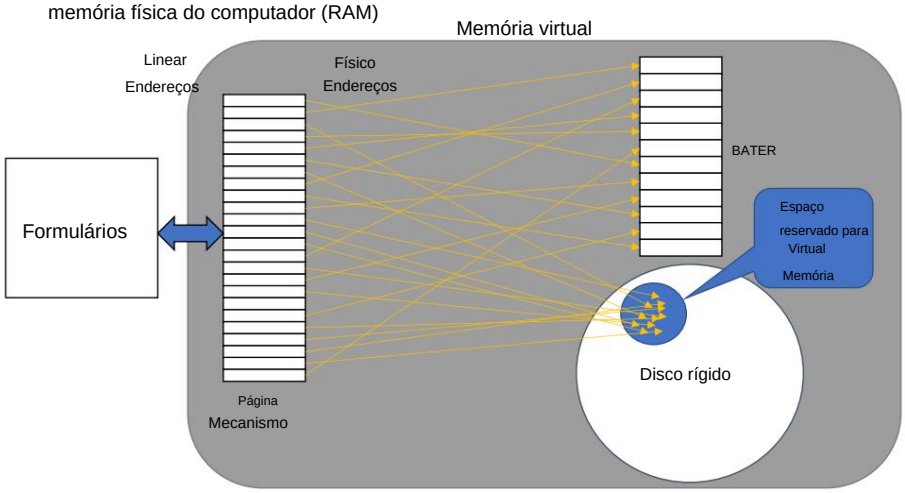
Memória virtual
Cache
Técnicas para melhorar o desempenho

1

Memória virtual

Usando a memória virtual, os aplicativos podem ter acesso a mais memória do que a memória física do computador (RAM)



2

Memória virtual

Quando um aplicativo precisa de memória, por exemplo 100 MB, o Sistema Reserva um espaço **contínuo** em endereços virtuais

O aplicativo recebe do Sistema Operacional um ponteiro para o primeiro byte

Usando um índice adicionado ao ponteiro, o aplicativo pode usar todos os 100 MB de memória

O aplicativo vê esta memória como uma memória física

O endereço nunca muda ao longo da execução do programa

O espaço de endereço é dividido em blocos de 4 KB

Cada bloco pode ser mapeado na RAM ou no disco rígido

O aplicativo não possui, ao mesmo tempo, todos os blocos na RAM

3

Memória virtual

Quando o aplicativo aponta para um bloco que está mapeado no disco rígido, o sistema lê esse bloco do disco rígido e o grava em qualquer bloco livre da RAM física

Logo depois disso, o aplicativo pode usar os dados

Esta é a razão pela qual o computador diminui o desempenho dos aplicativos ao usar a memória virtual

Se nenhum bloco de RAM estiver livre, o sistema transfere, pelo menos, um bloco de RAM para o disco rígido

Para melhorar o desempenho, o sistema mantém um grande número de blocos livres na RAM

Isso é importante para liberar espaço sempre que um aplicativo precisar de mais memória, e o sistema pode oferecer essa memória imediatamente.

4

Memória virtual

O sistema seleciona os blocos de RAM que não são usados há muito tempo para liberar, colocando-os no disco rígido

O sistema mantém um *mecanismo de paginação* que mapeia todos os blocos na RAM ou no disco rígido

A posição física de qualquer bloco pode mudar ao longo da execução do programa, mas essa mudança é totalmente transparente para os aplicativos, pois eles não sabem onde está, fisicamente, a memória utilizada

5

Memória cache

6

Memória cache

Princípio 90/10

É um princípio, não uma regra precisa...

Em geral, 90% do tempo que um programa usa, é usado apenas por 10% do código

Isso significa que as linhas de montagem são mais usadas do que outras

Em geral, 10% das variáveis que um programa usa, representam 90% do tempo total que um programa usa para todas as variáveis

Isso significa que são variáveis mais utilizadas do que outras

7

Princípio 90/10

- **Alguns códigos são executados com muito mais frequência do que outros código.** Por exemplo, algum código de tratamento de erros pode nunca ser usado. Algum código será executado apenas quando você iniciar seu programa. Outro código será executado repetidamente enquanto seu programa é executado.

- **Alguns códigos demoram *muito* mais para serem executados do que outros.** Por exemplo, uma única linha que executa uma consulta em um banco de dados ou extrai um arquivo da Internet provavelmente levará mais tempo do que milhões de operações matemáticas.

