

Memória interna e Código de Hamming

MAC0344 - Arquitetura de Computadores
Prof. Siang Wun Song

Slides usados: <https://www.ime.usp.br/~song/mac344/slides05-memory.pdf>

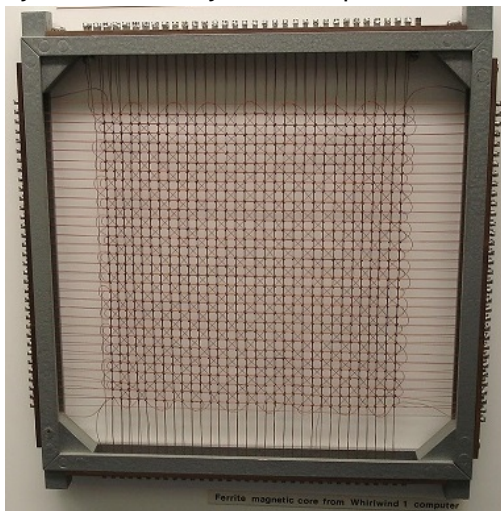
Baseado parcialmente em W. Stallings -
Computer Organization and Architecture

Memória interna e código de detecção/correção de erros

- Veremos Memória interna e código de detecção/correção de erros. Será passada a Lista 4 de exercícios.
- Ao final das aulas, vocês saberão
 - Todos os tipos de memória interna ou principal são feitos de Silício. As aulas sobre a Tecnologia VLSI - transistor MOS etc. - ajudam muito para entender esse assunto.
 - O que são memória dinâmica e estática, memória volátil e não-volátil. Memória ROM, Memória Flash, etc.
 - Há métodos de detectar erros de leitura de memória. Detectado o erro, a memória é lida de novo.
 - O melhor ainda é o método que detecta e corrige o erro, sem precisar ler de novo (código de Hamming).
 - Em particular, verão como esse método pode ser adaptado para corrigir erros de transmissão de grandes quantidades de dados entre dois pontos distantes.

Memória magnética de núcleo de ferrite

- Nos computadores antigos, a memória interna (RAM) era feita de núcleos de ferrite (*magnetic-core memory*). (Até hoje *core memory* é usado para indicar memória interna.)

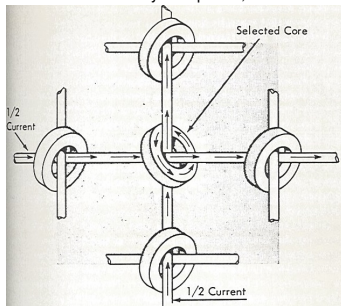


1024 bits

Source: Science
Museum, London

Memória magnética de núcleo de ferrite

Source: IBM Early Computers, MIT Press



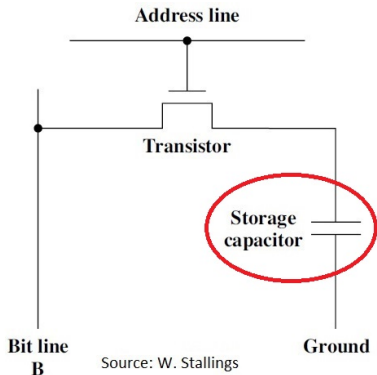
- A memória de núcleo de ferrite é do tipo **não-volátil**: o valor armazenado não se perde quando a energia é desligada e depois religada.

Memória DRAM e SRAM - ambas voláteis

- A memória é feita de semi-condutor (Silício).
- Pode ser de dois tipos:
 - **DRAM** ou Dynamic RAM
 - **SRAM** ou Static RAM
- DRAM e SRAM são ambas **voláteis**: o conteúdo se perde quando o computador é desligado e depois religado.
- Conteúdo criado na memória precisa ser gravado no disco periodicamente, para não perder tudo se a energia cair. Algumas editores já fazem isso automaticamente.

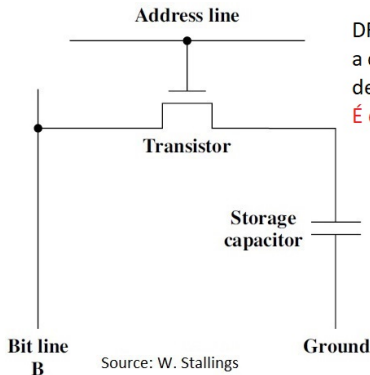
Memória interna DRAM - Dynamic RAM

- **DRAM** (*Dynamic RAM*): usada na memória principal.
- O **capacitor** com carga representa 1, senão representa 0.
- Quando carregado, a carga pode perder por vazamento.
- Para manter a carga de um capacitor que representa 1, aplica-se um pulso de **refrescamento** periodicamente.



Memória interna DRAM - Dynamic RAM

- **DRAM** (*Dynamic RAM*): usada na memória principal.
- O capacitor com carga representa 1, senão representa 0.
- Quando carregado, a carga pode perder por vazamento.
- Para manter a carga de um capacitor que representa 1, aplica-se um pulso de **refrescamento** periodicamente.



DRAM é **volátil** pois quando se desliga a eletricidade a carga se perde. Mas o pior é que, mesmo sem desligar, a carga também se perde por vazamento.

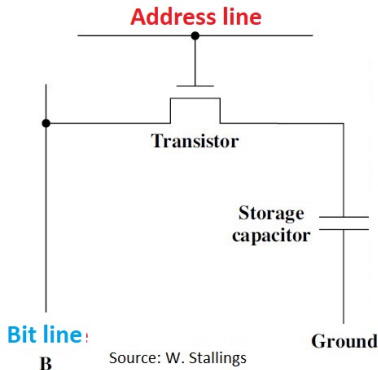
É como um balde furado com água:



Balde furado vai perdendo água. Precisa sempre encher mais água:
Refrescamento

Memória interna DRAM - Dynamic RAM

- **DRAM** (*Dynamic RAM*): usada na memória principal.
- O capacitor com carga representa 1, senão representa 0.
- Quando carregado, a carga pode perder por vazamento.
- Para manter a carga de um capacitor que representa 1, aplica-se um pulso de **refrescamento** periodicamente.



Address line controla o transistor para permitir acesso ao capacitor.

