Hierarquia de memória

Memórias Cache

Cap. 5: “Large and Fast: Exploiting Memory Hierarchy”

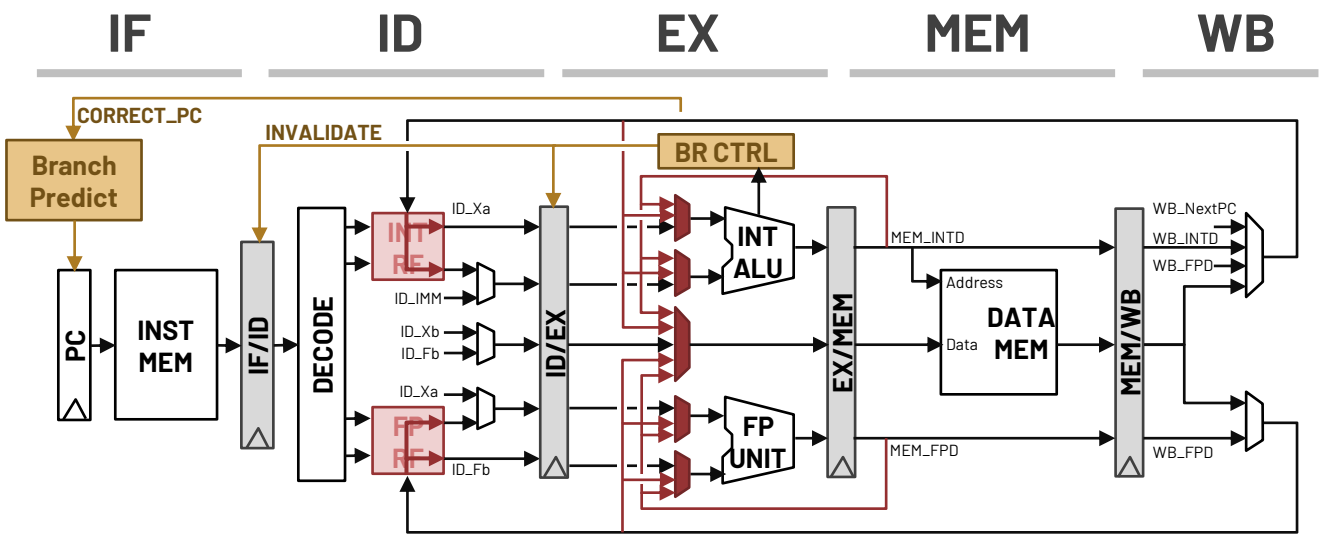
7 June, 2022

Desenho do processador

Revisão do capítulo anterior

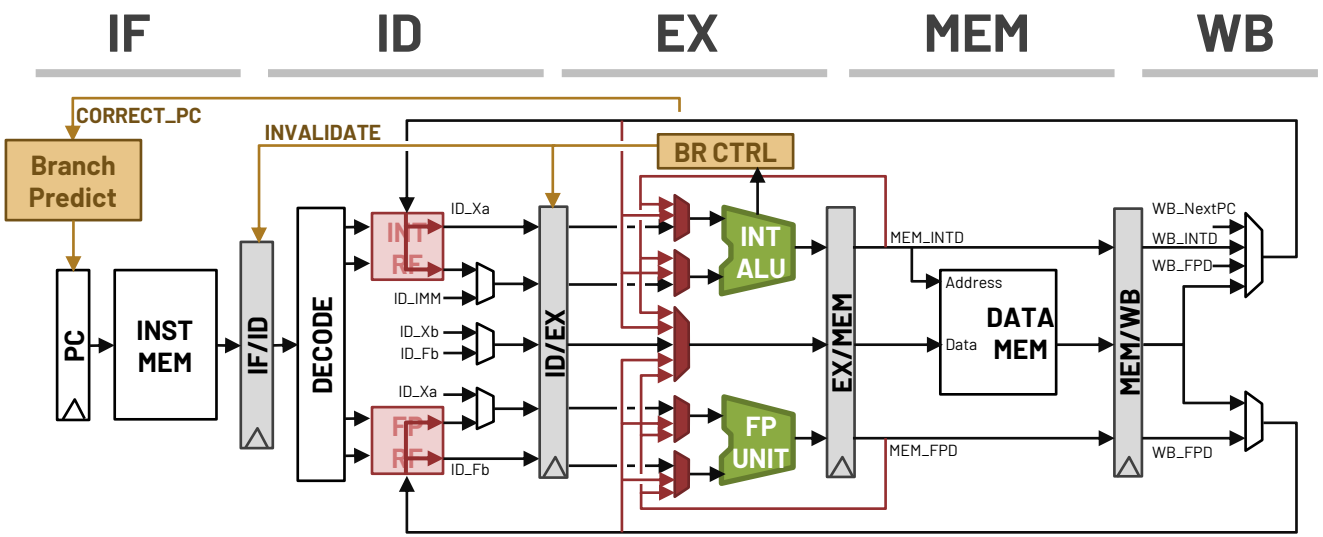
Arquitetura do processador (core)

Funcionamento em pipeline com caminhos de *forwarding* e predição de salto



Arquitetura do processador (core)

Funcionamento em pipeline com caminhos de *forwarding* e predição de salto

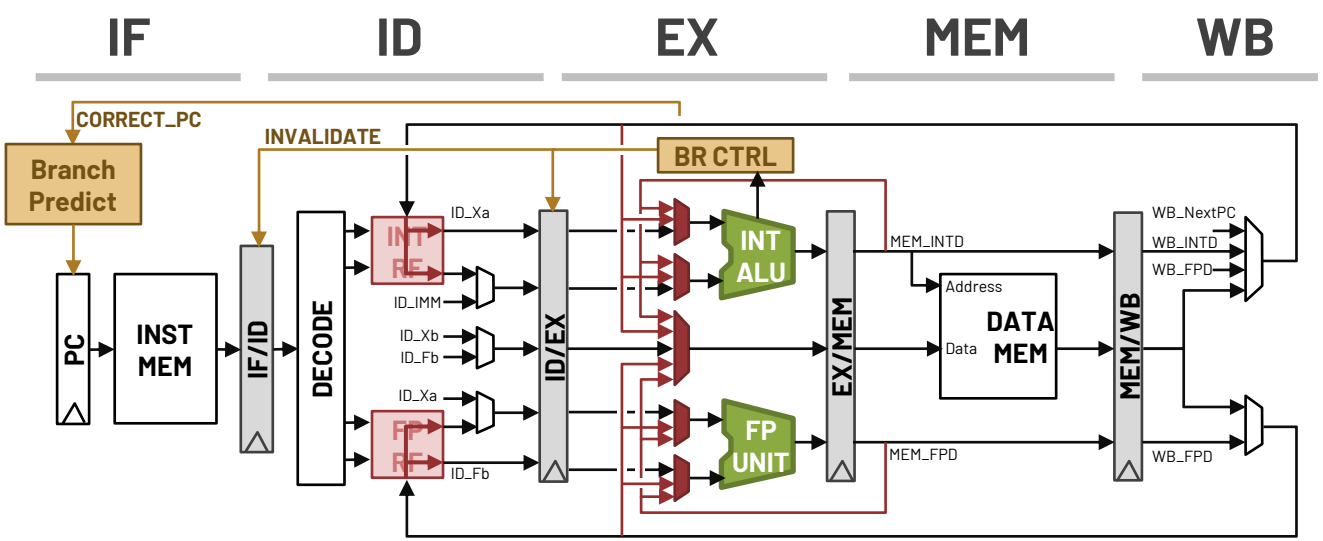


As unidades funcionais podem ser implementadas com pipeline, e cada uma, com um número de estágios diferente

Unidade funcional	#ciclos
Unidade lógica	1
Unidade de deslocamento	1
Somador/subtrator (inteiros)	1
Multiplicador	3
Divisor (inteiro/FP)	20
Somador/subtrator (FP)	3
Multiplicador (FP)	4

Arquitetura do processador (core)

Funcionamento em pipeline com caminhos de *forwarding* e predição de salto



As unidades funcionais podem ser implementadas com pipeline, e cada uma, com um número de estágios diferente

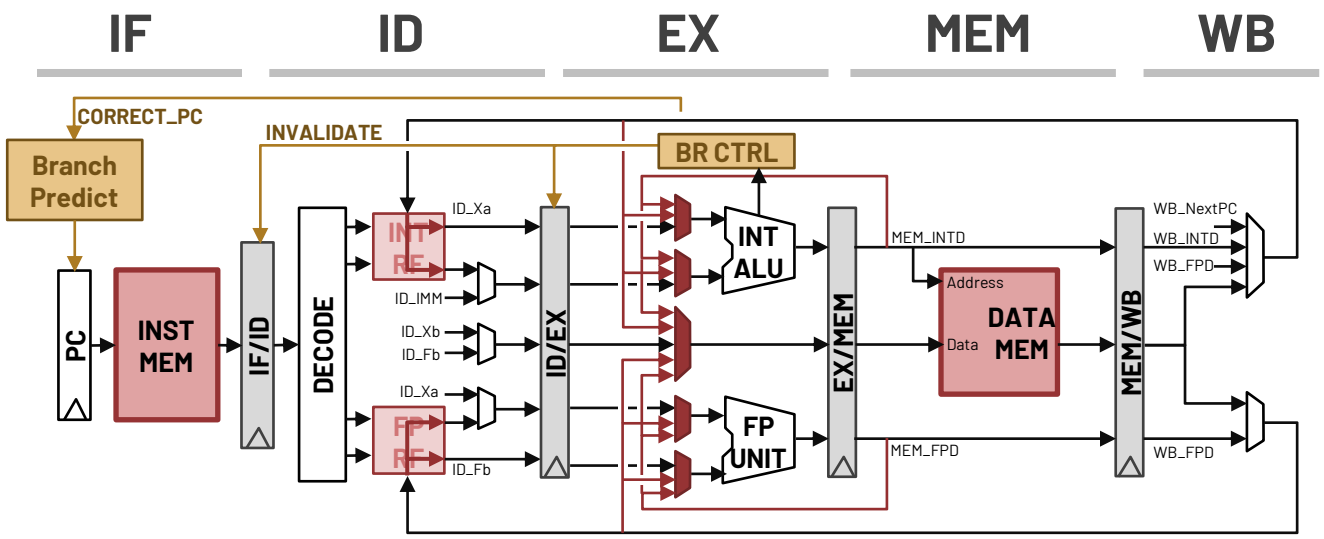
Unidade funcional	#ciclos
Unidade lógica	1
Unidade de deslocamento	1
Somador/subtrator (inteiros)	1
Multiplicador	3
Divisor (inteiro/FP)	20
Somador/subtrator (FP)	3
Multiplicador (FP)	4

Nota: nestes casos as instruções que não acedem à memória tipicamente fazem bypass ao estágio de MEM

Arquitetura do processador (core)

Funcionamento em pipeline com caminhos de *forwarding* e predição de salto

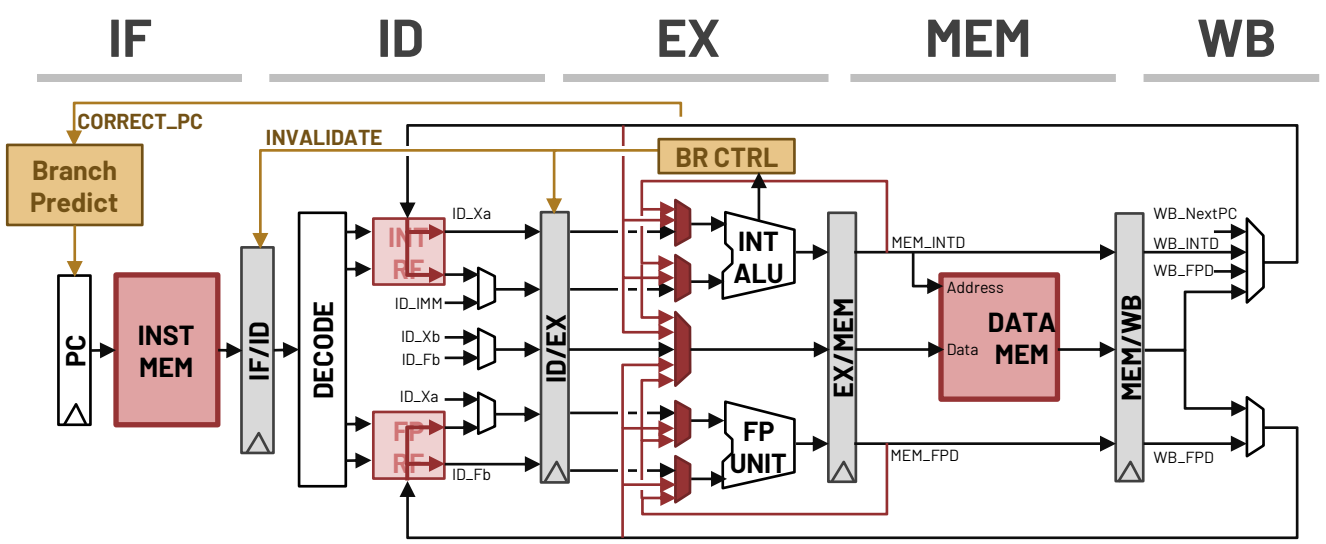
- Duas memórias?
- Leitura assíncrona com baixa latência?



Arquitetura do processador (core)

Funcionamento em pipeline com caminhos de *forwarding* e predição de salto

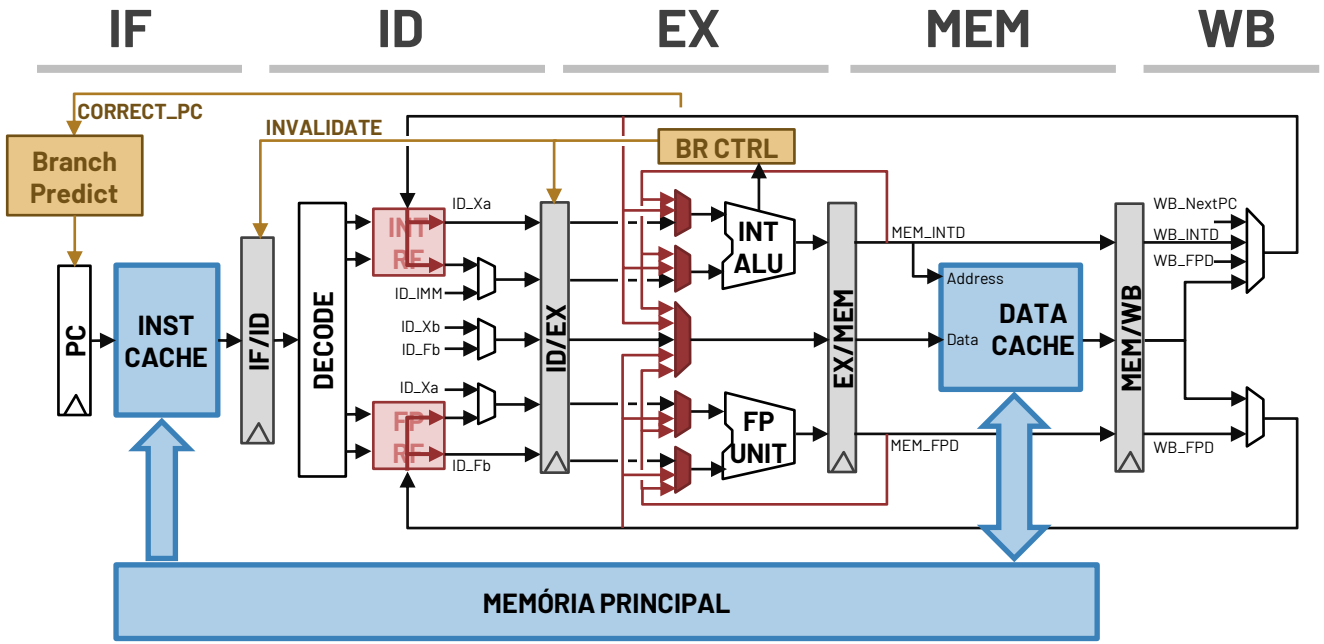
- Duas memórias? Geralmente, existe apenas uma memória principal (memória RAM)!
- Leitura assíncrona com baixa latência? Latência demasiado grande (~100 ciclos)!



Arquitetura do processador (core)

Funcionamento em pipeline com caminhos de *forwarding* e predição de salto

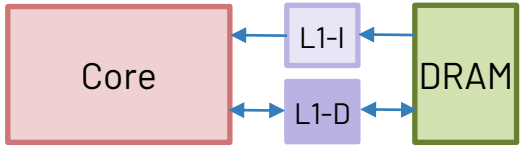
- Duas caches (pequenas ➔ baixa latência) que ligam à mesma memória principal
- Cada cache contém apenas um subconjunto das instruções ou dados

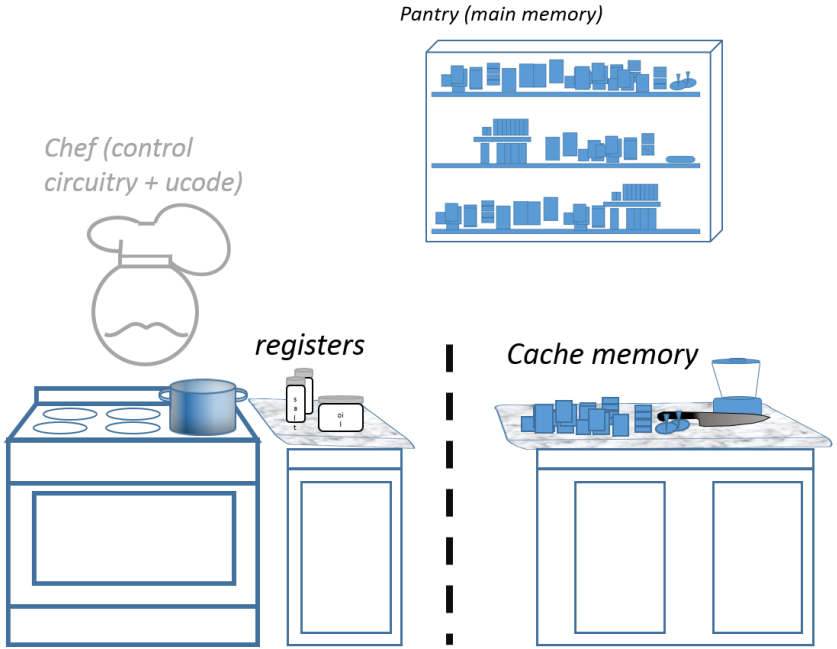


Arquitetura do processador

Hierarquia de memória

- Duas caches (pequenas ➔ baixa latência) que ligam à mesma memória principal
- Cada cache contém apenas um subconjunto das instruções ou dados





Arquitetura do processador

Hierarquia de memória

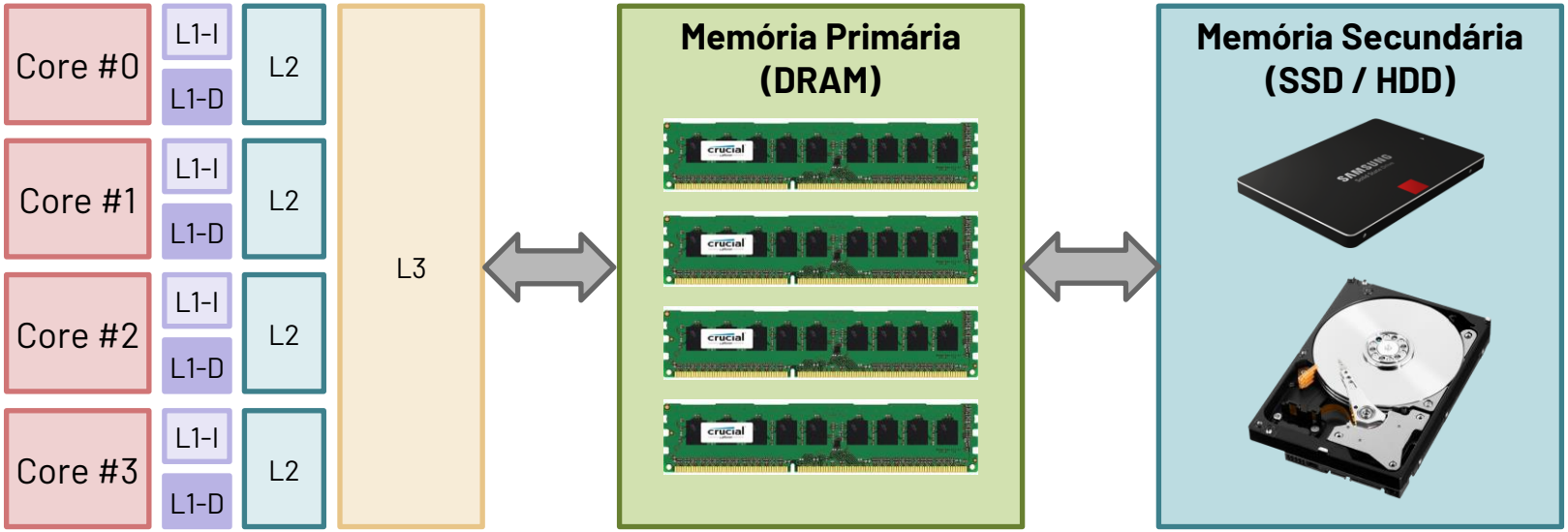
- Duas caches (pequenas ➔ baixa latência) que ligam à mesma memória principal
  - Embora as caches sejam de dimensão reduzida (valores típicos 32kB a 64kB), a latência é ainda assim demasiado grande para suportar frequências de funcionamento muito altas. Assim, o acesso à cache é tipicamente partido em 3-4 ciclos (i.e., para o caso de uma latência de 3 ciclos, o pipeline tem mais estágios correspondentes a IF1,IF2,IF3 e MEM1,MEM2,MEM3).

- Cada cache contém apenas um subconjunto das instruções ou dados
  - Se os dados que queremos consultar estiverem em cache, não é preciso pagar o "custo" de consultar o nível superior, ou a memória RAM

Arquitetura do processador

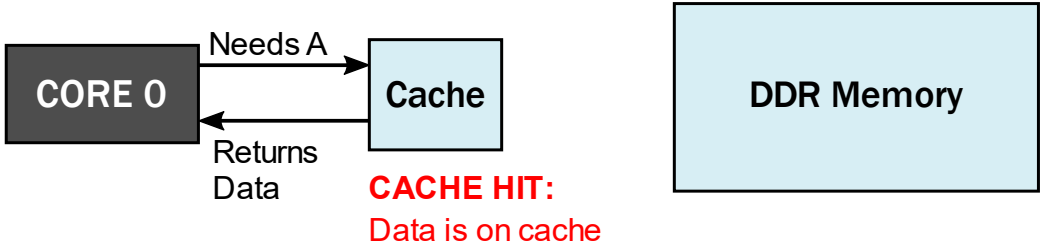
Hierarquia de memória

- Como as cache L1 têm uma dimensão reduzida, geralmente os sistemas contêm uma hierarquia com 2 a 3 níveis de cache (ex: L1, L2, L3). A cache L1 é a de <u>menor dimensão</u> e <u>menor latência</u>, portanto mais próxima do processador

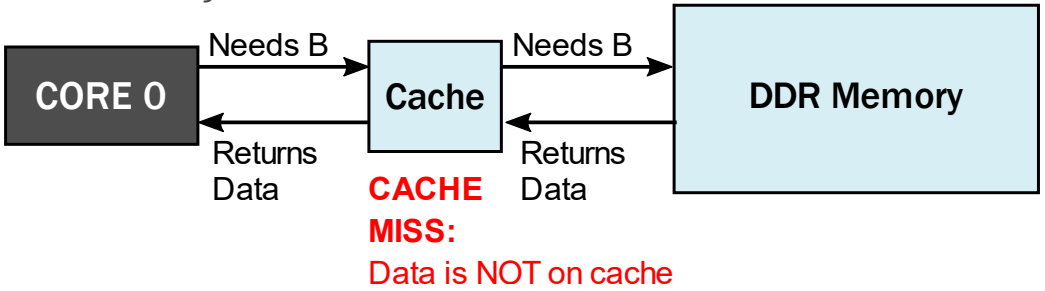


Princípio de funcionamento

- Quando o processador precisa de um dado, faz o pedido à cache
  - Se os dados estiverem na cache, devolve-os imediatamente

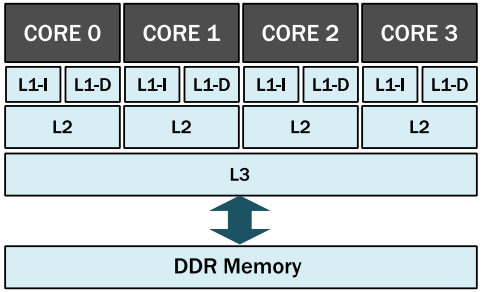


  - Se os dados NÃO estiverem na cache, faz o pedido à memória
    - Quando o nível seguinte devolver os dados,

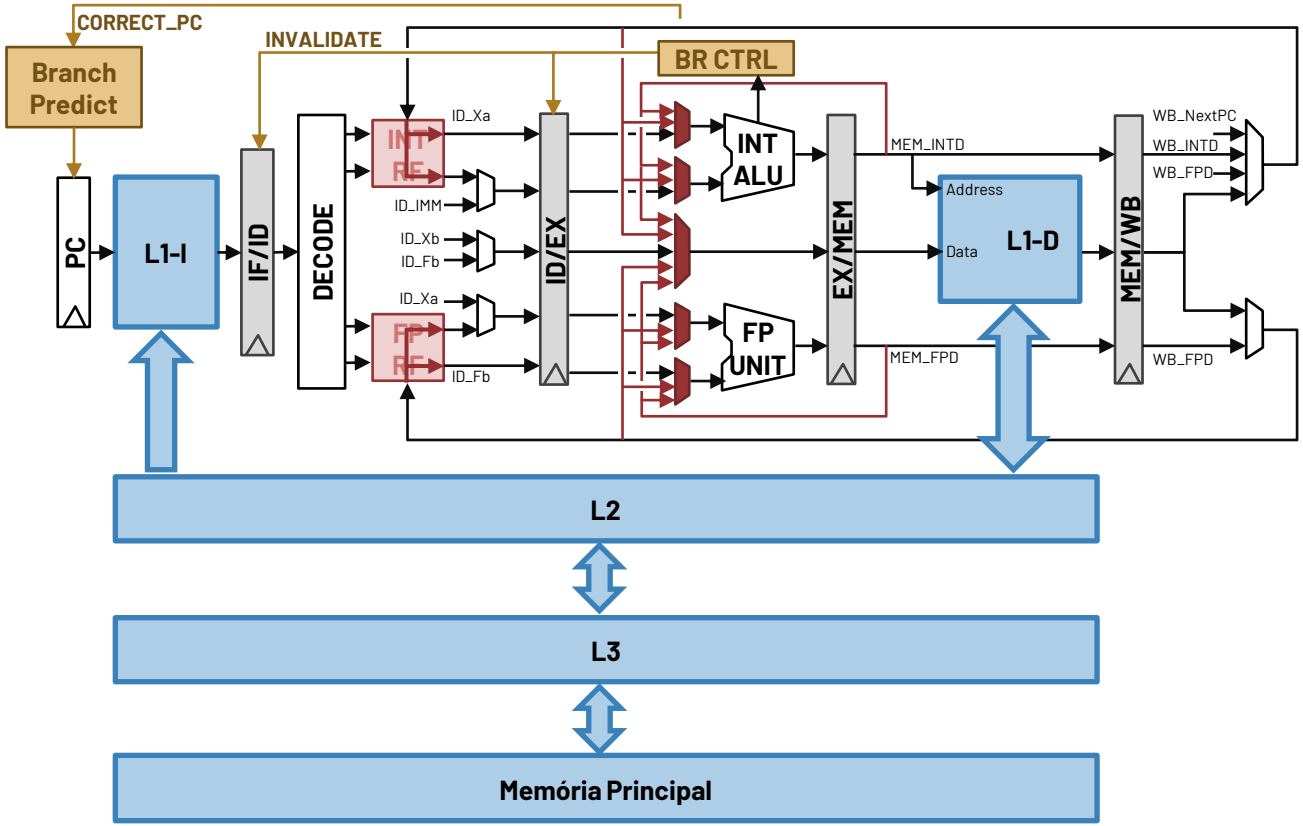


Arquitetura do processador

Hierarquia de memória:



Visão do core:



Quanto maiores e mais distantes estiverem as caches, maior a latência (#ciclos) de acesso.

TÉCNICO LISBOA

Princípio da localidade

Princípio da localidade

TÉCNICO LISBOA

- **Localidade temporal:**
  Se acedermos (leitura ou escrita) a um endereço A, é provável que num futuro próximo necessitemos de aceder novamente ao mesmo endereço
  - E.g., sequência de instruções em ciclos e acessos frequentes a estruturas de dados

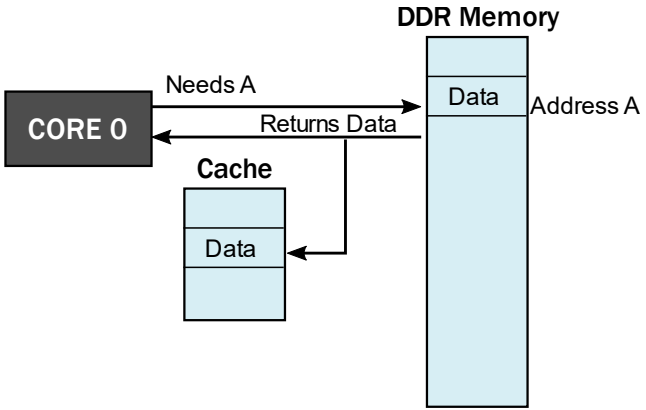
- **Localidade espacial:**
  Se acedermos (leitura ou escrita) a um endereço A, é provável que num futuro próximo necessitemos de aceder aos endereços adjacentes a A, i.e., A-1, A+1, A+2, ...
  - E.g., execução de código (sequência de instruções, na ausência de saltos), ou acessos a elementos num vetor ou outra estrutura de dados

- **Regra 90/10**
  Um programa passa 90% do tempo a executar 10% das instruções.

