UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ORGANIZAÇÃO DE COMPUTADORES

1) Um computador possui uma memória principal com capacidade para armazenar dados de 8 bits em cada uma das suas N células. O barramento de endereços tem 12 bits de tamanho. Quantos bytes poderão ser armazenados na memória.

O barramento de endereços tem 12 bits, ou seja, ele pode endereçar:

2^{12} = 4096 células de memória

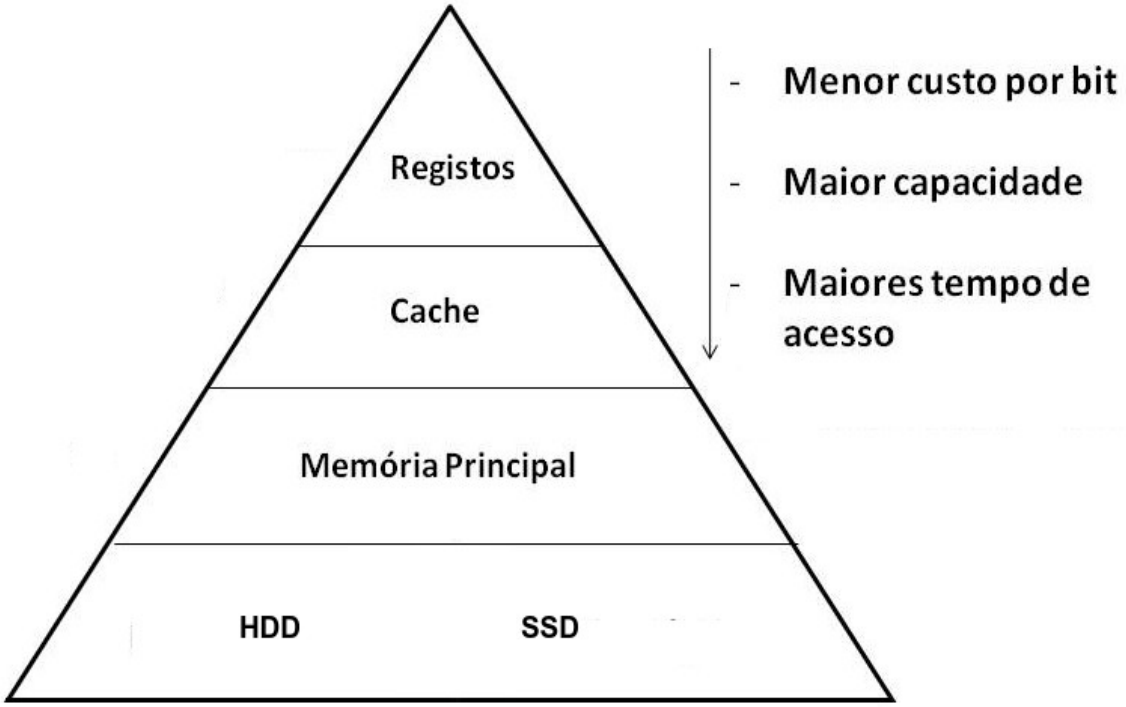
Cada célula armazena 8 bits, ou seja:

8 bits = 1 byte

Total de bytes = 4096 células × 1 byte por célula = 4096 bytes

2) Apresente a hierarquia de memória posicionando os diferentes tipos de memória. Porque a apresentação na forma de pirâmide é uma boa analogia da hierarquia de memória?

A hierarquia de memória é normalmente apresentada como uma pirâmide porque representa bem a relação entre tempo de acesso, capacidade de armazenamento, custo entre outras características de cada tipo de memória em um sistema computacional. A estrutura é assim:



3) Compare uma memória principal e uma memória cache em termos de tempo de acesso, capacidade e temporariedade de armazenamento de dados.