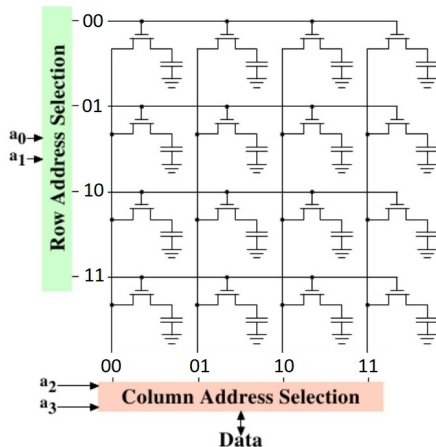
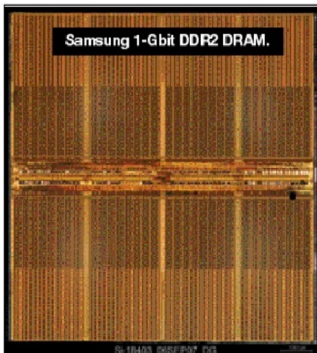
Usa-se voltagem alta no **Bit line** para escrever 1; voltagem baixa para 0.

Para ler, **Bit line** tira a carga do capacitor para ver se é 1 ou 0. A leitura descarrega o capacitor, cuja carga deve ser restaurada.

Memória interna DRAM - Dynamic RAM

Samsung 1-Gbit DRAM. Note a disposição regular dos bits, permitindo uma maior densidade (mais bits por unidade de área do Silício). [Source: Samsung 1-Gbit DRAM.](#)

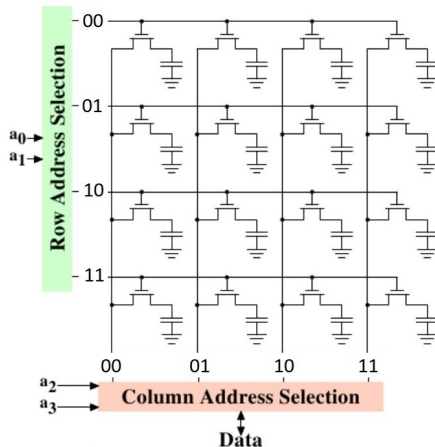
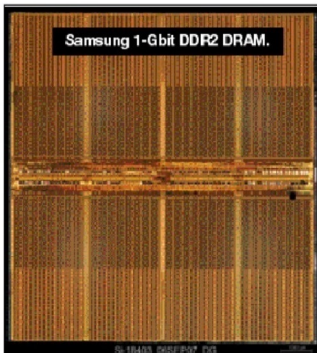
Modern DRAM chip with 8 internal memory banks.



Memória interna DRAM - Dynamic RAM

Samsung 1-Gbit DRAM. Note a disposição regular dos bits, permitindo uma maior densidade (mais bits por unidade de área do Silício). [Source: Samsung 1-Gbit DRAM.](#)

Modern DRAM chip with 8 internal memory banks.

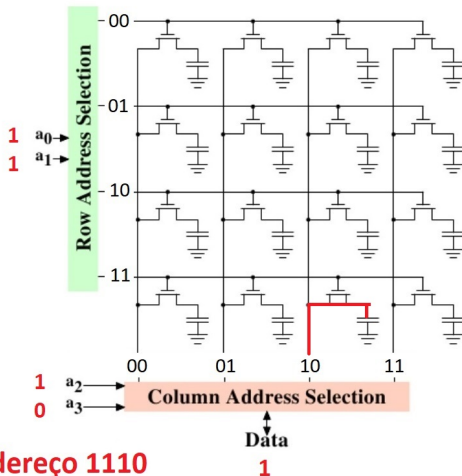
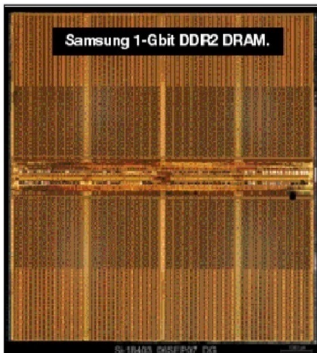


Queremos escrever 1 no endereço 1110

Memória interna DRAM - Dynamic RAM

Samsung 1-Gbit DRAM. Note a disposição regular dos bits, permitindo uma maior densidade (mais bits por unidade de área do Silício). [Source: Samsung 1-Gbit DRAM.](#)

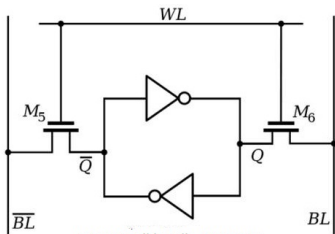
Modern DRAM chip with 8 internal memory banks.



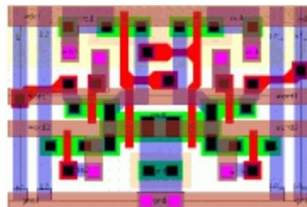
Queremos escrever 1 no endereço 1110

Memória SRAM - Static RAM

- **SRAM** (*Static RAM*): A memória estática mantém o dado inalterado, desde que haja energia. Não precisa de refrescamento.
- É usada na memória cache e registradores, sendo mais rápida, menos densa e mais custosa do que DRAM.
- Uma célula (um bit) SRAM pode ser implementada por duas portas inversoras (portas NÃO). A figura da direita mostra uma célula de um bit em CMOS.

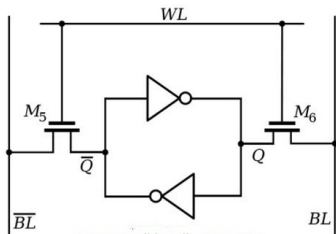


Source: Wikimedia Commons

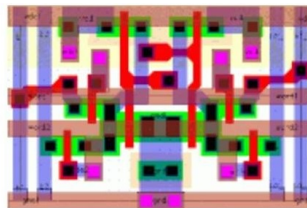


<https://iis-people.ee.ethz.ch/~kgf/aries/5.html>

Memória SRAM - Static RAM



Source: Wikimedia Commons



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

- Há dois estados estáveis:
 - Memória contendo 1: $Q = 1$ e $\overline{Q} = 0$
 - Memória contendo 0: $Q = 0$ e $\overline{Q} = 1$
- Para ler: Linha WL alta liga transistores $M5$ e $M6$:
Valor Q é transmitido para BL (e valor \overline{Q} para \overline{BL})
- Para escrever: Valor 1 ou 0 é colocado em BL e o complemento em \overline{BL} :
Linha WL alta liga transistores $M5$ e $M6$

Memória SRAM - Static RAM

Figura 1

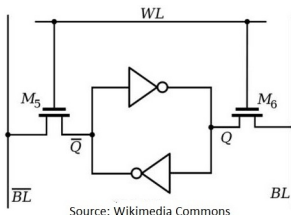
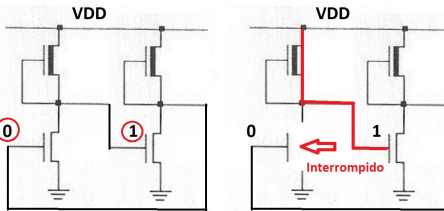


Figura 2



- Vejamos agora por quê duas portas inversoras assim podem garantir o valor 1 sem precisar de refrescamento.
- Figura 1: Porta da direita tem entrada 1, carregando o capacitor.
- A sua saída vale 0, que é conectado à entrada da porta da esquerda.
- Figura 2: Com a entrada igual a 0, o circuito na parte de baixo da porta da esquerda está interrompido. Assim, a corrente sai de VDD e alimenta o capacitor da direita. Portanto esse capacitor fica sempre carregado pela realimentação.