Exploração da localidade temporal

- **Localidade temporal:**
  Se acedermos (leitura ou escrita) a um endereço A, é provável que num futuro próximo necessitemos de aceder novamente ao mesmo endereço
  - E.g., sequência de instruções em ciclos e acessos frequentes a estruturas de dados

Quando acedemos ao dado/instrução no endereço A, já que temos de pagar o custo de o ir buscar à memória RAM, mais vale armazená-lo em memória cache.

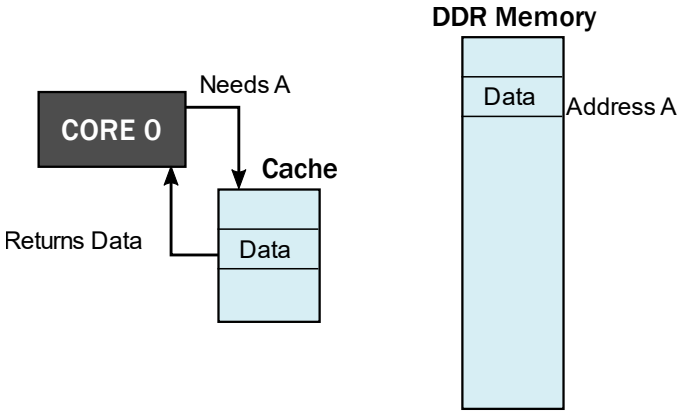


FIRST ACCESS:
Data access time: ~100 cycles

Exploração da localidade temporal

- **Localidade temporal:**
  Se acedermos (leitura ou escrita) a um endereço A, é provável que num futuro próximo necessitemos de aceder novamente ao mesmo endereço
  - E.g., sequência de instruções em ciclos e acessos frequentes a estruturas de dados

Quando acedemos ao dado/instrução no endereço A, já que temos de pagar o custo de o ir buscar à memória RAM, mais vale armazená-lo em memória cache.

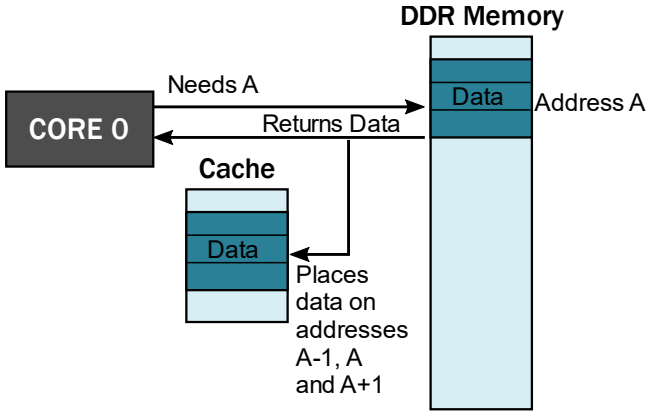


FOLLOWING ACCESS:
Data access time: ~3 cycles

Exploração da localidade espacial

- **Localidade espacial:**
  Se acedermos (leitura ou escrita) a um endereço A, é provável que num futuro próximo necessitemos de aceder aos endereços adjacentes a A, i.e., A-1, A+1, A+2, ...
  - E.g., execução de código (sequência de instruções, na ausência de saltos), ou acessos a elementos num vetor ou outra estrutura de dados

Quando acedemos ao dado/instrução no endereço A, já que temos de pagar o custo de o ir buscar à memória RAM, mais vale trazer também alguns dados/instruções contíguas.

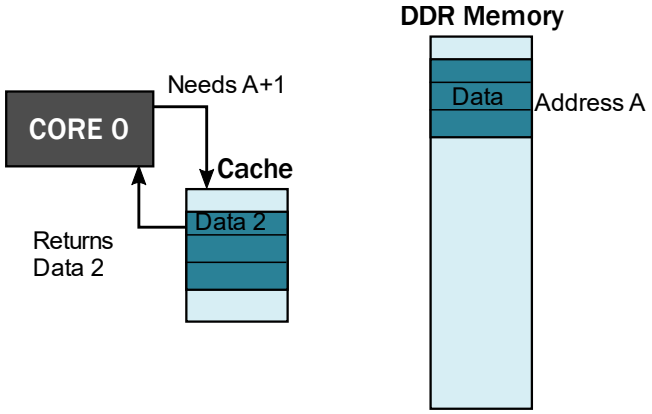


FIRST ACCESS:
Data access time: ~100 cycles

Exploração da localidade espacial

- **Localidade espacial:**
  Se acedermos (leitura ou escrita) a um endereço A, é provável que num futuro próximo necessitemos de aceder aos endereços adjacentes a A, i.e., A-1, A+1, A+2, ...
  - E.g., execução de código (sequência de instruções, na ausência de saltos), ou acessos a elementos num vetor ou outra estrutura de dados

Quando acedemos ao dado/instrução no endereço A, já que temos de pagar o custo de o ir buscar à memória RAM, mais vale trazer também alguns dados/instruções contíguas.



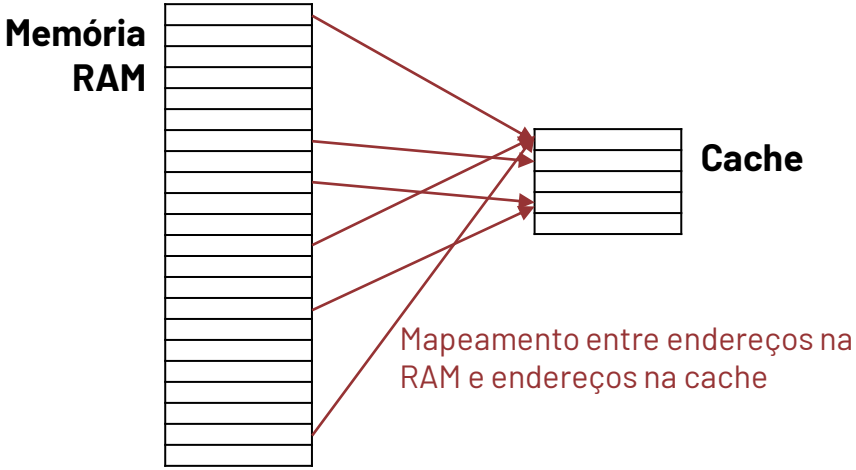
FOLLOWING ACCESS:
Data access time: ~3 cycles

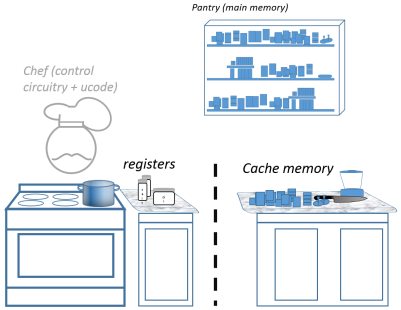
Organização das memórias cache

Princípio de funcionamento da cache

- Na prática, a cache mapeia os endereços da memória RAM (ex: 16GB) num conjunto limitado de entradas (ex: 1024 entradas)

- Naturalmente, este mapeamento dá origem a que múltiplas entradas da memória RAM correspondam à mesma entrada na cache.





Principio de funcionamento da cache

TÉCNICO LISBOA

- Na prática, a cache mapeia os endereços da memória RAM (ex: 16GB) num conjunto limitado de entradas (ex: 1024 entradas)

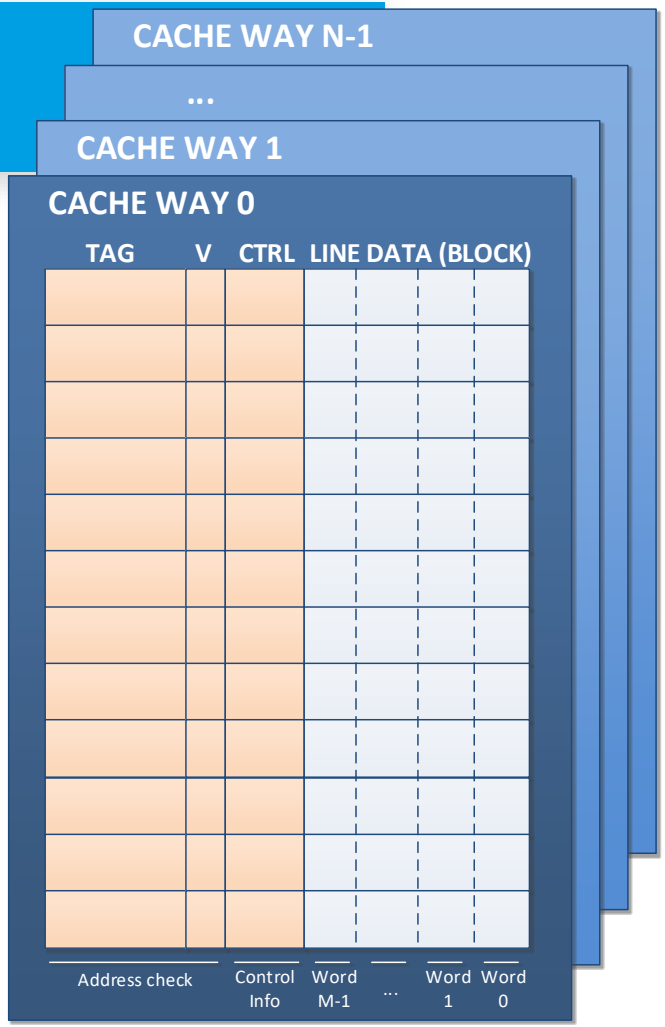
- Naturalmente, este mapeamento dá origem a que múltiplas entradas da memória RAM correspondam à mesma entrada na cache.

- Assim, para além de ser necessário guardar na cache os **dados**, é ainda preciso guardar uma **etiqueta** (TAG) que permite identificar se uma entrada na cache corresponde ao endereço pedido (A), ou a outro endereço (ex: A2, A3, A4, ...)

- Para explorar a localidade espacial, os blocos de dados na cache têm uma dimensão igual ou superior à das palavras do processador (i.e., ≥4B para RV32, ≥8B para RV64)

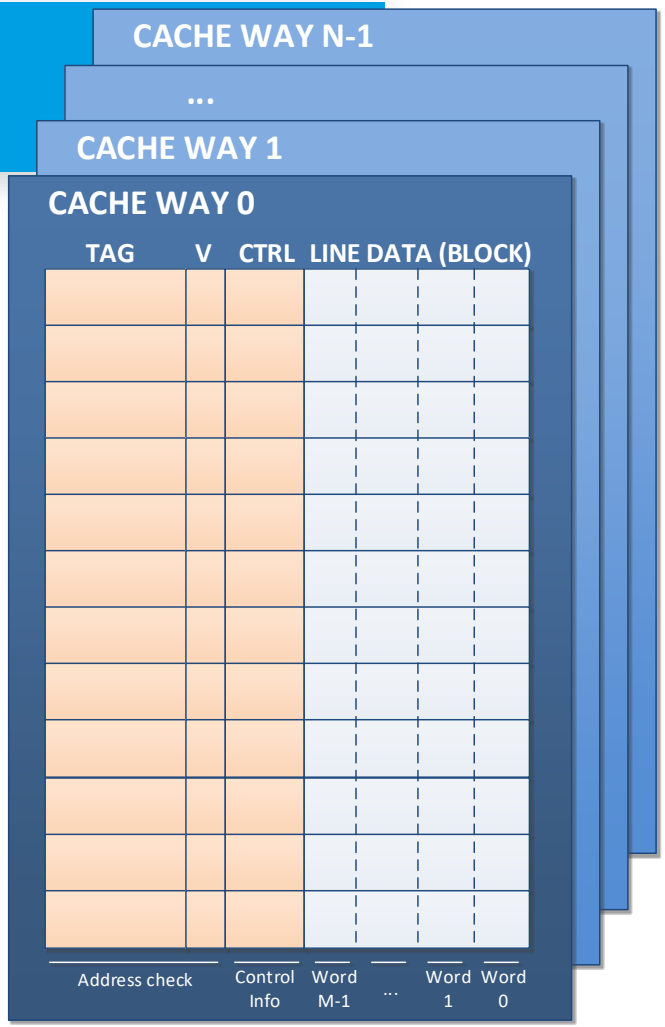
Organização das memórias cache

- Assim, uma cache corresponde a uma (ou mais) tabelas (designadas por vias), cada uma com os seguintes campos:
  - **Bloco de dados** → sequência de palavras de dados, correspondentes a endereços consecutivos em memória
  - **TAG** → etiqueta para confirmação do endereço
  - **V** → campo binário para identificar se a entrada é válida (V=1) ou inválida (V=0)
  - **CTRL** → sinais de controlo auxiliares (estudados posteriormente)



Organização das memórias cache

- Casos particulares:
  - **Cache de Mapeamento Direto:**
    Apenas uma via, mas com múltiplas entradas (linhas)
  - **Cache associativa**
    N Vias, cada uma com múltiplas linhas.
    A organização de cada uma das N vias é idêntica
  - **Cache completamente associativa**
    Múltiplas vias, mas cada uma contendo apenas uma linha

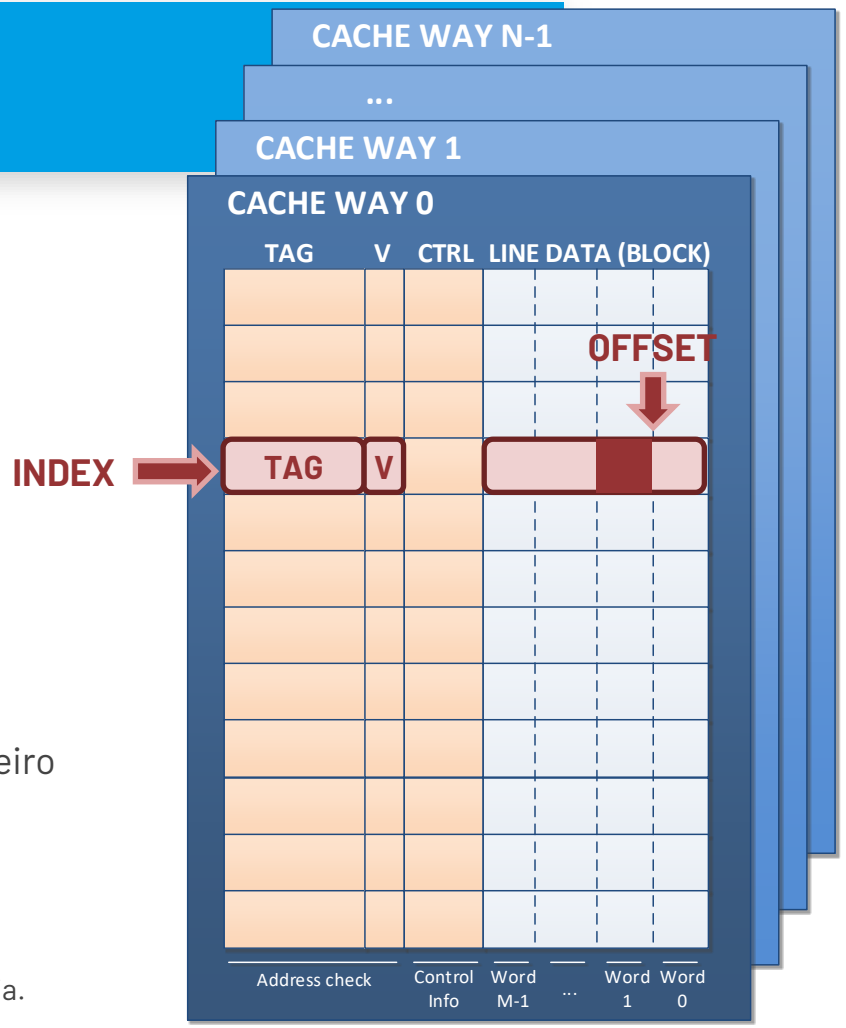


Organização das memórias cache

- Para realizar a tradução entre o endereço de memória (A) e localização na cache, é necessário:

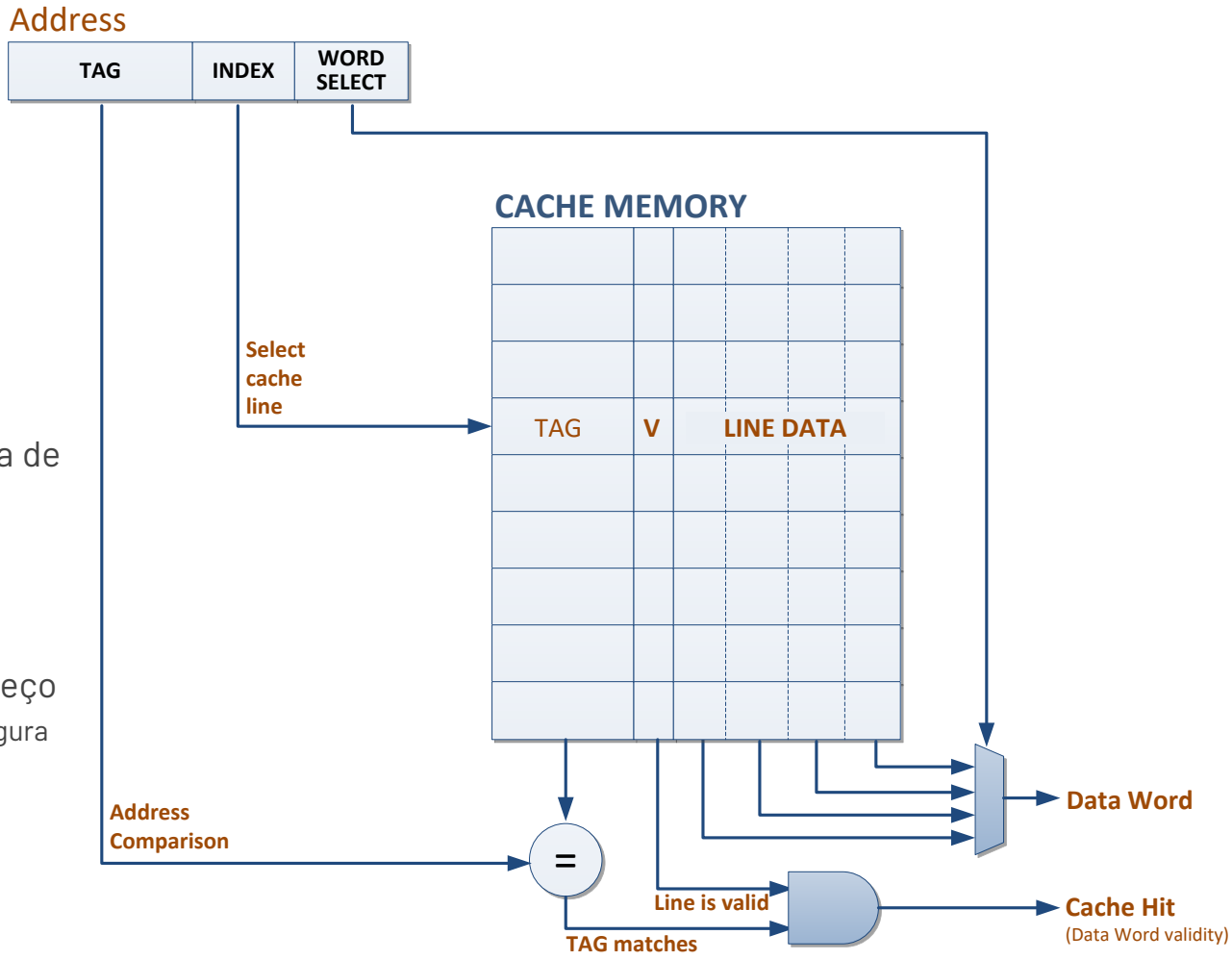
  1. Escolher a entrada na(s) tabela(s), i.e., **INDEX**
     O índice (ou INDEX) funciona como "endereço" na cache
  2. Verificar se a entrada é válida, i.e., se V=1
  3. Verificar se a entrada corresponde ao endereço pedido, i.e., confirmar o valor da **TAG**
  4. (se 2 e 3 forem verdadeiro ➔ **HIT**)
     ler a palavra pedida a partir da cache, sendo o primeiro byte indicado pelo valor de **OFFSET**

Nota: em caches associativas (i.e., quando o número de vias N >1), é necessário verificar se os pontos 2 e 3 são verdadeiros em alguma das N vias, e devolver a palavra (a partir do valor de OFFSET) dessa via. Por definição, o valor do endereço de memória A só pode estar numa única via.



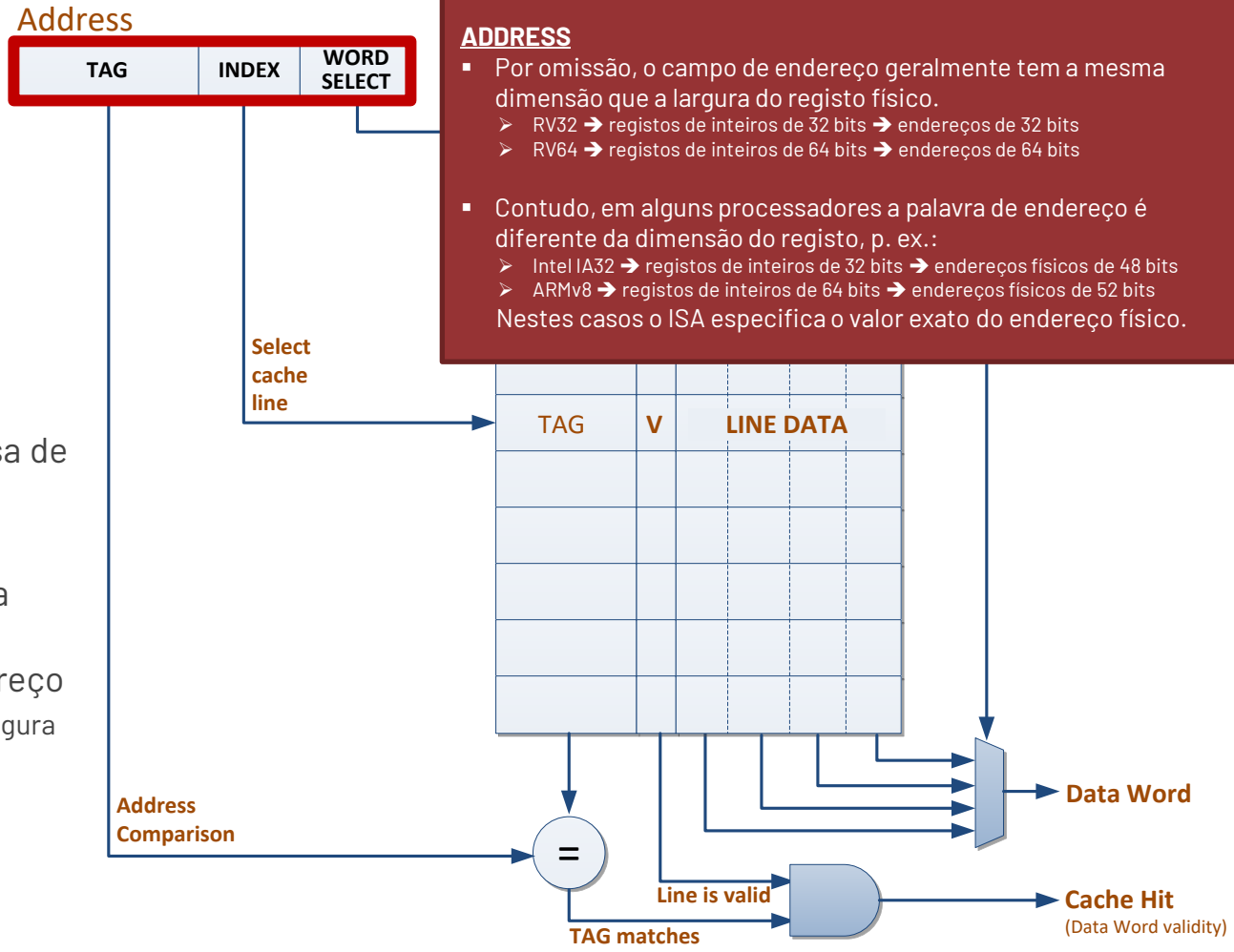
Mapeamento dos dados na cache

- Quando o processador precisa de um dado (ou instrução), faz o pedido à cache L1-D (ou L1-I)
- De acordo com a estrutura da cache L1-D (ou L1-I), esta decompõe a palavra de endereço em TAG, INDEX, OFFSET (na figura WORD SELECT)



Mapeamento dos dados na cache

- Quando o processador precisa de um dado (ou instrução), faz o pedido à cache L1-D (ou L1-I)
- De acordo com a estrutura da cache L1-D (ou L1-I), esta decompõe a palavra de endereço em TAG, INDEX, OFFSET (na figura WORD SELECT)



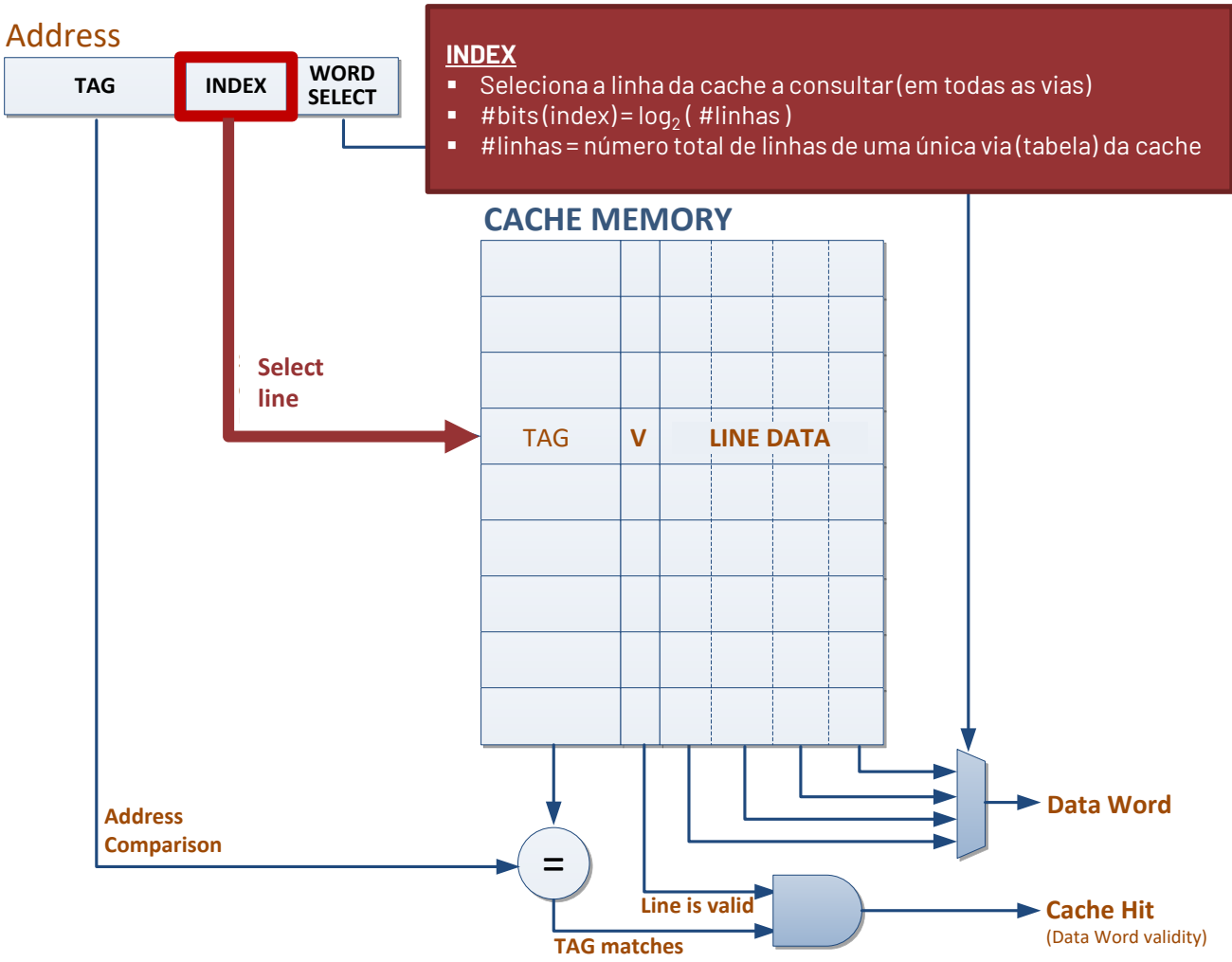
ADDRESS

- Por omissão, o campo de endereço geralmente tem a mesma dimensão que a largura do registo físico.
  - RV32 → registos de inteiros de 32 bits → endereços de 32 bits
  - RV64 → registos de inteiros de 64 bits → endereços de 64 bits
- Contudo, em alguns processadores a palavra de endereço é diferente da dimensão do registo, p. ex.:
  - Intel IA32 → registos de inteiros de 32 bits → endereços físicos de 48 bits
  - ARMv8 → registos de inteiros de 64 bits → endereços físicos de 52 bits

  Nestes casos o ISA especifica o valor exato do endereço físico.

