 11 em binário	8	4	2	1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

- Agora erre um bit nesse código e use a técnica para corrigir o erro.

y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}
1	1	1	1	0	0	0	0	1	1	0

$k_1 = 1, k_2 = 1, k_3 = 1, k_4 = 0$ e $k_4 k_3 k_2 k_1 = 0111$ Logo y_7 errado (devia ser 1).

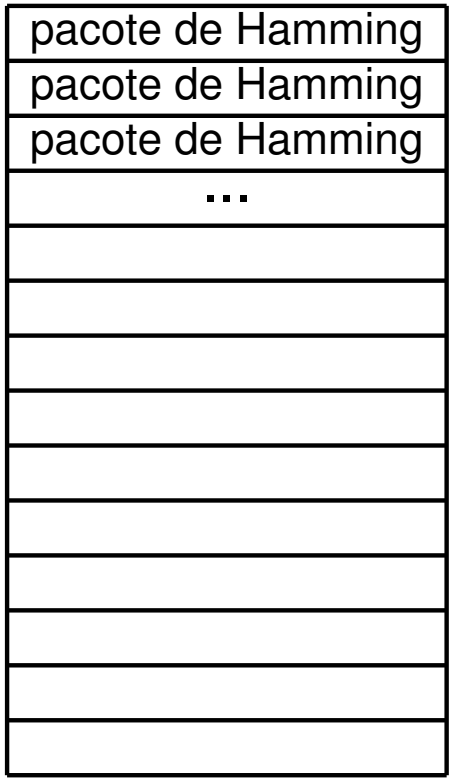
Erro em mais de 1 bit



Vamos ilustrar por um exemplo em comunicação de dados.

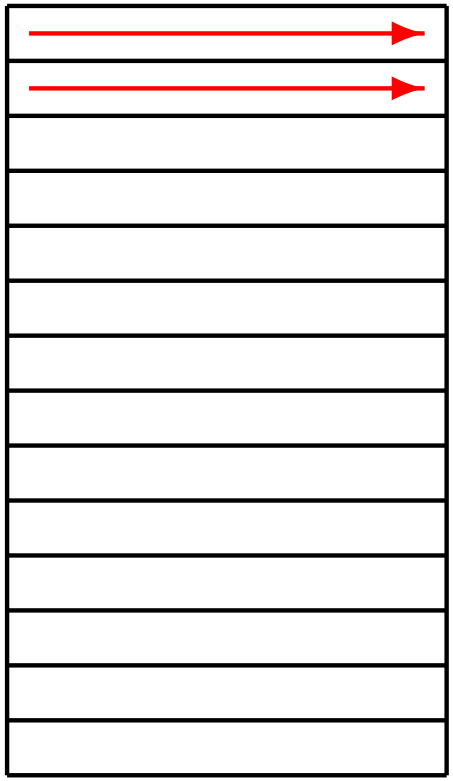
- Uma mensagem constituída de um número de pacotes (cada pacote tem *M* bits) deve ser enviada de um local a outro.
- O meio de transmissão é sujeito a chuvas e trovoadas :-) quando um raio pode danificar uma sequência de bits consecutivos.
- Não queremos apenas detectar erro de transmissão e pedir para retransmitir os pacotes errados. Queremos corrigir os erros.

Erro em mais de 1 bit



- Vamos acrescentar a cada pacote de M bits os K bits adicionais conforme estudamos no código de Hamming. Chamamos cada pacote assim incrementado de pacote de Hamming.
- Vamos considerar os pacotes de Hamming em uma matriz onde cada elemento é um pacote de Hamming.

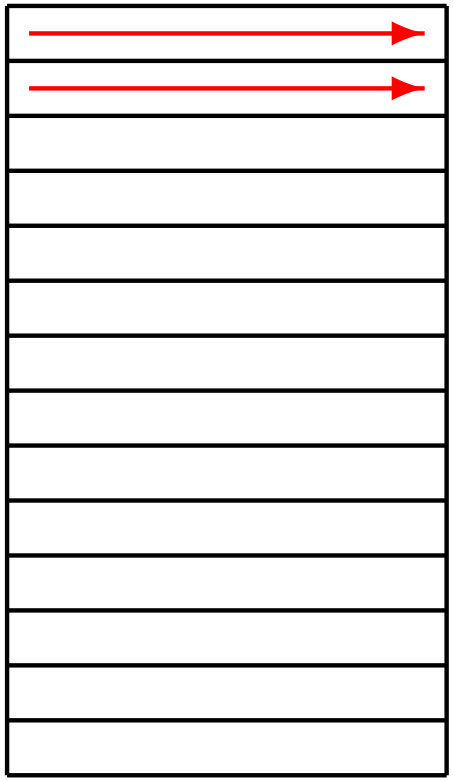
Erro em mais de 1 bit



- Se transimitirmos esses pacotes de Hamming sequencialmente, um a um, então o dano de um raio (que estraga uma série consecutiva de bits) pode ser irrecuperável. Nada adiatou :-(.

Agora vem a idéia brilhante :-).

