

# 03 - Hierarquia da Memória: Conceitos Fundamentais e Desempenho

Arquitectura de Computadores  
Mestrado Integrado em Engenharia Informática

Luís Paulo Santos

# Material de apoio (mesmo que para 04 – Hierarquia da Memória: Organização)

- *“Computer Organization and Design: The Hardware / Software Interface”*  
David A. Patterson, John L. Hennessy; 5th Edition, 2013
  - Secção 5.1 (pags. 372 .. 378) – Introduction
  - Secção 5.3 + 5.4 (pags. 383 .. 418) – The Basics of Caches + Performance
  - Secção 5.8 (pags. 454 .. 461) – A Common Framework for Memory Hierarchy
  - Secção 5.13 (pags. 471 .. 475) – Real Stuff: ARM Cortex-A8 and Intel Core i7
- *“Computer Systems: a Programmer's Perspective”*; Randal E. Bryant, David R. O'Hallaron--Pearson (2nd ed., 2011)
  - Secção 1.5 + 1.6 (pags. 12 .. 14) – Introduction
  - Capítulo 6 (pag. 560) – Preâmbulo
  - Secção 6.2 .. 6.7 (pags. 586 .. 630)

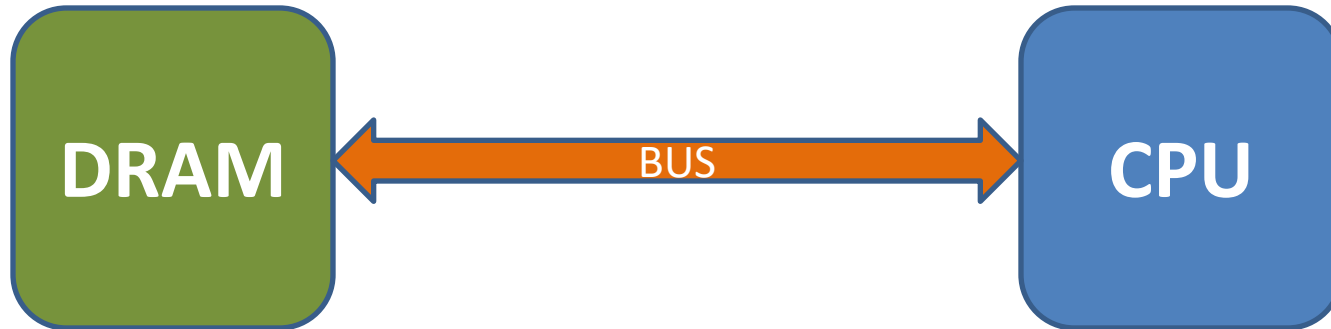
# Hiato Processador-Memória



Para cada instrução:

- 1.Ler instrução
- 2.Ler operando
- 3.Escrever Resultado

# Hiato Processador-Memória



Suponhamos um processador a executar um programa que consiste numa longa sequência de instruções inteiras:

```
addl reg, [Mem]
```

Se a instrução tiver 8 bytes de tamanho e cada inteiro 4 bytes a execução destas instruções implica um movimento de  $8 + 2 * 4 = 16$  bytes.

Se frequência = 2.5 GHz e o CPI=1 então são executadas  $2.5 * 10^9$  inst/seg  
A largura de banda necessária para manter o processador alimentado é de:

$$2.5 * 10^9 * 16 = \mathbf{40 \text{ GB/s}}$$

# Hiato Processador-Memória



Standard name (single channel)	Peak transfer rate
DDR2-400	3.2 GB/s
DDR2-800	6.4 GB/s
DDR3-1066	8.533 GB/s
DDR3-2133	17.067 GB/s
DDR4-1600	12.8 GB/s
DDR4-3200	25.6 GB/s
DDR5-xxxx	Ano 2020 ....