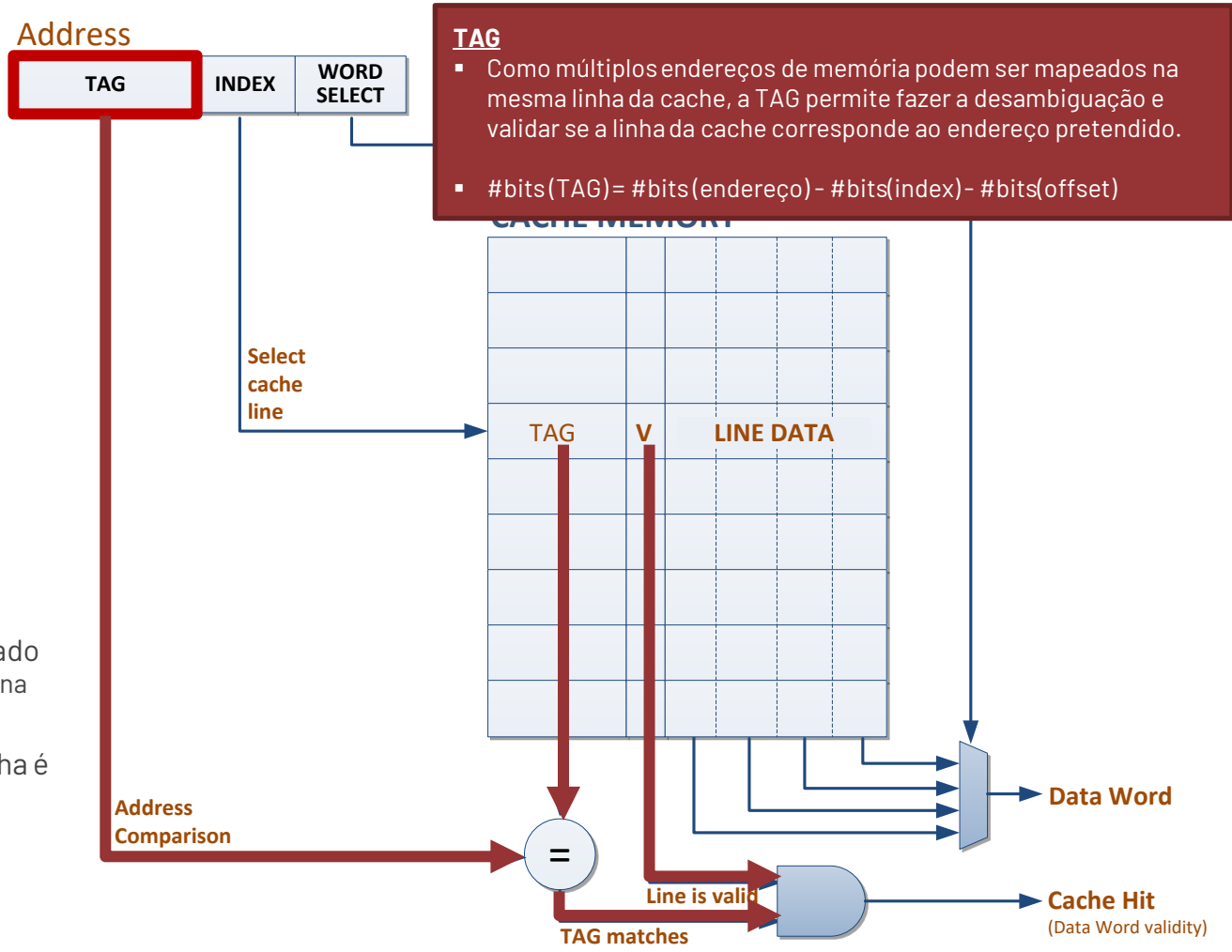
Mapeamento dos dados na cache

- O **INDEX** (índice) é usado para determinar a linha da cache que devemos consultar

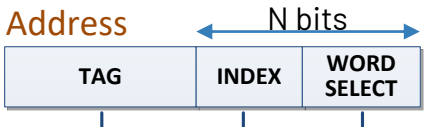


Mapeamento dos dados na cache

- O **INDEX** (índice) é usado para determinar a linha da cache que devemos consultar
- A **TAG** (etiqueta) valida se os dados/instruções na cache correspondem ao endereço indicado
  - Há múltiplos endereços mapeados na mesma linha
- O bit de validade (**V**) indica se a linha é válida

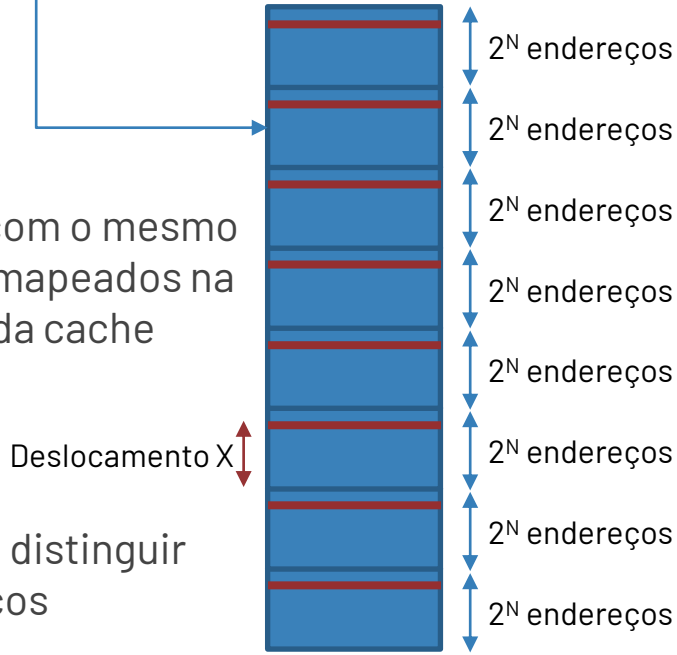


Mapeamento dos dados na cache



- O **INDEX** (índice) é usado para determinar a linha da cache que devemos consultar
- A **TAG** (etiqueta) valida se os dados/instruções na cache correspondem ao endereço indicado
  - Há múltiplos endereços mapeados na mesma linha
- O bit de validade (**V**) indica se a linha é válida

- O conjunto de N bits correspondentes ao conjunto dos campos INDEX+OFFSET definem um deslocamento dentro de um **espaço de memória** de 2^N endereços:

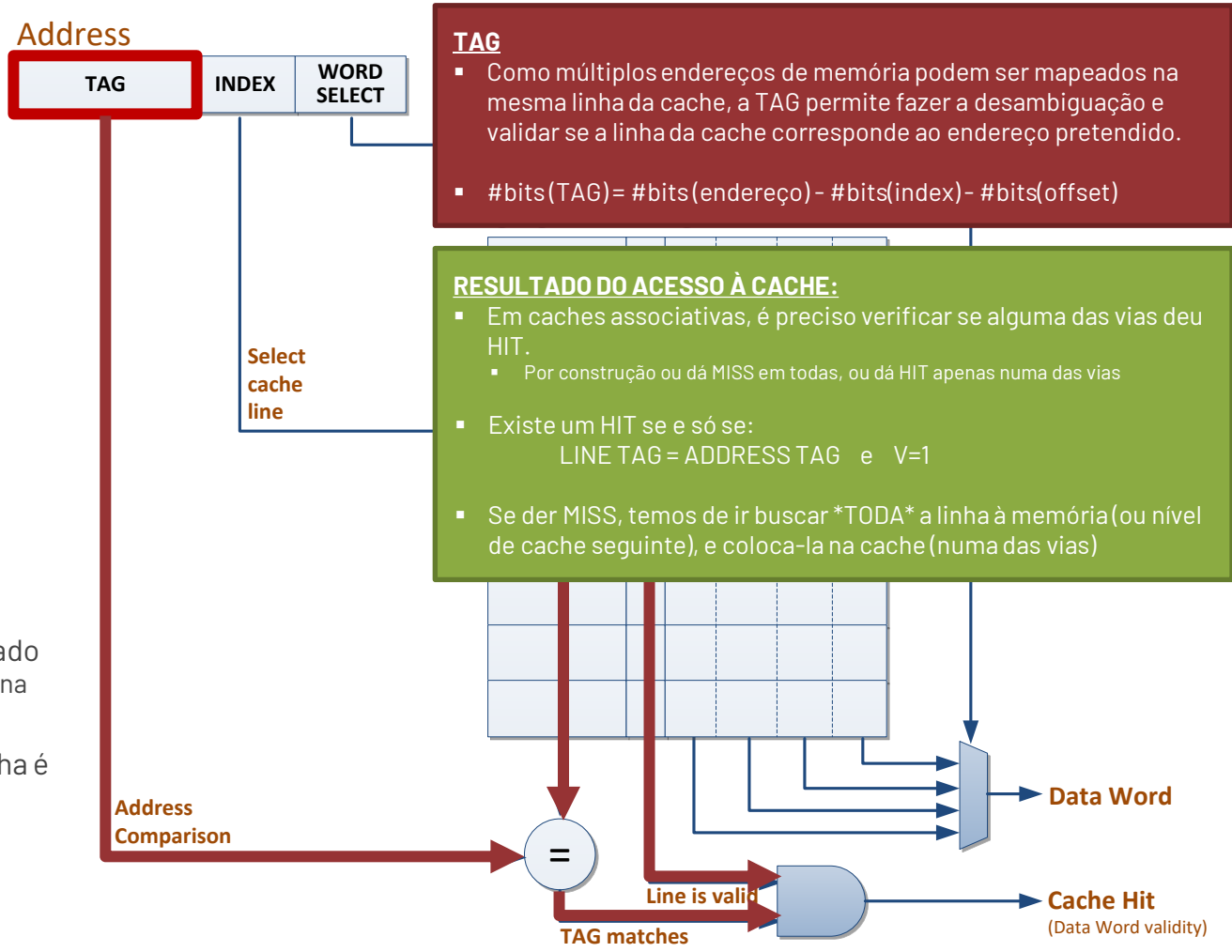


- Todos os endereços com o mesmo deslocamento X, são mapeados na mesma linha da linha da cache
- O campo TAG permite distinguir os diferentes endereços

TÉCNICO LISBOA

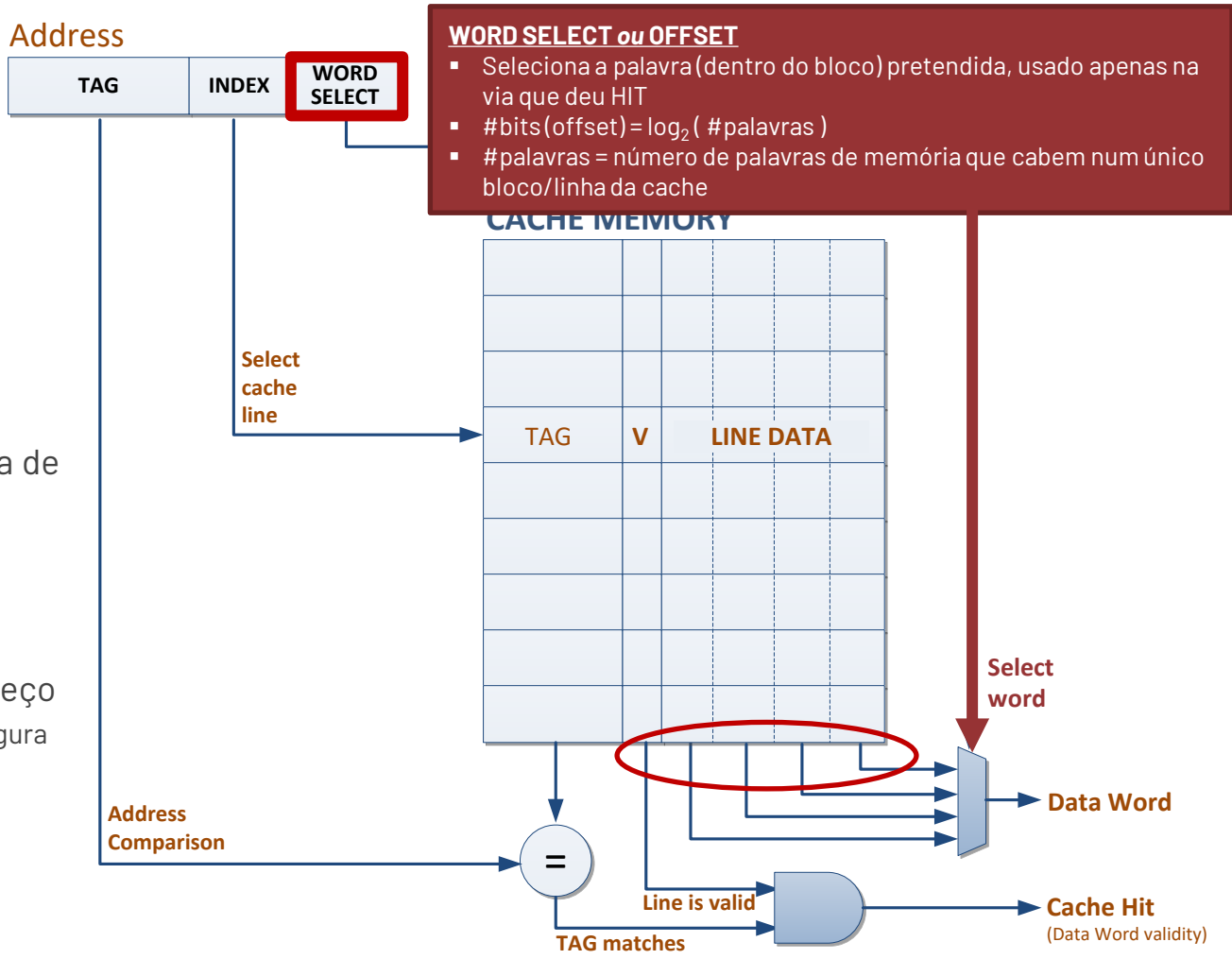
Mapeamento dos dados na cache

- O **INDEX** (índice) é usado para determinar a linha da cache que devemos consultar
- A **TAG** (etiqueta) valida se os dados/instruções na cache correspondem ao endereço indicado
  - Há múltiplos endereços mapeados na mesma linha
- O bit de validade (**V**) indica se a linha é válida



Mapeamento dos dados na cache

- Quando o processador precisa de um dado (ou instrução), faz o pedido à cache L1-D (ou L1-I)
- De acordo com a estrutura da cache L1-D (ou L1-I), esta decompõe a palavra de endereço em TAG, INDEX, OFFSET (na figura WORD SELECT)

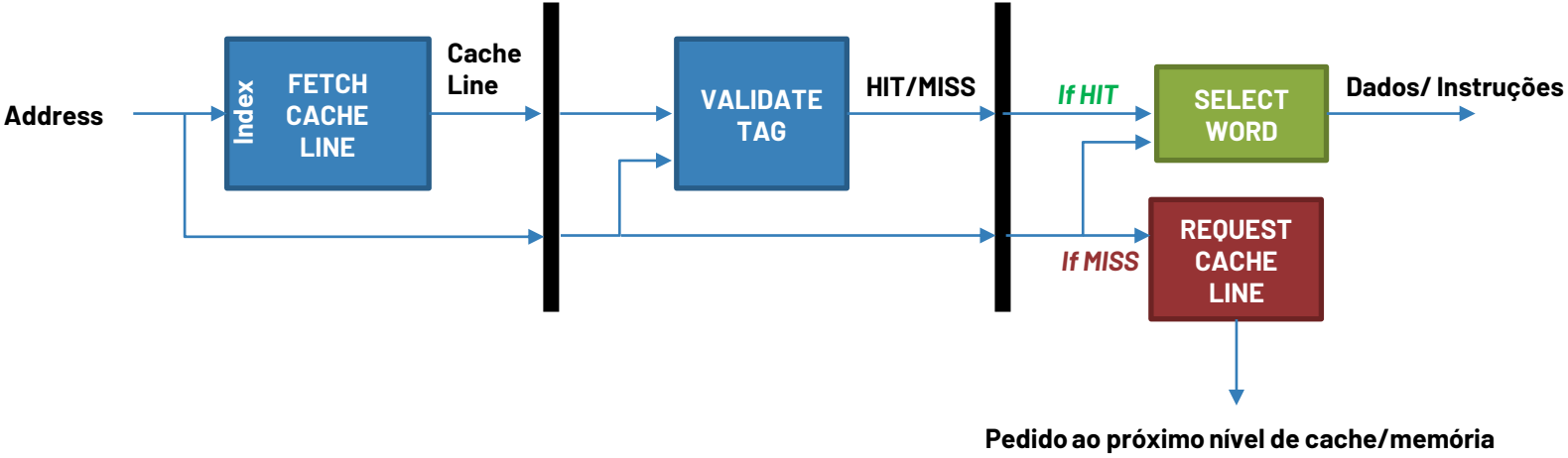


TÉCNICO LISBOA

Acesso em Pipeline

(com 3 estágios)

Possível acesso à cache em Pipeline



Decomposição da palavra de endereço

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

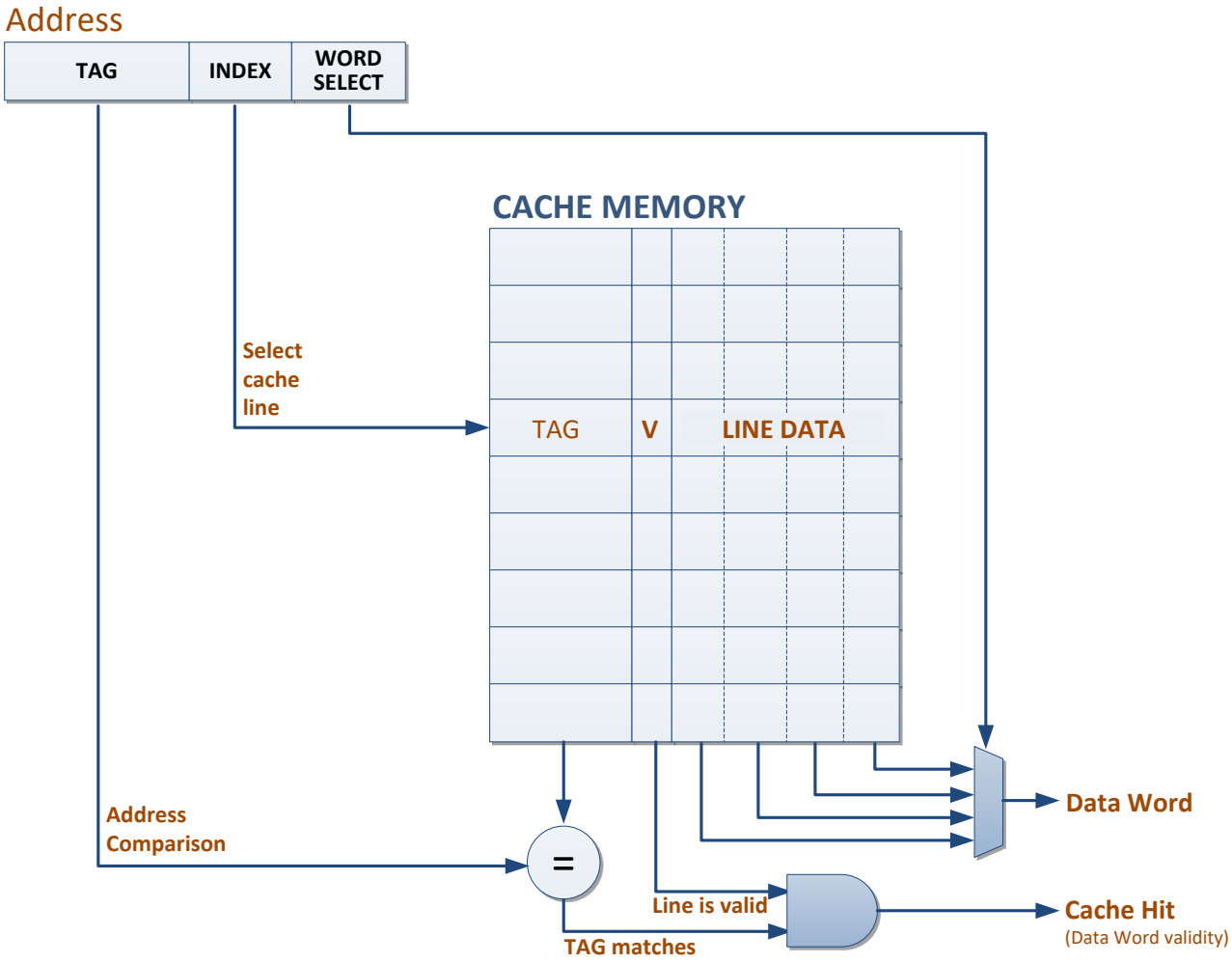
EXERCÍCIO #1

Decomposição da palavra de endereço

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B



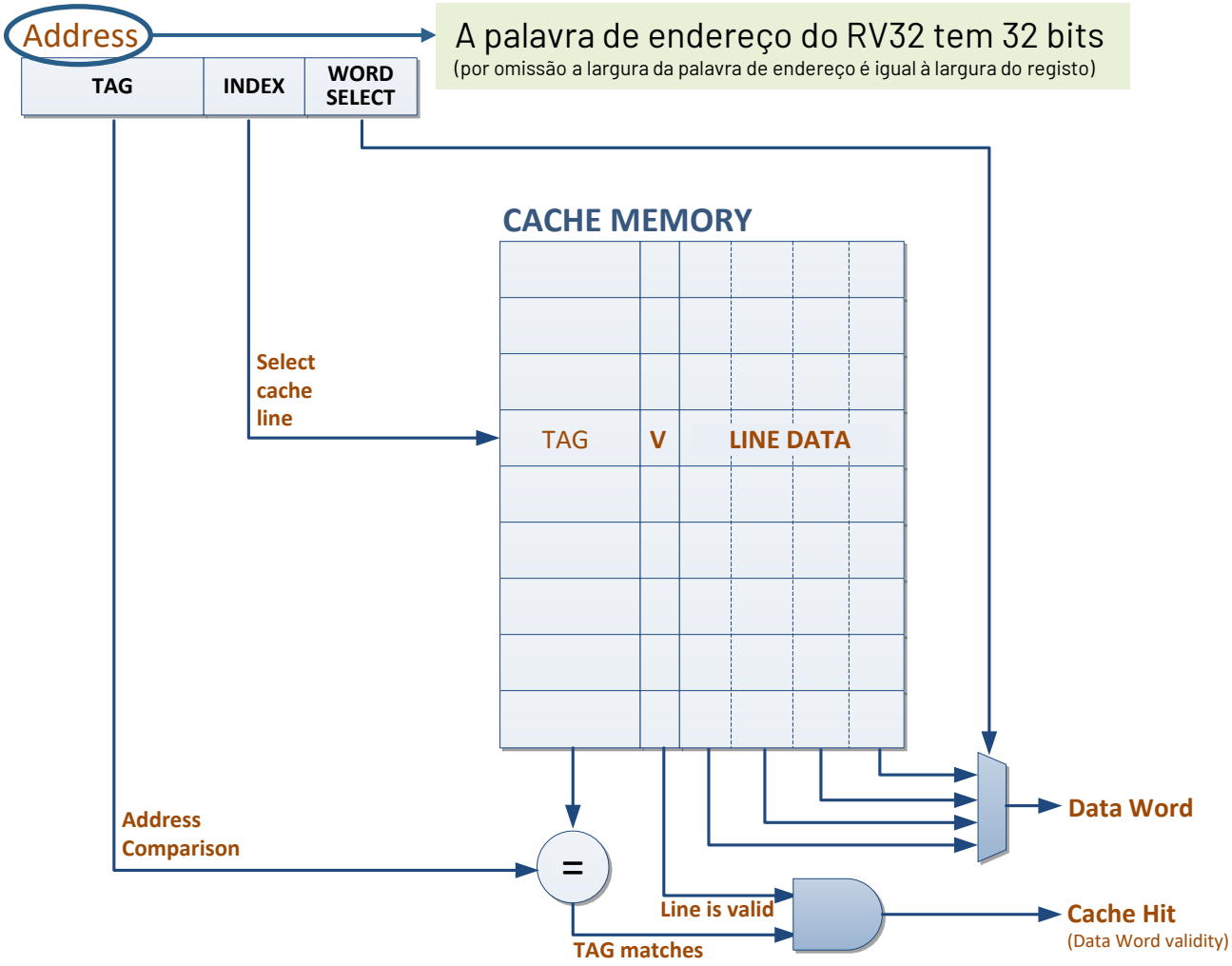
TÉCNICO LISBOA

Decomposição da palavra de endereço

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B



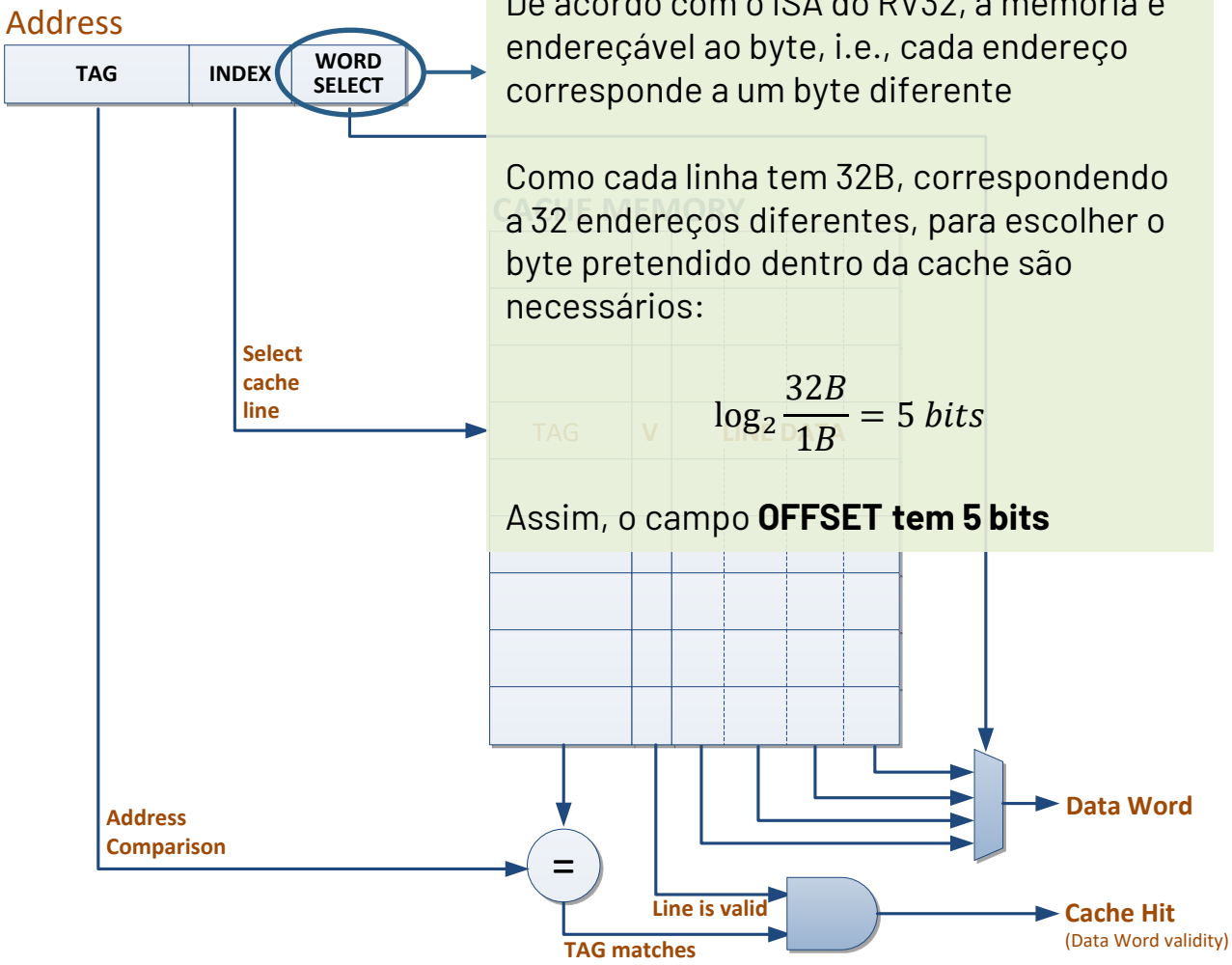
TÉCNICO LISBOA

Decomposição da palavra de endereço

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B



De acordo com o ISA do RV32, a memória é endereçável ao byte, i.e., cada endereço corresponde a um byte diferente

Como cada linha tem 32B, correspondendo a 32 endereços diferentes, para escolher o byte pretendido dentro da cache são necessários:

$$\log_2 \frac{32B}{1B} = 5\ bits$$

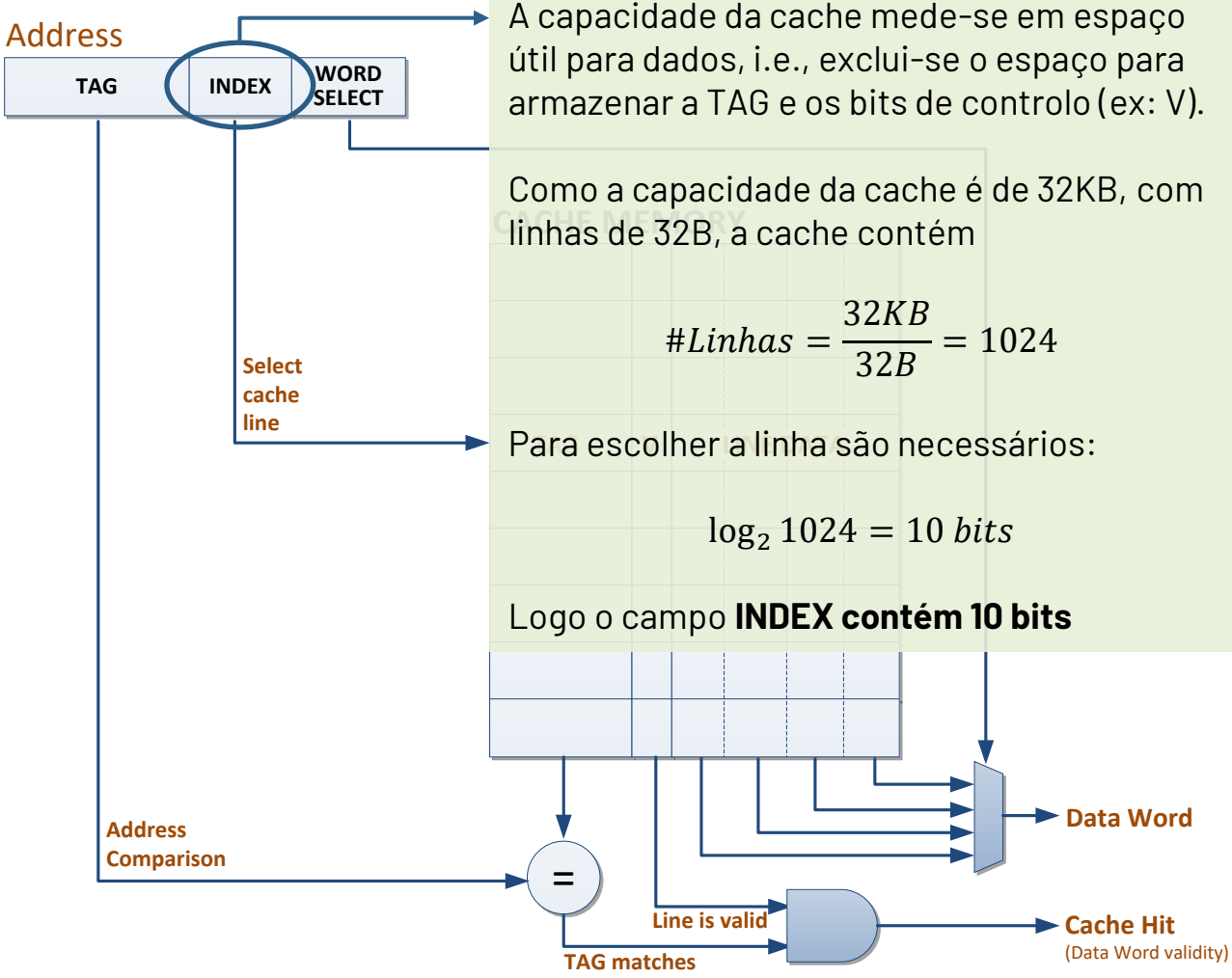
Assim, o campo **OFFSET tem 5 bits**

Decomposição da palavra de endereço

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B



A capacidade da cache mede-se em espaço útil para dados, i.e., exclui-se o espaço para armazenar a TAG e os bits de controlo (ex: V).