Comparação entre DRAM e SRAM.

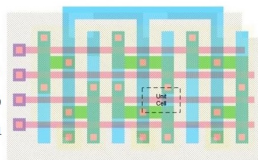
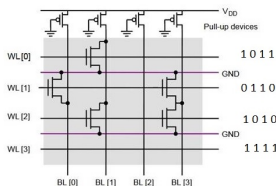
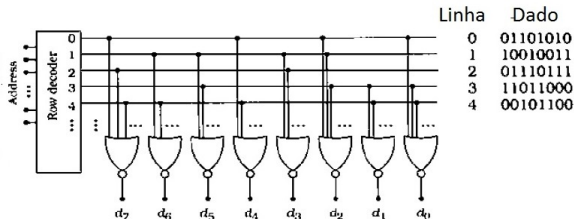
- **Ambas são voláteis.**
- A célula DRAM é mais simples e ocupa menos espaço que uma célula SRAM.
- Portanto DRAM é mais densa (mais células por unidade de área) e mais barata.
- Por outro lado, DRAM requer uma circuitaria de refrescamento. Para memórias grandes, esse custo fixo é mais que compensado 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, pré-gravado na fabricação, 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 fabricada com portas NOR ou NAND, com um *layout* denso.
- Como ROM não pode ser alterada, erro de um bit pode acarretar em descartar um lote inteiro.

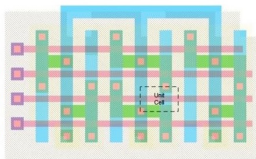
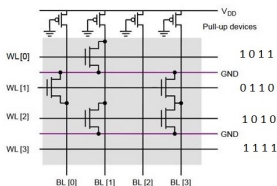
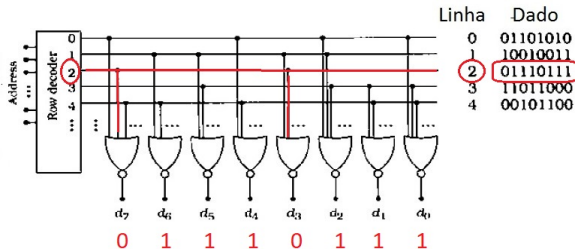
ROM baseado em portas NOR

- 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.



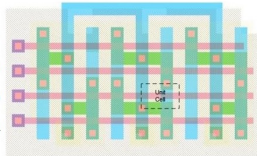
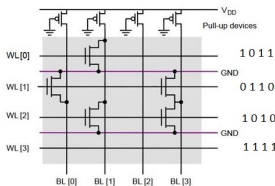
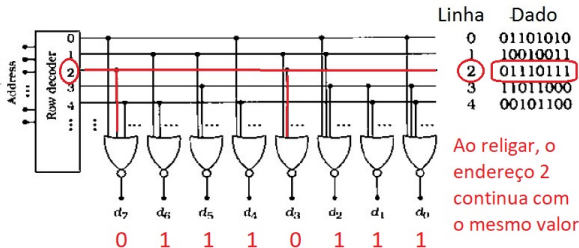
ROM baseado em portas NOR

- 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.



ROM baseado em portas NOR

- A ROM é **não-volátil**. Depois de desligar, ao religar os valores não se perdem. **Vocês podem explicar por quê?**
- É porque os valores estão codificados nas entradas das portas e não se perdem.

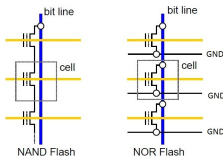


Tipos de ROM (*Read Only Memory*)

- **PROM** (*Programmable ROM*): pode ser escrita uma só vez, 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 Programmable ROM*): leitura e escrita é como numa PROM. Porém, antes de gravar um novo conteúdo, toda a memória é apagada antes por meio de radiação ultra-violeta.
- 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** 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.
- Solid State Drive ou SSD (“Disco de Estado Sólido”) usa a tecnologia de memória flash. Veremos SSD nas aulas sobre Memória Externa.



Detecção e correção de erros de memória

0	0	1	1	0
---	---	---	---	---

Memória gravada

0	1	1	1	0
---	---	---	---	---

Memória lida

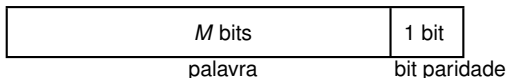
- Erros de leitura de memória podem ocorrer, por exemplo, por problemas de voltagem nas linhas ou radiações.
- Códigos de **detecção** e de **correção** são usados para detectar ou corrigir erros de memória.
 - Código de detecção de erro: o dado precisa ser lido novamente.
 - Código de correção de erro: o bit errado é corrigido. Não precisa ler de novo.

Detecção e correção de erros de memória

- Preparo do código:
 - Para uma palavra original de M bits que queremos gravar na memória, calculamos K bits adicionais, obtidos em função dos M bits originais.
 - Forma-se um código de $M + K$ bits. Esse código é gravado na memória.
- Detecção:
 - Para uma palavra de M bits, acrescentamos $K = 1$ bit a mais, que depende dos M bits.
- Correção:
 - Para uma palavra de M bits, acrescentamos K bits a mais, onde K depende de M . Veremos isso.
 - Se há apenas um bit errado no código formado por $M + K$ bits, então o método que veremos consegue dizer qual esse bit errado e pode corrigi-lo sem ter que ler de novo.

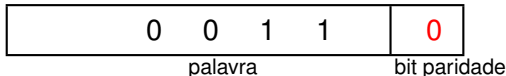
Código de detecção de erro

Um **bit paridade** ($K = 1$) é acrescentado a cada palavra original da memória de M bits. O código formado tem $M + 1$ bits.



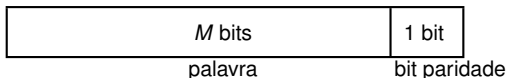
O bit paridade é escolhido de tal modo que o número de 1's do código formado ($M + 1$ bits) é par.

Exemplo 1: A palavra contém um número par de 1's. Então o bit paridade vale **0**. Assim, o código ($4 + 1$ bits) contém um número par de 1's.