8

Princípio 90/10

A questão é que, **se você precisa que seu programa seja executado mais rápido, provavelmente apenas um pequeno número de linhas é significativo para que isso aconteça.**

9

Memória cache

Princípio de continuamente

Se um programa acessar um byte específico em um endereço em um determinado momento, há uma grande probabilidade de que em tempo próximo o programa acesse os bytes do endereço próximo.

10

Memória cache

Princípio da temporalidade

Se um programa acessar um byte específico em um determinado momento, há grande probabilidade de que em um tempo próximo o programa acesse novamente esse byte.

11

Por que usar memória cache?

Para acelerar o computador

Isso é possível se o cache de memória puder reduzir o tempo da CPU para obter dados da memória

Imagine um trabalhador montando rodas em uma linha de montagem de automóveis
Se ele não tiver as rodas e parafusos por perto, ele deve gastar muito tempo para retirá-los do depósito.

Neste caso, ele gasta mais tempo para fazer o trabalho dela

12

Operação de Cache

- A CPU solicita o conteúdo da localização da memória
- Verifique o cache desses dados
- Se presente, obtenha do cache –> (rápido) •

Caso contrário, leia o bloco necessário da memória principal para o cache -> (lento)

- Em seguida, entregue do cache para a CPU

13

O que é um bloco?

A memória cache é dividida em blocos com o mesmo tamanho

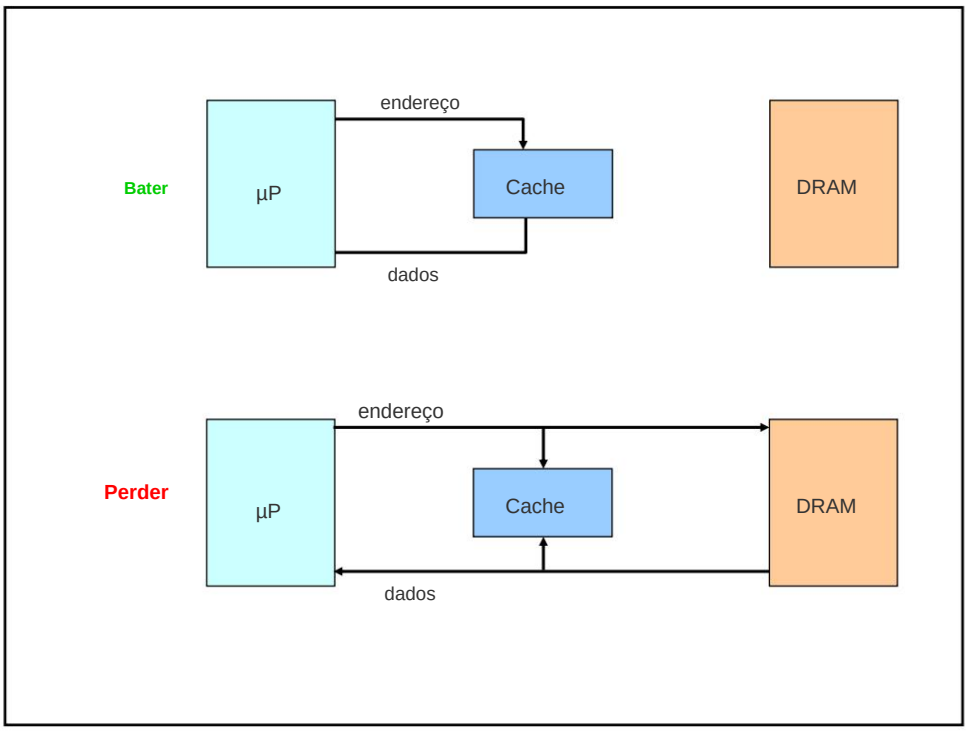
Sempre que a CPU precisar de um byte específico e esse byte não estiver no cache, o sistema obtém da memória um bloco inteiro de dados, em vez de um único byte

Neste caso, temos uma **"perda de caixa"** e um aumento de tempo necessário para obter o bloco inteiro da memória

De acordo com os princípios de continuamente e temporalidade é altamente provável que nas próximas vezes a CPU acesse o mesmo byte ou os bytes próximos

Neste caso, temos um **"golpe de caixa"** e o tempo para a CPU obter os dados é muito baixo

14



15

O que é um bloco?