Como a capacidade da cache é de 32KB, com linhas de 32B, a cache contém

$$\#Linhas = \frac{32KB}{32B} = 1024$$

Para escolher a linha são necessários:

$$\log_2 1024 = 10\ bits$$

Logo o campo **INDEX contém 10 bits**

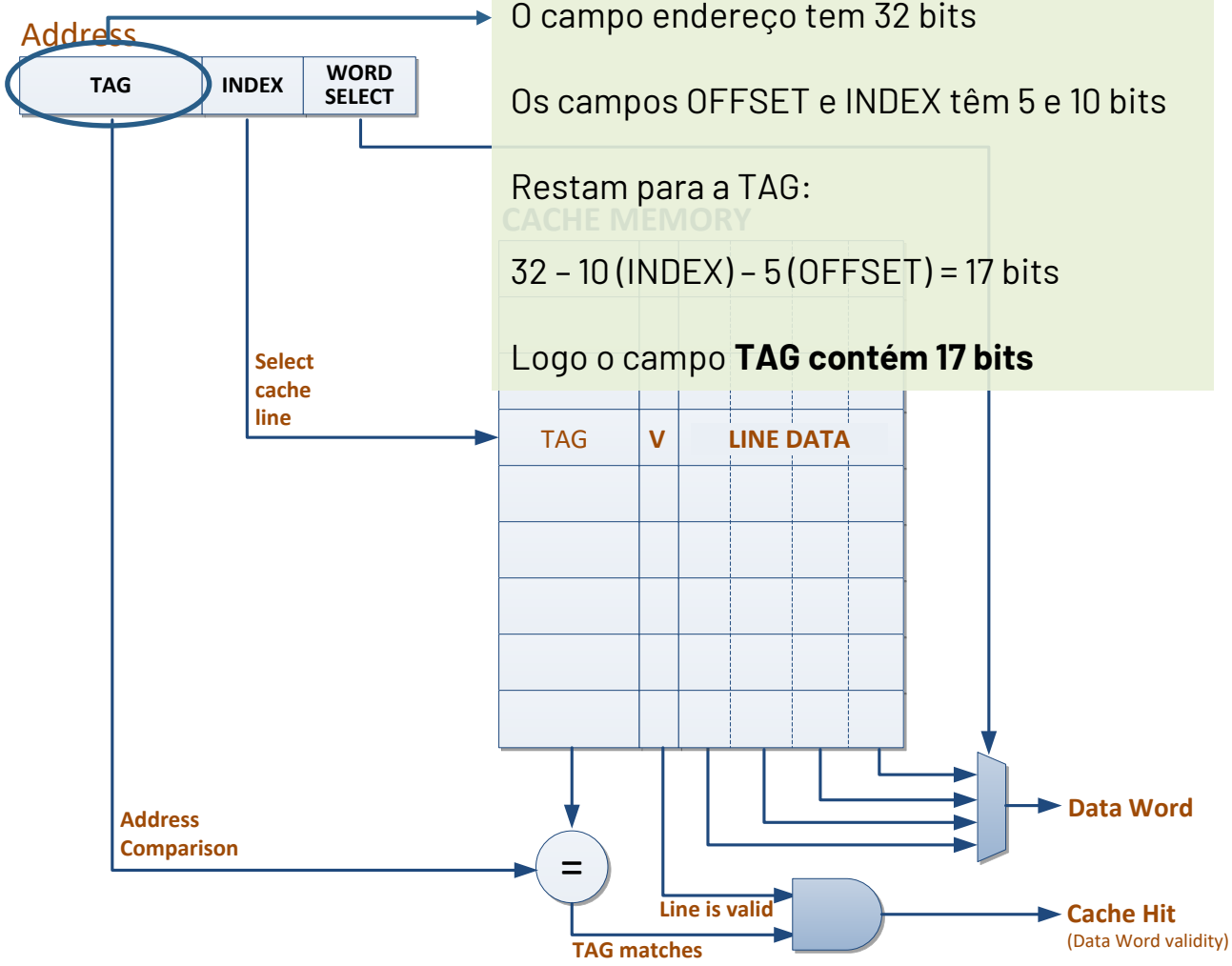
TÉCNICO LISBOA

Decomposição da palavra de endereço

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B



O campo endereço tem 32 bits

Os campos OFFSET e INDEX têm 5 e 10 bits

Restam para a TAG:

32 – 10 (INDEX) – 5 (OFFSET) = 17 bits

Logo o campo **TAG contém 17 bits**

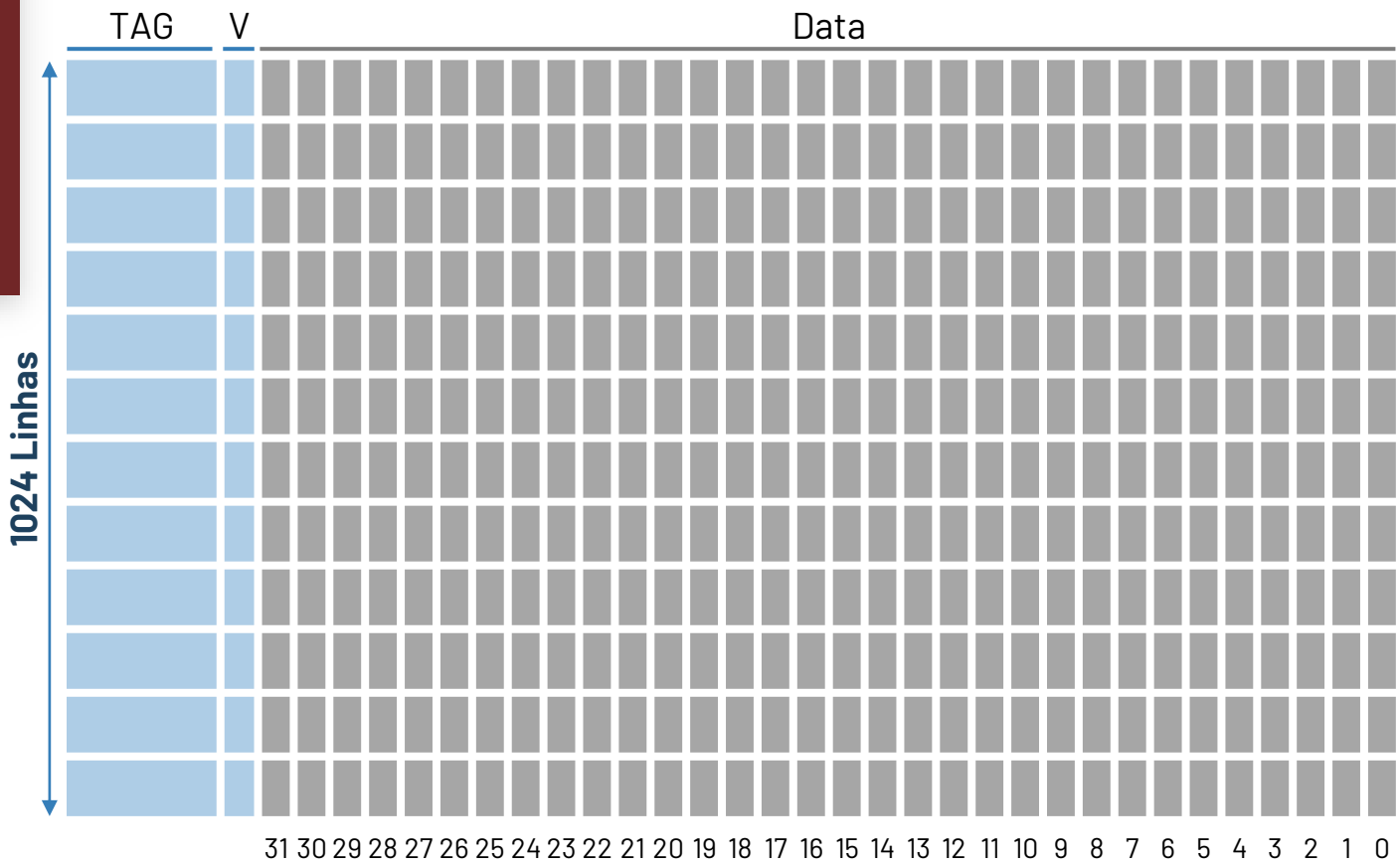
TÉCNICO LISBOA

Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

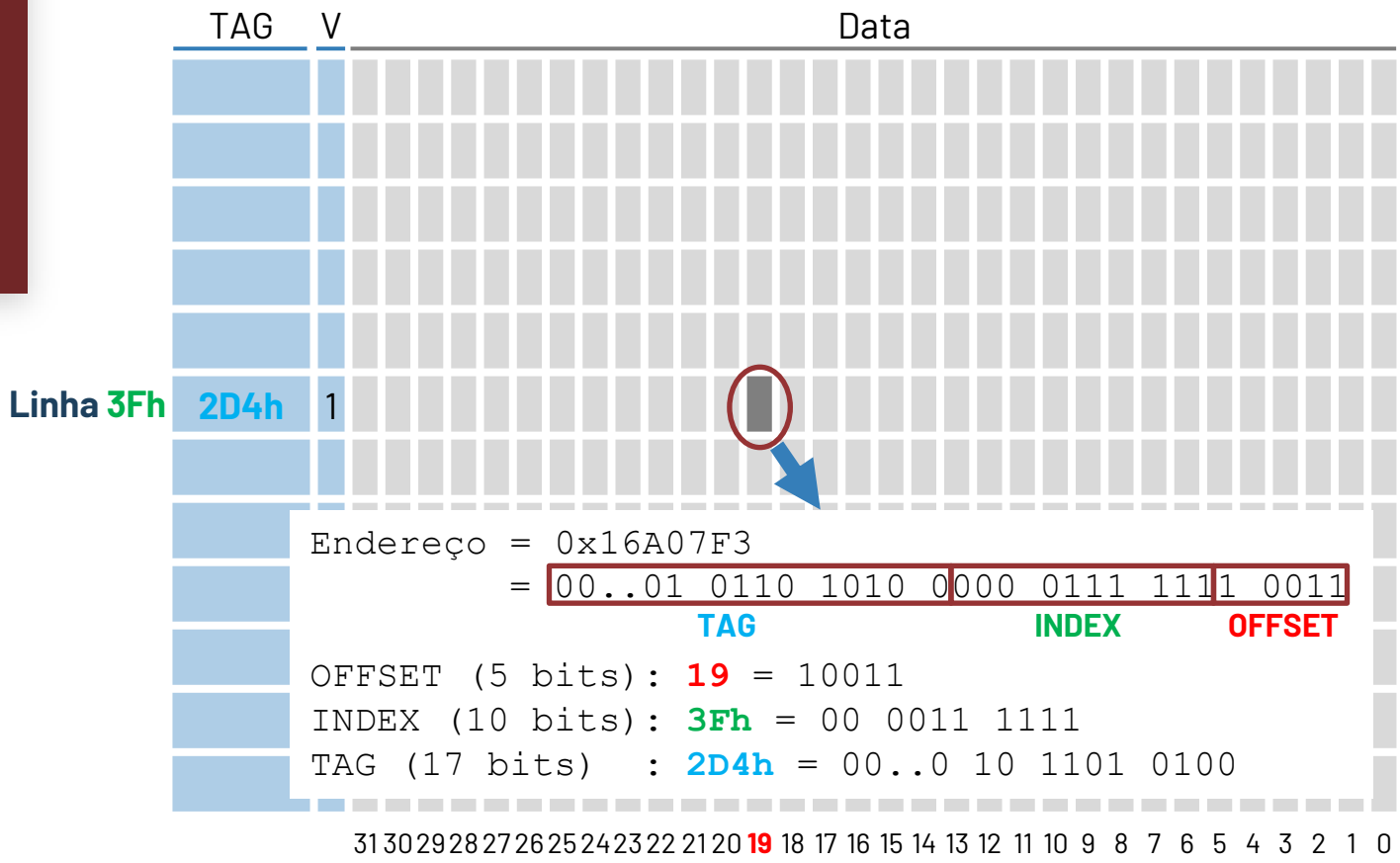


Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B



TÉCNICO LISBOA

Estatística de acesso à cache

- Define-se a taxa de sucesso no acesso à cache (HIT RATE) como:

$$HIT\ RATE = \frac{\#HITS}{\#ACESSOS}$$

- De forma análoga, define-se o MISS RATE como:

$$MISS\ RATE = \frac{\#MISSES}{\#ACESSOS}$$

- Note-se que:

$$MISS\ RATE = 1 - HIT\ RATE$$

Nota: o hit rate e miss rate são geralmente representados como uma percentagem.

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

EXERCÍCIO #2

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Nota: existem duas caches L1:

- L1-I para instruções
- L1-D para dados

Assim, para o acesso aos dados podemos ignorar o acesso às instruções

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

X10 = 512
X1 1= 0x13F8
X12 = 4088 = 0xFF8
X13 = 0

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

Acede ao endereço 0x13F8:

TAG	INDEX	OFFSET
00…00	0001 0011 1111	1000

Vamos aceder à linha 9Fh.

Assumindo a cache inicialmente vazia, i.e., o bit de validade (V) é zero em todas as linhas, vamos ter um **MISS**!

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

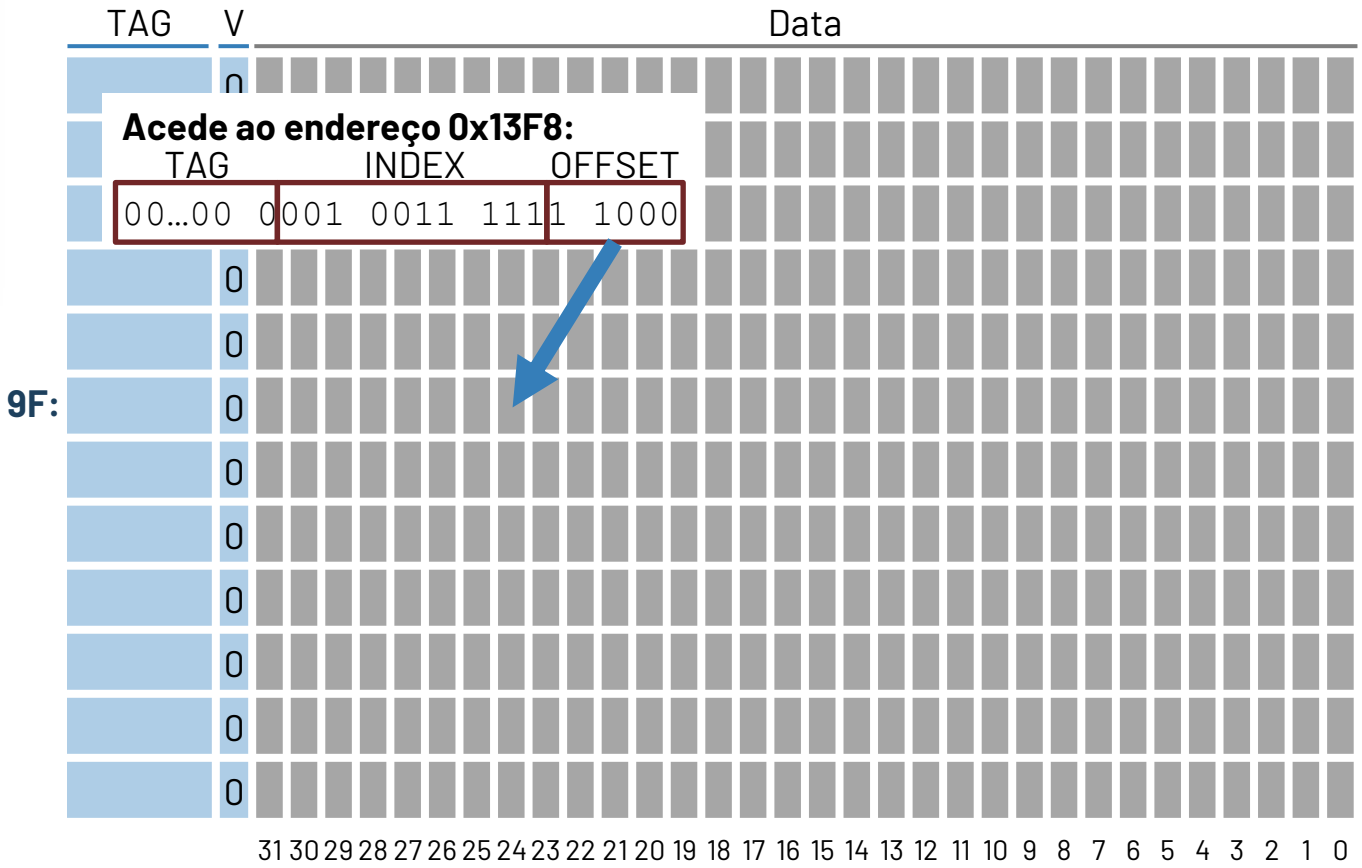
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado inicial da cache:



TÉCNICO LISBOA

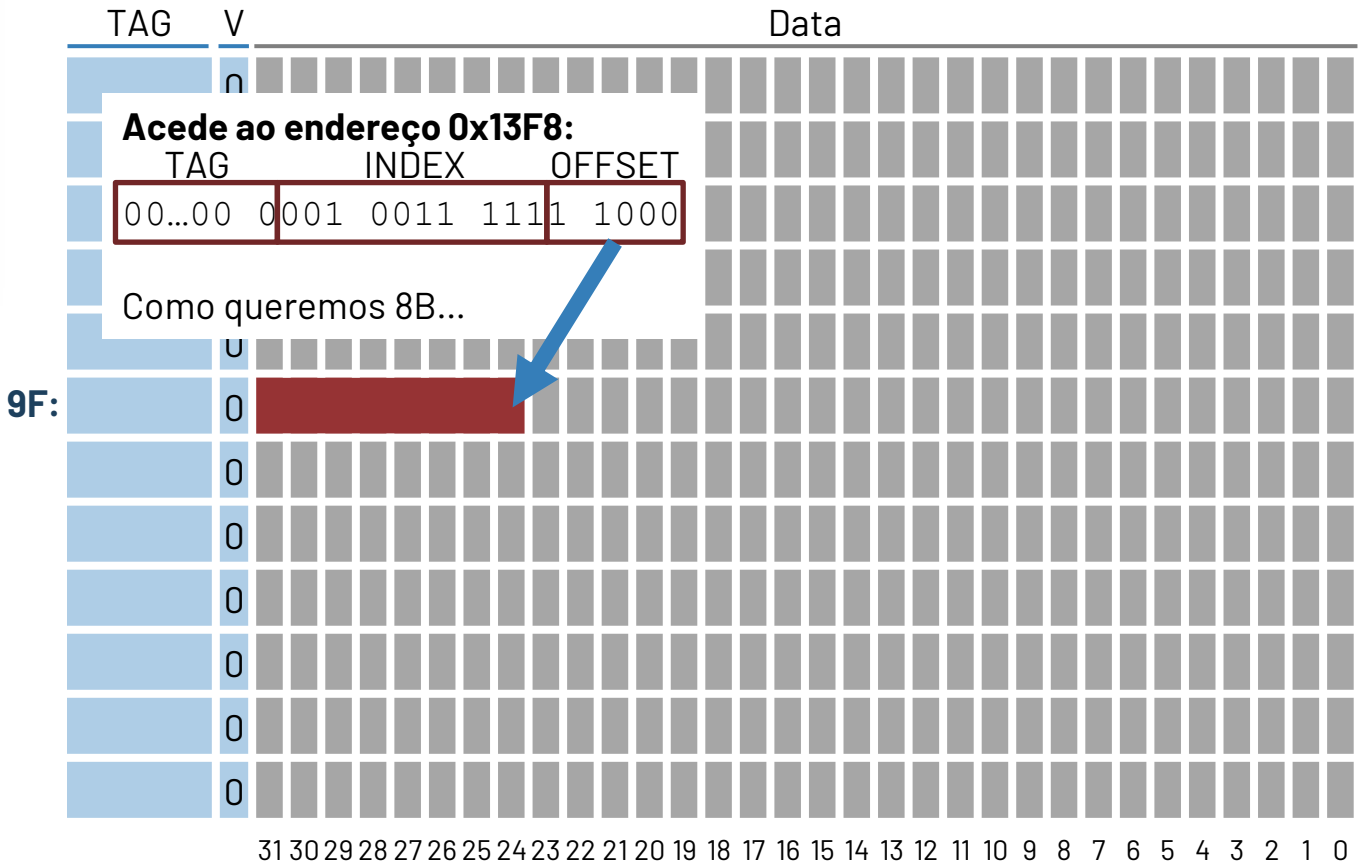
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado inicial da cache:



TÉCNICO LISBOA

Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache:

	TAG	V	Data			
		0				
		0				
9F:	0h	1	**Vector[511]**	**Vector[510]**	**Vector[509]**	**Vector[508]**
		0				

Acede ao endereço 0x13F8:

TAG	INDEX	OFFSET
00…00 0	001 0011	111 1 1000

Como queremos 8B…

Para devolver os dados ao core, a cache começa por ir buscar ao nível superior os dados correspondentes a TODA a linha

Ao preencher a linha, preenche o bit V=1 e o campo TAG com o valor correspondente

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

X10 = 511
X11 = 0x13F0

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

Acede ao endereço 0x13F0:

TAG	INDEX	OFFSET
00…00 0	001 0011 111	1 0000

Vamos novamente aceder à linha 9Fh.

Como esta entrada foi preenchida com o último acesso, vamos ter um **HIT**.

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

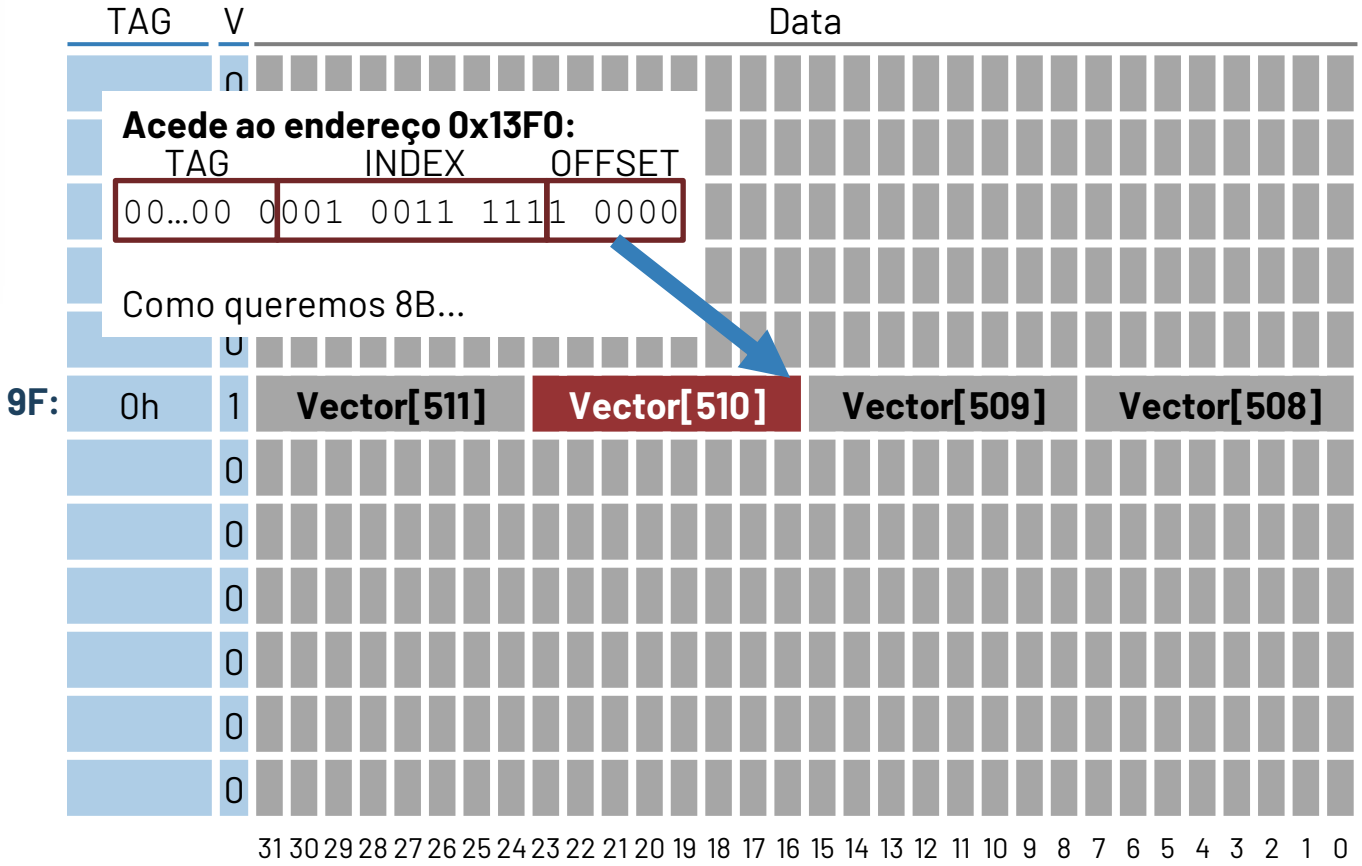
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache:



TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

X0 = 510
X1 = 0x13E8

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

Acede ao endereço 0x13E8:

TAG	INDEX	OFFSET
00…00 0	001 0011 111	0 1000

Vamos aceder novamente à linha 9Fh.

Como esta entrada já foi preenchida vamos ter um **HIT**.

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache:

	TAG	V	Data			
		0				
		0				
		0				
		0				
		0				
9F:	0h	1	**Vector[511]**	**Vector[510]**	**Vector[509]**	**Vector[508]**
		0				
		0				
		0				
		0				
		0				
		0				

Acede ao endereço 0x13E8:

TAG	INDEX	OFFSET
00…00	0001 0011 111	0 1000

Como queremos 8B…

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

X10 = 509
X11 = 0x13E0

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

Acede ao endereço 0x13E0:

TAG	INDEX	OFFSET
00…00 0	001 0011 111	0 0000

Vamos aceder novamente à linha 9Fh.