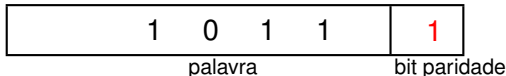
Código de detecção de erro

Um **bit paridade** ($K = 1$) é acrescentado a cada palavra original da memória de M bits. O código formado tem $M + 1$ bits.



O bit paridade é escolhido de tal modo que o número de 1's do código formado ($M + 1$ bits) é par.

Exemplo 2: A palavra contém um número ímpar de 1's. Então o bit paridade vale **1**. Assim, o código ($4 + 1$ bits) contém um número par de 1's.



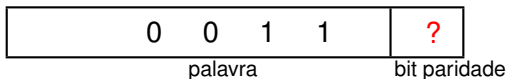
Código de detecção de erro

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 $x_5 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$

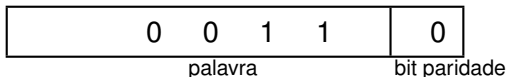
onde \oplus representa a operação *ou-exclusivo*.



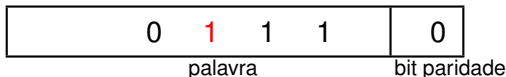
o bit paridade $x_5 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$

Como detectar erro

Exemplo: Preparado o código ($M + 1$ bits), o código é gravado na memória.



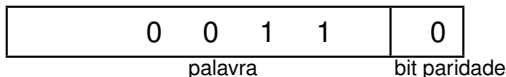
Suponha que ao ler a memória, temos o seguinte caso.



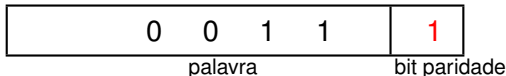
Calculamos a paridade do código lido: paridade **ímpar**: erro!
Tem que ler de novo.

Como detectar erro

Exemplo: Preparado o código ($M + 1$ bits), o código é gravado na memória.



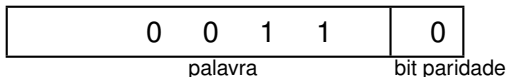
Suponha que ao ler a memória, temos o seguinte caso.



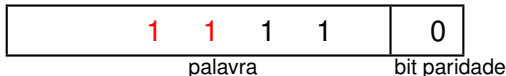
Calculamos a paridade do código lido: paridade **ímpar**: erro!
Tem que ler de novo.

Como detectar erro

Exemplo: Preparado o código ($M + 1$ bits), o código é gravado na memória.



Suponha que ao ler a memória, temos o seguinte caso.



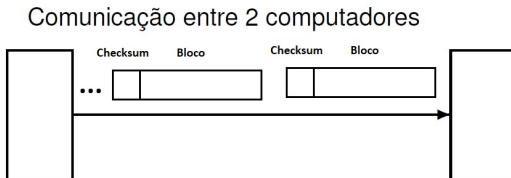
Paridade deu **par**: OK? O método não detecta erro quando há um número par de bits errados.

Código de detecção de erro



- Uso de bit paridade para detectar erro é um método simples.
- É usado em fitas magnéticas. A cada bloco de dados, um bit paridade é gravado.
- Na leitura, ao detectar um erro de paridade, o bloco de dados precisa ser lido de novo.

Erros de transmissão entre 2 computadores



- Um esquema semelhante é usado em comunicação de dados entre 2 computadores.
- Os dados são organizados em blocos. Depois de cada bloco é introduzido um *checksum* que é função dos dados do bloco.
- Para cada bloco recebido, é recalculado o *checksum* usando a mesma função.
- O *checksum* calculado é comparado com o *checksum* recebido. Se diferente, então o bloco deve ser retransmitido.

Como foi o meu **aprendizado**?

Vamos fazer um exercício juntos.

- O método de usar um bit de paridade sempre funciona? Isto é, o método sempre detecta quando há bits errados?
- Se a sua resposta é não, então em que situação o método funciona? E em que situação o método não funciona?

Código de correção de erro - código de Hamming

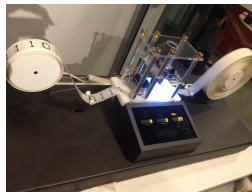
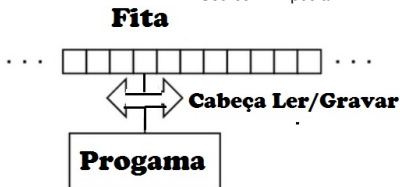


Source: Wikipedia

*Hamming queria fazer Engenharia, mas não tinha recursos. Acabou fazendo Matemática pois conseguiu uma bolsa na Universidade de Chicago, 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 arrependeu 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.*

Turing Award - em homenagem a Alan Turing

Source: Wikipedia



- Alan Mathison Turing (1912 - 1954)
- Inglês, matemático, cientista da computação, cripto-analista. Considerado fundador da Teoria da Computação.
- Máquina de Turing, Computabilidade, Indecidibilidade (Problema da Parada)
- Turing Award: prêmio anual criado em 1966, considerado o Prêmio Nobel da Computação. Em 2014, o prêmio aumentou para US\$ 1 milhão.

Código de correção de erro - código de Hamming

- O código de detecção pode detectar erro, mas não se sabe qual bit está errado. O dado precisa ser lido de novo 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.
- É usado para corrigir erro de memória.
- Para erros de disco RAID, é usado um código de Hamming estendido que é capaz de corrigir erro de 1 bit e detecção de erros em 2 bits.
- Veremos como usar o código de Hamming para erros em comunicação de dados entre computadores.

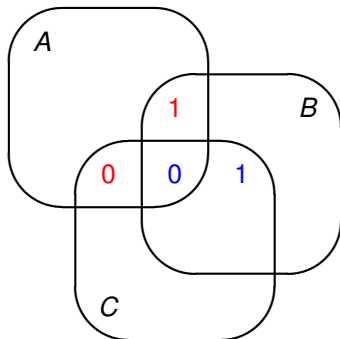
Código de correção de erro - código de Hamming

Vamos começar com um exemplo simples.

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

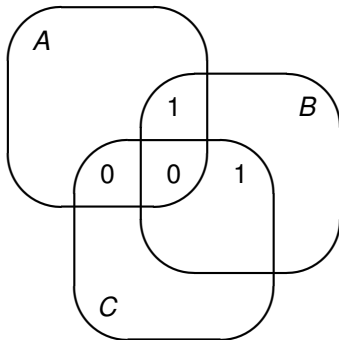
Nos próximos slides, ilustramos o método com diagrama apenas para fim didático, no caso específico de palavra de 4 bits. O método com diagrama não serve para o caso geral em que a palavra possui mais bits. Mostramos como tratar do caso geral mais tarde.

