Detecção e correção de erros

Lei de Murphy aplicada à manipulação de dados: Detecção e correção de erros.

Como não é possível prever quando e como eles vão acontecer, logo Erros Acontecem!!!

Como proceder?

- Detecção de erros: bits de paridade
- Correção: códigos de *hamming*, neste caso a *informação é suficiente para a recuperação.*

Bits de Paridade

Tornam sempre par ou ímpar o número de 1s em um byte

- Simples, permite detecção de erros individuais;
- Normalmente implementado em hardware.

Exemplo

- Paridade par: 01011010 ➔ 01011010 **0**
- Paridade ímpar: 01011010 ➔ 01011010 **1**

Códigos Corretores de Erros

Vários bits de paridade são acrescentados segundo regras especiais. Essa redundância extra é tal que certos erros podem ser corrigidos. Correção de bit único.

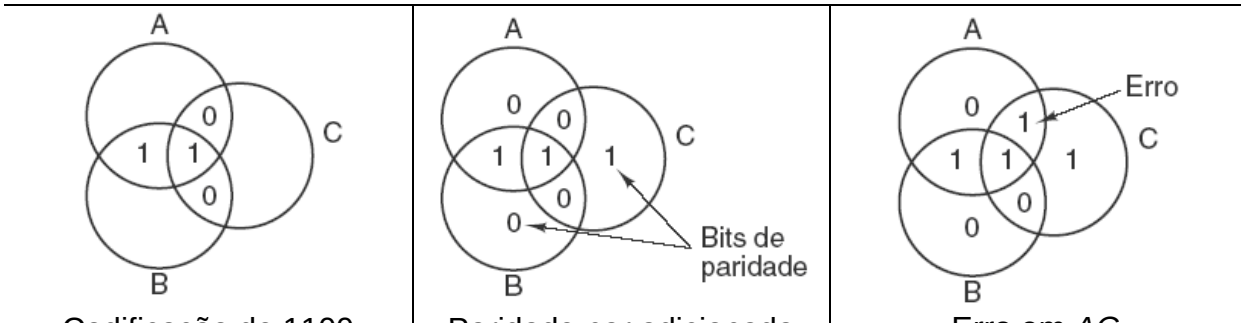
- **m** bits de dados, **r** bits de redundância
- **n** = **m**+**r** bits por palavra
- $2^{m}(n+1) \leq 2^{n} \rightarrow (m+r+1) \leq 2^{r}$

Códigos corretores de erros

Tamanho da Palavra	Bits de Verificação	Tamanho Total	Percentagem sobrecarga
8	**4**	**12**	**50**
16	**5**	**21**	**31**
32	**6**	**38**	**19**
64	**7**	**71**	**11**
128	**8**	**136**	**6**
256	**9**	**265**	**4**
512	**10**	**522**	**2**

Figura 1: Número de bits de verificação para corrigir um único erro.

Códigos corretores de erros



Codificação de 1100 | Paridade par adicionada | Erro em *AC*

Figura 2: Verificação de Erro para corrigir um único bit.

Correção de Erros em Memória

Todo o sistema de memória de semicondutor está sujeito a erros, podendo ser falhas graves ou moderadas. Por falha grave entendemos um defeito físico permanente, onde a região de memória atingida não tem mais condições de armazenar informações, permanecendo permanentemente em 0 ou 1 ou variando constantemente. Causas: defeito de fabricação, desgaste, etc.

Por outro lado, uma falha moderada apenas afetará o conteúdo da(s) posição(ões) de memória atingida, sem danificar fisicamente o espaço.

Causas: fornecimento de energia (oscilações), partículas alfa (radiação), etc.

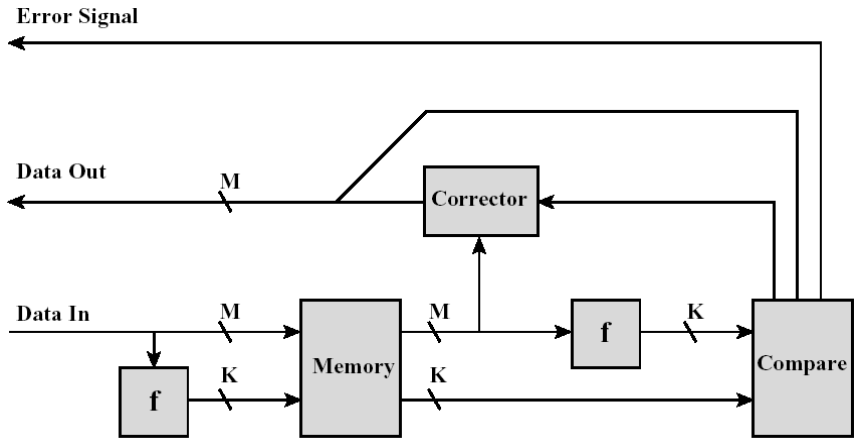


Figura 3: Código de correção de erros.