Como esta entrada já foi preenchida vamos ter um **HIT**.

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache:

	TAG	V	Data
		0	
		0	
		0	
		0	
		0	
9F:	0h	1	**Vector[511]** **Vector[510]** **Vector[509]** **Vector[508]**
		0	
		0	
		0	
		0	
		0	
		0	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Acede ao endereço 0x13E0:

TAG	INDEX	OFFSET
00…00	0001 0011 111	0 0000

Como queremos 8B…

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

X10 = 508
X11 = 0x13D8

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

Acede ao endereço 0x13D8:

TAG	INDEX	OFFSET
00…00 0	001 0011 1101	1000

Vamos aceder a uma linha diferente, i.e., linha 9Eh.

Como esta entrada ainda não foi preenchida vamos ter um **MISS**.

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```


TÉCNICO LISBOA

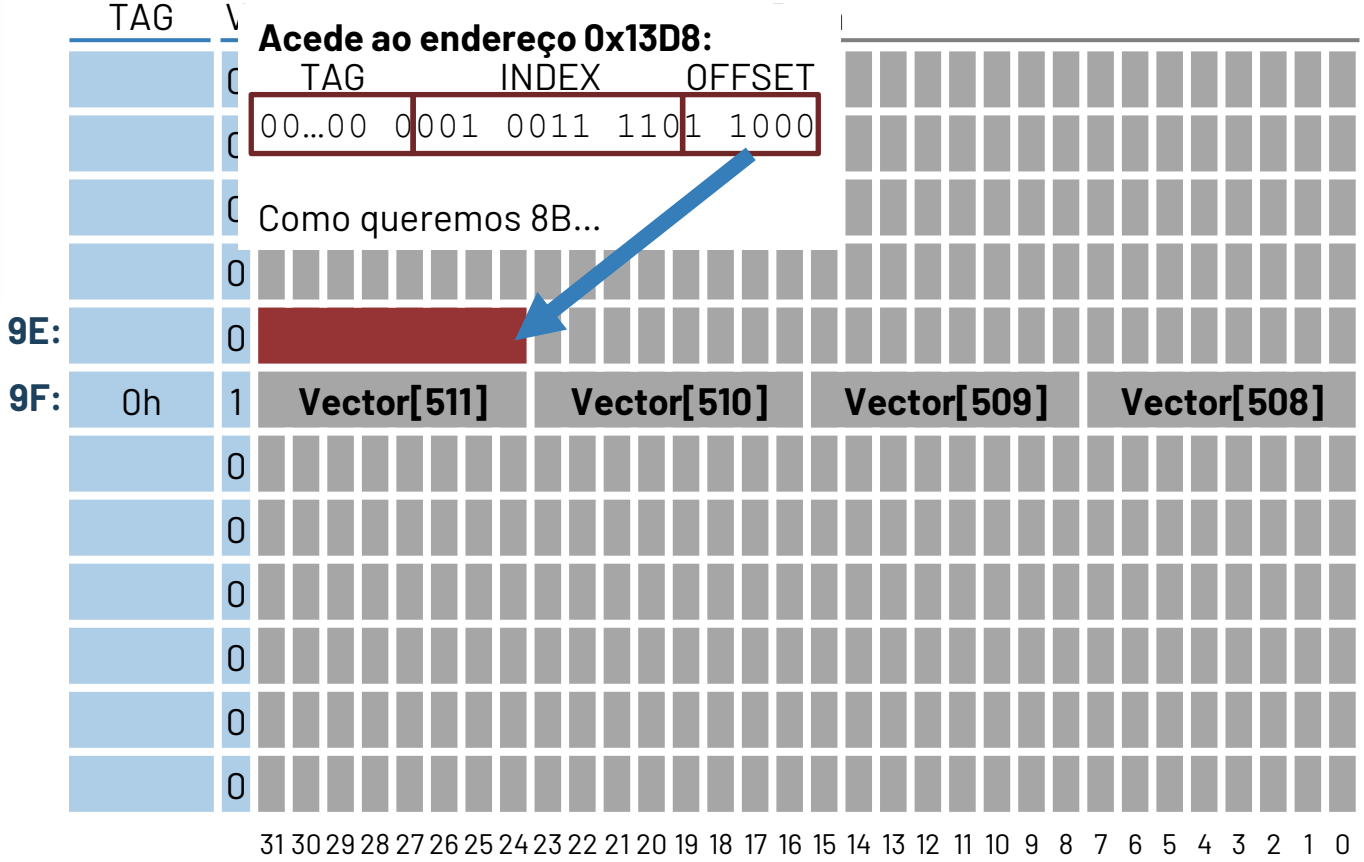
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache antes do acesso ao endereço 13D8h:



TÉCNICO LISBOA

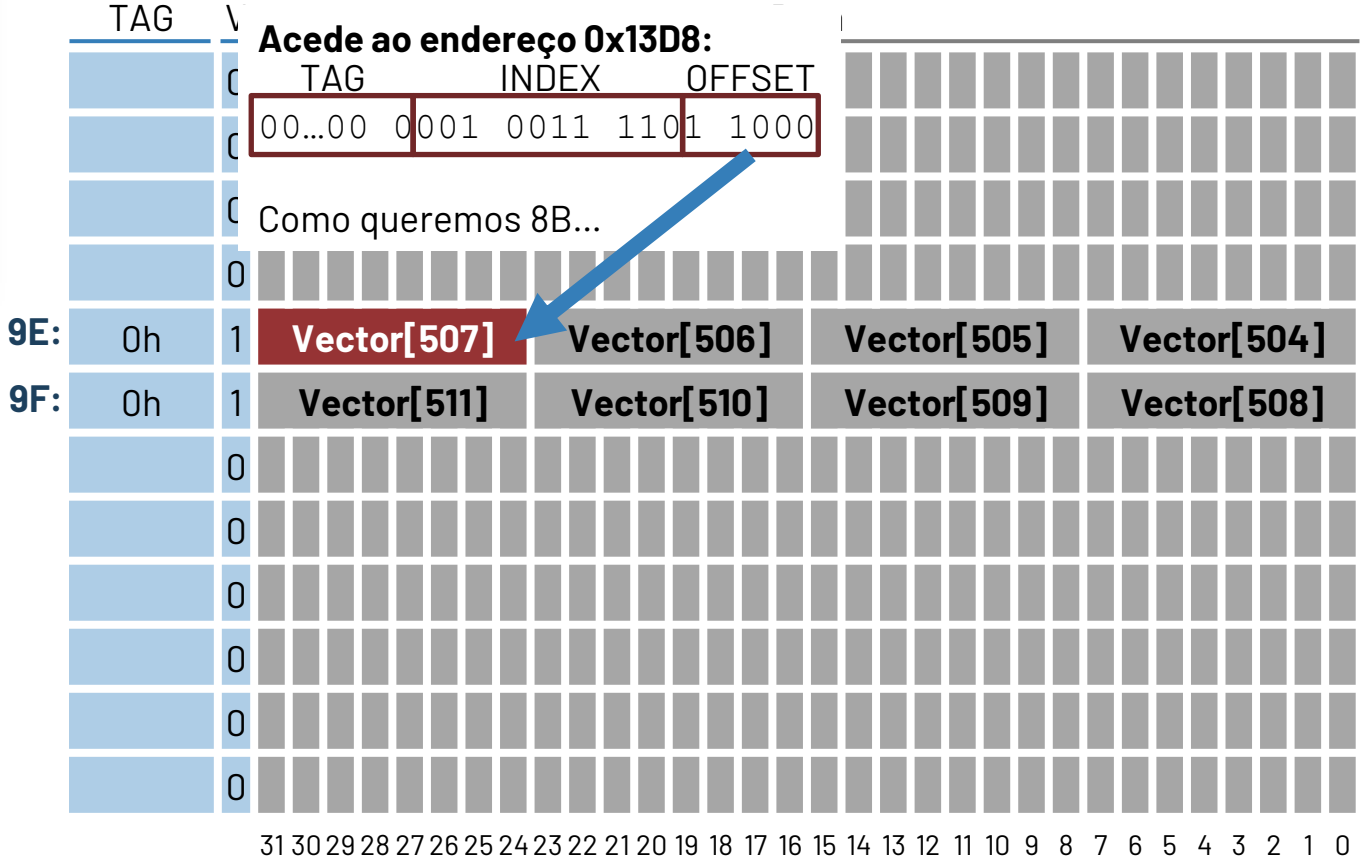
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache após o acesso ao endereço 13D8h:



TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

Para a resolução do exercício, admita que o primeiro elemento do vetor encontra-se no endereço 400h, i.e., Vector=0x400

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Acesso aos endereço 0x13D0, 0x13C8, 0x13C0:

TAG	INDEX	OFFSET
00…00	0001 0011 1101	1000
00…00	0001 0011 1101	1000
00…00	0001 0011 1101	1000

Vamos ter **HIT** (sempre a mesma linha)

TÉCNICO LISBOA

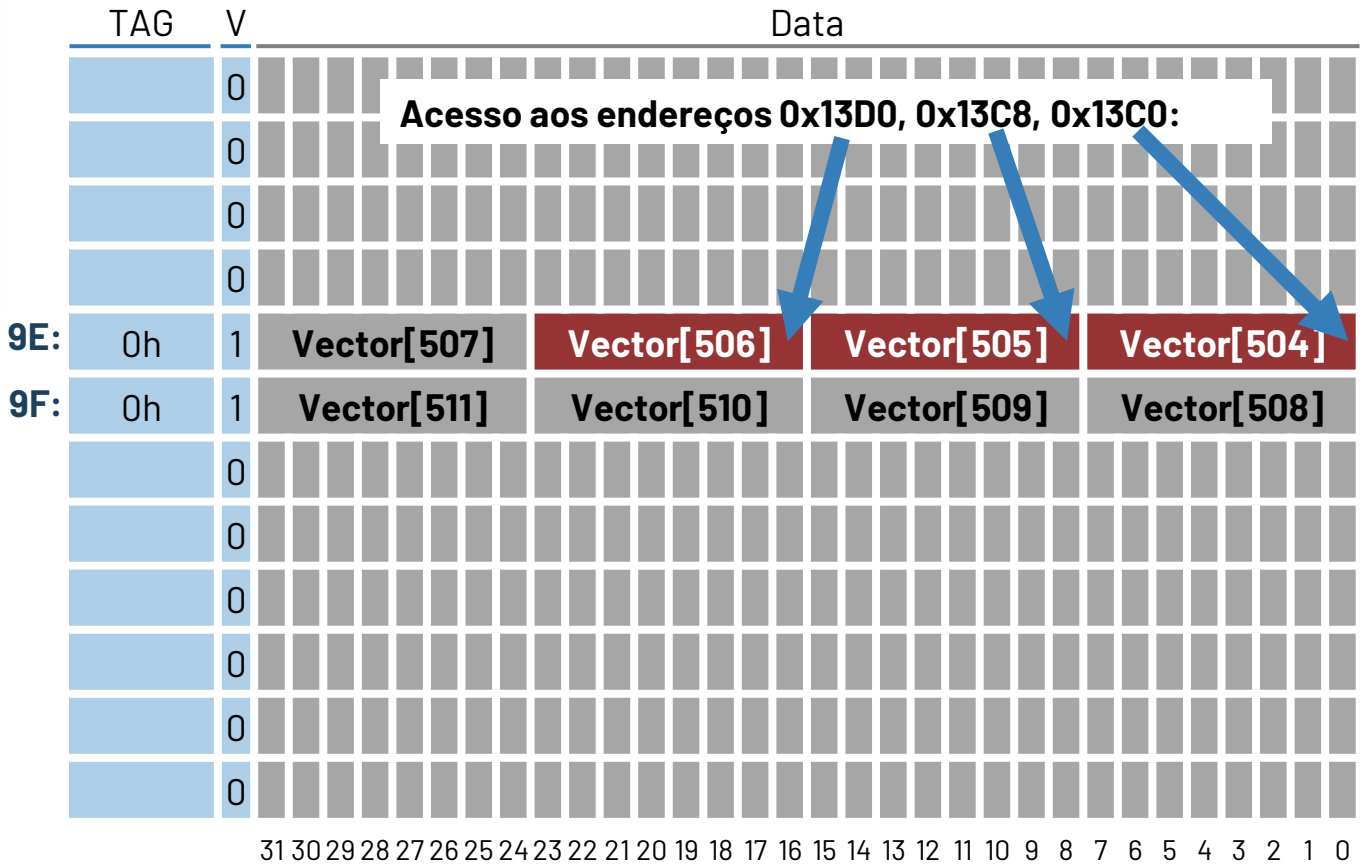
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache após o acesso ao endereço 13D8h:



TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso aos dados para o troço de código indicado.

A taxa de sucesso (HIT RATE):

$$HIT\ RATE = \frac{\#HITS}{\#ACESSOS}$$

- Acesso a um vetor com 512 posições
- No primeiro acesso temos um MISS
- Nos 3 seguintes temos um HIT
- A padrão repete-se em múltiplos de 4

$$\#ACESSOS = 512$$

$$\#HITS = 3 \times \frac{512}{4} = 384$$

$$HIT\ RATE\ (HR) = \frac{384}{512} = \frac{3}{4} = 75\%$$

$$MISS\ RATE = 1 - HR = 1/4 = 25\%$$

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          slli       x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Tempo (latência) médio de acesso aos dados

- No acesso aos dados, primeiro consulta-se a cache, se houver um MISS consultamos o nível seguinte.

- Para um sistema com apenas um nível de cache obtém-se:

$$T_{Acesso} = T_{L1} + MR_{L1} \times T_{RAM}$$

Onde

MR_{L1} corresponde ao MISS RATE da cache (L1)

Tempo (latência) médio de acesso aos dados

Sistemas hierárquicos com vários níveis de cache

- No acesso aos dados, primeiro consulta-se a cache, se houver um MISS consultamos o nível seguinte.

- Para um sistema com 2 níveis de cache:

$$T_{Acesso} = T_{L1} + MR_{L1} \times (T_{L2} + MR_{L2} \times T_{RAM})$$

Onde

MR_{L1}, MR_{L2} corresponde ao MISS RATE das caches L1 e L2

Tempo (latência) médio de acesso aos dados

Sistemas hierárquicos com vários níveis de cache

- No acesso aos dados, primeiro consulta-se a cache, se houver um MISS consultamos o nível seguinte.

- Para um sistema com 3 níveis de cache:

$$T_{Acesso} = T_{L1} + MR_{L1} \times \left(T_{L2} + MR_{L2} \times (T_{L3} + MR_{L3} \times T_{RAM})\right)$$

Onde

$MR_{L1}, MR_{L2}, MR_{L3}$ corresponde ao MISS RATE das caches L1, L2 e L3

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

EXERCÍCIO #3

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine o tempo médio de acesso aos dados

Para a resolução do exercício, admita:

1. Um sistema com apenas 1 nível de cache
2. A latência de acesso à cache L1 é de 2,5 ns
3. A latência de acesso à memória principal é de 100 ns
4. O processador opera a uma frequência de 1GHz

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          sll        x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine o tempo médio de acesso aos dados

Para a resolução do exercício, admita:

1. Um sistema com apenas 1 nível de cache
2. A latência de acesso à cache L1 é de 2,5 ns
3. A latência de acesso à memória principal é de 100 ns
4. O processador opera a uma frequência de 1GHz

$$T_{CLK} = \frac{1}{1GHz} = 1ns$$

$$Lat_{L1} = \left\lceil \frac{2{,}5ns}{T_{CLK}} \right\rceil = \left\lceil \frac{2{,}5ns}{1ns} \right\rceil = 3\ ciclos$$

$$Lat_{RAM} = \left\lceil \frac{100ns}{T_{CLK}} \right\rceil = \left\lceil \frac{100ns}{1ns} \right\rceil = 100\ ciclos$$

```
Vector:   .double   <list of values>

          li        x10,512
          la        x11,Vector
          addi      x12,x10,-1
          sll       x12,x12,3
          add       x11,x11,x12
          fcvt.d.w  f0,x0
          ble       x10,x0,fim
loop:     fld       f1,0(x11)
          fadd.d    f0,f0,f1
          addi      x11,x11,-8
          addi      x10,x10,-1
          bgt       x10,x0,loop
fim:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine o tempo médio de acesso aos dados

Para a resolução do exercício, admita:

1. Um sistema com apenas 1 nível de cache
2. A latência de acesso à cache L1 é de 2,5 ns
3. A latência de acesso à memória principal é de 100 ns
4. O processador opera a uma frequência de 1GHz

$$Lat_{Acesso} = Lat_{L1} + MR_{L1} \times Lat_{RAM}$$

$$Lat_{Acesso} = 3 + 0.25 \times 100 = 28\ ciclos$$

$$T_{Acesso} = Lat_{Acesso} \times T_{CLK} = 28 \times 1ns = 28ns$$

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          sll        x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine o tempo médio de acesso aos dados

Para a resolução do exercício, admita:

1. Um sistema com apenas 1 nível de cache
2. A latência de acesso à cache L1 é de 2,5 ns
3. A latência de acesso à memória principal é de 100 ns
4. O processador opera a uma frequência de 1GHz

COM cache:

$$T_{Acesso} = Lat_{Acesso} \times T_{CLK} = 28 \times 1ns = 28ns$$

SEM cache (só com mem. principal):

$$T_{Acesso} = Lat_{Acesso} \times T_{CLK} = 100 \times 1ns = 100ns$$

ou seja: neste caso, a cache proporciona uma aceleração de cerca de 100/28=3,57 vezes!!!

```
Vector:   .double    <list of values>

          li         x10,512
          la         x11,Vector
          addi       x12,x10,-1
          sll        x12,x12,3
          add        x11,x11,x12
          fcvt.d.w   f0,x0
          ble        x10,x0,fim
loop:     fld        f1,0(x11)
          fadd.d     f0,f0,f1
          addi       x11,x11,-8
          addi       x10,x10,-1
          bgt        x10,x0,loop
fim:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

EXERCÍCIO #4

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine se existe alguma vantagem em introduzir uma cache L2, com uma latência de 10 ciclos de relógio.

Considere que o miss rate na cache L2 é de 50%.

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine se existe alguma vantagem em introduzir uma cache L2, com uma latência de 10 ciclos de relógio.

Considere que o miss rate na cache L2 é de 50%.

$$Lat_{Acesso} = Lat_{L1} + MR_{L1} \times (Lat_{L2} + MR_{L2} \times Lat_{RAM})$$
$$= 3 + 0.25 \times (10 + 0.5 \times 100) = 18\ ciclos$$

No acesso aos dados perdem-se (em media) menos 10 ciclos!

Nota: para um acesso individual, sempre que há um miss, perdem-se mais 10 ciclos. Contudo, em média, a introdução da cache é vantajosa, já que:

- 50% das vezes dá miss, perdem-se mais 10 ciclos ☹
- 50% das vezes dá hit, perdem-se menos 90 ciclos ☺

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

EXERCÍCIO #5

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

```
N:        .word     512
Vector:   .double   <list of values>

          lw        x10,N
          la        x11,Vector
          addi      x12,x10,-1
          sll       x12,x12,3
          add       x11,x11,x12
          fcvt.d.w  f0,x0
          ble       x10,x0,fim
loop:     fld       f1,0(x11)
          fadd.d    f0,f0,f1
          addi      x11,x11,-8
          addi      x10,x10,-1
          bgt       x10,x0,loop
fim:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

Pseudo-instrução: `auipc + lw`

Pseudo-instrução: `auipc + addi`

Pseudo-instrução: `bge`

Pseudo-instrução: `blt`

```
N:          .word     512
Vector:     .double   <list of values>
```

Address			
00h		lw	x10,N
08h		la	x11,Vector
10h		addi	x12,x10,-1
14h		sll	x12,x12,3
18h		add	x11,x11,x12
1Ch		fcvt.d.w	f0,x0
20h		ble	x10,x0,fim
24h	loop:	fld	f1,0(x11)
28h		fadd.d	f0,f0,f1
2Ch		addi	x11,x11,-8
30h		addi	x10,x10,-1
34h		bgt	x10,x0,loop
	fim:		

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

Acede ao endereço 00h:

TAG	INDEX	OFFSET
00…00 0	000 0000 000	0 0000

Assumindo a cache inicialmente vazia, vamos ter um **MISS**!

Preenchemos a linha 0 da cache…

```
N:          .word      512
Vector:     .double    <list of values>
Address
00h                    lw         x10,N
08h                    la         x11,Vector
10h                    addi       x12,x10,-1
14h                    sll        x12,x12,3
18h                    add        x11,x11,x12
1Ch                    fcvt.d.w   f0,x0
20h                    ble        x10,x0,fim
24h         loop:      fld        f1,0(x11)
28h                    fadd.d     f0,f0,f1
2Ch                    addi       x11,x11,-8
30h                    addi       x10,x10,-1
34h                    bgt        x10,x0,loop
            fim:
```


TÉCNICO LISBOA

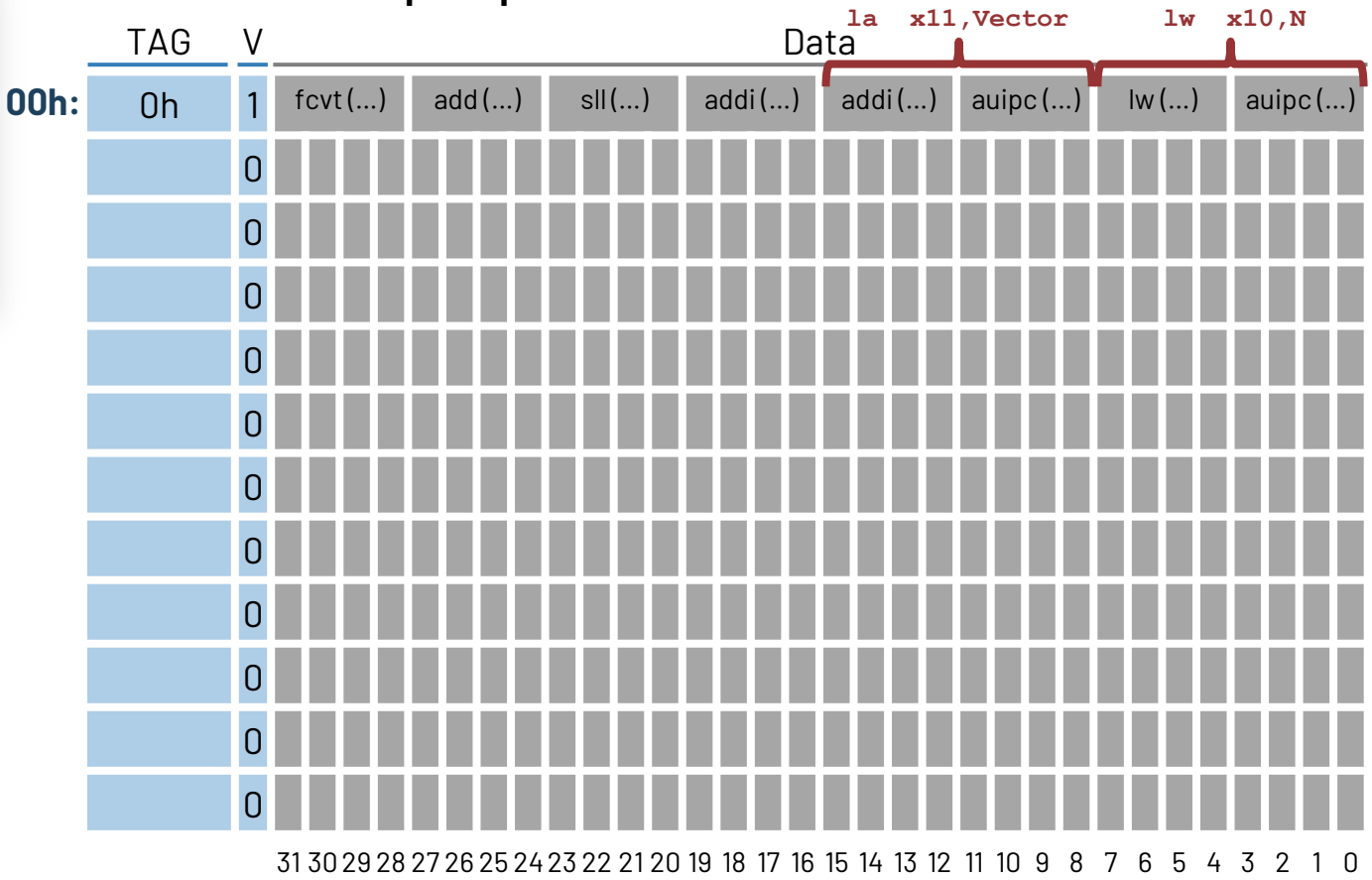
Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache L1-I após o primeiro acesso:



TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

Os próximos 7 acessos vão dar HIT

```
        N:          .word       512
        Vector:     .double     <list of values>
```

	Address		
MISS / HIT	00h	lw	x10,N
HIT / HIT	08h	la	x11,Vector
HIT	10h	addi	x12,x10,-1
HIT	14h	sll	x12,x12,3
HIT	18h	add	x11,x11,x12
HIT	1Ch	fcvt.d.w	f0,x0
	20h	ble	x10,x0,fim
	24h	loop: fld	f1,0(x11)
	28h	fadd.d	f0,f0,f1
	2Ch	addi	x11,x11,-8
	30h	addi	x10,x10,-1
	34h	bgt	x10,x0,loop
		fim:	

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

Acede ao endereço 20h:

TAG	INDEX	OFFSET
00…00 0	000 0000 001	0 0000

Corresponde a uma linha diferente

Como a linha ainda não foi preenchida vai dar **MISS**.

Vamos preencher a linha 1 da cache…

```
        N:          .word       512
        Vector:     .double     <list of values>
Address
00h                 lw          x10,N
08h                 la          x11,Vector
10h                 addi        x12,x10,-1
14h                 sll         x12,x12,3
18h                 add         x11,x11,x12
1Ch                 fcvt.d.w    f0,x0
20h                 ble         x10,x0,fim
24h     loop:       fld         f1,0(x11)
28h                 fadd.d      f0,f0,f1
2Ch                 addi        x11,x11,-8
30h                 addi        x10,x10,-1
34h                 bgt         x10,x0,loop
        fim:
```


TÉCNICO LISBOA

Estrutrura da cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Estado da cache L1-I após acesso ao endereço 20h:

	TAG	V	Data							
00h:	0h	1	fcvt(...)	add(...)	sll(...)	addi(...)	addi(...)	auipc(...)	lw(...)	auipc(...)
01h:	0h	1	?	?	blt(...)	addi(...)	addi(...)	fadd.d(...)	fld(...)	bge(...)
		0			`bgt x10,x0,loop`				`ble x10,x0,fim`	
		0								
		0								
		0								
		0								
		0								
		0								
		0								
		0								
		0								

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

Os próximos acessos vão dar **HIT**

Como a cache já está preenchida as próximas iterações do ciclo vão dar sempre **HIT**

Nota: isto só acontece porque nestes exemplos as linhas nunca saem da cache

```
                N:          .word       512
                Vector:     .double     <list of values>
```

	Address			
MISS / HIT	00h		lw	x10,N
HIT / HIT	08h		la	x11,Vector
HIT	10h		addi	x12,x10,-1
HIT	14h		sll	x12,x12,3
HIT	18h		add	x11,x11,x12
HIT	1Ch		fcvt.d.w	f0,x0
MISS	20h		ble	x10,x0,fim
HIT	24h	loop:	fld	f1,0(x11)
HIT	28h		fadd.d	f0,f0,f1
HIT	2Ch		addi	x11,x11,-8
HIT	30h		addi	x10,x10,-1
HIT	34h		bgt	x10,x0,loop
		fim:		

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 32KB
- Linhas com 32B

Determine a taxa de sucesso no acesso às instruções para o troço de código indicado.

Preâmbulo:

- 2 MISS, 7 HITs, 9 Acessos

Ciclo:

- 5 HITs, 5 Acessos

Total:

- 2 MISS
- 5x512 HITs
- 2+5x512 Acessos

$$HIT\ RATE\ (HR) = \frac{2567}{2569} = 99.92\%$$

$$MISS\ RATE\ (MR) = \frac{2}{2569} = 0.08\% = 1 - HR$$