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



- Seja a palavra dada **1010**. Vamos colocar esses bits na figura acima assim:
- **1** = $(A \cap B - C)$ **0** = $(A \cap C - B)$
1 = $(B \cap C - A)$ **0** = $(A \cap B \cap C)$

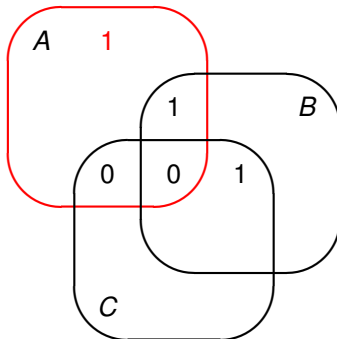
Como obter os $K = 3$ bits adicionais



- Vamos acrescentar 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) vão formar o **código de Hamming**. Vejamos.

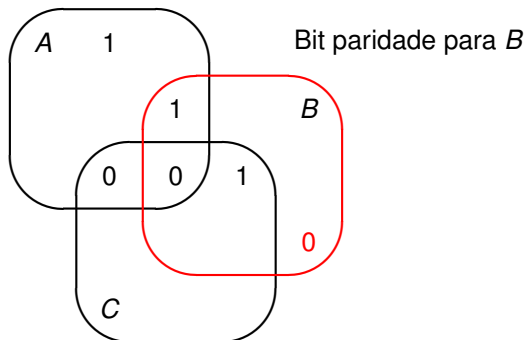
Como obter os $K = 3$ bits adicionais

Bit paridade para A



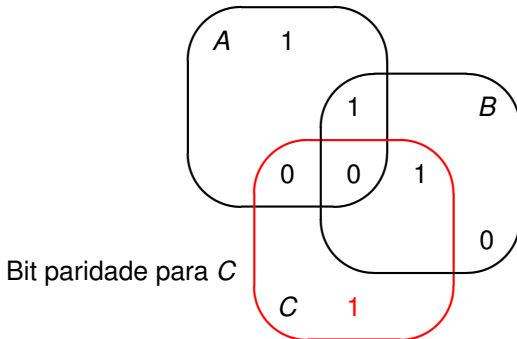
- Acrescentamos um bit de paridade em cada uma das 3 regiões vazias acima para dar paridade par em A, B, e C.
- Os 7 bits (4 da palavra original e 3 adicionais) formam o **código de Hamming**.

Como obter os $K = 3$ bits adicionais



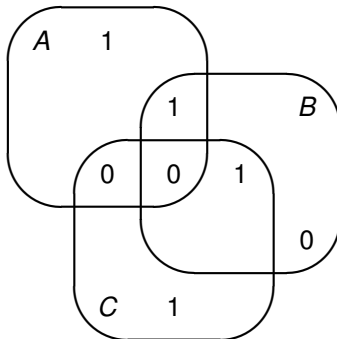
- Acrescentamos um bit de paridade em cada uma das 3 regiões vazias acima para dar paridade par em A , B , e C .
- Os 7 bits (4 da palavra original e 3 adicionais) formam o **código de Hamming**.

Como obter os $K = 3$ bits adicionais



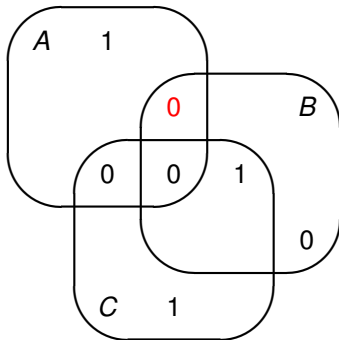
- Acrescentamos um bit de paridade em cada uma das 3 regiões vazias acima para dar paridade par em A , B , e C .
- Os 7 bits (4 da palavra original e 3 adicionais) formam o **código de Hamming**.

Como obter os $K = 3$ bits adicionais



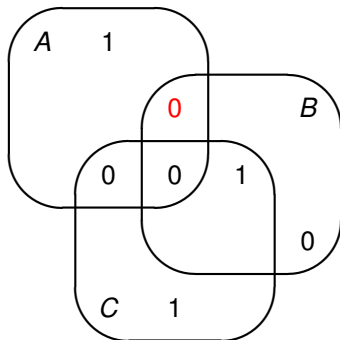
- O código de Hamming obtido (com $M + K$ bits) é gravado na memória.
- Vamos ler esse código e supor que no máximo um bit lido errado. Vamos mostrar como saber qual o bit lido errado.

Erro em 1 bit da palavra original



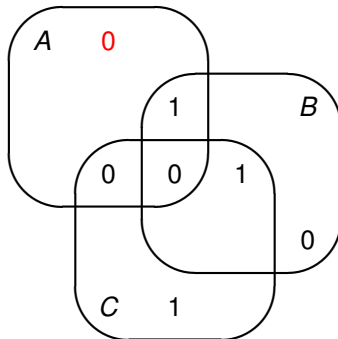
- Erro de 1 bit na palavra original pode ser localizado e corrigido.
- Tal erro pode ser detectado de modo simples, como se segue.

Erro em 1 bit da palavra original



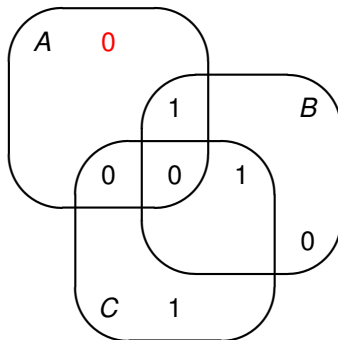
- Calculamos os bits de paridade:
Região **A**: paridade **errada**. Região **B**: paridade **errada**.
- Região **C**: paridade OK. Temos **2 paridades erradas**. Logo região **AB** errada e o bit de **AB** deve ser 1.

Erro em 1 dos K bits adicionais



- Qualquer um dos 7 bits pode estar errado, por exemplo, o erro pode ser de um dos K bits adicionais.
- Tal erro pode ser detectado e corrigido como no caso anterior, como se segue.

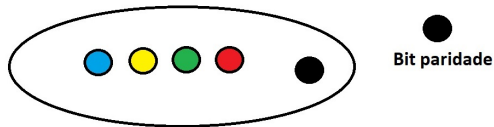
Erro em 1 dos K bits adicionais



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

Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado. Não consideramos erros de 2 ou mais bits.



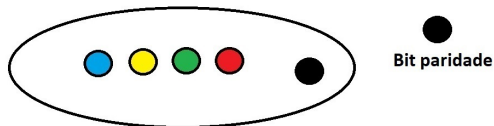
Para os M bits originais, se usamos um bit paridade:

Na leitura, se dá erro de paridade: **não sabemos qual bit está errado.**

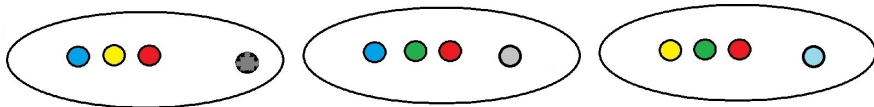
 **Um deles é o culpado**

Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado.
Não consideramos erros de 2 ou mais bits.



