Memória Principal
(RAM – Random Access Memory)

É a parte do computador onde são armazenados programas e dados. É na memória que os processadores lêem e gravam informações.

-- // --

Memórias são compostas por células, as quais têm determinadas características:

Cada uma contém apenas uma informação (um byte).
Todas as células têm o mesmo tamanho, isto é, o mesmo número de bits.

-- //--

Memórias são compostas por células, as quais têm determinadas características (continuação):

Cada uma tem apenas um endereço, isto é, um número associado pelo qual a célula é referenciada pelos programas.
O endereçamento é sequencial.
Células consecutivas têm endereços consecutivos.

--//--

Se a memória contém n células, ela apresentará endereços de 0 (zero) a n-1.

O número de bits utilizados para endereçamento está relacionado ao número máximo de células endereçáveis na memória, e é independente do número de bits por célula.

Exemplo:
Para endereçarmos 12 células (de 0 a 11) precisamos de, no mínimo, 4 bits: de 0000 a 1011.

--//--

Se a célula é composta de k bits, ela pode conter apenas 2k combinações diferentes de bits.

Exemplo:
Para k=1 → 21 = 2 → 0 ou 1
Para k=2 → 22 = 4 → 00, 01, 10 ou 11

--//--

Temos uma memória com 212 células de 8 bits cada. Temos outra memória com 212 células de 60 bits cada.

Quantos bits de endereçamento precisaremos em cada um dos casos?

--//--

Ordenação dos Bytes

Uma palavra pode ser composta por mais de um byte.

Os bytes de uma palavra podem ser numerados da esquerda para a direita, ou vice-versa, dependendo do projeto do fabricante.

--//--

O computador em que a numeração dos bytes das palavras atribui um número maior (endereço do byte dentro da palavra) à parte mais significativa da palavra é chamado de BIG ENDIAN:

Na estrutura Big Endian os bits 1102 estarão no byte 3, e na estrutura Little Endian estarão no byte 0 (zero), apesar de que, em ambos os casos, o endereço da palavra será 0 (zero).

--//--

Como o Processador se comunica com a Memória?

Vamos analisar as duas situações possíveis:

A. O Processador precisa ler um byte na memória.

B. O Processador precisa gravar um dado na memória.
Para isto vamos relembrar os dois registradores específicos anteriormente citados:

Registrador de Instruções (IR)
Contém a instrução a ser executada.

Registrador Contador de Programa (PC)
Contém endereço da próxima instrução a ser executada.

--//--

1º passo:
Carregar o endereço do byte desejado.

2º passo:
Enviar um sinal de leitura.

3º passo:
A controladora da memória “puxa” o endereço desejado.

4º passo:
A memória carrega o conteúdo solicitado no IR.

5º passo:
A UC “puxa” o conteúdo disponibilizado.

--//--

1º passo:
Carregar no PC o endereço onde o conteúdo deverá ser armazenado.

2º passo:
Carregar no IR o conteúdo que deverá ser armazenado.

3º passo:
Enviar um sinal de armazenamento.

4º passo:
A controladora da memória “puxa” o endereço solicitado para o armazenamento.

5º passo:
A controladora da memória “puxa” o conteúdo e armazena-o no endereço solicitado.

--//--

Códigos com correção de erros

Quando um dado transita de um módulo para outro dentro da máquina (por exemplo: da memória para o HD), pode sofrer alterações indevidas por interferências externas: variação de tensão, defeito no hardware, interferência de ondas eletromagnéticas, etc.

Daí surge a necessidade de adotarmos algoritmos para detectar e/ou corrigir estes bits “trocados” indevidamente no byte em trânsito.

--//--

Bit de Paridade:

Foi uma forma pioneira de resolver o problema da perda da integridade de um byte
Consistia em adotar um bit a mais (além dos 8 bits que compõem o byte) para detectar de houve dano ao byte.
Era um processo de eficiência fraca, mas melhor que nenhum.

Paridade Par
Quantidade par de 1s no byte, incluindo o(s) bit(s) de paridade.

Paridade Ímpar
Quantidade ímpar de 1s no byte, incluindo o(s) bit)s) de paridade.

Portanto, se um bit fosse trocado durante a transmissão, o número de 1s no receptor estará com a paridade errada, e o receptor saberá que ocorreu um erro no trajeto.

Por que a eficiência deste algoritmo era fraca?

Se houvesse uma dupla inversão de bits contrários o algoritmo não detectava: de 0 para 1 e de 1 para 0 simultaneamente.
O algoritmo poderia detectar o erro de paridade, mas não podia corrigi-lo.

--//--

Código de Hamming:

É um método para detectar e corrigir erros de transmissão.
Admite somente uma inversão indevida de bit.
A um byte de n bits adiciona-se k bits de paridade formando um novo byte de n+k bits.
Este byte maior é utilizado somente para o trajeto entre dois módulos.

As posições dos bits (dentro do byte) são numeradas de 1 até n+k.
Todos os bits cuja posição física no byte sejam uma potência de 2 são de paridade, e não de dado.
Este algoritmo é genérico, pode ser aplicado a qualquer conjunto de caracteres.

Cada bit de paridade checa posições específicas do caractere, de modo que a quantidade total de 1s esteja de acordo com o critério de paridade adotado (par ou ímpar).
Cada bit é checado pelos bits de paridade P1, P2, ..., Pj tais que P1 + P2 + ... + Pj = n, ou seja:

Utilizaremos como exemplo o conjunto de caracteres ASCII, ou seja, cada byte tem 7 bits.

O bit da 5ª posição é checado pelos bits de paridade P1 e P4 pois 1 + 4 = 5.
O bit da 11ª posição é checado pelos bits de paridade P1, P2 e P8 pois 1 + 2 + 8 = 11.

Desta forma, e ainda utilizando o ASCII como exemplo:

O bit P1 checa os bits de posição 1, 3, 5, 7, 9 e 11
O bit P2 checa os bits de posição 2, 3, 6, 7, 10 e 11
O bit P4 checa os bits de posição 4, 5, 6 e 7
O bit P8 checa os bits de posição 8, 9, 10 e 11

--//--

Distância de Hamming:

É um método para detectar e corrigir erros de transmissão.
Admite várias inversões indevidas simultâneas de de bits.

$$\text{Desperdício} = \frac{n}{2^{n-1}} \times 100$$