Característica	Memória Cache	Memória Principal (RAM)
Tempo de acesso	Muito rápido (alguns nanossegundos)	Razoavelmente rápido (mais lento que cache)
Capacidade	Pequena (KB a poucos MB)	Grande (GB, como 8GB, 16GB, etc.)
Temporariedade dos dados	Armazena dados **temporários e mais usados recentemente**, que serão acessados com frequência em breve	Armazena dados **temporários em uso pelo sistema operacional e aplicatições**, mas com menor frequência de acesso imediato

4) Explique os conceitos de localidade espacial e localidade temporal. Como estes conceitos foram utilizados na implementação das memórias cache?

Localidade Espacial: Se uma informação foi acessada, é provável que informações próximas a ela (em endereço de memória) também sejam acessadas em breve.

Exemplos: - Acesso sequencial e consecutivo das instruções na memória pelo PC

- A leitura de um vetor/array elemento por elemento: vetor[0], vetor[1], vetor[2]...

Aplicação na cache: Quando a cache carrega um dado, ela traz junto um bloco de memória adjacente (conjunto de bytes vizinhos), antecipando que eles serão necessários logo.

Isso é feito através das linhas de cache, que armazenam blocos inteiros da memória principal, permitindo que se carregue na cache não apenas o dado do endereço atual solicitado mas também os dados dos endereços adjacentes.

Localidade Temporal: Se uma informação (dado ou instrução) foi acessada recentemente, é provável que ela seja acessada novamente em breve.

Exemplo: - Um laço for que acessa repetidamente a mesma variável ou instrução.

Aplicação na cache: A memória cache mantém os dados mais recentemente acessados, esperando que eles sejam usados novamente logo.

5) Descreva detalhadamente cada uma das ações que ocorrem durante o acesso de leitura da memória considerando a existência de uma memória cache que utiliza mapeamento associativo?

No mapeamento associativo, qualquer bloco da memória principal pode ser armazenado em qualquer linha da cache. Isso oferece flexibilidade máxima e aproveitamento ideal do espaço da cache, mas exige mais recursos para a busca simultânea em todas as linhas da cache.

Passo a passo: Acesso de leitura na cache com mapeamento associativo

1. CPU solicita o endereço de memória

A CPU precisa ler um dado localizado em um determinado endereço da memória principal.

Esse endereço é dividido logicamente em:

RÓTULO (TAG): Identifica qual bloco da RAM está armazenado.

DESLOCAMENTO (OFFSET): Indica a posição do dado dentro do bloco ou linha da cache

2. Cache faz busca pela TAG

Como o mapeamento é associativo, a busca ocorre em todas as linhas da cache simultaneamente.

A cache compara o RÓTULO do endereço requisitado com os RÓTULOS armazenados em cada uma de suas linhas.

Esse processo é feito com comparadores atuando em paralelo.

3. Verificação de HIT ou MISS

HIT (acerto):

O RÓTULO foi encontrada em alguma linha válida da cache.

O dado está na cache → o acesso é feito diretamente da cache (rápido).

MISS (falha):

O RÓTULO não foi encontrado em nenhuma linha.

O dado não está na cache → é necessário buscá-lo na memória principal.

4. Se HIT: Leitura direta

O processador acessa o dado usando o DESLOCAMENTO dentro do bloco localizado.

O acesso é concluído rapidamente, com tempo típico de acesso à cache em nanossegundos.

5. Se MISS: Acesso à memória principal

A cache solicita o bloco correspondente ao endereço da RAM.

A memória principal envia o bloco completo.

O dado requisitado é lido e:

O bloco é armazenado na cache, substituindo outro bloco conforme a política de substituição (por exemplo, LRU – Least Recently Used).

6. Atualização da cache

A linha da cache que receber o novo bloco terá seu RÓTULO atualizada com o RÓTULO do bloco carregado.

Um bit de validade (valid bit) é ativado indicando que a linha está ocupada com um bloco válido.