A Figura 1 acima mostra o processo de detecção e correção de erros. O processo acontece da seguinte forma: quando um novo dado é armazenado na memória, realiza-se um cálculo sobre o mesmo (representado pelo bloco **f**), para produção de um código que será armazenado juntamente com os dados. Portanto, se a palavra de dados possui M bits e o código K bits, o tamanho final da palavra armazenada será M+K bits.

Ao ler uma palavra da memória, o código armazenado também é lido e utilizado para detectar possíveis erros. Isso é feito da seguinte forma: um novo código é gerado a partir dos bits de dados da palavra, se o mesmo for igual ao que está armazenado é sinal de que não teve erros.

Esta comparação é feita bit a bit utilizando a operação lógica ou-exclusivo sobre as duas entradas. Quando um erro é detectado, pode acontecer duas situações:

1. É possível corrigi-lo ou
2. Não é possível corrigi-lo. Estes códigos são denominados de **códigos de correção de erros**.

Código de Hamming

É o código de correção de erros mais simples, projetado por Richard Hamming. A Figura 2 ilustra, através de diagramas de Venn, o uso desse código para palavras de 4 bits (M=4). Os 4 bits de dados são atribuídos a compartimentos internos (Figura 3.a), os compartimentos restantes são preenchidos com os chamados bits de paridade, escolhidos de modo que o número total de 1s em seu círculo seja par (Figura 3.b).

Portanto, se algum dos bits for modificado fica fácil à detecção do erro. Por exemplo, na Figura 3.c encontramos divergências nos círculos A e C, mas não no B. Como apenas um dos 7 compartimentos pertence a A e C e não a B, o erro pode ser corrigido alterando o bit desse compartimento.

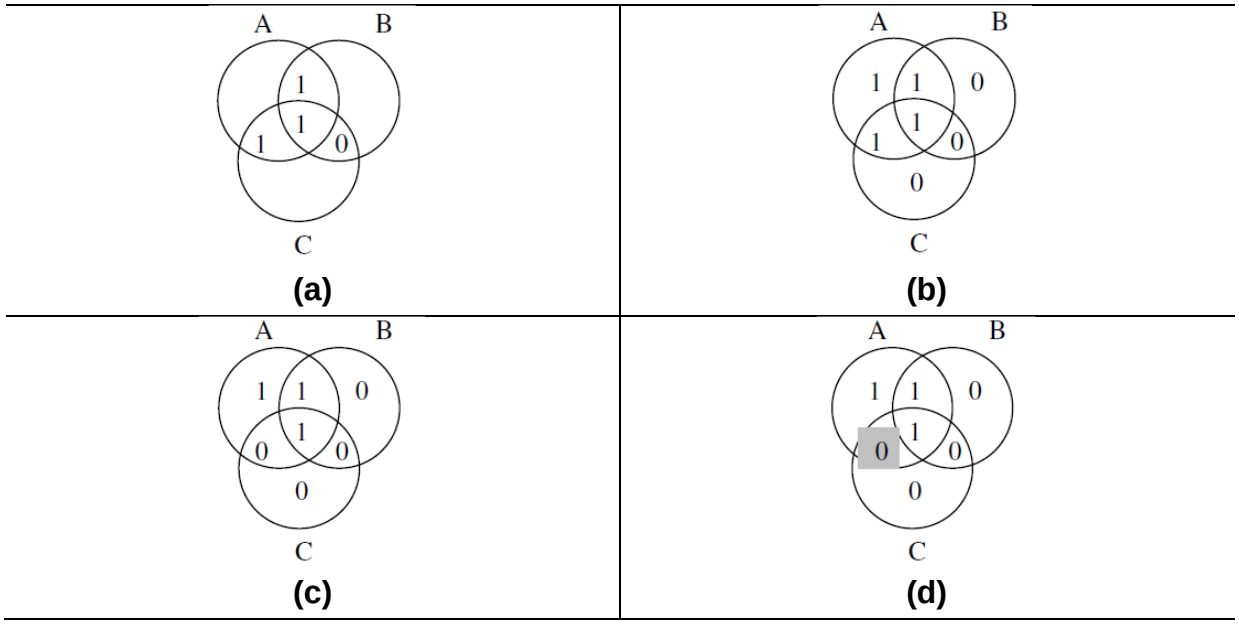


Figura 4: Correção de Erros de Hamming

Vamos considerar um código para detectar e corrigir erros que ocorram em um único bit, usando palavras de 8 bits.

Inicialmente, precisamos determinar o tamanho que o código deve ter. Ao realizar a comparação com as duas entradas de tamanho k, usando o ou-exclusivo, obtemos como resultado bits 0 ou 1.