Imagine uma caixa grande dividida em pequenas caixas para guardar parafusos

A caixa grande é o cache e as caixas pequenas são os blocos

Sempre que o trabalhador precisa de um parafuso específico que não está na caixa grande, ele vai até o depósito e pega uma caixinha inteira com os parafusos necessários

Na próxima vez que ele precisar do mesmo parafuso, esses parafusos estão na caixa

16

Qual é o melhor tamanho para os blocos?

Se o tamanho do bloco aumentar

- O número de blocos no cache diminui

  (número de blocos = tamanho do cache/tamanho do bloco)

- Diminuir o número de blocos diminuir a variedade de dados (se tivermos apenas um bloco no cache, teremos apenas um tipo de “parafusos”)
- Tendo baixa variedade de dados, cresce o número de faltas de cache
- Precisa de mais tempo para transferir todo o bloco da memória para o cache

17

Qual é o melhor tamanho para os blocos?

Se o tamanho do bloco diminuir

- O número de blocos no cache aumenta e aumenta a variedade de dados
- O número de bytes no bloco diminui (se o tamanho for apenas um byte sempre que precisarmos de outro byte, devemos obtê-lo da memória tendo sempre um cache miss)

Isso não é eficiente

18

Qual é o melhor tamanho para os blocos?

Temos de encontrar compromissos…

O tamanho de bloco mais usado é de 64 bytes e 128 bytes

O tamanho do bloco é sempre uma potência de 2

Em geral, quando o tamanho do cache é maior, o tamanho do bloco também é maior

19

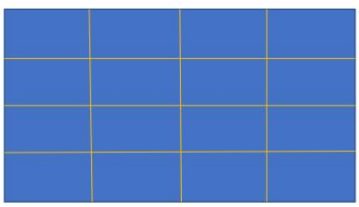
Onde colocar cada bloco?

Suponha novamente que temos uma caixa grande com 16 blocos para colocar parafusos

Você pode organizar a caixa pelo tamanho dos parafusos

Cada bloco tem apenas um tamanho de parafuso de cada vez

Em momentos diferentes, cada bloco pode ter tamanho de parafuso diferente



20

Onde colocar cada bloco?

Por exemplo:

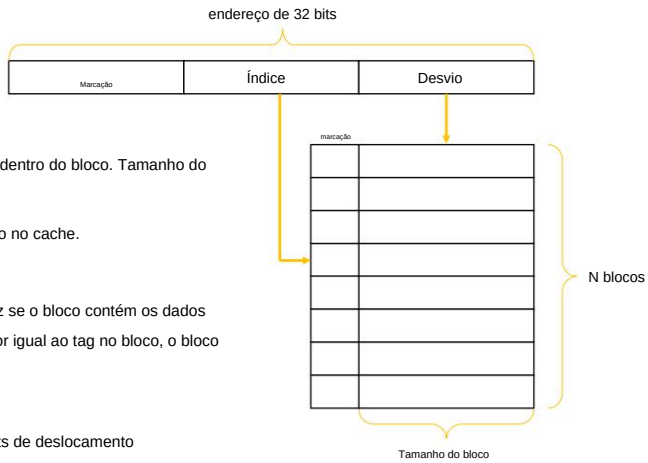
- o bloco 0 pode ter parafusos de tamanho 1, 17, 33, 49, …
- o bloco 1 pode ter parafusos de tamanho 2, 18, 34, 50, …
- o bloco 2 pode ter parafusos de tamanho 3, 19, 35, 51, …
- …
- o bloco 15 pode ter parafusos de tamanho 16, 32, 48, 64, …

Se precisarmos do parafuso tamanho 50, sabemos que esse parafuso só pode estar no bloco 1. Só precisamos confirmar se o bloco 1 contém o parafuso tamanho 50.