Resumindo

CPU → endereço lógico

↓

Extrai RÓTULO e DESLOCAMENTO

↓

Compara RÓTULO do endereço com RÓTULO de todas as linhas da cache

↓

→ [HIT] → Acessa dado na linha da cache da cache usando o DESLOCAMENTO

↓

→ [MISS] → Busca o bloco que corresponde o endereço na RAM

↓

Substitui linha na cache

↓

Atualiza TAG e valid bit

↓

Acessa dado na cache usando o DESLOCAMENTO

6) Supondo um processador com barramento de endereços de 24 bits, qual a quantidade de células de memória endereçáveis por este processador?

Com **N bits**, ele pode gerar até 2^N endereços únicos:

2^{24} =16.777.216 células de memoria

Se **cada célula endereçada armazena 1 byte**

2^{24} células = 2^{24} bytes = 16 MB

7) Qual a política de substituição de dados implementada no mapeamento direto?

No **mapeamento direto**, **não existe uma política de substituição no sentido tradicional**, como acontece nos mapeamentos associativos.
A substituição de dados é **determinística e automática**: O **novo bloco sempre substitui o bloco que ocupa a linha mapeada pelo seu endereço.**

8) Quais as diferenças e as implicações nas políticas de escrita da memória cache: escrita em ambas e escrita no retorno?

Escrita em ambas (Write-Through): Sempre que a CPU escreve um dado na cache, o mesmo dado também é imediatamente escrito na memória principal

Implicações:

Vantagem	Desvantagem
Mantém a memória principal **sempre atualizada**	**Mais lenta**, pois cada escrita atinge dois locais

Vantagem	Desvantagem
Fácil de implementar	Aumenta o tráfego no barramento de memória
Menor risco de perda de dados em falhas	Desempenho menor em programas que escrevem muito

Escrita no retorno (Write-Back): A CPU escreve apenas na cache. A memória principal só é atualizada quando o bloco modificado é substituído na cache.

Implicações:

Vantagem	Desvantagem
Reduz o tráfego no barramento	A memória principal pode **ficar desatualizada** por um tempo
Melhor desempenho em escritas frequentes	Implementação mais complexa (controle de dirty bits)
Economiza energia e tempo	Em caso de falha, dados não salvos podem ser perdidos

9) Cite políticas de substituição que podem ser implementadas em um sistema que utiliza memória cache. Explique cada uma delas e apresente um exemplo numérico que diferencie-as.

Quando ocorre um **cache miss** e a cache está cheia, é necessário **escolher qual bloco será removido (substituído)** para dar lugar ao novo bloco. As **políticas de substituição** determinam **qual bloco da cache será removido**.

1. LRU (Least Recently Used) – Menos recentemente usado

   Remove o bloco que ficou mais tempo sem ser acessado.

   Assume localidade temporal: se não foi usado recentemente, é improvável que será em breve.

   Vantagem: Desempenho geralmente bom.

   Desvantagem: Requer controle de histórico de acesso → mais complexa de implementar.

2. FIFO (First In, First Out) – Primeiro que entrou, primeiro que sai

   Remove o bloco mais antigo, ou seja, aquele que está há mais tempo na cache, independentemente de uso recente.

   Vantagem: Fácil de implementar com filas.

   Desvantagem: Pode remover blocos que ainda estão sendo usados.

3. LFU (Least Frequently Used) – Menos frequentemente usado

   Remove o bloco que foi menos acessado no total.

Ideal para padrões estáveis de acesso.

Vantagem: Boa em cargas previsíveis.

Desvantagem: Pode manter blocos antigos que foram acessados muito no passado, mas não mais úteis agora.

4. Random (Aleatória)

Escolhe aleatoriamente um bloco da cache para remover.

Vantagem: Implementação simples, sem sobrecarga.

Desvantagem: Pode remover um bloco importante por acaso.

Cada política tem impacto direto no desempenho do sistema, dependendo do padrão de acesso aos dados:

LRU costuma oferecer melhor desempenho em geral, mas é mais complexa.

FIFO é simples, mas pode descartar blocos ainda úteis.

LFU é boa com dados frequentemente reutilizados.