```
N:          .word     512
Vector:     .double   <list of values>
```

	Address		
MISS / HIT	00h	lw	x10,N
HIT / HIT	08h	la	x11,Vector
HIT	10h	addi	x12,x10,-1
HIT	14h	sll	x12,x12,3
HIT	18h	add	x11,x11,x12
HIT	1Ch	fcvt.d.w	f0,x0
MISS	20h	ble	x10,x0,fim
HIT	24h	loop: fld	f1,0(x11)
HIT	28h	fadd.d	f0,f0,f1
HIT	2Ch	addi	x11,x11,-8
HIT	30h	addi	x10,x10,-1
HIT	34h	bgt	x10,x0,loop
		fim:	

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

EXERCÍCIO #6

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

Admita os seguintes endereços para a localização dos vetores VA e VB:

VA = &(VA[0]) = 1000h
VB = &(VB[0]) = 2000h

E ainda as seguintes latências de acessos aos dados:

Latência da L1 = 4 ciclos
Latência da DRAM = 50 ciclos

1. **Determine a taxa de sucesso no acesso aos <u>dados</u> para o troço de código indicado.**

2. **Determine o tempo médio de acesso aos dados**

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

Bloco de dados de 16B, i.e., 16 endereços de 1 byte. Para selecionar o byte, precisamos de:

$$\log_2 \frac{16B}{1B} = 4 \text{ bits}$$

OFFSET ➔ 4 bits

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

4KB de dados, distribuídos em linhas de 16B, o que corresponde a 4KB/16B=$2^{12}/2^4=2^8$=256 linhas

Para endereçar 256 linhas são necessários

$\log_2(256) = 8$ bits

INDEX ➔ 8 bits

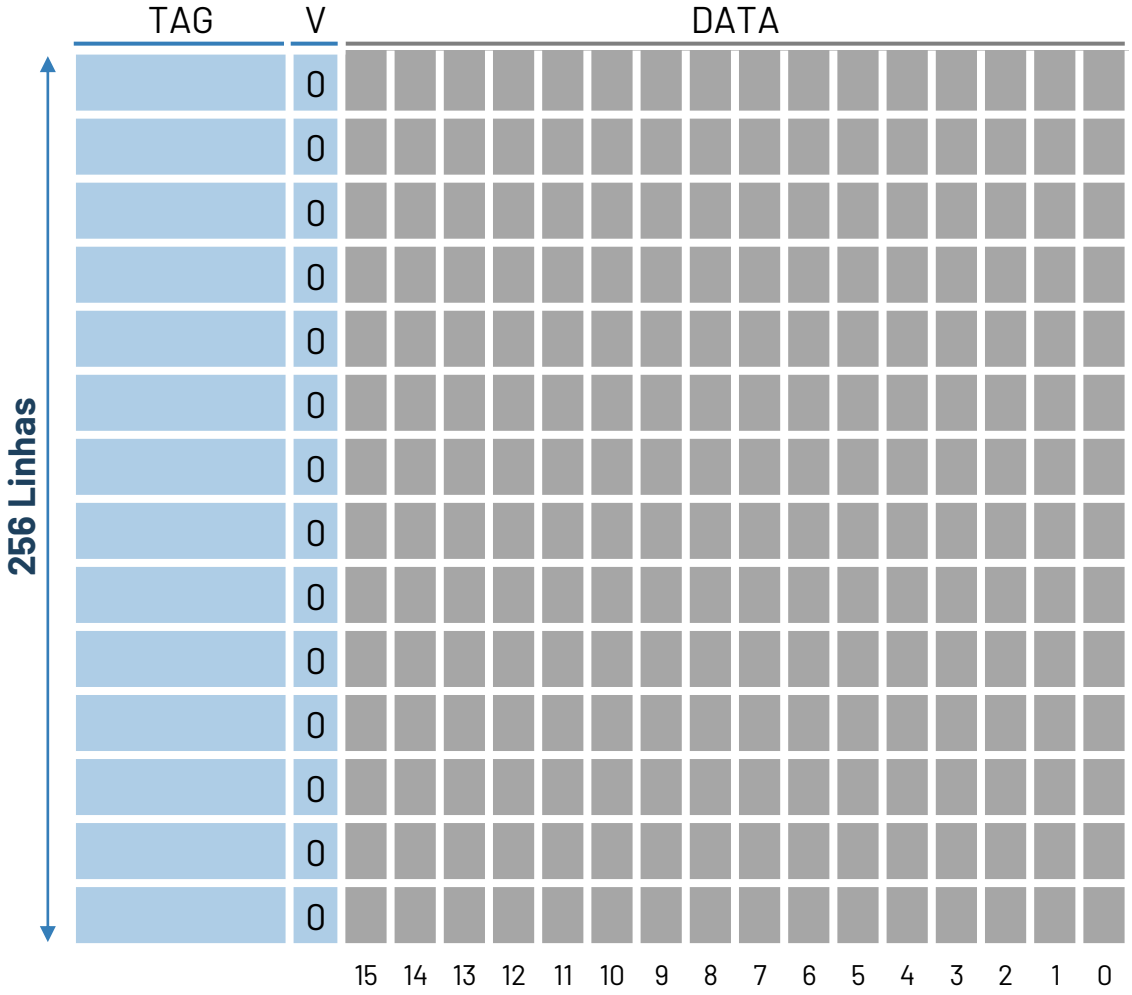
```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Estrutrura da cache

ESTADO INICIAL DA CACHE:

ADDRESS (32 bits)

TAG	INDEX	OFFSET
20 bits	8 bits	4 bits



TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

Admita os seguintes endereços para a localização dos vetores VA e VB:

VA = &(VA[0]) = 1000h
VB = &(VB[0]) = 2000h

X10 = 1024
X11 = 0x1000
X12 = 0x2000
X13 = 0xFFC
X14 = 0x1FFC
X15 = 0x2FFC
X16 = 0

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

0 0001h	FFh	Ch
20 bits	8 bits	4 bits

Como a cache está inicialmente vazia, i.e., todas as entradas têm V=0, o acesso à entrada FFh corresponde necessariamente a um **MISS**!

Vamos preencher a linha...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 1FFCh

Estrutrura da cache

Preenchimento da cache após o primeiro acesso:

ADDRESS (32 bits)

0 0001h	FFh	Ch
20 bits	8 bits	4 bits

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	**1h**	**1**	**VA[1023]**	**VA[1022]**	**VA[1021]**	**VA[1020]**
			ADDRESS: 1FFCh	**ADDRESS: 1FF8h**	**ADDRESS: 1FF4h**	**ADDRESS: 1FF0h**

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

0 0002h	FFh	Ch
20 bits	8 bits	4 bits

Vamos verificar o estado da cache...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 2FFCh

Estrutrura da cache

Endereço pedido: 2FFCh

A linha selecionada (FFh), é válida, mas a TAG não é igual. Assim, temos um **MISS**!

ADDRESS (32 bits)

0 0002h	FFh	Ch
20 bits	8 bits	4 bits

A TAG é diferente!

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	1h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]
			ADDRESS: 1FFCh	**ADDRESS:** 1FF8h	**ADDRESS:** 1FF4h	**ADDRESS:** 1FF0h

TÉCNICO LISBOA

Estrutrura da cache

Endereço pedido: 2FFCh

Vamos colocar os dados em cache... como corresponde à mesma linha vamos substituir esta linha...

ADDRESS (32 bits)

0 0002h	FFh	Ch
20 bits	8 bits	4 bits

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	2h	1	VB[1023]	VB[1022]	VB[1021]	VB[1020]
			ADDRESS: 2FFCh	**ADDRESS:** 2FF8h	**ADDRESS:** 2FF4h	**ADDRESS:** 2FF0h

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

0 0001h	FFh	8h
20 bits	8 bits	4 bits

Vamos verificar o estado da cache...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 1FF8h

Estrutrura da cache

Endereço pedido: 1FFCh

A linha selecionada (FFh), é válida, mas a TAG não é igual. Assim, temos um **MISS**!

ADDRESS (32 bits)

0 0001h	FFh	8h
20 bits	8 bits	4 bits

A TAG é diferente!

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	2h	1	VB[1023]	VB[1022]	VB[1021]	VB[1020]
			ADDRESS: 2FFCh	**ADDRESS:** 2FF8h	**ADDRESS:** 2FF4h	**ADDRESS:** 2FF0h

Estrutrura da cache

Endereço pedido: 1FF8h

Vamos colocar os dados em cache... como corresponde à mesma linha vamos substituir esta linha...

ADDRESS (32 bits)

0 0001h	FFh	8h
20 bits	8 bits	4 bits

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	1h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]
			ADDRESS: 1FFCh	**ADDRESS:** 1FF8h	**ADDRESS:** 1FF4h	**ADDRESS:** 1FF0h

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

0 0002h	FFh	8h
20 bits	8 bits	4 bits

Vamos verificar o estado da cache...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 2FF8h

TÉCNICO LISBOA

Estrutrura da cache

Endereço pedido: 2FF8h

A linha selecionada (FFh), é válida, mas a TAG não é igual. Assim, temos um **MISS**!

ADDRESS (32 bits)

0 0002h	FFh	8h
20 bits	8 bits	4 bits

A TAG é diferente!

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	1h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]
			ADDRESS: 1FFCh	**ADDRESS:** 1FF8h	**ADDRESS:** 1FF4h	**ADDRESS:** 1FF0h

Estrutrura da cache

Endereço pedido: 2FFCh

Vamos colocar os dados em cache... como corresponde à mesma linha vamos substituir esta linha...

ADDRESS (32 bits)

0 0002h	FFh	8h
20 bits	8 bits	4 bits

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
		0				
FFh:	2h	1	VB[1023]	VB[1022]	VB[1021]	VB[1020]
			ADDRESS: 2FFCh	**ADDRESS:** 2FF8h	**ADDRESS:** 2FF4h	**ADDRESS:** 2FF0h

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Mapeamento direto
- Capacidade para 4KB
- Linhas com 16B

Embora seja um exemplo feito "à medida", ilustra um caso típico de padrões de acesso à memória com conflitos.

Em consequência todos os acessos dão **MISS** na cache!

$MISS\ RATE = 100\%$, $HIT\ RATE = 0\%$

Como resolver o problema dos conflitos?

Vamos experimentar uma cache associativa...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- **2 vias (i.e., 2 tabelas)**
- Capacidade para 4KB
- Linhas com 16B

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

Bloco de dados de 16B, i.e., 16 endereços de 1 byte. Para selecionar o byte, precisamos de:

$$\log_2 \frac{16B}{1B} = 4 \text{ bytes}$$

OFFSET ➔ 4 bits

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

4KB de dados, distribuídos em linhas de 16B, o que corresponde a 4KB/16B=256 linhas

Como existem duas vias (tabelas), as linhas distribuem-se de igual forma por ambas.

Assim, cada via tem 128 linhas. Para as endereçar são necessários

$\log_2(128) = 7$ bits

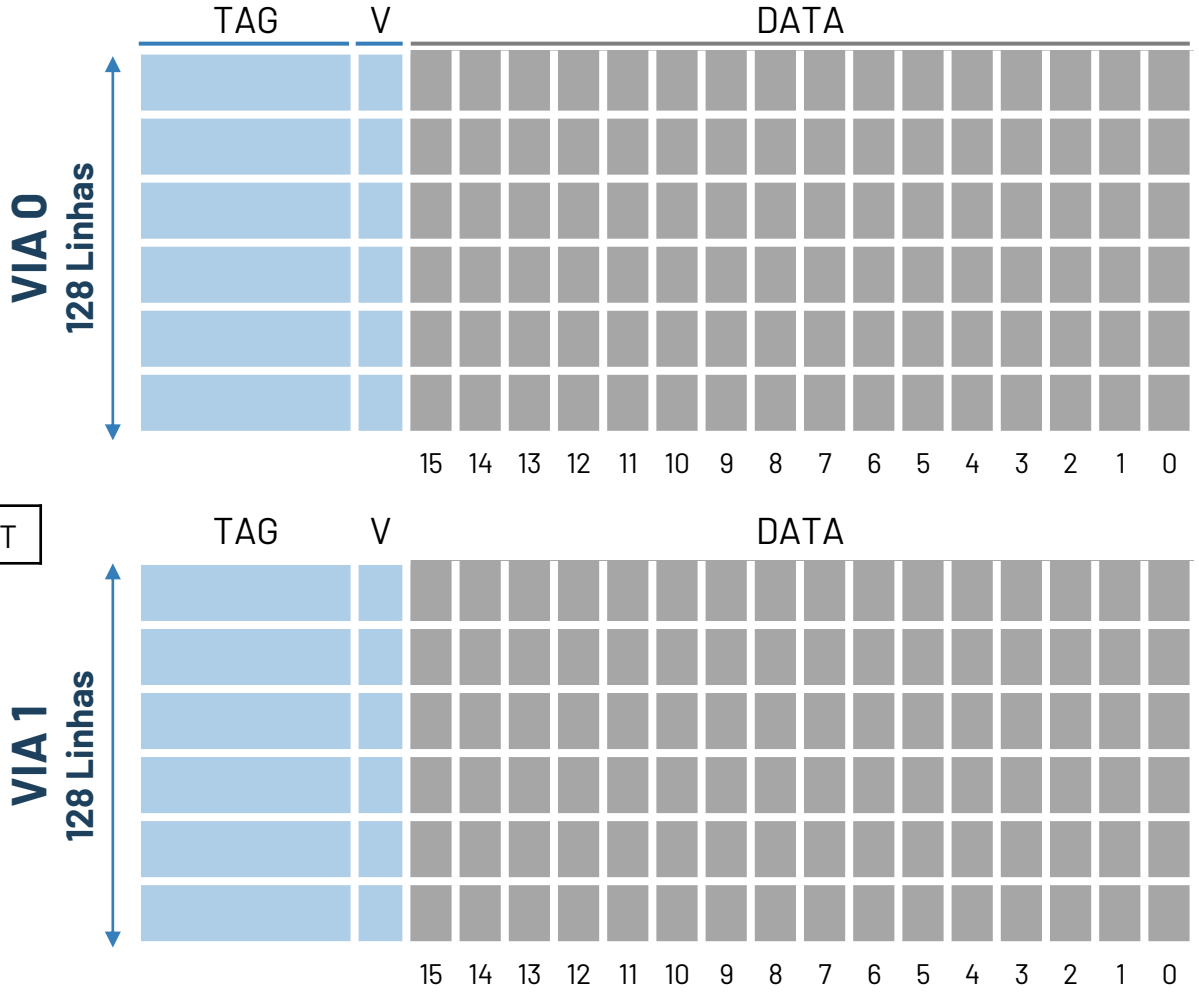
INDEX ➔ 7 bits

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Estrutrura da cache

ADDRESS (32 bits)

TAG	INDEX	OFFSET
21 bits	7 bits	4 bits



Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

3h	7Fh	Ch
21 bits	7 bits	4 bits

Como a cache está inicialmente vazia, i.e., todas as entradas têm V=0, o acesso à entrada 7Fh corresponde necessariamente a um **MISS**!

Vamos preencher a linha...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 1FFCh

Estrutrura da cache

ADDRESS (32 bits)

3h	7F	Ch
21 bits	7 bits	4 bits

Endereço pedido: 1FFCh

MISS....

Ambas as vias estão vazias... escolhemos uma delas

VIA 0

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
7Fh:	3h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

VIA 1

TAG	V	DATA
	0	
	0	
	0	
	0	
	0	
	0	

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

5h	7Fh	Ch
21 bits	7 bits	4 bits

Vamos verificar o estado da cache...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 2FFCh

TÉCNICO LISBOA

Estrutrura da cache

ADDRESS (32 bits)

5h	7F	Ch
21 bits	7 bits	4 bits

Endereço pedido: 2FFCh

MISS....

A via 0 está preenchida, vamos preencher a via 1

VIA 0

	TAG	V	DATA													
		0														
		0														
		0														
		0														
		0														
7Fh:	3h	1	VA[1023]				VA[1022]				VA[1021]				VA[1020]	
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

VIA 1

	TAG	V	DATA													
		0														
		0														
		0														
		0														
		0														
7Fh:	5h	1	VB[1023]				VB[1022]				VB[1021]				VB[1020]	
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

0 0003h	7Fh	8h
21 bits	7 bits	4 bits

Vamos verificar o estado da cache...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
        lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 1FF8h

TÉCNICO LISBOA

Estrutrura da cache

ADDRESS (32 bits)

3h	7F	8h
21 bits	7 bits	4 bits

Endereço pedido: 1FF8h

HIT na via 0

VIA 0

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
7Fh:	3h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

VIA 1

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
7Fh:	5h	1	VB[1023]	VB[1022]	VB[1021]	VB[1020]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

ADDRESS (32 bits)

5h	7Fh	8h
21 bits	7 bits	4 bits

Vamos verificar o estado da cache...

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Endereço 2FF8h

TÉCNICO LISBOA

Estrutrura da cache

ADDRESS (32 bits)

5h	7F	8h
21 bits	7 bits	4 bits

Endereço pedido: 2FF8h

HIT na via 1

VIA 0

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
7Fh:	3h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

VIA 1

	TAG	V	DATA			
		0				
		0				
		0				
		0				
		0				
7Fh:	5h	1	VB[1023]	VB[1022]	VB[1021]	VB[1020]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

Próximos acessos:

ADDRESS (32 bits): 1FF4h

3h	7Fh	4h

HIT na via 0

ADDRESS (32 bits): 2FF4h

5h	7Fh	4h

HIT na via 1

ADDRESS (32 bits): 1FF0h

3h	7Fh	0h

HIT na via 0

ADDRESS (32 bits): 2FF0h

5h	7Fh	0h

HIT na via 1

ADDRESS (32 bits): 1FECh

3h	7Eh	Ch

MISS... Preenche via 0

ADDRESS (32 bits): 2FECh

5h	7Eh	Ch

MISS... Preenche via 1

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop

fim:
```

Estrutrura da cache

VIA 0

	TAG	V	DATA			
		0				
		0				
		0				
		0				
7Eh:	3h	1	VA[1019]	VA[1018]	VA[1017]	VA[1016]
7Fh:	3h	1	VA[1023]	VA[1022]	VA[1021]	VA[1020]
			15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

VIA 1

	TAG	V	DATA			
		0				
		0				
		0				
		0				
7Eh:	5h	1	VB[1019]	VB[1018]	VB[1017]	VB[1016]
7Fh:	5h	1	VB[1023]	VB[1022]	VB[1021]	VB[1020]
			15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- 2 vias (i.e., 2 tabelas)
- Capacidade para 4KB
- Linhas com 16B

Resultado:

Em cada 8 acessos:

- 4 para o array VA
- 4 para o array VB

Ocorrem:

- 2 **MISS** (um para cada array)
- 6 **HIT** (3 para cada array).

Assim:

$$HIT\ RATE = \frac{6}{8} = 75\%$$

```
VA:     .word   <list of values>
VB:     .word   <list of values>

        li      x10,1024 #N
        la      x11,VA
        la      x12,VB
        addi    x13,x10,-1
        sll     x13,x13,2
        add     x14,x11,x13
        add     x15,x12,x13
        mv      x16,x0
        ble     x10,x0,fim
loop:   lw      x17,0(x14)
        lw      X18,0(x15)
        mul     x17,x17,x18
        add     x16,x16,x17
        addi    x14,x14,-4
        addi    x15,x15,-4
        addi    x10,x10,-1
        bgt     x10,x0,loop
fim:
```

Caches de mapeamento associativo

Vantagens e desvantagens do aumento da associatividade

TÉCNICO LISBOA

- Em geral, quanto maior a associatividade, menor é o número de conflitos na cache

  Embora na maioria dos casos esta regra seja verdade, é fácil arranjar exemplos onde uma cache com menos vias (ex: mapeamento direto) tem um maior HIT RATE que uma cache com mais vias (ex: completamente associativa)

- Contudo, o aumento do número de vias geralmente leva a:
  - Maior tempo de acesso (i.e., uma latência maior)
  - Mais recursos de hardware (é necessário lógica para leitura, comparação e seleção dos dados para cada uma das vias)
  - Maior área de silício e maior consumo de potência.

- Geralmente as caches têm um número de vias limitado (ex: 2-8 vias)
  - As caches completamente associativas, geralmente tem uma dimensão reduzida (ex: 16 ou 32 entradas)

Caches de mapeamento associativo (#vias>1)

Política de substituição

- Em caso de MISS, qual das vias deve ser escolhida?
  - Se a linha selecionada estiver livre (V=0) em pelo menos uma via ➔ escolher essa via!

- O que fazer se essa linha já estiver ocupada em <u>todas</u> as vias?

Caches de mapeamento associativo (#vias>1)

Política de substituição

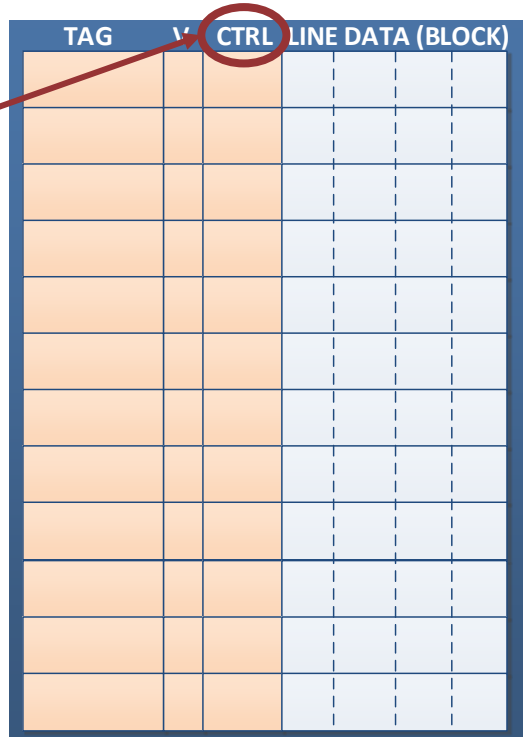
- Existem várias políticas de substituição:
  - **First-In First-Out (FIFO)**

    ➔ Substituir o bloco que está na cache há mais tempo

    Requer a introdução de bits adicionais de controlo com a ordem de entrada na cache. Por exemplo, para uma cache com 4 vias, podemos usar 2 bits para indicar:

    00 – inserido na cache há menos tempo
    01
    10
    11 – inserido na cache há mais tempo (<u>linha a substituir</u>)

    Sempre que acedemos a um bloco de uma via, teremos de alterar em todas as vias (i.e., vias 0,1,2,3), a ordem de entrada na cache



Caches de mapeamento associativo (#vias>1)

Política de substituição

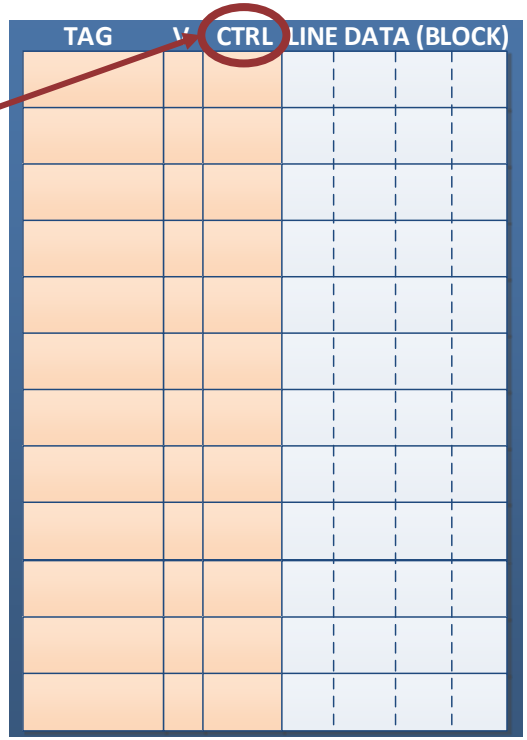
- Existem várias políticas de substituição:
  - **Least Recently Used (LRU)**

    ➔ Substituir o bloco que está há mais tempo sem ser acedido

    Requer a introdução de bits adicionais de controlo com a ordem de acesso. Por exemplo, para uma cache com 4 vias, podemos usar 2 bits para indicar:

    00 – usado há menos tempo
    01
    10
    11 – usado há mais tempo (<u>linha a substituir</u>)

    Sempre que acedemos a um bloco de uma via, teremos de alterar em todas as vias (i.e., vias 0,1,2,3), a ordem de acesso



Caches de mapeamento associativo (#vias>1)

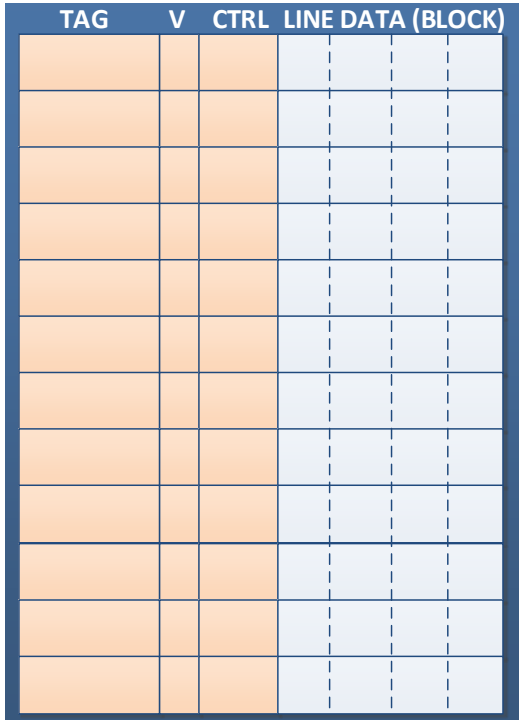
Política de substituição

- Existem várias políticas de substituição:
  - **Random replacement**

    ➔ Escolher uma via ao acaso

    Teoricamente simples e em geral não é significativamente pior que os outros dois.

    Não requer manter qualquer estado adicional (i.e., não obriga há existência de mais bits de controlo)



Caches de mapeamento associativo (#vias>1)

Política de substituição

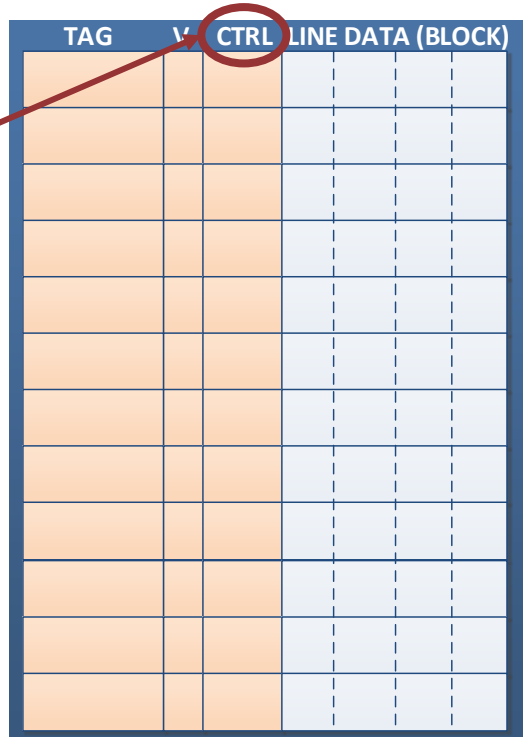
- Existem várias políticas de substituição:
  - **Pseudo-Least Recently Used (Pseudo-LRU)**

    ➔ **Não** substituir o bloco que foi acedido há menos tempo

    Requer a introdução um único bit adicional para indicar qual a via que contém o bloco acedido há menos tempo:

    1 – último a ser usado
    0- outros

    Para #vias>2, esta política apenas indica qual a via cuja linha NÃO deve ser substituida. Entre as restantes vias alternativas, escolhemos uma política secundária (ex: random, round-robin)



Escrita de dados

Política de escrita e de alocação

- Políticas de escrita:
  - **Write-back**: a escrita é realizada na cache
  - **Write-through**: a escrita não é realizada na cache.
    (mas se o dado estiver em cache, este também é atualizado)

- Políticas de alocação:
  - **Write allocate**: uma escrita obriga à alocação dos dados na cache
    ➔ conta para o número de acessos e gera um HIT/MISS
  - **Write not-allocate**: uma escrita nunca leva à alocação dos dados na cache
    ➔ não conta para a contagem do número de acessos, e não gera HIT/MISS

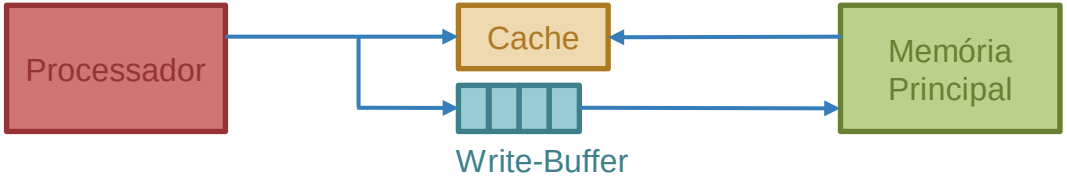
- Combinações típicas:
  - Write-back, write allocate
  - Write-through, write not-allocate

Escrita de dados

Política de escrita e de alocação

- Políticas de escrita:
  - **Write-back**: a escrita é realizada na cache
  - **Write-through**: a escrita não é realizada na cache.
    (mas se o dado estiver em cache, este também é atualizado)

- Para reduzir a latência das escritas, é habitual o uso de um write-buffer.

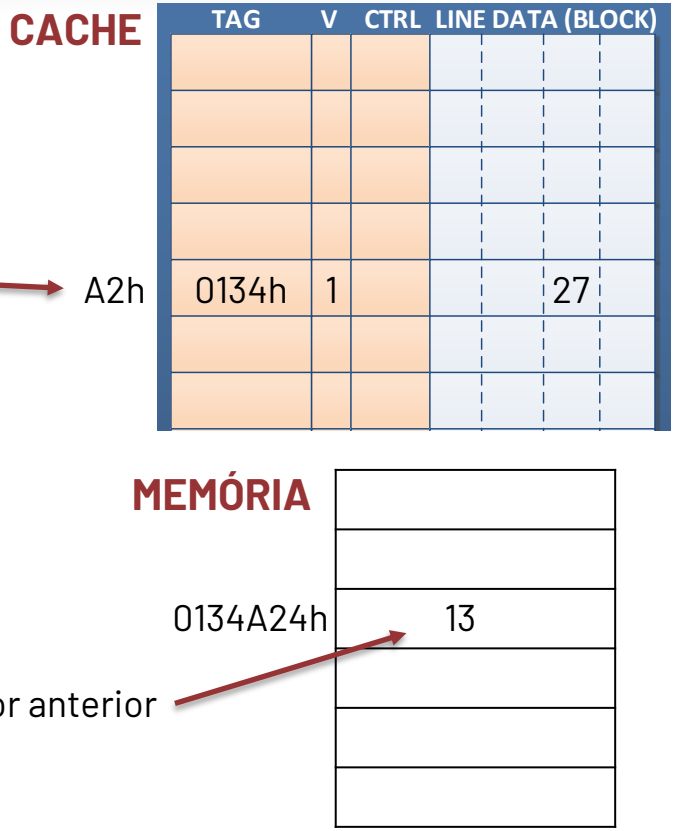


- O write-buffer permite que a cache continue a funcionar enquanto a escrita (demorada...) no nível seguinte da hierarquia de memória é realizado.
- Desta forma, as escritas passam a ser vistas como operações com latência nula.
- Problema: o que acontece se surgir uma instrução de LOAD logo a seguir ao STORE enquanto a escrita ainda está a decorrer?

Escrita de dados

Política de escrita e de alocação

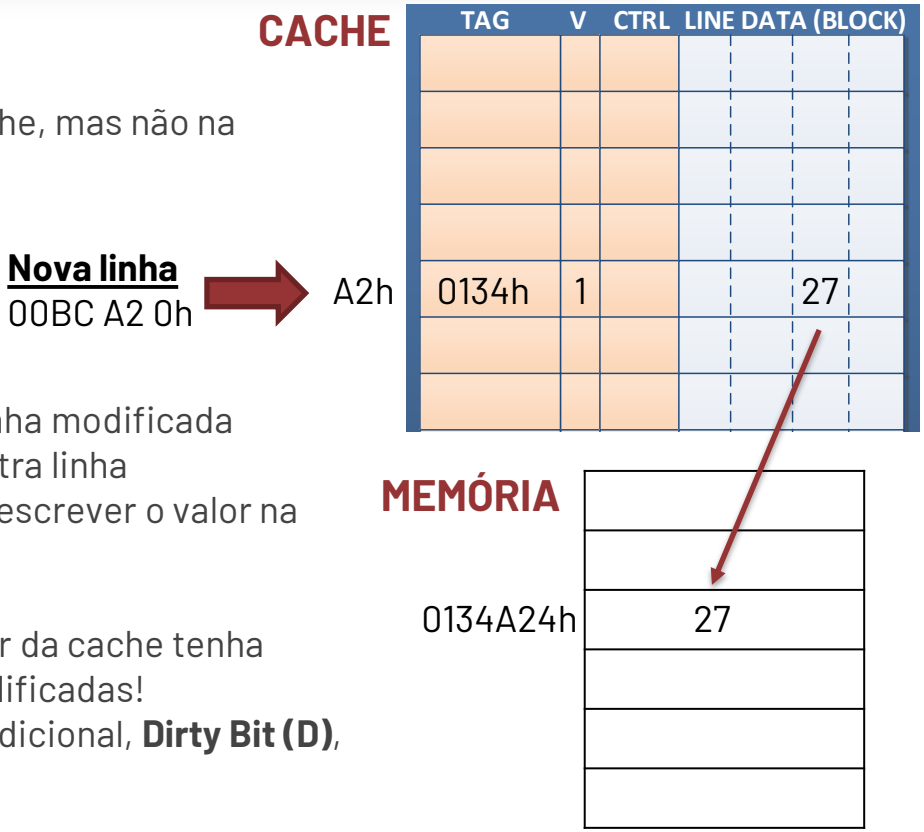
- Políticas de escrita:
  - **Write-back**: a escrita é realizada na cache, mas não na memória principal
  - Ex: sw x5,0(x10)



Escrita de dados

Política de escrita e de alocação

- Políticas de escrita:
  - **Write-back**: a escrita é realizada na cache, mas não na memória principal
  - Ex: sw x5,0(x10)



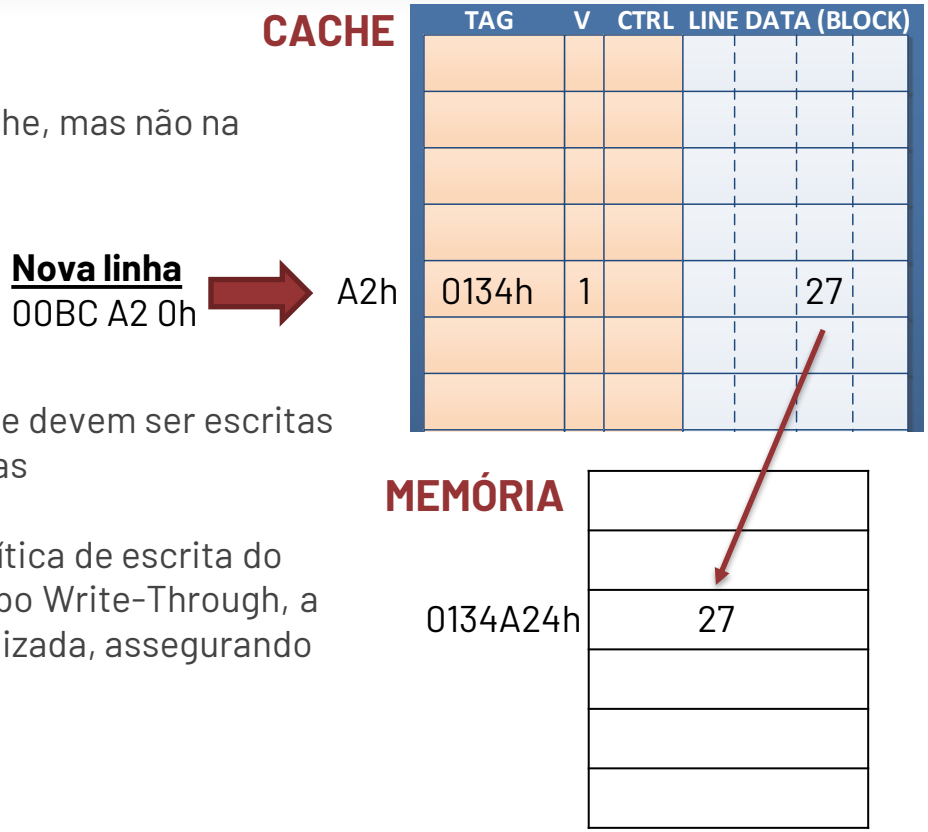
- Substituição:
  - Na sequência de um MISS na cache, a linha modificada anteriormente é substituída por uma outra linha
  - Para garantir a coerência, é necessário escrever o valor na memória antes de alterar a linha!
  - Contudo, é necessário que o controlador da cache tenha informação relativamente às linhas modificadas!
  - **Solução:** Introduzir um bit de controlo adicional, **Dirty Bit (D)**, que indica quais as linhas modificadas.

Escrita de dados

Política de escrita e de alocação

- Políticas de escrita:
  - **Write-back**: a escrita é realizada na cache, mas não na memória principal
  - Ex: sw x5,0(x10)

- **Dirty Bit (D)**:
  - O Dirty Bit (D=1) indica quais as linhas que devem ser escritas em memória antes de serem substituídas
  - O Dirty Bit só e necessário para uma política de escrita do tipo Write-Back. Com uma política do tipo Write-Through, a memória (ou cache seguinte) já foi atualizada, assegurando desde logo a coerência dos dados.



Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

EXERCÍCIO #7

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Admita os seguintes endereços para a localização dos vetores VA, VB e VC:

VA = 1000h; VB = 2000h; VC = 3000h

e ainda as seguintes latências de acessos aos dados:

Latência da L1 = 3 ciclos
Latência da DRAM = 100 ciclos

1. **Determine qual a combinação de política de escrita e alocação de dados mais vantajosa.**

2. **Para cada caso, determine o número ótimo de vias, de forma a minimizar o número de MISSes.**