Se este bloco não tiver o parafuso tamanho 50, devemos esvaziar esse bloco e colocar lá os parafusos tamanho 50

21

Mapeamento direto



O offset seleciona o byte desejado dentro do bloco. Tamanho do bloco = 2 bits de deslocamento

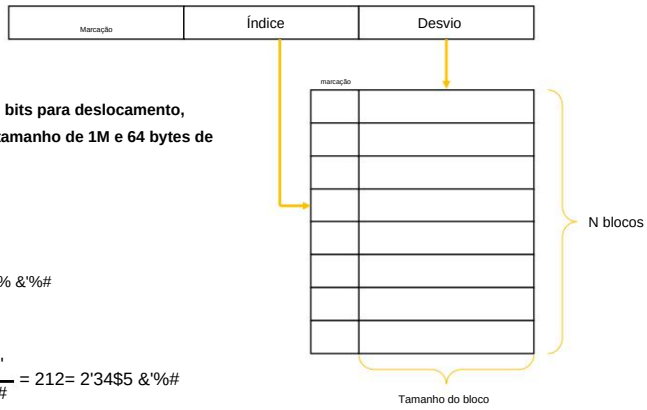
O índice seleciona o bloco desejado no cache.
N blocos = 2 bits de índice

A tag é um identificador que nos diz se o bloco contém os dados desejados. Se o tag no endereço for igual ao tag no bloco, o bloco como o dado desejado.

Bits de tag = 32 - bits de índice - bits de deslocamento

22

Mapeamento direto



Exercício: Encontre o número de bits para deslocamento, índice e tag para um cache com tamanho de 1M e 64 bytes de tamanho de bloco

Tamanho do cache=1M=220

Tamanho do bloco=64=26 = 2!""#$% &'%#

Bits de deslocamento = 6

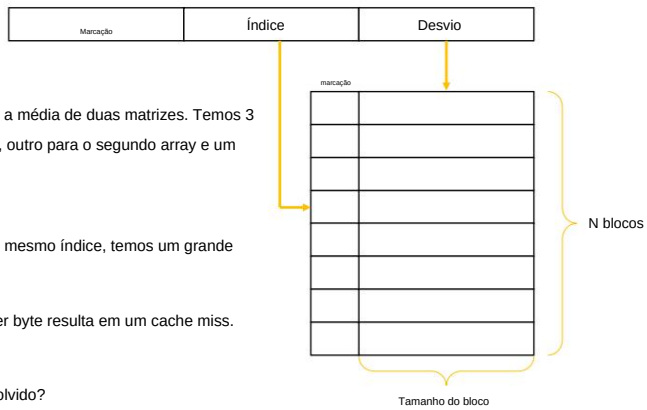
N blocos=()*+$ #',$ / -.!*/ #',$ = 0!" / 0# = 212= 2'34$5 &'%#

Bits de índice = 14

Bits de tag = 32 bits de índice de bits de deslocamento = 32-14-6 = 12

23

Problemas de mapeamento direto



Considere o exercício para calcular a média de duas matrizes. Temos 3 ponteiros, um para o primeiro array, outro para o segundo array e um terceiro para o resultado.

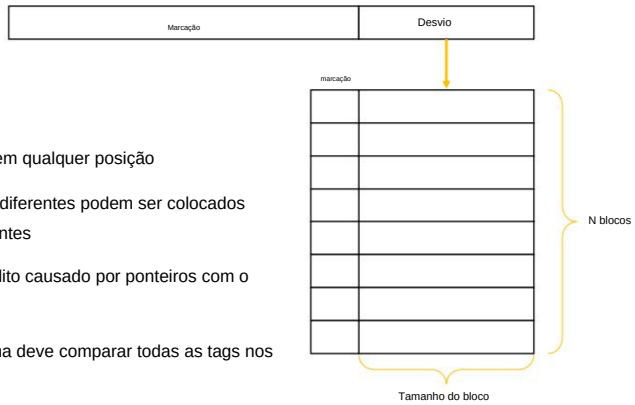
Se todos esses ponteiros tiverem o mesmo índice, temos um grande problema!

Nesse caso, todo acesso a qualquer byte resulta em um cache miss.

Como esse problema pode ser resolvido?

24

Mapeamento associativo (sem índice)



Neste método não há índice

Cada bloco pode ser colocado em qualquer posição

Agora quaisquer dois ponteiros diferentes podem ser colocados em quaisquer dois blocos diferentes

Isso resolve o problema de conflito causado por ponteiros com o mesmo índice

Para encontrar o bloco, o sistema deve comparar todas as tags nos blocos com a tag de endereço