Random é útil em hardware simples, mas menos eficiente.

Exemplo Numérico Comparativo

Situação:

- Cache com 4 linhas: capacidade para 4 blocos ao mesmo tempo.
- Sequência de acesso aos blocos: A, B, C, D, A, E

Objetivo:

- ver qual bloco será removido ao tentar inserir o bloco E, quando a cache já estiver cheia com 4 blocos

Passo a passo até a inserção de E:

1. Acessa A → entra na cache
2. Acessa B → entra na cache
3. Acessa C → entra na cache
4. Acessa D → entra na cache
5. Acessa A → já está na cache (**HIT**)
   Agora a cache está cheia com: A, B, C, D
6. Acessa E → não está na cache (**MISS**) → precisa **substituir um bloco**

Aplicação das políticas na substituição do bloco:

1. LRU (Least Recently Used)

- Últimos acessos foram: C, D, A
- B foi o **último a ser usado**, lá atrás.

  Remove B
- **Cache final:** A, C, D, E

2. FIFO (First In, First Out)

- Ordem de entrada: A (1º), B (2º), C (3º), D (4º)
- O mais antigo é A, **mas ele foi acessado novamente recentemente — isso não importa** no FIFO.

  Remove A
- **Cache final:** B, C, D, E

3. LFU (Least Frequently Used)

- Contagem de acessos:
  - A = 2 vezes
  - B, C, D = 1 vez cada
- Há um empate entre B, C e D. Vamos supor que o sistema remove **aleatoriamente entre os empatados → C**

  Remove C
- **Cache final:** A, B, D, E

4. Random (Aleatória)

- Escolhe aleatoriamente entre os 4 blocos (A, B, C, D)
- Suponha que sorteia e remove D

  Remove D
- **Cache final:** A, B, C, E

Resumo final do exemplo com cache de 4 linhas:

Política	Bloco removido	Cache final
LRU	B	A, C, D, E
FIFO	A	B, C, D, E
LFU	C (empatado)	A, B, D, E
Random	D	A, B, C, E

10) Uma memória principal tipo DDR funciona em um barramento com largura de 64 bits rodando a 200MHZ com ciclos de wait-states 4-1-1-1. Pergunta-se: Qual a largura de banda máxima teórica do sistema? b) Qual a largura de banda efetiva do mesmo?

Entendendo os dados:

Tipo de memória: DDR (Double Data Rate)

Frequência do barramento: 200 MHz

Ciclos de acesso (wait-states): 4-1-1-1

Cada acesso (burst) transfere 4 palavras consecutivas

DDR transfere 2 palavras por ciclo de clock real (por isso o “Double Data Rate”)

a) Largura de banda máxima teórica (LBMT)

LBMT = Taxa de transferência × largura do barramento de dados

Considerando a largura de 64 bits = 8 bytes, e que DDR transfere 2 dados por ciclo:

Taxa de transferências = 200 MHz × 2=400 milhões de transferências/seg

Assim:

LBMT = 400 $\times 10^6$ transf/seg × 8 bytes/transf = 3200 MB/seg

b) Largura de banda efetiva (LBE)

Wait-states 4-1-1-1 no modo Burst significa:

Primeira transferência: 4 ciclos

Próximas 3 transferências: 1 ciclo cada

Total para 4 transferências: 4 + 1 + 1 + 1 = 7 ciclos

Assim para 4 transferências considerando a LBMT precisamos 4 ciclos e para as mesmas 4 transferências considerando o modo Burst precisamos 7 ciclos:

LBE = LBMT * (ciclos LBMT / ciclos Burst)

LBE = 3200 MB/seg * (4 / 7) = 1828,57 MB/seg

11) Apresente e explique as 4 categorias de melhora da performance da cache existentes de acordo com Patterson & Hannessy.

Segundo **Patterson & Hennessy**, **existem quatro categorias principais para melhorar o desempenho da memória cache**. Essas categorias atacam **as causas principais dos misses (falhas de cache)** e/ou tentam reduzir seu impacto no desempenho.