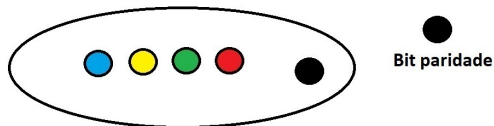
Formamos K subconjuntos: para cada subconjunto calculamos um bit paridade



Após a leitura, vamos verificar se há erro de paridade em cada um dos subconjuntos

Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado.
Não consideramos erros de 2 ou mais bits.

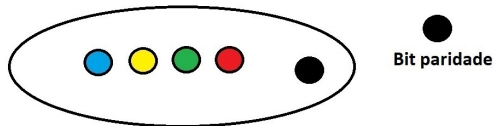


Formamos K subconjuntos: para cada subconjunto calculamos um bit paridade



Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado.
Não consideramos erros de 2 ou mais bits.

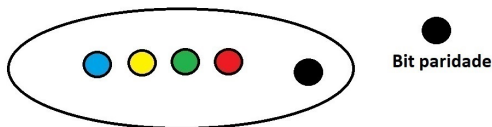


Formamos K subconjuntos: para cada subconjunto calculamos um bit paridade



Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado.
Não consideramos erros de 2 ou mais bits.

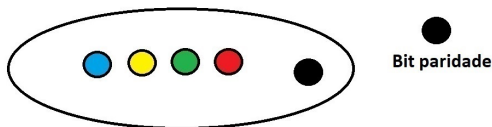


Formamos K subconjuntos: para cada subconjunto calculamos um bit paridade



Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado.
Não consideramos erros de 2 ou mais bits.

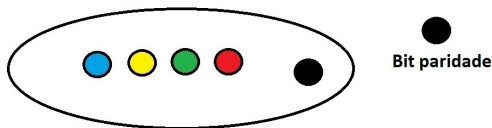


Formamos K subconjuntos: para cada subconjunto calculamos um bit paridade



Explicação: como funciona o código de Hamming?

Supomos que após a leitura do código de Hamming, apenas um bit pode estar errado. Não consideramos erros de 2 ou mais bits.



Formamos K subconjuntos: para cada subconjunto calculamos um bit paridade

Quanto vale K para dado M ?

Veremos:

Como formar os K subconjuntos?

Como formalizar tudo?

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 palavra está correta.
 - Se 1 ou mais paridades erradas: um dos $M + K$ bits foi lido errado.


Uma palavra de M bits precisa de K bits adicionais

Calculadas K paridades, cada uma pode estar correta ou errada.

		K Paridades							
		1	2	3	4	...	K-2	K-1	K
0 significa paridade correta	0	0	0	0			0	0	0
	0	0	0	0			0	0	1
	0	0	0	0			0	1	0
	0	0	0	0			0	1	1
1 significa paridade errada	0	0	0	0			1	0	0
	0	0	0	0			1	0	1
	0	0	0	0			1	1	0
	0	0	0	0			1	1	1
		.							
		.							
		.							
		1	1	1	1		1	1	1

Uma palavra de M bits precisa de K bits adicionais

Calculadas K paridades, cada uma pode estar correta ou errada.

		K Paridades							
		1	2	3	4	...	K-2	K-1	K
0 significa paridade correta	0	0	0	0			0	0	0
	0	0	0	0			0	0	1
	0	0	0	0			0	1	0
	0	0	0	0			0	1	1
1 significa paridade errada	0	0	0	0			1	0	0
	0	0	0	0			1	0	1
	0	0	0	0			1	1	0
	0	0	0	0			1	1	1
 2^k possibilidades									
		1	1	1	1		1	1	1

Uma palavra de M bits precisa de K bits adicionais

Calculadas K paridades, cada uma podendo estar correta ou errada.

0 significa
paridade correta

1 significa
paridade errada

2^k possibilidades

K Paridades

1	2	3	4	...	K-2	K-1	K
0	0	0	0		0	0	0
0	0	0	0		0	0	1
0	0	0	0		0	1	0
0	0	0	0		0	1	1
0	0	0	0		1	0	0
0	0	0	0		1	0	1
0	0	0	0		1	1	0
0	0	0	0		1	1	1
				.			
				.			
				.			
1	1	1	1		1	1	1

Sem erro: 1 possibilidade

Com erro:
 $2^k - 1$ possibilidades

Uma palavra de M bits precisa de K bits adicionais

Calculadas K paridades, cada uma podendo estar correta ou errada.

0 significa
paridade correta

1 significa
paridade errada

↓

2^k possibilidades

K Paridades

1	2	3	4	...	K-2	K-1	K
0	0	0	0		0	0	0
0	0	0	0		0	0	1
0	0	0	0		0	1	0
0	0	0	0		0	1	1
0	0	0	0		1	0	0
0	0	0	0		1	0	1
0	0	0	0		1	1	0
0	0	0	0		1	1	1
				.			
				.			
				.			
1	1	1	1		1	1	1

Sem erro: 1 possibilidade

Com erro:
 $2^k - 1$ possibilidades

O erro pode estar em um dos $M + K$ bits.

Portanto:

$$2^k - 1 \geq M + K$$

Uma palavra de M bits precisa de K bits adicionais

- Devemos ter: $2^K - 1 \geq M + K$.
- 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 $2^4 - 1 - 4 = 11 \geq 8$.
- Então dado M , como calculamos K ?
 - Dado M , obtemos o menor K que satisfaz $2^K - 1 - K \geq M$.

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 ver como calcular o 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

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

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*:

$$x_1 = x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11}$$

$$x_2 = x_3 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11}$$

$$x_4 = x_5 \oplus x_6 \oplus x_7 \oplus x_{12}$$

$$x_8 = x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12}$$

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

Considere o conjunto dos 12 bits:

$$\{X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10} X_{11} X_{12}\}$$

A ideia é

- Primeiro escolhemos vários subconjuntos desse conjunto de bits
- Vamos exigir que a paridade seja par para cada um desses subconjuntos
- Ao invés de um bit paridade, teremos vários bits de paridade. Isso será útil conforme veremos mais tarde.

Vejamos como escolher esses subconjuntos.

Explicação

Considere o conjunto dos 12 bits: $\{x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}\}$

Escrevemos os números 0 a 12 em binário:

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

São 4 colunas na tabela. Vamos ter 4 subconjuntos. Vejamos como obter o primeiro subconjunto.