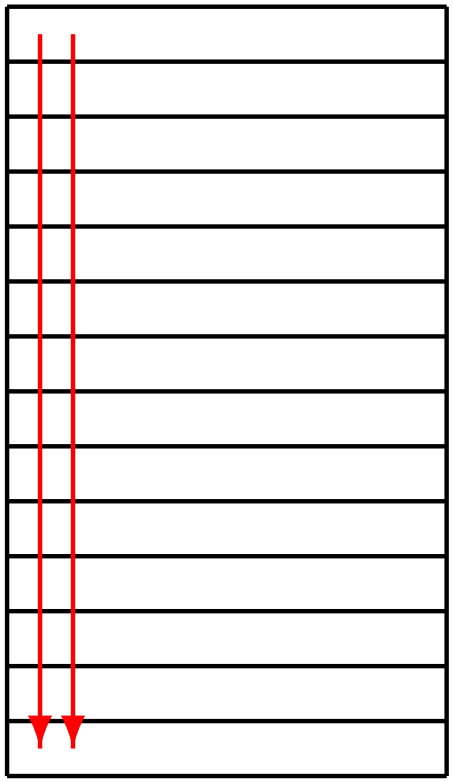
Erro em mais de 1 bit



- Se transimitirmos esses pacotes de Hamming sequencialmente, um a um, então o dano de um raio (que estraga uma série consecutiva de bits) pode ser irrecuperável. Nada adiatou :-(.

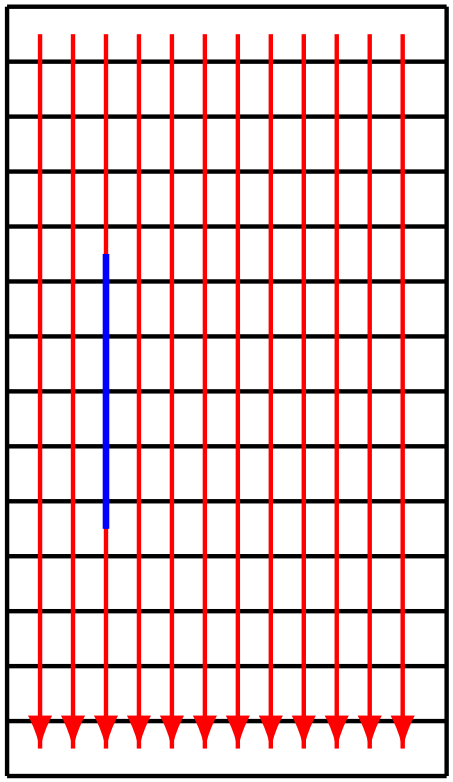
Agora vem a idéia brilhante :-).

A idéia brilhante



- Basta transmitirmos a matriz por coluna. No outro lado da recepção coletamos os bits recebidos para reconstruir a matriz.
- Agora aplicamos método de Hamming para cada pacote de Hamming recebido.

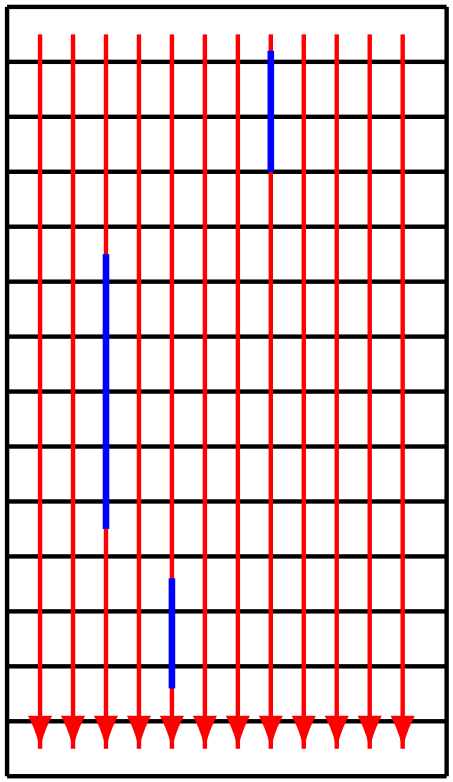
A idéia brilhante



Cor azul = bits danificaods

- Basta transmitirmos a matriz por coluna. No outro lado da recepção coletamos os bits recebidos para reconstruir a matriz.
- Cada bit errado (bit azul na figura) está num pacote de Hamming. Por isso, podemos corrigi-los.

A idéia brilhante



Cor azul = bits danificaods

- Basta transmitirmos a matriz por coluna. No outro lado da recepção coletamos os bits recebidos para reconstruir a matriz.
- Com sorte, pode até aguentar erros de várias sequências de bits. Basta não ter mais um erro em cada pacote.

Como foi o meu aprendizado?

Marque as afirmações corretas.

1. DRAM e SRAM são ambas voláteis, mas SRAM precisa de circuitaria de refrescamento para repor as cargas que se perdem por vazamento.
2. DRAM e SRAM são ambas não-voláteis e assim seu conteúdo não se perde mesmo sem energia elétrica.
3. DRAM e SRAM são ambas voláteis, mas DRAM precisa de circuitaria de refrescamento para repor as cargas que se perdem por vazamento.
4. A memória ROM é não-volátil mas EPROM é volátil.
5. Todos os tipos de memória ROM são não-voláteis.
6. A memória flash pode ser regravada mas somente pelo fabricante.
7. O número de ciclos de escrita numa memória flash é grande mas não é ilimitado.
8. O uso de um bit de paridade pode corrigir erros de memória quando há apenas um bit errado.
9. O código de Hamming serve para corrigir erros de memória quando há apenas um bit errado.

Como foi o meu aprendizado?

Desejo usar o códido de Hamming para corrigir erro de memória.

Qual das duas alternativas está mais adequada?

- Dvemos escolher M bem grande, digamos 2^{10}. Assim, nesse caso, usaremos apenas 11 bits adicionais, uma grande economia.
- Para proteger contra erro de leitura de um dado grande, digamos 2^{10} de bits, o melhor é dividir esse dado em blocos menores e para cada um deles usar o código de Hamming.