Isso pode levar muito tempo e não é eficiente

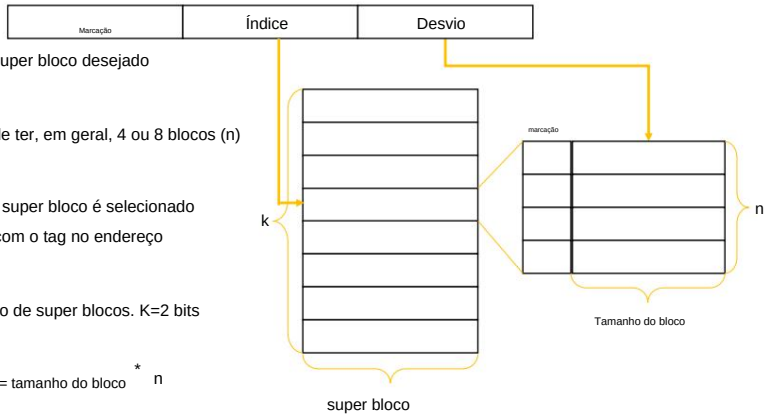
25

Cache de memória

Como podemos encontrar um novo método de mapeamento para o cache que tenha as vantagens do mapeamento direto e do mapeamento associativo, mas não tenha suas desvantagens?

26

Mapeamento associativo por blocos



O índice seleciona o super bloco desejado

Cada super bloco pode ter, em geral, 4 ou 8 blocos (n)

Cada bloco dentro do super bloco é selecionado comparando seu tag com o tag no endereço

K representa o número de super blocos. K=2 bits de índice

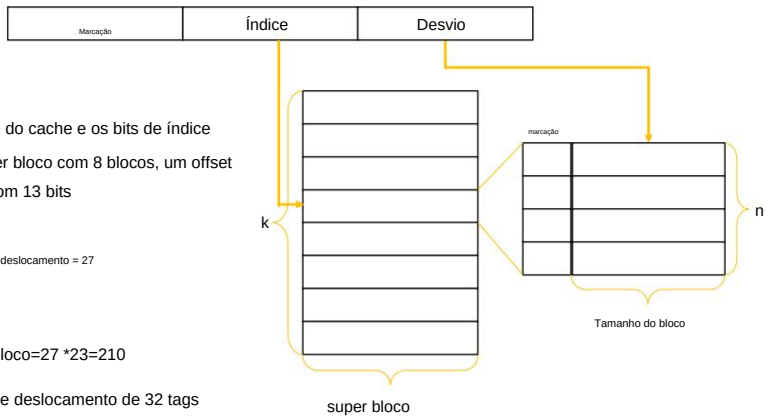
Super tamanho do bloco = tamanho do bloco * n

Tamanho do cache = tamanho do super bloco * k

Tamanho do bloco = 2 bits de deslocamento

27

Mapeamento associativo por blocos



Exercício

Encontre o tamanho do cache e os bits de índice se tivermos um super bloco com 8 blocos, um offset de 7 bits e um tag com 13 bits

Tamanho do bloco = 2 bits de deslocamento = 27

n=8=23

Super tamanho do bloco=27 *23=210

Bits de índice=bits de deslocamento de 32 tags

Bits de índice=32-13-7=12

Tamanho do cache =212*210=222=4MB

28

Como o processo grava no cache?

Em geral os programas fazem mais leituras na memória que escreve

O cache pode ter políticas diferentes para leitura e gravação

Para escrever é possível implementar as seguintes políticas:

- Escreva diretamente na memória
  - Se o bloco estiver no cache, escreva também no bloco
  - Se o bloco não estiver no cache, carregue e grave-o
  - Se o bloco não estiver no cache, não o carregue

  Nestes modos quando um bloco deve ser alterado não é necessário copiá-lo para o ram

- Escreva apenas no bloco
  - Se o bloco não estiver no cache, carregue e grave-o

  Neste modo quando um bloco deve ser alterado é necessário copiá-lo contendo para o ram se eles foram alterados

29

Qual bloco substituir?