Explicação

Considere o conjunto dos 12 bits: $\{x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}\}$

Escrevemos os números 0 a 12 em binário:

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

O primeiro subconjunto é (olhe a cor vermelha):

$\{x_1 x_3 x_5 x_7 x_9 x_{11}\}$

Explicação

Considere o conjunto dos 12 bits: $\{x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}\}$

Escrevemos os números 0 a 12 em binário:

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

Para $\{x_1 x_3 x_5 x_7 x_9 x_{11}\}$ ter paridade par, basta fazer

$$x_1 = x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11}$$

Temos assim uma regra simples para calcular x_1, x_2, x_4 e x_8
Veremos nos próximos slides.

Uma regra simples para chegar às fórmulas

$$X_1 = X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{11}$$

$$X_2 = X_3 \oplus X_6 \oplus X_7 \oplus X_{10} \oplus X_{11}$$

$$X_4 = X_5 \oplus X_6 \oplus X_7 \oplus X_{12}$$

$$X_8 = X_9 \oplus X_{10} \oplus X_{11} \oplus X_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

$$X_1 = X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{11}$$

$$X_2 = X_3 \oplus X_6 \oplus X_7 \oplus X_{10} \oplus X_{11}$$

$$X_4 = X_5 \oplus X_6 \oplus X_7 \oplus X_{12}$$

$$X_8 = X_9 \oplus X_{10} \oplus X_{11} \oplus X_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

$$X_1 = X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{11}$$

$$X_2 = X_3 \oplus X_6 \oplus X_7 \oplus X_{10} \oplus X_{11}$$

$$X_4 = X_5 \oplus X_6 \oplus X_7 \oplus X_{12}$$

$$X_8 = X_9 \oplus X_{10} \oplus X_{11} \oplus X_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

$$X_1 = X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{11}$$

$$X_2 = X_3 \oplus X_6 \oplus X_7 \oplus X_{10} \oplus X_{11}$$

$$X_4 = X_5 \oplus X_6 \oplus X_7 \oplus X_{12}$$

$$X_8 = X_9 \oplus X_{10} \oplus X_{11} \oplus X_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

$$X_1 = X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{11}$$

$$X_2 = X_3 \oplus X_6 \oplus X_7 \oplus X_{10} \oplus X_{11}$$

$$X_4 = X_5 \oplus X_6 \oplus X_7 \oplus X_{12}$$

$$X_8 = X_9 \oplus X_{10} \oplus X_{11} \oplus X_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

O código de Hamming calculado é gravado na memória:

$$X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10} X_{11} X_{12}$$

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

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

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

Senão o bit y_i (onde i é o valor decimal de $k_4k_3k_2k_1$) está errado. Mais tarde vamos mostrar o porquê.

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

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

Correção de erro

De onde vieram essas fórmulas?

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

De onde vieram essas fórmulas?

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

De onde vieram essas fórmulas?

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

De onde vieram essas fórmulas?

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

0 a 12 em binário	8	4	2	1
0	0	0	0	0
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

Suponha ao ler um código de Hamming, os seguintes bits foram lidos. (Supomos apenas um bit pode estar errado.)

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

Calculamos os 4 bits de paridade k_1, k_2, k_3, k_4 pelas fórmulas vistas:

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

Suponha $k_4 k_3 k_2 k_1 = 0111$ **entao o bit y_7 está errado**. Por quê?

- $k_1 = 1$ significa que o bit errado **está** no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- $k_2 = 1$ significa que o bit errado **está** no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- $k_3 = 1$ significa que o bit errado **está** no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- $k_4 = 0$ significa que o bit errado **não está** no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Portanto em $\{y_7\}$

Nos próximos slides vamos explicar:

Se $k_4k_3k_2k_1 = 0111$ **entao o bit y_7 está errado**. *Por quê?*

O que se segue não faz parte da etapa para descobrir o bit errado, mas tão somente uma explicação sobre o método.

Correção de erro

Seja $k_4 k_3 k_2 k_1 = 0111$.

- O bit errado está no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também está no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também está no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado não está no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Vamos calcular isso.

$A \cap B \cap C - D$	8	4	2	1
0	0	0	0	0
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

Seja $k_4 k_3 k_2 k_1 = 0111$.

- O bit errado **está** no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também está no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também está no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado não está no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Vamos calcular isso.

	A	\cap	B	\cap	C	$-$	D	8	4	2	1
0								0	0	0	0
1	1							0	0	0	1
2								0	0	1	0
3	3							0	0	1	1
4								0	1	0	0
5	5							0	1	0	1
6								0	1	1	0
7	7							0	1	1	1
8								1	0	0	0
9	9							1	0	0	1
10								1	0	1	0
11	11							1	0	1	1
12								1	1	0	0

Correção de erro

Seja $k_4 k_3 k_2 k_1 = 0111$.

- O bit errado está no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também **está** no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também está no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado não está no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Vamos calcular isso.

	A	∩	B	∩	C	-	D	8	4	2	1
0								0	0	0	0
1	1							0	0	0	1
2			2					0	0	1	0
3	3		3					0	0	1	1
4								0	1	0	0
5	5							0	1	0	1
6			6					0	1	1	0
7	7		7					0	1	1	1
8								1	0	0	0
9	9							1	0	0	1
10			10					1	0	1	0
11	11		11					1	0	1	1
12								1	1	0	0

Correção de erro

Seja $k_4 k_3 k_2 k_1 = 0111$.

- O bit errado está no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também está no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também **está** no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado não está no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Vamos calcular isso.

	A	∩	B	∩	C	-	D	8	4	2	1
0								0	0	0	0
1	1							0	0	0	1
2			2					0	0	1	0
3	3		3					0	0	1	1
4					4			0	1	0	0
5	5				5			0	1	0	1
6			6		6			0	1	1	0
7	7		7		7			0	1	1	1
8								1	0	0	0
9	9							1	0	0	1
10			10					1	0	1	0
11	11		11					1	0	1	1
12					12			1	1	0	0

Correção de erro

Seja $k_4 k_3 k_2 k_1 = 0111$.

- O bit errado está no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também está no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também está no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado **não está** no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Vamos calcular isso.

	A	∩	B	∩	C	-	D	8	4	2	1
0								0	0	0	0
1	1							0	0	0	1
2			2					0	0	1	0
3	3		3					0	0	1	1
4					4			0	1	0	0
5	5				5			0	1	0	1
6			6		6			0	1	1	0
7	7		7		7			0	1	1	1
8						8		1	0	0	0
9	9					9		1	0	0	1
10			10			10		1	0	1	0
11	11		11			11		1	0	1	1
12					12	12		1	1	0	0

Correção de erro

Seja $k_4 k_3 k_2 k_1 = 0111$.

