Cache e Hamming

Exercício 1:

Papel da Memória Cache: A memória cache é uma memória de acesso ultrarrápido situada entre a CPU e a memória principal (RAM). Seu objetivo é armazenar dados e instruções frequentemente acessados pela CPU para reduzir o tempo de acesso. A cache melhora significativamente o desempenho do sistema ao minimizar a latência do acesso à memória, pois ela armazena uma cópia de dados ou instruções que a CPU usa repetidamente.

L1 mais rápida que L2 e L3:

- Cache L1 (nível 1) é mais rápida porque está localizada diretamente dentro do processador, próxima aos núcleos da CPU. Ela é menor em capacidade (geralmente de 16 KB a 64 KB), mas é extremamente rápida para fornecer dados e instruções de maneira eficiente.
- Cache L2 (nível 2) é maior e mais lenta que a L1, mas ainda está fisicamente próxima ao processador (geralmente dentro do chip ou próximo). A capacidade da L2 varia entre 128 KB a 512 KB ou mais.
- Cache L3 (nível 3) é maior ainda (geralmente de 1 MB a 8 MB ou mais), mas é mais lenta que a L1 e L2. Ela pode ser compartilhada entre vários núcleos da CPU.

Impacto do Cache Hit e Cache Miss:

- Cache Hit ocorre quando os dados solicitados pela CPU estão presentes na cache. Isso reduz significativamente o tempo de acesso, pois o processador obtém os dados muito rapidamente.
- Cache Miss ocorre quando os dados solicitados não estão na cache, o que força a CPU a acessar a memória principal (RAM), um processo muito mais lento. Isso afeta o desempenho negativamente, pois o tempo de acesso à RAM é muito maior.

Exercício 2:

a) **LRU (Least Recently Used):**

- O algoritmo LRU substitui o bloco de cache que não é acessado há mais tempo. Ele mantém um histórico das referências feitas aos blocos de cache e, ao ocorrer uma falha de cache, o bloco que foi menos recentemente utilizado é o escolhido para ser substituído.
- Vantagem: Boa escolha em cenários em que dados recentemente acessados são mais prováveis de serem acessados novamente.

b) **FIFO (First In, First Out):**

- O algoritmo FIFO substitui o bloco mais antigo no cache, ou seja, o primeiro bloco que entrou no cache será o primeiro a ser substituído quando ocorrer uma falha de cache.
- Desvantagem: Não leva em consideração o quão frequentemente ou recentemente os dados foram acessados, o que pode resultar em substituições ineficientes.

c) **Aleatório:**

- O algoritmo aleatório escolhe um bloco de cache para substituição de forma aleatória, sem considerar a ordem de acesso ou qualquer outra métrica.
- Desvantagem: Pode não ser muito eficiente, pois pode escolher blocos que ainda são necessários com frequência.

Situação em que o algoritmo FIFO pode não ser eficiente:

- FIFO pode não ser eficiente em cenários onde há uma alta taxa de reutilização de dados em blocos de cache. Por exemplo, em sistemas com acesso a dados que formam um padrão cíclico (por exemplo, os dados A, B, C, D, E são acessados repetidamente), FIFO pode substituir um bloco que foi acessado mais recentemente, prejudicando o desempenho, enquanto LRU poderia manter o bloco que será acessado em seguida.

Exercício 3:

Processo de Mapeamento Direto: No mapeamento direto, a memória principal é dividida em blocos, e a cache também é dividida em linhas. Cada linha de cache pode armazenar dados de um único bloco da memória principal. A posição de um bloco de memória em cache é determinada por uma função de mapeamento. A maneira mais simples é usar o módulo da divisão entre o número total de blocos de memória e o número de linhas de cache.

- Se a memória principal tem 16 blocos e a cache tem 4 linhas, podemos usar o mapeamento direto para determinar a linha da cache para o bloco de endereço 10.
- Dividimos o número de blocos de memória pelo número de linhas da cache: $16 \div 4 = 4$
- Isso significa que cada linha de cache pode armazenar 4 blocos diferentes da memória principal.
- Para encontrar a linha de cache para o bloco 10, usamos a operação módulo: $10 \bmod 4 = 2$
- Portanto, o bloco 10 será mapeado para a linha 2 da cache.

Exercício 4:

a) Calcule o código de Hamming correspondente, inserindo os bits de paridade necessários nas posições corretas. Apresente o código final obtido.

Primeiro, a palavra de dados é **11011010** (8 bits). O código de Hamming insere bits de paridade em posições específicas (potências de 2) para detectar e corrigir erros. As posições de paridade são P1, P2, P4, P8.

Passos para calcular o código de Hamming:

- A palavra original: 11011010 (8 bits).
- Inserir 4 bits de paridade: P1,P2,P4,P8
- Agora temos uma palavra de 12 bits: **P1 P2 1 P4 1 1 0 P8 1 0 1 0**.

Agora, calculamos os bits de paridade:

- **P1** (bit de paridade na posição 1) verifica os bits 1, 3, 5, 7, 9, 11:
  $P1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$, logo, P1 = 1
- **P2** (bit de paridade na posição 2) verifica os bits 2, 3, 6, 7, 10, 11:
  $P2 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$, logo P2 = 0
- **P4** (bit de paridade na posição 4) verifica os bits 4, 5, 6, 7, 12:
  $P4 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$, logo P4 = 0
- **P8** (bit de paridade na posição 8) verifica os bits 8, 9, 10, 11, 12:
  $P8 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$, logo P8 = 1

Código final de Hamming: **101101101010**.

b) **Simule um erro na transmissão, alterando o bit D5 dessa palavra. Utilize o código de Hamming para identificar a posição do erro e corrigir o dado corrompido.**

- A palavra transmitida é **101101101010**, e o erro ocorre no bit D5, ou seja, o bit na posição 5 (que é 0).
- Vamos verificar os bits de paridade.
  - $P1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$, logo, está correto.
  - $P2 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$, logo, está correto.
  - $P4 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$, logo, está correto.
  - $P8 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1$, logo, está correto.
- Com base nas verificações dos bits de paridade, identificamos que o erro ocorreu no **bit D5** (na posição 5).
- Corrigindo o erro, o dado original seria **11011010**.

Essas são as explicações para os exercícios. Se precisar de mais detalhes, estarei por aqui!