1. Reduzir a taxa de misses (miss rate)

Diminuir a **frequência com que a cache falha** ao encontrar o dado pedido.

Como?

- Aumentar o **tamanho da cache** (mais blocos armazenados).
- Usar **mapeamento associativo por conjunto** (reduz conflitos).
- Usar políticas inteligentes de **substituição de blocos** (como LRU).
- Aplicar **prefetching** (trazer dados antecipadamente com base em padrões).

Impacto:

Melhora a eficiência da cache e reduz os acessos lentos à memória principal.

2. Reduzir o tempo de acesso à cache (hit time)

Diminuir o tempo necessário para obter um dado **que está na cache** (quando ocorre um **hit**).

Como?

- Usar **caches menores ou mais simples** (acesso mais rápido).
- Implementar **cache L1 com baixa latência**.
- Usar **arquiteturas paralelas ou pipelines** no acesso à cache.

Impacto:

Mesmo com alta taxa de acerto, se o tempo de acesso for alto, o desempenho global sofre.

3. Reduzir o tempo de penalidade por miss (miss penalty)

Diminuir o tempo que se perde ao **buscar um dado na memória principal** após uma falha (miss).

Como?

- Usar **memória principal mais rápida** (ex: DDR5 em vez de DDR3).
- Usar **cache de múltiplos níveis (L1, L2, L3)**.
- Implementar **substituição não bloqueante (non-blocking cache)** que permite continuar processando enquanto espera o dado.

Impacto:

Minimiza o impacto negativo dos misses no desempenho geral do sistema.

4. Aumentar a taxa de acertos (hit rate) com prefetching inteligente

Trazer dados **antecipadamente** para a cache com base em padrões de acesso, **antes que o processador os solicite**.

Como?

- Implementar **prefetching por hardware** (detectando acessos sequenciais, por exemplo).
- Implementar **prefetching por software** (instruções explícitas de pré-busca no código).
- Usar **heurísticas adaptativas** que ajustam o prefetch conforme o comportamento do programa.

Impacto:

Reduz significativamente o número de misses **sem esperar o pedido explícito da CPU**.

12) O que é *memória virtual*? Para que é usada? Quem dá suporte? Quem gerencia? Quais as técnicas de implementação?

A memória virtual é uma abstração da memória principal que permite que um processo (programa em execução) acredite que dispõe de um espaço de endereçamento contínuo e grande, mesmo que fisicamente ele não esteja todo na RAM. Em outras palavras: a memória virtual **dá a ilusão de que há mais memória do que a física disponível**, usando o armazenamento secundário (como o disco) como extensão da RAM.

Principais objetivos:

Isolamento de processos: Cada processo trabalha como se tivesse sua própria memória privada — sem interferir em outros.

Execução de programas maiores que a RAM física: Parte do programa pode ficar na RAM, outra parte no disco, e o sistema vai trazendo as partes conforme necessário (paginação).

Facilidade para o programador: Ele não precisa se preocupar com os endereços físicos pois trabalha com endereços virtuais simples e contínuos.

S**egurança e proteção**: Um processo não pode acessar a memória de outro processo.

Quem dá suporte é o hardware, através da TLB, do registrador apontador da tabela de páginas, dos modos de execução (modo kernel e modo usuário), da chamada ao sistema operacional para tratar falhas de acesso.

Quem gerencia é o Sistema Operacional, pois mantém as tabelas de páginas (por processo), decide o que vai para a RAM e o que vai para o disco, trata page faults (quando o dado está no disco, não na RAM), aplica políticas de substituição de páginas, entre outras funcionalidades de gerencia.

Técnicas de implementação da memória virtual:

1. Paginação (Paging)

2. Segmentação (Segmentation)

3. Paginação com segmentação

13) Quais são os recursos que devem ser implementados no processador para dar suporte a Memória Virtual?