```
float *A, *B, *C;
int   i;
…
for (i=0; i<1024; i++)
  C[i] = B[i]*(i>0?B[i-1]:1.0) + A[i];
```

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Linhas com 16B ➔ offset de 4 bits

Capacidade para 4 KB ➔ 256 linhas

Número máximo de bits para índex: 8
(cache de mapeamento direto)

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

Sequência de acessos:

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	100 0	200 0	-	300 0
1	100 4	200 4	200 0	300 4
2	100 8	200 8	200 4	300 8
3	100 C	200 C	200 8	300 C
4	101 0	201 0	200 C	301 0
5	101 4	201 4	201 0	301 4
6	101 8	201 8	201 4	301 8
7	101 C	201 C	201 8	301 C
8	102 0	202 0	201 C	302 0
9	102 4	202 4	200 0	302 4
10	102 8	202 8	200 4	302 8
11	102 C	202 C	200 8	302 C
…	…	…	…	…

```
float *A, *B, *C;
int   i;
…
for (i=0; i<1024; i++)
  C[i] = B[i]*(i>0?B[i-1]:1.0) + A[i];
```

```
A:      .float    <list of values>
B:      .float    <list of values>
C:      .zero     4*1024

        li        x10,1024      #N
        la        x11,VA
        la        x12,VB
        la        x13,VC
        mv        x5,x0         #i=0
        addi      x14,x0,1
        fcvt.s.w  f0,x14       #f0=1.0
        bgt       x5,x10,end    #i>N?
loop:   flw       f1,0(x11)     #A[i]
        flw       f2,0(x12)     #B[i]
        fmv.s     f3,f0
        blt       x5,x0,skip    #i<=0?
        flw       f3,-4(x12)
skip:   fmul.s    f2,f2,f3
        fadd.s    f1,f2,f1
        fsw       f1,0(x13)     #C[i]=
        addi      x11,x11,4
        addi      x12,x12,4
        addi      x13,x13,4
        addi      x5,x5,1       #i++
        blt       x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

Sequência de acessos:

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	1 00 0	2 00 0	-	3 00 0
1	1 00 4	2 00 4	2 00 0	3 00 4
2	1 00 8	2 00 8	2 00 4	3 00 8
3	1 00 C	2 00 C	2 00 8	3 00 C
4	1 01 0	2 01 0	2 00 C	3 01 0
5	1 01 4	2 01 4	2 01 0	3 01 4
6	1 01 8	2 01 8	2 01 4	3 01 8
7	1 01 C	2 01 C	2 01 8	3 01 C
8	1 02 0	2 02 0	2 01 C	3 02 0
9	1 02 4	2 02 4	2 00 0	3 02 4
10	1 02 8	2 02 8	2 00 4	3 02 8
11	1 02 C	2 02 C	2 00 8	3 02 C
...	...	...	...	...

Para uma cache de mapeamento direto, todos os acessos numa dada iteração acedem à mesma linha...

```
A:      .float    <list of values>
B:      .float    <list of values>
C:      .zero     4*1024

        li        x10,1024      #N
        la        x11,VA
        la        x12,VB
        la        x13,VC
        mv        x5,x0         #i=0
        addi      x14,x0,1
        fcvt.s.w  f0,x14       #f0=1.0
        bgt       x5,x10,end    #i>N?
loop:   flw       f1,0(x11)     #A[i]
        flw       f2,0(x12)     #B[i]
        fmv.s     f3,f0
        blt       x5,x0,skip    #i<=0?
        flw       f3,-4(x12)
skip:   fmul.s    f2,f2,f3
        fadd.s    f1,f2,f1
        fsw       f1,0(x13)     #C[i]=
        addi      x11,x11,4
        addi      x12,x12,4
        addi      x13,x13,4
        addi      x5,x5,1       #i++
        blt       x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

Sequência de acessos:

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	100 0	200 0	-	300 0
1	100 4	200 4	200 0	300 4
2	100 8	200 8	200 4	300 8
3	100 C	200 C	200 8	300 C
4	101 0	201 0	200 C	301 0
5	101 4	201 4	201 0	301 4
6	101 8	201 8	201 4	301 8
7	101 C	201 C	201 8	301 C
8	102 0	202 0	201 C	302 0
9	102 4	202 4	200 0	302 4
10	102 8	202 8	200 4	302 8
11	102 C	202 C	200 8	302 C
...	...	...	...	...

Para evitar conflitos, são precisas pelo menos 3 vias. Como 256 linhas não podem ser distribuídas em 3 vias, são precisas pelo menos 4 vias.

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

Resultado dos acessos

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	100 0 M	200 0 M	-	300 0 M
1	100 4 H	200 4 H	200 0 H	300 4 H
2	100 8 H	200 8 H	200 4 H	300 8 H
3	100 C H	200 C H	200 8 H	300 C H
4	101 0 M	201 0 M	200 C H	301 0 M
5	101 4 H	201 4 H	201 0 H	301 4 H
6	101 8 H	201 8 H	201 4 H	301 8 H
7	101 C H	201 C H	201 8 H	301 C H
8	102 0 M	202 0 M	201 C H	302 0 M
9	102 4 H	202 4 H	200 0 H	302 4 H
10	102 8 H	202 8 H	200 4 H	302 8 H
11	102 C H	202 C H	200 8 H	302 C H
...	...	...	...	...

M MISS

H HIT

```
A:      .float    <list of values>
B:      .float    <list of values>
C:      .zero     4*1024

        li        x10,1024      #N
        la        x11,VA
        la        x12,VB
        la        x13,VC
        mv        x5,x0         #i=0
        addi      x14,x0,1
        fcvt.s.w  f0,x14       #f0=1.0
        bgt       x5,x10,end    #i>N?
loop:   flw       f1,0(x11)     #A[i]
        flw       f2,0(x12)     #B[i]
        fmv.s     f3,f0
        blt       x5,x0,skip    #i<=0?
        flw       f3,-4(x12)
skip:   fmul.s    f2,f2,f3
        fadd.s    f1,f2,f1
        fsw       f1,0(x13)     #C[i]=
        addi      x11,x11,4
        addi      x12,x12,4
        addi      x13,x13,4
        addi      x5,x5,1       #i++
        blt       x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

MISS RATE = 768/4092=18.77%

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	100 0 M	200 0 M	-	300 0 M
1	100 4 H	200 4 H	200 0 H	300 4 H
2	100 8 H	200 8 H	200 4 H	300 8 H
3	100 C H	200 C H	200 8 H	300 C H
4	101 0 M	201 0 M	200 C H	301 0 M
5	101 4 H	201 4 H	201 0 H	301 4 H
6	101 8 H	201 8 H	201 4 H	301 8 H
7	101 C H	201 C H	201 8 H	301 C H
8	102 0 M	202 0 M	201 C H	302 0 M
9	102 4 H	202 4 H	200 0 H	302 4 H
10	102 8 H	202 8 H	200 4 H	302 8 H
11	102 C H	202 C H	200 8 H	302 C H
...	...	...	...	...

Loop: 256 grupos de 4 iterações
Grupo #1 (i=0,1,2,3): 3 MISSes, 15 acessos
Grupo #n (i>3): 3 MISSes, 16 acessos

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

MISS RATE = 768/4092=18.77%

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	100 0 M	200 0 M	-	300 0 M
1	100 4 H	200 4 H	200 0 H	300 4 H
2	100 8 H	200 8 H	200 4 H	300 8 H
3	100 C H	200 C H	200 8 H	300 C H
4	101 0 M	201 0 M	200 C H	301 0 M
5	101 4 H	201 4 H	201 0 H	301 4 H
6	101 8 H	201 8 H	201 4 H	301 8 H
7	101 C H	201 C H	201 8 H	301 C H
8	102 0 M	202 0 M	201 C H	302 0 M
9	102 4 H	202 4 H	200 0 H	302 4 H
10	102 8 H	202 8 H	200 4 H	302 8 H
11	102 C H	202 C H	200 8 H	302 C H
...	...	...	...	...

Loop: 256 grupos de 4 iterações
Grupo #1 (i=0,1,2,3): 3 MISSes, 15 acessos
Grupo #n (i>3): 3 MISSes, 16 acessos

Miss Rate = (3+3*255)/(15+16*255) = 18.71%

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```


TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Through, Write Not-Allocate

Sequência de acessos:

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	100 0	200 0	-	-
1	100 4	200 4	200 0	-
2	100 8	200 8	200 4	-
3	100 C	200 C	200 8	-
4	101 0	201 0	200 C	-
5	101 4	201 4	201 0	-
6	101 8	201 8	201 4	-
7	101 C	201 C	201 8	-
8	102 0	202 0	201 C	-
9	102 4	202 4	200 0	-
10	102 8	202 8	200 4	-
11	102 C	202 C	200 8	-
...	...	...	...	-

Como os stores não contam (devido à presença do **write-buffer**), bastam 2 vias

```
A:      .float    <list of values>
B:      .float    <list of values>
C:      .zero     4*1024

        li        x10,1024      #N
        la        x11,VA
        la        x12,VB
        la        x13,VC
        mv        x5,x0         #i=0
        addi      x14,x0,1
        fcvt.s.w  f0,x14       #f0=1.0
        bgt       x5,x10,end    #i>N?
loop:   flw       f1,0(x11)     #A[i]
        flw       f2,0(x12)     #B[i]
        fmv.s     f3,f0
        blt       x5,x0,skip    #i<=0?
        flw       f3,-4(x12)
skip:   fmul.s    f2,f2,f3
        fadd.s    f1,f2,f1
        fsw       f1,0(x13)     #C[i]=
        addi      x11,x11,4
        addi      x12,x12,4
        addi      x13,x13,4
        addi      x5,x5,1       #i++
        blt       x5,x10,loop   #i<N?
end:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Through, Write Not-Allocate

Resultado dos acessos

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	1 00 0 M	2 00 0 M	-	-
1	1 00 4 H	2 00 4 H	2 00 0 H	-
2	1 00 8 H	2 00 8 H	2 00 4 H	-
3	1 00 C H	2 00 C H	2 00 8 H	-
4	1 01 0 M	2 01 0 M	2 00 C H	-
5	1 01 4 H	2 01 4 H	2 01 0 H	-
6	1 01 8 H	2 01 8 H	2 01 4 H	-
7	1 01 C H	2 01 C H	2 01 8 H	-
8	1 02 0 M	2 02 0 M	2 01 C H	-
9	1 02 4 H	2 02 4 H	2 00 0 H	-
10	1 02 8 H	2 02 8 H	2 00 4 H	-
11	1 02 C H	2 02 C H	2 00 8 H	-
...	...	...	...	-

M MISS

H HIT

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Through, Write Not-Allocate

Resultado dos acessos

i	lw A[i]	lw B[i]	lw B[i-1]	sw C[i]
0	1 00 0 M	2 00 0 M	-	-
1	1 00 4 H	2 00 4 H	2 00 0 H	-
2	1 00 8 H	2 00 8 H	2 00 4 H	-
3	1 00 C H	2 00 C H	2 00 8 H	-
4	1 01 0 M	2 01 0 M	2 00 C H	-
5	1 01 4 H	2 01 4 H	2 01 0 H	-
6	1 01 8 H	2 01 8 H	2 01 4 H	-
7	1 01 C H	2 01 C H	2 01 8 H	-
8	1 02 0 M	2 02 0 M	2 01 C H	-
9	1 02 4 H	2 02 4 H	2 00 0 H	-
10	1 02 8 H	2 02 8 H	2 00 4 H	-
11	1 02 C H	2 02 C H	2 00 8 H	-
...	...	...	...	-

Loop: 256 grupos de 4 iterações
Grupo #1 (i=0,1,2,3): 2 MISSes, 11 acessos
Grupo #n: 2 MISSes, 12 acessos

Miss Rate = (2+2*255)/(11+12*255) = 16.67%

```
A:      .float    <list of values>
B:      .float    <list of values>
C:      .zero     4*1024

        li        x10,1024      #N
        la        x11,VA
        la        x12,VB
        la        x13,VC
        mv        x5,x0         #i=0
        addi      x14,x0,1
        fcvt.s.w  f0,x14       #f0=1.0
        bgt       x5,x10,end    #i>N?
loop:   flw       f1,0(x11)     #A[i]
        flw       f2,0(x12)     #B[i]
        fmv.s     f3,f0
        blt       x5,x0,skip    #i<=0?
        flw       f3,-4(x12)
skip:   fmul.s    f2,f2,f3
        fadd.s    f1,f2,f1
        fsw       f1,0(x13)     #C[i]=
        addi      x11,x11,4
        addi      x12,x12,4
        addi      x13,x13,4
        addi      x5,x5,1       #i++
        blt       x5,x10,loop   #i<N?
end:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 4KB
- Linhas com 16B

Write-Back, Write Allocate

MISS RATE ≈ 18.71%

Tempo médio de acesso aos dados:

T = 3 + 0.1871 x 100 = 21.71 ciclos

Latência total de acesso (~#stalls):

Assumindo que sempre que temos miss, vamos ter *stalls* até que o pedido seja satisfeito, mas que os *hits* não geram *stalls*.

Lat_{Total} ≈ 18.71 stalls/acesso x 4095 acessos ≈ 77k ciclos

Write-Through, Write Not-Allocate

MISS RATE ≈ 16.67%

Tempo médio de acesso aos dados:

T = 3 + 0.1667 x 100 = 19.67 ciclos

Latência total de acesso (~#stalls):

Lat_{Total} = 16.67 x 3071 ≈ 51k ciclos

```
A:      .float     <list of values>
B:      .float     <list of values>
C:      .zero      4*1024

        li         x10,1024      #N
        la         x11,VA
        la         x12,VB
        la         x13,VC
        mv         x5,x0         #i=0
        addi       x14,x0,1
        fcvt.s.w   f0,x14       #f0=1.0
        bgt        x5,x10,end    #i>N?
loop:   flw        f1,0(x11)     #A[i]
        flw        f2,0(x12)     #B[i]
        fmv.s      f3,f0
        blt        x5,x0,skip    #i<=0?
        flw        f3,-4(x12)
skip:   fmul.s     f2,f2,f3
        fadd.s     f1,f2,f1
        fsw        f1,0(x13)     #C[i]=
        addi       x11,x11,4
        addi       x12,x12,4
        addi       x13,x13,4
        addi       x5,x5,1       #i++
        blt        x5,x10,loop   #i<N?
end:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 16KB
- Linhas com 32B

EXERCÍCIO #8

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];


for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i];
```

a) Esboce a organização da memória cache, indicando a decomposição dos bits de endereço em TAG, INDEX e OFFSET

b) Indique a taxa de falhas para o troço de código indicado, assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];
```

3 1 2

A[0],B[0],C[0],A[2],B[2],C[2],...

```
for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i];
```

3 1 2

C[0],D[0],C[0],C[2],D[2],C[2],...

a) Esboce a organização da memória cache, indicando a decomposição dos bits de endereço em TAG, INDEX e OFFSET

b) Indique a taxa de falhas para o troço de código indicado, **assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.**

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];


for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i];
```

a) Esboce a organização da memória cache, indicando a decomposição dos bits de endereço em TAG, INDEX e OFFSET

TAG=20 bits, INDEX=6 bits, OFFSET=6 bits

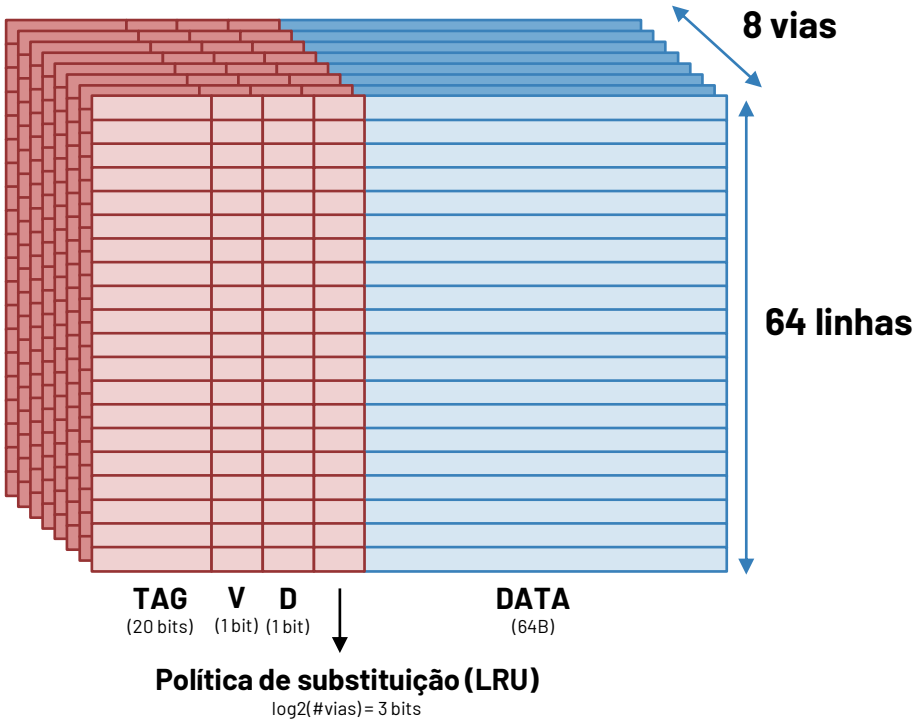
Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

Address

TAG: 20 bits	INDEX: 6 bits	OFFSET: 6 bits

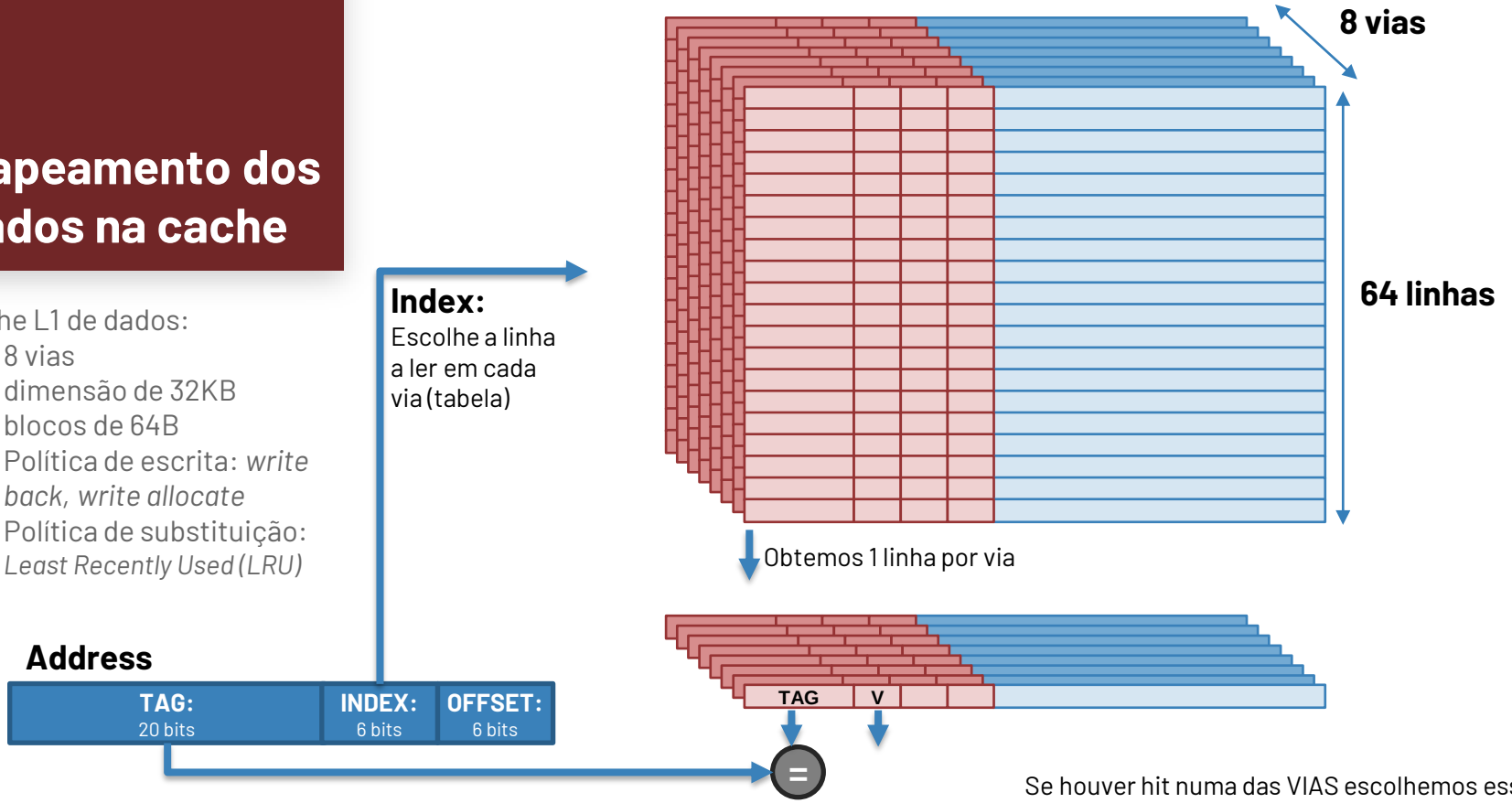
Estrutura da cache:
- 8 vias (tabelas) todas com a mesma estrutura.



Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

Estrutura da cache:
- 8 vias (tabelas) todas com a mesma estrutura.



Comparamos as TAGs de cada linha e verificamos os bits de validade. Usamos tantos comparadores, quanto o número de vias.