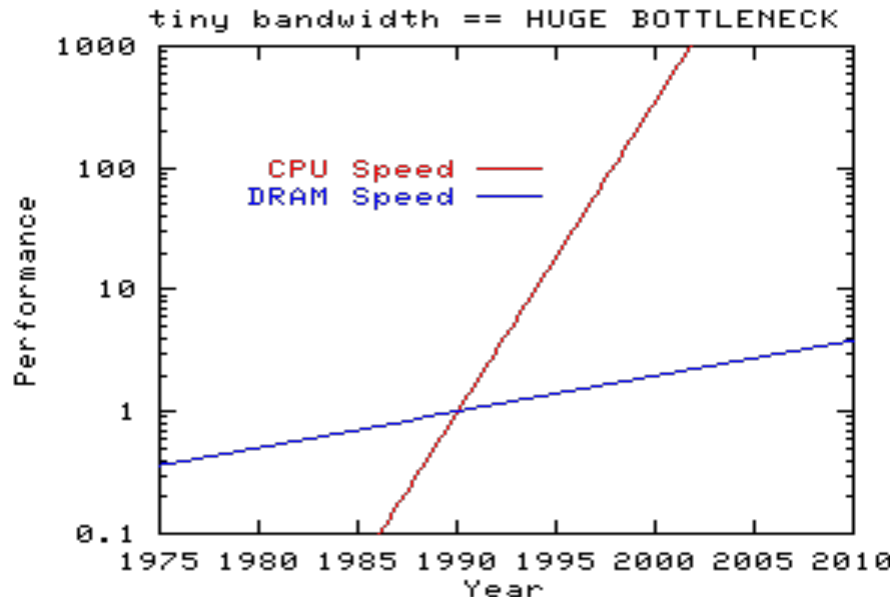
Largura de banda exigida neste exemplo:  $2.5 \cdot 10^9 \cdot 16 = \mathbf{40\ GB/s}$

Hiato processador-memória:

“A memória é incapaz de alimentar o processador com instruções e dados a uma taxa suficiente para o manter constantemente ocupado”

# Hiato Processador-Memória

- O desempenho dos micro-processadores tem vindo a aumentar a uma taxa de cerca de 60% ao ano.
- O desempenho das memórias tem vindo a aumentar a uma taxa de perto de 10% ao ano [1,2]



The STREAM benchmark

<http://www.cs.virginia.edu/stream/ref.html>

[1] "The Processor-Memory bottleneck: Problems and Solutions."; Nihar R. Mahapatra and Balakrishna Venkatrao, ACM  
(<http://www.acm.org/crossroads/xrds5-3/pmgap.html>)

[2] "The Memory Gap and the Future of High Performance Memories"; Maurice V. Wilkes, ACM  
(<http://www.cl.cam.ac.uk/research/dtg/attarchive/pub/docs/URL/tr.2001.4.pdf>)

# Hiato Processador-Memória

- As diferentes taxas de aumento do desempenho destes dois componentes essenciais levam a um aumento do hiato Processador-Memória (*“the memory gap”*) com o tempo
  - Em 1990 um acesso à memória central custava entre 8 a 32 ciclos do relógio
  - Em 2000 custava, numa estação Alpha 21264 667 MHz, cerca de 128 ciclos
  - O custo de cada acesso (medido em ciclos) tende a duplicar cada vez que o desempenho dos processadores duplica, isto é, cada período de [1,5 .. 2] anos
- O hiato processador-memória é um dos principais obstáculos à melhoria do desempenho dos sistemas de computação

# Hiato Processador-Memória

- Dynamic RAM (DRAM)
  - 1 condensador +1 transistor por *bit*
    - (alta densidade -> alta capacidade e baixo custo relativo)
  - Não persistente, *refresh* periódico (-> tempos de acesso elevados)
- Static RAM (SRAM)
  - 6 transistores por *bit* (baixa densidade)
    - (baixa densidade -> menor capacidade e alto custo relativo)
  - Muito persistente: *bistable* (-> tempos de acesso reduzidos)

	Transistors per bit	Relative access time	Persistent?	Sensitive?	Relative cost
SRAM	6	1×	Yes	No	100×
DRAM	1	10×	No	Yes	1×