No mapeamento direto

- O bloco apontado pelo índice (única possibilidade)

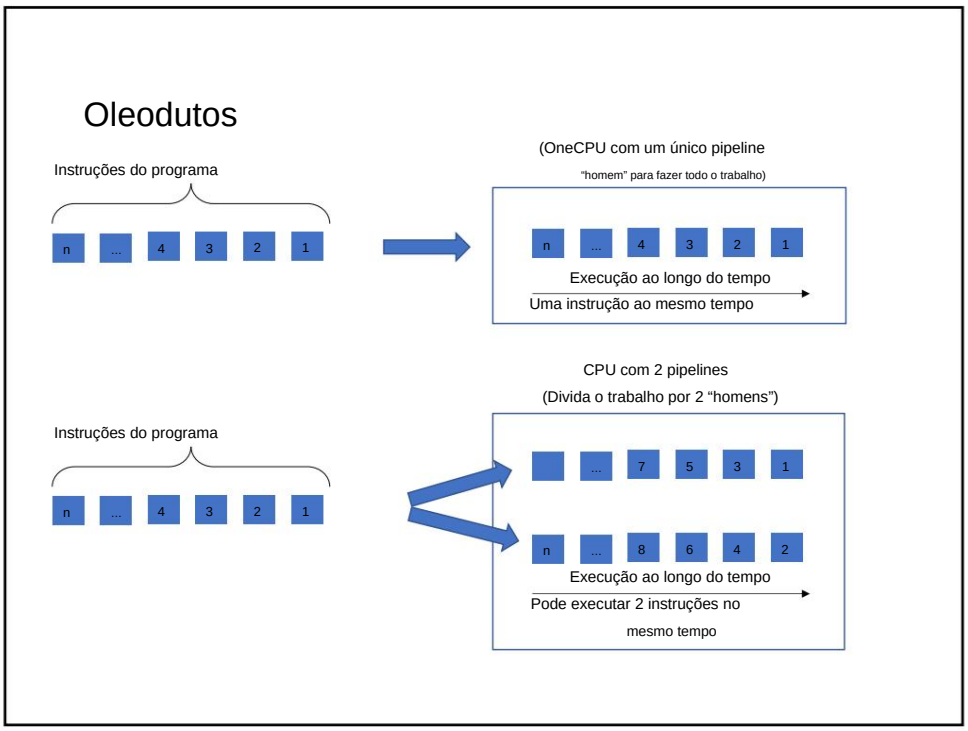
No mapeamento associativo

- O bloco que não é usado há muito tempo
- Bloco aleatório

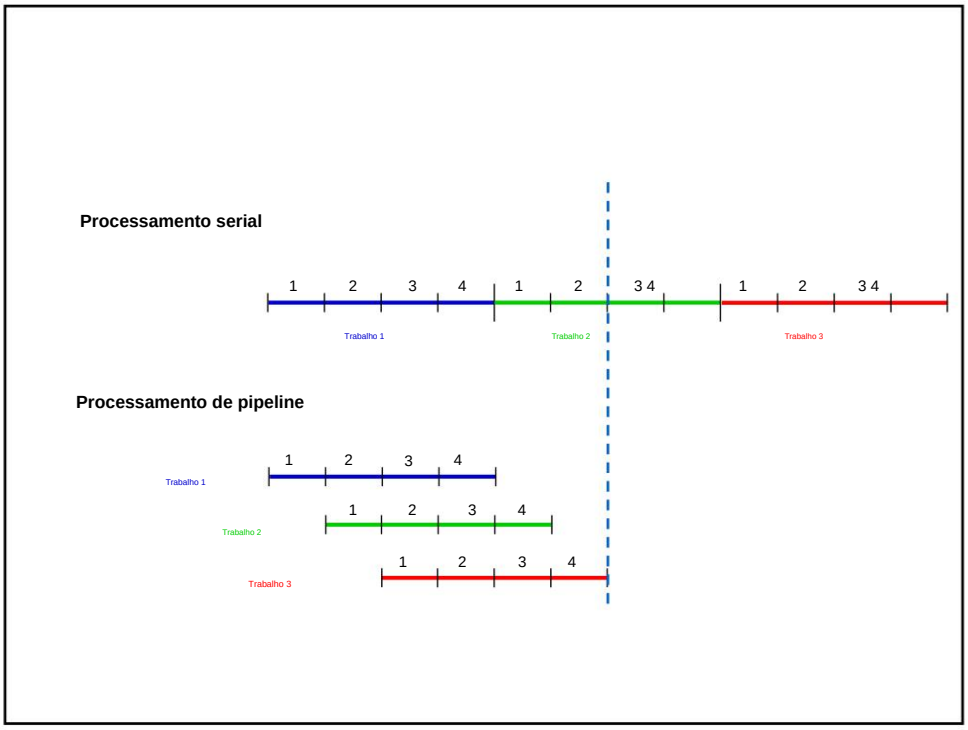
No mapeamento associativo por blocos

- Dentro do super bloco
  - O bloco que não é usado há muito tempo
  - Bloco aleatório