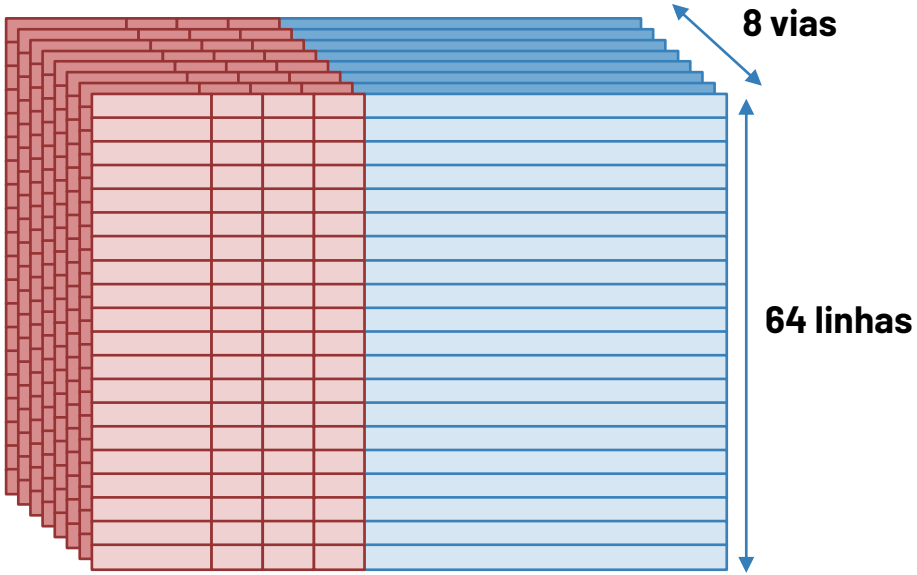
Se houver hit numa das VIAS escolhemos essa via. Caso contrário, vamos pedir a linha correspondente ao nível seguinte (cache/memória)

TÉCNICO LISBOA

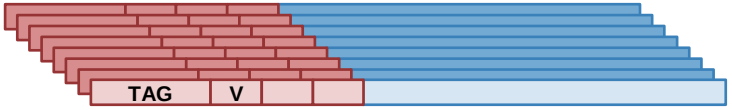
Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

Estrutura da cache:
- 8 vias (tabelas) todas com a mesma estrutura.



Obtemos 1 linha por via



Index: Escolhe a linha a ler em cada via (tabela)

Address

TAG: 20 bits	INDEX: 6 bits	OFFSET: 6 bits

Offset: seleciona o primeiro byte (dentro do bloco de dados) da palavra a ler

Via com HIT

TÉCNICO LISBOA

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];


for ( i=0 ; i<1024 ; i
    C[2*i] = C[2*i] *
```

Começa por aceder a A[0]: linha 0, offset 0. Gera um MISS. Vamos preencher por exemplo a Via 0. Como a linha tem capacidade para 64B, vamos preencher a linha 0 com 16 palavras: A[0],A[1],...,A[15].

Como a variável i é incrementada em 2 em cada iteração do loop, apenas os elementos pares são úteis.

b) Indique a taxa de falhas para o troço de código indicado, assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.

Assumindo que a cache está inicialmente vazia...

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];


for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i];
```

Acedemos a B[0]: linha 0, offset 0. Gera um MISS.

Como a via 0 está ocupada, vamos preencher a via 1 com 16 palavras: B[0],B[1],...,B[15].

b) Indique a taxa de falhas para o troço de código indicado, assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.

Assumindo que a cache está inicialmente vazia...

TÉCNICO LISBOA

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
   C[i] = 2*A[i] + 4*B[i];


for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i]
```

Acedemos a C[0]: linha 0, offset 0. Gera um MISS.

Como as vias 0 e 1 estão ocupadas, vamos preencher a via 2 com 16 palavras: C[0],C[1],...,C[15]. De seguida atualizamos C[0] com o novo valor.

b) Indique a taxa de falhas para o troço de código indicado, assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.

Assumindo que a cache está inicialmente vazia...

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // ende
register int i;       // vari

for ( i=0 ; i<2048 ; i+=2 )
   C[i] = 2*A[i] + 4*B[i];

for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i]
```

2ª a 7ª iteração do loop:
- A[2],...,A[14]: linha 0, offset 0. 7 HITs na via 0
- B[2],...,B[14]: linha 0, offset 0. 7 HITs na via 1
- C[2],...,C[14]: linha 0, offset 0. 7 HITs na via 2

Resultado:
A cada 8 iterações temos:
- 3x1 MISS (uma falha por cada vector)
- 3x7 HITs

NOTA:
- Cada vetor tem 2048 elementos x 4 B = 8KB
- Cada via da cache tem 32KB/8 = 4KB.
- Na via 0 cabem A[0] a A[1023]. Quando chegamos a A[1024] precisamos de ocupar uma nova via.
- A[1024]...A[2047] → Via 3
- B[1024]...B[2047] → Via 4
- C[1024]...C[2047] → Via 5

b) Indique a taxa de falhas para o troço
as variáveis são lidas da memória e e
indicada no código.

Assumindo que a cache está inicialmente vazia...

TÉCNICO LISBOA

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];

for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i];
```

Quando chegamos ao segundo loop temos os vetores A, B e C na cache.

O vetor D vai ocupar as vias 6 e 7. A cada miss temos 7 hits.

O acesso ao vetor C vai dar sempre HIT.

b) Indique a taxa de falhas para o troço de código indicado, assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.

Assumindo que a cache está inicialmente vazia...

Mapeamento dos dados na cache

Cache L1 de dados:
- 8 vias
- dimensão de 32KB
- blocos de 64B
- Política de escrita: *write back, write allocate*
- Política de substituição: *Least Recently Used (LRU)*

```
int A[2048];          // endereço base: 0x0F 0000
int B[2048];          // endereço base: 0x0F 1000
int C[2048];          // endereço base: 0x0F 2000
int D[2048];          // endereço base: 0x0F 3000
register int i;       // variáveis alocadas em registos

for ( i=0 ; i<2048 ; i+=2 )
    C[i] = 2*A[i] + 4*B[i];
```

→ **3x1024 acessos**
3x1024/8 Misses

```
for ( i=0 ; i<1024 ; i+=1 )
    C[2*i] = C[2*i] * D[2*i];
```

→ **3x1024 acessos**
1024/8 Misses (todos no vector D)

b) Indique a taxa de falhas para o troço de código indicado, assumindo que as variáveis são lidas da memória e escritas em memória pela ordem indicada no código.

Taxa de falhas:

$$MISS\ RATE = \frac{3 \times \frac{1024}{8} + \frac{1024}{8}}{3 \times 1024 + 3 \times 1024} = \frac{1}{12} \approx 8.33\%$$

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 32B
- Linhas com 8B
- Política de substituição: *Least Recently Used (LRU)*

EXERCÍCIO #9

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 32B
- Linhas com 8B
- Política de substituição: *Least Recently Used (LRU)*

Compare o desempenho de uma cache de mapeamento direto e uma cache completamente associativa.

Para tal, determine a taxa de sucesso no acesso às <u>instruções</u> para o troço de código indicado para cada um dos casos.

Indique ainda o conteúdo da cache após a primeira execução da instrução "B.GT loop"

```
        A:      .float    <list of values>
        B:      .float    <list of values>
        C:      .zero     4*1024
Address:
00h             li        x10,1024      #N
04h             la        x11,VA
08h             la        x12,VB
0Ch             la        x13,VC
10h             mv        x5,x0         #i=0
14h             addi      x14,x0,1
18h             fcvt.s.w  f0,x14       #f0=1.0
1Ch             bgt       x5,x10,end    #i>N?
20h     loop:   flw       f1,0(x11)     #A[i]
24h             flw       f2,0(x12)     #B[i]
28h             fmv.s     f3,f0
2Ch             ble       x5,x0,skip    #i<=0?
30h             flw       f3,-4(x12)
34h     skip:   fmul.s    f2,f2,f3
38h             fadd.s    f1,f2,f1
3Ch             fsw       f1,0(x13)     #C[i]=
40h             addi      x11,x11,4
44h             addi      x12,x12,4
48h             addi      x13,x13,4
4Ch             addi      x5,x5,1       #i++
50h             blt       x5,x10,loop   #i<N?
54h     end:
```

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 32B → **4 linhas**
- Linhas com 8B → **2 inst./linha**
- Política de substituição: *Least Recently Used (LRU)*

PRIMEIRA ITERAÇÃO

Map. Direto			Completamente Associativa		
RES.	LINHA	TAG	RES.	VIA	TAG
M	0	0	M	0	0h
H	0	0	H	0	0h
M	1	0	M	1	1h
H	1	0	H	1	1h
M	2	0	M	2	2h
H	2	0	H	2	2h
M	3	0	M	3	3h
H	3	0	H	3	3h
M	0	1	M	0	4h
H	0	1	H	0	4h
M	1	1	M	1	5h
H	1	1	H	1	5h
M	2	1	M	2	6h
M	3	1	M	3	7h
H	3	1	H	3	7h
M	0	2	M	0	8h
H	0	2	H	0	8h
M	1	2	M	1	9h
H	1	2	H	1	9h
M	2	2	M	2	Ah

```
A:      .float   <list of values>
B:      .float   <list of values>
C:      .zero    4*1024
```

Address:				
00h		li	x10,1024	#N
04h		la	x11,VA	
08h		la	x12,VB	
0Ch		la	x13,VC	
10h		mv	x5,x0	#i=0
14h		addi	x14,x0,1	
18h		fcvt.s.w	f0,x14	#f0=1.0
1Ch		bgt	x5,x10,end	#i>N?
20h	loop:	flw	f1,0(x11)	#A[i]
24h		flw	f2,0(x12)	#B[i]
28h		fmv.s	f3,f0	
2Ch		blt	x5,x0,skip	#i<=0?
30h		flw	f3,-4(x12)	
34h	skip:	fmul.s	f2,f2,f3	
38h		fadd.s	f1,f2,f1	
3Ch		fsw	f1,0(x13)	#C[i]=
40h		addi	x11,x11,4	
44h		addi	x12,x12,4	
48h		addi	x13,x13,4	
4Ch		addi	x5,x5,1	#i++
50h		blt	x5,x10,loop	#i<N?
54h	end:			

TÉCNICO LISBOA

Estrutrura da cache

Após a 1ª iteração do loop:

Mapeamento direto

Via única:	TAG	V	DATA	
Linha 0:	2h	1	addi x12,x12,4	addi x11,x11,4
Linha 1:	2h	1	addi x5,x5,1	addi x13,x13,4
Linha 2:	2h	1	???	blt x5,x10,loop
Linha 3:	1h	1	fsw f1,0(x13)	fadd.s f1,f2,f1
			7 6 5 4	3 2 1 0

Hits: 9
Misses: 11
Acessos: 20

Completamente associativa

1 linha/via	TAG	V	DATA	
Via 0:	8h	1	addi x12,x12,4	addi x11,x11,4
Via 1:	9h	1	addi x5,x5,1	addi x13,x13,4
Via 2:	Ah	1	???	blt x5,x10,loop
Via 3:	7h	1	fsw f1,0(x13)	fadd.s f1,f2,f1
			7 6 5 4	3 2 1 0

Hits: 9
Misses: 11
Acessos: 20

TÉCNICO LISBOA

Mapeamento dos dados na cache

Considere uma cache L1 para um processador com ISA RV32G

Características da memória cache:

- Capacidade para 32B → **4 linhas**
- Linhas com 8B → **2 inst./linha**
- Política de substituição: *Least Recently Used (LRU)*

SEGUNDA ITERAÇÃO

Map. Direto			Completamente Associativa		
RES.	LINHA	TAG	RES.	VIA	TAG
M	0	1	M	3	4h
H	0	1	H	3	4h
M	1	1	M	0	5h
H	1	1	H	0	5h
M	2	1	M	1	6h
H	2	1	H	1	6h
H	3	1	M	2	7h
H	3	1	H	2	7h
M	0	2	M	3	8h
H	0	2	H	3	8h
M	1	2	M	0	9h
H	1	2	H	0	9h
M	2	2	M	1	Ah

```
A:      .float   <list of values>
B:      .float   <list of values>
C:      .zero    4*1024
```

Address:				
00h		li	x10,1024	#N
04h		la	x11,VA	
08h		la	x12,VB	
0Ch		la	x13,VC	
10h		mv	x5,x0	#i=0
14h		addi	x14,x0,1	
18h		fcvt.s.w	f0,x14	#f0=1.0
1Ch		bgt	x5,x10,end	#i>N?
20h	loop:	flw	f1,0(x11)	#A[i]
24h		flw	f2,0(x12)	#B[i]
28h		fmv.s	f3,f0	
2Ch		blt	x5,x0,skip	#i<=0?
30h		flw	f3,-4(x12)	
34h	skip:	fmul.s	f2,f2,f3	
38h		fadd.s	f1,f2,f1	
3Ch		fsw	f1,0(x13)	#C[i]=
40h		addi	x11,x11,4	
44h		addi	x12,x12,4	
48h		addi	x13,x13,4	
4Ch		addi	x5,x5,1	#i++
50h		blt	x5,x10,loop	#i<N?
54h	end:			

TÉCNICO LISBOA

Estrutrura da cache

Após a 2ª iteração do loop:

Mapeamento direto

Via única:

	TAG	V	DATA	
0:	2h	1	addi x12,x12,4	addi x11,x11,4
1:	2h	1	addi x5,x5,1	addi x13,x13,4
2:	2h	1	???	blt x5,x10,loop
3:	1h	1	fsw f1,0(x13)	fadd.s f1,f2,f1
			7 6 5 4	3 2 1 0

Hits: 16
Misses: 17
Acessos: 33

Completamente associativa

	TAG	V	DATA	
Via 0:	9h	1	addi x5,x5,1	addi x13,x13,4
Via 1:	Ah	1	???	blt x5,x10,loop
Via 2:	7h	1	fsw f1,0(x13)	fadd.s f1,f2,f1
Via 3:	8h	1	addi x12,x12,4	addi x11,x11,4
			7 6 5 4	3 2 1 0

Hits: 15
Misses: 18
Acessos: 33

TÉCNICO LISBOA

Estrutrura da cache

Após a 3ª iteração do loop:

Mapeamento direto

Via única:

	TAG	V	DATA							
0:	2h	1	addi x12,x12,4				addi x11,x11,4			
1:	2h	1	addi x5,x5,1				addi x13,x13,4			
2:	2h	1	???				blt x5,x10,loop			
3:	1h	1	fsw f1,0(x13)				fadd.s f1,f2,f1			
			7	6	5	4	3	2	1	0

Hits: 23
Misses: 23
Acessos: 46

Completamente associativa

	TAG	V	DATA							
Via 0:	Ah	1	???				blt x5,x10,loop			
Via 1:	7h	1	fsw f1,0(x13)				fadd.s f1,f2,f1			
Via 2:	8h	1	addi x12,x12,4				addi x11,x11,4			
Via 3:	9h	1	addi x5,x5,1				addi x13,x13,4			
			7	6	5	4	3	2	1	0

Hits: 21
Misses: 25
Acessos: 46

TÉCNICO LISBOA

Alinhamento das palavras em memória

Impacto no acesso à cache

RISC-V Instruction Set Architecture (ISA)

Dimensão das palavras na memória

...	
000Fh	Byte 15
000Eh	Byte 14
000Dh	Byte 13
000Ch	Byte 12
000Bh	Byte 11
000Ah	Byte 10
0009h	Byte 9
0008h	Byte 8
0007h	Byte 7
0006h	Byte 6
0005h	Byte 5
0004h	Byte 4
0003h	Byte 3
0002h	Byte 2
0001h	Byte 1
0000h	Byte 0

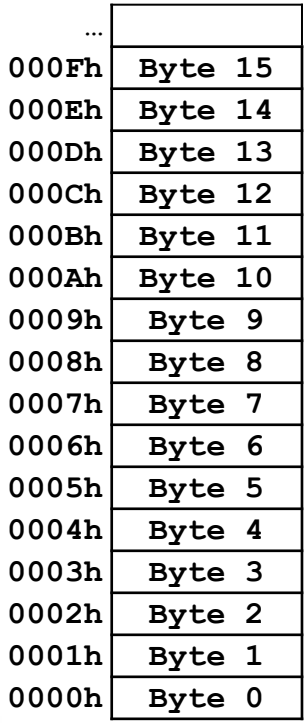
...	
000Fh	Half-Word 7
000Eh	
000Dh	Half-Word 6
000Ch	
000Bh	Half-Word 5
000Ah	
0009h	Half-Word 4
0008h	
0007h	Half-Word 3
0006h	
0005h	Half-Word 2
0004h	
0003h	Half-Word 1
0002h	
0001h	Half-Word 0
0000h	

...	
000Fh	Word 3 / SP FP 3
000Eh	
000Dh	
000Ch	
000Bh	Word 2 / SP FP 2
000Ah	
0009h	
0008h	
0007h	Word 1 / SP FP 1
0006h	
0005h	
0004h	
0003h	Word 0 / SP FP 0
0002h	
0001h	
0000h	

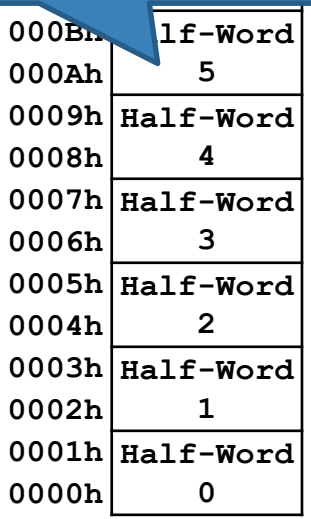
...	
000Fh	Double Word 1 / DP FP 1
000Eh	
000Dh	
000Ch	
000Bh	
000Ah	
0009h	
0008h	
0007h	Double Word 0 / DP FP 0
0006h	
0005h	
0004h	
0003h	
0002h	
0001h	
0000h	

RISC-V Instruction Set Architecture (ISA)

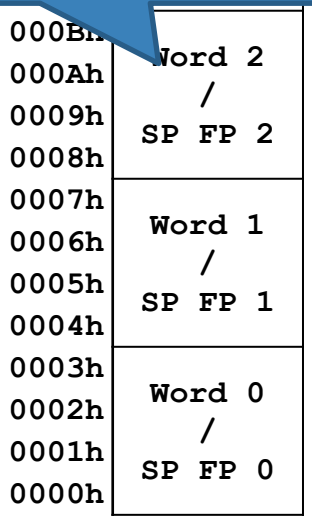
Alinhamento das palavras na memória



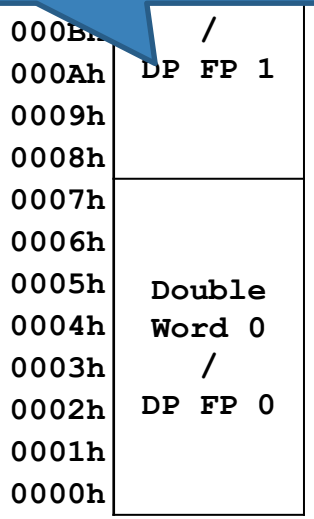
O primeiro byte de cada Half-word é par, i.e., o bit menos significativo do endereço é zero.



O primeiro byte de cada Word é múltiplo de 4, i.e., os 2 bits menos significativos do endereço são zero.



O primeiro byte de cada Double Word é múltiplo de 8, i.e., os 3 bits menos significativos do endereço são zero.



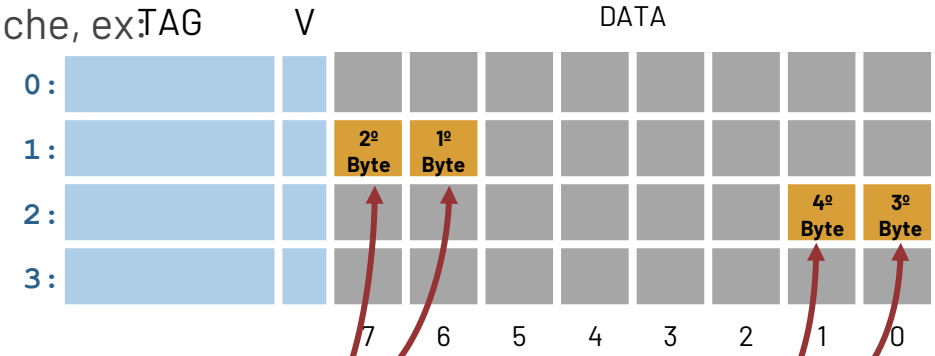
Acesso a variáveis não alinhadas em memória

- Considere um acesso não alinhado à cache, ex:

```
lw      x5,0(X10)
```

Endereço base = 12Eh

4 bytes



Endereço do 1° Byte: 12Eh → TAG = 9, INDEX = 1, OFFSET = 6
Endereço do 2° Byte: 12Fh → TAG = 9, INDEX = 1, OFFSET = 7
Endereço do 3° Byte: 130h → TAG = 9, INDEX = 2, OFFSET = 0
Endereço do 4° Byte: 131h → TAG = 9, INDEX = 2, OFFSET = 1

- Se o endereço não estiver alinhado em memória é possível que a palavra esteja partida em duas linhas da cache!!!!
  - Resulta em dois acessos (cada um pode dar HIT ou MISS) em vez de um único acesso

Otimização de código para a cache

Se eu não quiser desenhar processadores... qual a relevância das caches?

Exemplo de programa

Mesmo programa, diferentes estruturas de dados

Programa A

```
struct DATA{
   int a,b,c,d,e,f,g,h;
};

#define DATA_LEN
64*1024*1024
...

Struct DATA
myData[DATA_LEN];

...

for (i=0;i<DATA_LEN;i++) {
  myData[i].a = myData[i].b;
}
```

Programa B

```
struct DATA{
   int a, b;
};

#define DATA_LEN
64*1024*1024
...

Struct DATA
myData[DATA_LEN];

...

for (i=0;i<DATA_LEN;i++) {
  myData[i].a = myData[i].b;
}
```


TÉCNICO LISBOA

Exemplo de programa

Mesmo programa, diferentes estruturas de dados

Programa A

```
struct DATA{
   int a,b,c,d,e,f,g,h;
};

#define DATA_LEN
64*1024*1024
...

Struct DATA
myData[DATA_LEN];

...

for (i=0;i<DATA_LEN;i++) {
  myData[i].a = myData[i].b;
}
```

Programa B

```
struct DATA{
   int a, b;
};

#define DATA_LEN
64*1024*1024
...

Struct DATA
myData[DATA_LEN];

...

for (i=0;i<DATA_LEN;i++) {
  myData[i].a = myData[i].b;
}
```

Intel i7-6700K @4GHz

176 ms | **142 ms**

Programa A	Programa B
struct DATA {	struct DATA {

Ex
pr

Me
dif
de

O bloco do processador é de 64 bytes, o que corresponde a 16 inteiros de 4B

Linha de cache no programa A: carregamento de 2 estruturas

Linha de cache no programa B: carregamento de 8 estruturas

O MISS RATE no acesso aos dados é:
caso A=1/4=25% caso B=1/16=6,25%

Nota: o impacto na performance não é tão grande como seria de esperar devido à existência de um mecanismo de prefetch, que tenta adivinhar a sequência de acessos e faz pre-load das próximas linhas, e porque o Intel i7-6700K é um processador com execução fora de ordem, o que permite parcialmente esconder a latência dos acessos.

Intel i7-6700K @4GHz **176 ms** **142 ms**

Programa C

```c
#define ROW_SIZE  10*1024
#define COL_SIZE  10*1024

long long myMatrix[ROW_SIZE*COL_SIZE];

...
for (i=0 ; i < ROW_SIZE ; i++) {
  /* for each column */
  for (j=0 ; j < COL_SIZE ; j++) {
      /* for each row */
      myMatrix[i*COL_SIZE + j]++;
  }
}
```

Programa D

```c
#define ROW_SIZE  10*1024
#define COL_SIZE  10*1024

long long myMatrix[ROW_SIZE*COL_SIZE];

...
for (j=0 ; j < COL_SIZE ; j++) {
  /* for each row */
  for (i=0 ; i < ROW_SIZE ; i++) {
    /* for each column */
    myMatrix[i*COL_SIZE + j]++;
  }
}
```

A matriz está organizada em memória guardando dois elementos vizinhos de uma dada linha em endereços adjacentes.

O elemento 11 é armazenado num endereço adjacente ao elemento 10

O elemento 28 é armazenado num endereço adjacente ao elemento 27

O elemento 48 é armazenado num endereço adjacente ao elemento 47

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

A matriz está organizada em memória guardando dois elementos vizinhos de uma dada linha em endereços adjacentes.

O programa é mais eficiente se efectuar as operações por:

[A] Colunas (programa C)

[B] Linhas (programa D)

[C] É indiferente

[D] Isto é tudo muito estranho

Intel i7-6700K @ 4GHz

A matriz está organizada em memória guardando dois elementos vizinhos de uma dada linha em endereços adjacentes.

O programa é mais eficiente se efectuar as operações por:

[A] Colunas (programa C)

Por cada acesso à memória são carregadas para a cache 7 operandos

Miss Rate = 12.5%

Intel i7-6700K @ 4GHz

A matriz está organizada em memória guardando dois elementos vizinhos de uma dada linha em endereços adjacentes.

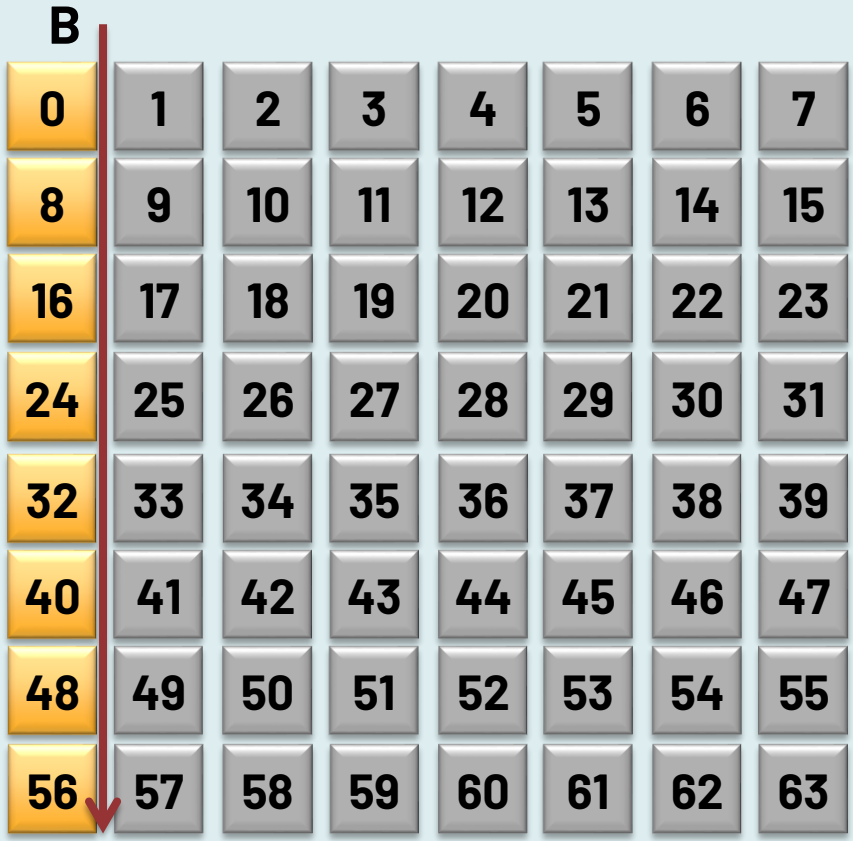
O programa é mais eficiente se efectuar as operações por:

[B] Linhas (programa D)

Por cada acesso à memória são carregadas para a cache 7 operandos

No entanto, como a matriz é de grandes dimensões, estes dados são retirados da cache antes de serem utilizados

Miss Rate = 100%



Intel i7-6700K @ 4GHz

Programa C

```c
#define ROW_SIZE  10*1024
#define COL_SIZE  10*1024

long long myMatrix[ROW_SIZE*COL_SIZE];

...
for (i=0 ; i < ROW_SIZE ; i++) {
  /* for each column */
  for (j=0 ; j < COL_SIZE ; j++) {
      /* for each row */
      myMatrix[i*COL_SIZE + j]++;
  }
}
```

172 ms

Programa D

```c
#define ROW_SIZE  10*1024
#define COL_SIZE  10*1024

long long myMatrix[ROW_SIZE*COL_SIZE];

...
for (j=0 ; j < COL_SIZE ; j++) {
  /* for each row */
  for (i=0 ; i < ROW_SIZE ; i++) {
    /* for each column */
    myMatrix[i*COL_SIZE + j]++;
  }
}
```

967 ms

Intel i7-6700K @ 4GHz

Programa C

```
#define ROW_SIZE  10*1024
#define COL_SIZE  10*1024

long long myMatri

...
for (i=0 ; i < RO
  /* for each col
  for (j=0 ; j <
      /* for each
      myMatrix[i*
  }
}
```

Programa D

```
#define ROW_SIZE  10*1024
#define COL_SIZE  10*1024

_SIZE*COL_SIZE];

E ; j++) {

IZE ; i++) {
 */
E + j]++;
}
```

CONCLUSÃO:

É fundamental que o programador do software tenha consciência dos mecanismos de hardware do processador para que o programa desenvolvido tenha o melhor desempenho possível.

172 ms

967 ms

Intel i7-6700K @ 4GHz