Para que um processador possa suportar memória virtual, ele precisa oferecer recursos de hardware específicos que permitam a tradução de endereços, o controle de proteção de memória, e a interação eficiente com o sistema operacional.

Recursos que o processador deve implementar para suportar memória virtual:

1. MMU (Memory Management Unit)

A principal unidade de hardware para suporte à memória virtual.

Realiza a tradução de endereços virtuais para endereços físicos.

Faz checagens de proteção (leitura/escrita/execução).

Detecta e gera exceções para page faults.

Trabalha em conjunto com a TLB e com as tabelas de página fornecidas pelo sistema operacional.

2. TLB (Translation Lookaside Buffer)

Cache especial da MMU.

Armazena traduções recentes de endereços virtuais → físicos.

Reduz a latência da tradução de endereços (acesso rápido).

Pode ser dividido em TLBs separados para instruções e dados (iTLB e dTLB).

3. Suporte a Modos de Execução (User vs. Kernel Mode)

Permite que o processador opere em dois modos distintos:

Modo usuário: acesso restrito à memória.

Modo kernel: acesso completo (usado pelo SO).

Isso evita que programas de usuário acessem áreas protegidas da memória.

4. Registradores de Gerenciamento de Memória

Esses registradores ajudam a controlar a MMU e o acesso à memória:

Page Table Base Register (PTBR): Aponta para o início da tabela de páginas.

Control registers (como o CR3 em arquiteturas x86): usados para configurar o gerenciamento de memória.

Bits de controle como:

Bit de presença (valid bit): indica se a página está na RAM.

Bit de modificação (dirty bit): indica se a página foi alterada.

Bits de proteção (leitura, escrita, execução).

5. Unidade de Exceções (para Page Faults)

Detecta acessos inválidos ou ausentes na RAM.

Gera uma exceção de page fault, que é tratada pelo sistema operacional.

Permite carregar páginas do disco para a RAM sob demanda.

6. Suporte a Tabelas de Página por Níveis (multi-level page tables) (opcional)

Essencial para trabalhar com endereços virtuais de 32 ou 64 bits.

Reduz o consumo de memória com tabelas de páginas muito grandes.

O processador precisa de lógica para percorrer vários níveis da tabela (nível 1, 2, 3...).

14) Explique como se dá o processo de tradução do endereço lógico em físico com e sem TLB em um sistema que implementa memória virtual usando paginação. Apresente um exemplo em que cada página tem 4 KB, os endereços virtuais são de 40 bits e os endereços físicos são de 32 bits.

Tradução usando Paginação

Endereço virtual (ou lógico): Endereço usado pelo programa.

Dividido em:

Número da página (page number)

Deslocamento (offset): posição dentro da página

Endereço físico: Endereço real na memória RAM

Dividido em:

Número do quadro (frame number)

Deslocamento (mesmo valor do offset)

Tradução sem TLB (Translation Lookaside Buffer)

1. A CPU acessa um endereço virtual.

2. A MMU não encontra o dado na TLB, então acessa diretamente a tabela de páginas (mais lento).

3. Obtém o número do quadro correspondente à página.

4. Combina com o offset para formar o endereço físico.

5. O dado é acessado.

Tradução com TLB (Translation Lookaside Buffer)

1. A CPU acessa um endereço virtual.
2. A MMU consulta a TLB.
3. Se a entrada estiver na TLB (TLB hit):
   - O número da página é traduzido rapidamente para o número do quadro.
   - Concatena-se com o offset → endereço físico.
4. Se for um TLB miss:
   - A MMU consulta a tabela de páginas na RAM.
   - Atualiza a TLB com essa nova tradução.