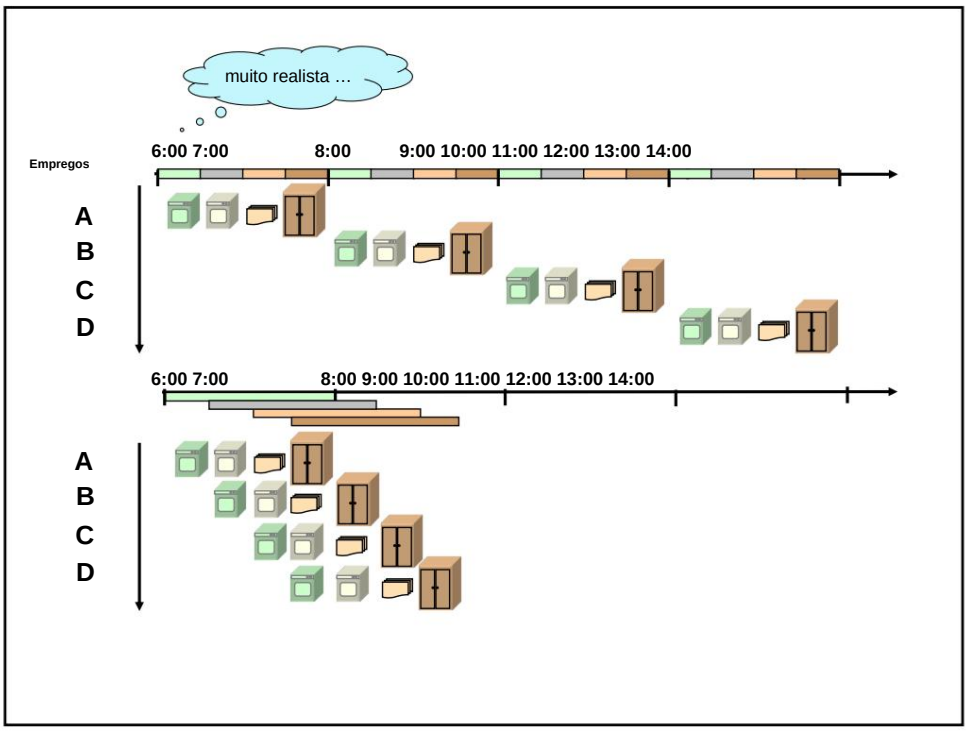
30



31



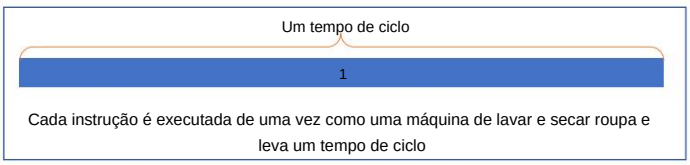
32



33

Estados em pipelines

Um pipeline de estado



Cada instrução é executada de uma vez como uma máquina de lavar e secar roupa e leva um tempo de ciclo

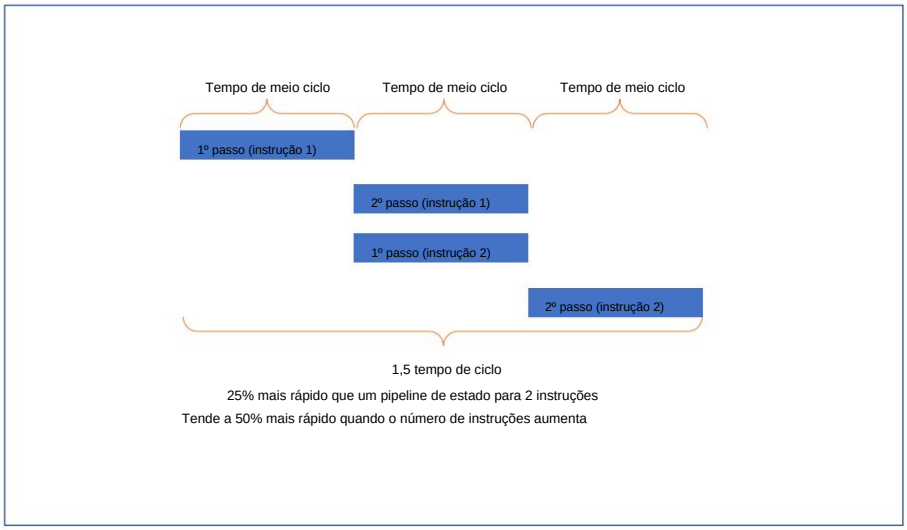
Pipeline de dois estados

Cada instrução é dividida em duas etapas. Cada etapa é executada em um hardware dedicado e independente, como uma máquina de lavar para lavar e outra secadora para secar.
O tempo total para a execução da instrução é o mesmo (ou aproximado).
Nenhum benefício para uma única instrução!