- O bit errado está no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também está no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também está no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado não está no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$. Portanto o bit errado é y_7 .

	A	\cap	B	\cap	C	$-$	D	8	4	2	1
0								0	0	0	0
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	7		7		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

Seja $k_4 k_3 k_2 k_1 = 0111$ Portanto BIT ERRADO: 7.

- O bit errado está no conjunto $A = \{y_1, y_3, y_5, y_7, y_9, y_{11}\}$
- O bit errado também está no conjunto $B = \{y_2, y_3, y_6, y_7, y_{10}, y_{11}\}$
- O bit errado também está no conjunto $C = \{y_4, y_5, y_6, y_7, y_{12}\}$
- E o bit errado não está no conjunto $D = \{y_8, y_9, y_{10}, y_{11}, y_{12}\}$
- O bit errado portanto está em $A \cap B \cap C - D$.

$A \cap B \cap C - D$	8	4	2	1
0	0	0	0	0
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

Recapitulando: lido o código de Hamming, a verificação se há erro é assim:

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11}$$

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11}$$

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 \oplus y_{12}$$

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{12}$$

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

Não é necessário calcular $A \cap B \cap C - D$, que serve apenas para mostrar que o método funciona.

Lista de Exercícios 4

- Fazer e entregar por email a [Lista de Exercícios 4](#).
- Há prazo para entrega. Recomendo não demorar muito. Bom fazer logo com a matéria fresquinha na cabeça.

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 permite 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 de 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?

Vamos fazer um exercício juntos.

- Parte 1: Escolha um número M entre 5 a 11. Invente um número de M bits e escreva o código de Hamming.
- Parte 2: Agora erre um bit nesse código e use a técnica para corrigir o erro. (Obviamente não é para olhar a primeira parte para descobrir qual bit errado :-)

Continuamos esse exercício nos próximos slides.

Como foi o meu aprendizado?

Parte 1: Escolha um número M entre 5 a 11. Invente um número de M bits e escreva o código de Hamming.

- Por exemplo escolhemos $M = 7$ e seja o dado de 7 bits como sendo 1001110.
- Pela tabela vista anteriormente (slide no. 60), para 7 bits de dados precisamos acrescentar 4 bits adicionais.
- Mas podemos deduzir que precisamos de mais 4 bits, assim.
- Escrevemos uma linha com $x_1, x_2, x_3, x_4, x_5, x_6, \dots$
- Deixamos de lado os x_i onde i é potência de 2 (i.e. $x_1, x_2, x_4, x_8, \text{etc.}$) e preenchemos os bits do número dado.

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

- Notamos que precisamos usar x_1, x_2, x_4, x_8 e portanto 4 bits adicionais. Vamos agora calcular esses 4 bits.

Como foi o meu aprendizado?

Cálculo de x_1 :

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

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$x_1 = x_3 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11} = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

Como foi o meu aprendizado?

Cálculo de x_2 :

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

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$x_2 = x_3 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11} = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

Como foi o meu **aprendizado**?

Cálculo de x_4 :

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	?	0	0	1		1	1	0

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$x_4 = x_5 \oplus x_6 \oplus x_7 = 0 \oplus 0 \oplus 1 = 1$$

Como foi o meu **aprendizado**?

Cálculo de x_8 :

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	?	1	1	0

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$x_8 = x_9 \oplus x_{10} \oplus x_{11} = 1 \oplus 1 \oplus 0 = 0$$

Como foi o meu **aprendizado**?

Resposta da Parte 1: o código de Hamming calculado foi:

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

Parte 2: Agora erre um bit nesse código e use a técnica para corrigir o erro.

Os bits lidos y_i são os seguintes. Note que o bit 7 foi lido erradamente.

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

Vamos agora ver como detectamos e corrigimos o erro.

Como foi o meu **aprendizado**?

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

Calculamos k_1 :

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$k_1 = y_1 \oplus y_3 \oplus y_5 \oplus y_7 \oplus y_9 \oplus y_{11} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

Como foi o meu **aprendizado**?

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

Calculamos k_2 :

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$k_2 = y_2 \oplus y_3 \oplus y_6 \oplus y_7 \oplus y_{10} \oplus y_{11} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

Como foi o meu aprendizado?

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

Calculamos k_3 :

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$k_3 = y_4 \oplus y_5 \oplus y_6 \oplus y_7 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

Como foi o meu aprendizado?

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

Calculamos k_4 :

0 a 11 em binário	8	4	2	1
0	0	0	0	0
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

$$k_4 = y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

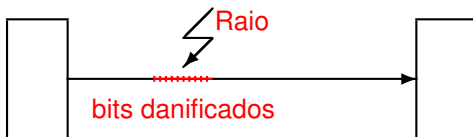
Como foi o meu aprendizado?

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

Obtivemos $k_4k_3k_2k_1 = 0111$ (7 em decimal).

Portanto o bit y_7 está errado. Ao invés de 0 corrigimos para 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

pacote de Hamming
pacote de Hamming
pacote de Hamming
...

- 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**.
- Colocamos os pacotes de Hamming em uma matriz.

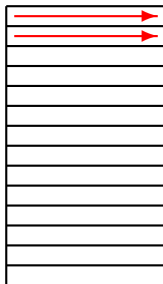
Erro em mais de 1 bit



- Se transmitirmos 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 adiantou :-).

Agora vem a **ideia brilhante** :-).

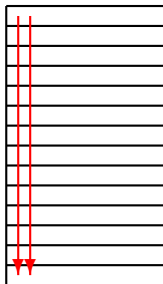
Erro em mais de 1 bit



- Se transmitirmos esses **pacotes de Hamming** sequencialmente, um a um, então o dano de um raio (que estraga uma série consecutiva de bits) pode ser irrecoverável. Nada adiantou :-).

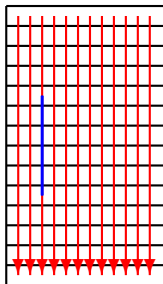
Agora vem a **ideia brilhante** :-).

A ideia 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.

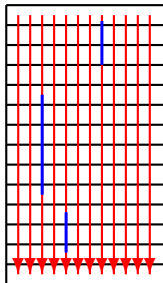
A idéia brilhante



Cor azul = bits danificados

- 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 danificados

- 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 código de Hamming para corrigir erro de memória.

Qual das duas alternativas está mais adequada?

- Devemos 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.

Próximo assunto: Memória externa

- Próximo assunto: Memória externa
- Disco magnético (HD), Discos RAID (Redundant Array of Independent Disks), SSD (*Solid State Disks*), etc.
- Não percam!