Exemplo:

Parâmetros fornecidos:

- Tamanho da página = 4 KB = 2^{12} bytes
- Endereço virtual = 40 bits
- Endereço físico = 32 bits

Divisão do endereço virtual de 40 bits:

- Offset:
  - Como a página tem 4 KB → 2^{12}, então: 12 bits para o offset
- Número da página virtual:

  40 bits (total) − 12 bits (offset) = 28 bits

Divisão do endereço físico de 32 bits:

- Offset: permanece com 12 bits
- Número do quadro físico (frame number):

  32 bits (total) − 12 bits (offset) = 20 bits
- Ou seja, a memória RAM pode conter até:

  2^{20} quadros = 1 M quadros
- Tamanho da memória física = número de quadros x tamanho de cada quadro

  Tamanho da memória física = 1 Mquadros x 4 KB/ quadro = 4 GBytes

  Tamanho da memória física = 2^{20} x 2^{12} = 2^{32} = 4 GBytes

Exemplo prático:

Suponha:

- **Endereço virtual gerado pela CPU**:
  `0x000C345678 (em hexadecimal)`

- em binário temos:

  `0000 0000 0000 1100 0011 0100 0101 0110 0111 1000`

1. Separar os 28 bits mais significativos (page number):

`0000 0000 0000 1100 0011 0100 01  → Page Number (28 bits)`

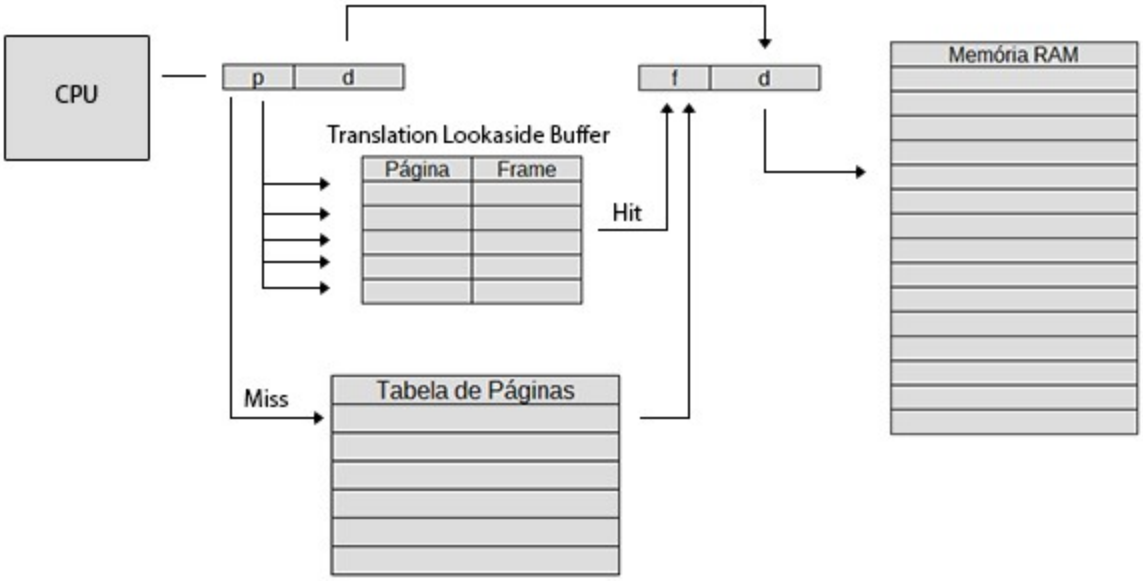
2. Separar os 12 bits finais (offset):

`0101 0110 0111 1000  → Offset (12 bits)`

3. Consulta:

- Verifica na TLB se o page number `0000 0000 0000 1100 0011 0100 01` está na TLB
- Se **TLB hit**:
  - pega o número do quadro diretamente e forma o endereço físico:

    Frame Number (20 bits) + Offset (12 bits) = Endereço físico (32bits)
- Se **TLB miss**:
  - A MMU consulta a tabela de páginas na RAM usando o page number
  - Obtém o frame number
  - Atualiza a TLB com essa entrada
  - Acessa o endereço físico usando o frame number + offset


CPU
p
d
f
d
Memória RAM
Translation Lookaside Buffer
Página
Frame
Hit
Miss
Tabela de Páginas