34

Estados em pipelines

Pipeline de dois estados, duas instruções

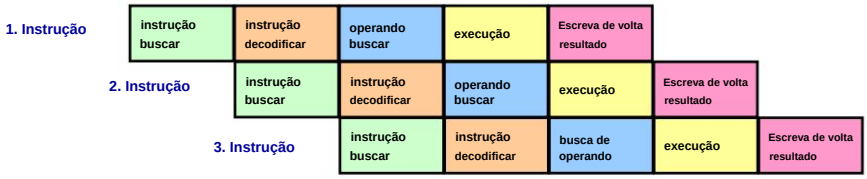


25% mais rápido que um pipeline de estado para 2 instruções
Tende a 50% mais rápido quando o número de instruções aumenta

35

Execução sequencial:

Canalização:



36

Técnicas para melhorar o desempenho

Previsão de ramificação\execução especulativa

```
próximo: mov [esi], al
         incluindo
         inc esi
         dezembro ecx
         jnz próximo
         movimento, 'A'
```

A instrução *jnz next* bloqueia todas as instruções sucessivas porque logo após esta instrução terminar o CPU conhece a próxima instrução

Se a CPU puder prever como uma ramificação é feita, ela poderá inicializar a execução da próxima instrução

Se a previsão for confirmada, a CPU ganha tempo

Caso contrário, a CPU suspende a execução da instrução e executa a instrução correta

37

previsões

• Profissional

- Capaz de eliminar uma ramificação e, portanto, a previsão de ramificação associada Æ aumentando a distância entre as previsões incorretas.
- O comprimento de execução de um bloco de código é aumentado Æ melhor agendamento do compilador.

• Contra

- A predicação afeta o conjunto de instruções, adiciona uma porta ao arquivo registrador e complica a execução da instrução.
  - Instruções predicadas que

são descartadas ainda consomem recursos do processador; especialmente o buscar largura de banda.

• A predicação é mais eficaz quando as dependências de controle podem ser eliminadas, como em um if-then com um pequeno corpo then.

• O uso de instruções predicadas é limitado quando o fluxo de controle envolve mais do que uma simples sequência alternativa.

38

Técnicas para melhorar o desempenho

Execução fora de ordem

```
mov al,bl
add al,cl
mov ah,'A'
```

A CPU pode alterar a ordem de execução

```
para mov
al,bl mov
ah,'A' add al,cl
```

Mudando a ordem, a CPU pode executar essas 3 instruções em menos tempo, pois a instrução mov ah,'A' pode ser executada em paralelo com a instrução mov al,bl

39

Técnicas para melhorar o desempenho

Registros renomeados

```
adicionar
eax,5 adicionar
ebx,eax mover eax,2
```

Essas 3 instruções possuem dependência entre elas.

O segundo apenas pode ser executado após o primeiro e o terceiro apenas pode ser executado após o segundo.

A CPU pode resolver esta dependência internamente renomeando o registrador eax.

40

Técnicas para melhorar o desempenho

Registros renomeados

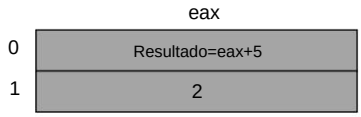
Cada registrador possui internamente uma caixa dupla

Por vez, apenas uma caixa (0 ou 1) está ativa

adicionar	se a caixa ativa for 0 coloque o resultado na
eax,5; adicionar	caixa '0 use o valor na caixa ativa (0)
ebx,eax; mover eax,2;	mude a caixa ativa para 1 e coloque nela o valor 2

Esta instrução pode ser executada em paralelo com a segunda

Este processo é completamente independente para o programador



41

Técnicas para melhorar o desempenho

1. **Previsão de ramificação\execução especulativa**
2. **Execução fora de ordem**
3. **Registros renomeados**

42