A palavra resultante é denominada *palavra síndrome* e armazena valores entre 0 e $2^k - 1$. O valor 0 indica que nenhum erro foi detectado, e os $2^k - 1$ valores restantes podem ser utilizados para indicar qual é o bit errado, caso exista um erro. Como pode ocorrer um erro em qualquer um dos M bits de dados ou dos K bits de teste, devemos ter:

$$\mathbf{2^k - 1 \geq M + K}$$

Essa equação fornece o número de bits de código necessários para corrigir erros em um único bit, usando uma palavra com M bits de dados.

	Correção de Erro Único		**Correção de Erro Único/detecção de Erro Duplo**	
Bits de dados	**Bits de Teste**	**% de Aumento**	**Bits de Teste**	**% de Aumento**
8	**4**	**50,0**	**5**	**62,5**
16	**5**	**31,25**	**6**	**37,5**
32	**6**	**18,75**	**7**	**21,875**
64	**7**	**10,94**	**8**	**12,5**
128	**8**	**6,25**	**9**	**7,03**
256	**9**	**3,52**	**10**	**3,91**

Figura 5: Aumento do tamanho da palavra com correção de erros.

A porcentagem de aumento diz respeito ao percentual acrescido ao tamanho da palavra através do uso dos bits de teste. Ex. Para M = 8 precisamos de K = 4, que representa 50 % dos 8 bits de dados.

Assim, para uma palavra síndrome de 4 bits desejamos que:

1. Se todos os bits da palavra síndrome têm valor 0s, não ocorreu erro.
2. Se a palavra síndrome tem apenas um bit com valor 1, ocorreu erro em um dos 4 bits de teste. Nenhuma correção é necessária.

3. Se a palavra síndrome contém mais de um bit com valor 1, o valor numérico da palavra síndrome indica a posição do bit que ocorreu erro. A palavra é corrigida invertendo-se o valor desse bit do dado.

Para satisfazer as características citadas, os bits de dados e de testes são organizados em uma palavra de 12 bits, conforme mostra a Figura 6 as posições dos bits são numeradas de 1 a 12. As posições de bits cujo número é uma potência de 2 são reservadas como bits de teste.

1	2	3	4	5	6	7	8	9	10	11	12	Posição do Bit
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**Número da Posição**
0	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	
0	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	
1	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	
C1	**C2**		**C4**				**C8**					**Bit de Teste**
		M1		**M2**	**M3**	**M4**		**M5**	**M6**	**M7**	**M8**	**Bit de Dados**

Figura 6: Arranjo de bits de dados e bits de teste.

Para calcular os bits de testes usamos o seguinte esquema (⊕ Representa OU-Exclusivo):

C1 = M1 ⊕ M2 ⊕ M4 ⊕ M5 ⊕ M7

C2 = M1 ⊕ M3 ⊕ M4 ⊕ M6 ⊕ M7

C4 = M2 ⊕ M3 ⊕ M4 ⊕ M8

C8 = M5 ⊕ M6 ⊕ M7 ⊕ M8

Cada bit de teste opera sobre todas as posições de bits de dados cujo número contém um valor 1 na coluna de posição correspondente. Assim, as posições 3, 5, 7, 9 e 11 de bits de dados contêm, todas elas, o termo 2^0; as posições de bit 3, 6, 7, 10 e 11 contêm o termo 2^1; as posições de bit 5, 6, 7 e 12 contêm o termo 2^2; e as posições de bit 9, 10, 11 e 12 contêm o termo 2^3. Em outras palavras, a posição de bit n é testada pelos bits Ci, tal que Σ i = n.

Por exemplo, a posição 7 é testada pelos bits nas posições 4, 2 e 1, pois 7 = 4 + 2 + 1.

Exemplo: suponha que a palavra de 8 bits dada como entrada é 00111001, com bit de dados M1 na posição mais a direita. Os cálculos feitos são apresentados a seguir:

C1 = 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 = **1**	**C4** = 0 ⊕ 0 ⊕ 1 ⊕ 0 = **1**
C2 = 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 = **1**	**C8** = 1 ⊕ 1 ⊕ 0 ⊕ 0 = **0**

Se o 3º bit M3 foi alterado de 0 para 1, tem um erro. Qdo bits de testes são recalculados, vem:

C1 = 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 = **1**	**C4** = 0 ⊕ 1 ⊕ 1 ⊕ 0 = **0**
C2 = 1 ⊕ 1 ⊕ 1 ⊕ 1 ⊕ 0 = **0**	**C8** = 1 ⊕ 1 ⊕ 0 ⊕ 0 = **0**

Qdo os novos bits de testes são comparados como os anteriores, forma-se a palavra síndrome:

C8	C4	C2	C1	
0	**1**	**1**	**1**	
0	**0**	**0**	**1**	⊕
0	**1**	**1**	**0**	

O resultado é 0110, o que indica que o bit de posição 6, que corresponde ao 3º bit de dados M3 está errado.