[Computers Systems: A Programmers' Perspective; Bryant & Hallaron; Pearson, 2nd ed.; 2011]



# Hiato Processador-Memória

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
\$/MB	19,200	2900	320	256	100	75	60	320
Access (ns)	300	150	35	15	3	2	1.5	200

(a) SRAM trends

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
\$/MB	8000	880	100	30	1	.1	0.06	130,000
Access (ns)	375	200	100	70	60	50	40	9
Typical size (MB)	0.064	0.256	4	16	64	2000	8,000	125,000

(b) DRAM trends

Metric	1980	1985	1990	1995	2000	2003	2005	2010	2010:1980
Intel CPU	8080	80286	80386	Pent.	P-III	Pent. 4	Core 2	Core i7	—
Clock rate (MHz)	1	6	20	150	600	3300	2000	2500	2500
Cycle time (ns)	1000	166	50	6	1.6	0.30	0.50	0.4	2500
Cores	1	1	1	1	1	1	2	4	4
Eff. cycle time (ns)	1000	166	50	6	1.6	0.30	0.25	0.10	10,000

(d) CPU trends

[Computers Systems: A Programmers' Perspective; Bryant & Hallaron; Pearson, 2nd ed.; 2011]

# Localidade

## **princípio da localidade:**

*“Os programas bem escritos tendem a aceder a dados que estão próximos (em termos de endereço de memória) de outros dados acedidos recentemente, bem como a referenciar repetidamente os mesmos dados.”*

## **consequência:**

num determinado período de tempo os acessos à memória concentram-se num subconjunto bem localizado do espaço de endereçamento.

O **princípio da localidade** divide-se em 2 componentes:

- **Localidade temporal**
- **Localidade espacial**

# Localidade Temporal

**Localidade Temporal** – um elemento de memória acedido pelo CPU será, com grande probabilidade, acedido de novo num futuro próximo.

**Exemplos:** tanto as instruções dentro dos ciclos, como as variáveis usadas como contadores de ciclos, são acedidas repetidamente em curtos intervalos de tempo.

```
for (i=0 ; i< N ; i++)  
    a[i] += i;
```

Quais os elementos (código e variáveis) deste programa que exibem boa localidade temporal?

# Localidade Espacial

**Localidade Espacial** – se um elemento de memória é acedido pelo CPU, então elementos com endereços na proximidade serão, com grande probabilidade, acedidos num futuro próximo.

**Exemplos:** as instruções são acedidas em sequência, assim como, na maior parte dos programas os elementos dos *arrays*.

```
for (i=0 ; i< N ; i++)  
    a[i] += i;
```

Quais os elementos (código e variáveis) deste programa que exibem boa localidade espacial?

# Localidade

C: os elementos de um vector multidimensional são armazenados *row-wise*

```
int a[3][4];
```

0	4	8	12	16	20	24	28	32	36	40	44
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
for (j=0 ; j< 4 ; j++)  
    for (i=0 ; i< 3 ; i++)  
        a[i][j]++;
```

# Localidade

C: os elementos de um vector multidimensional são armazenados *row-wise*

```
int a[3][4];
```

0	4	8	12	16	20	24	28	32	36	40	44
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
for (i=0 ; i< 3 ; i++)  
    for (j=0 ; j< 4 ; j++)  
        a[i][j]++;
```

# Hierarquia de Memória: Tecnologia e Localidade

- Tecnologia das memórias:
  - Diferentes tecnologias têm tempos de acesso muito diferentes;
  - Tecnologias mais rápidas; menos capacidade e mais caras que as mais lentas
  - O hiato CPU vs. memória tem aumentado e tende a aumentar
- Software
  - Programas bem escritos exibem boa localidade, isto é, tendem a concentrar os seus acessos a um subconjunto do espaço de endereçamento e a aceder repetidamente aos mesmos endereços

## **Hierarquia de Memória**

Dotar a máquina de vários níveis de memória, tão mais rápidos (mais caros e menor capacidade) quanto mais perto se encontram do processador.

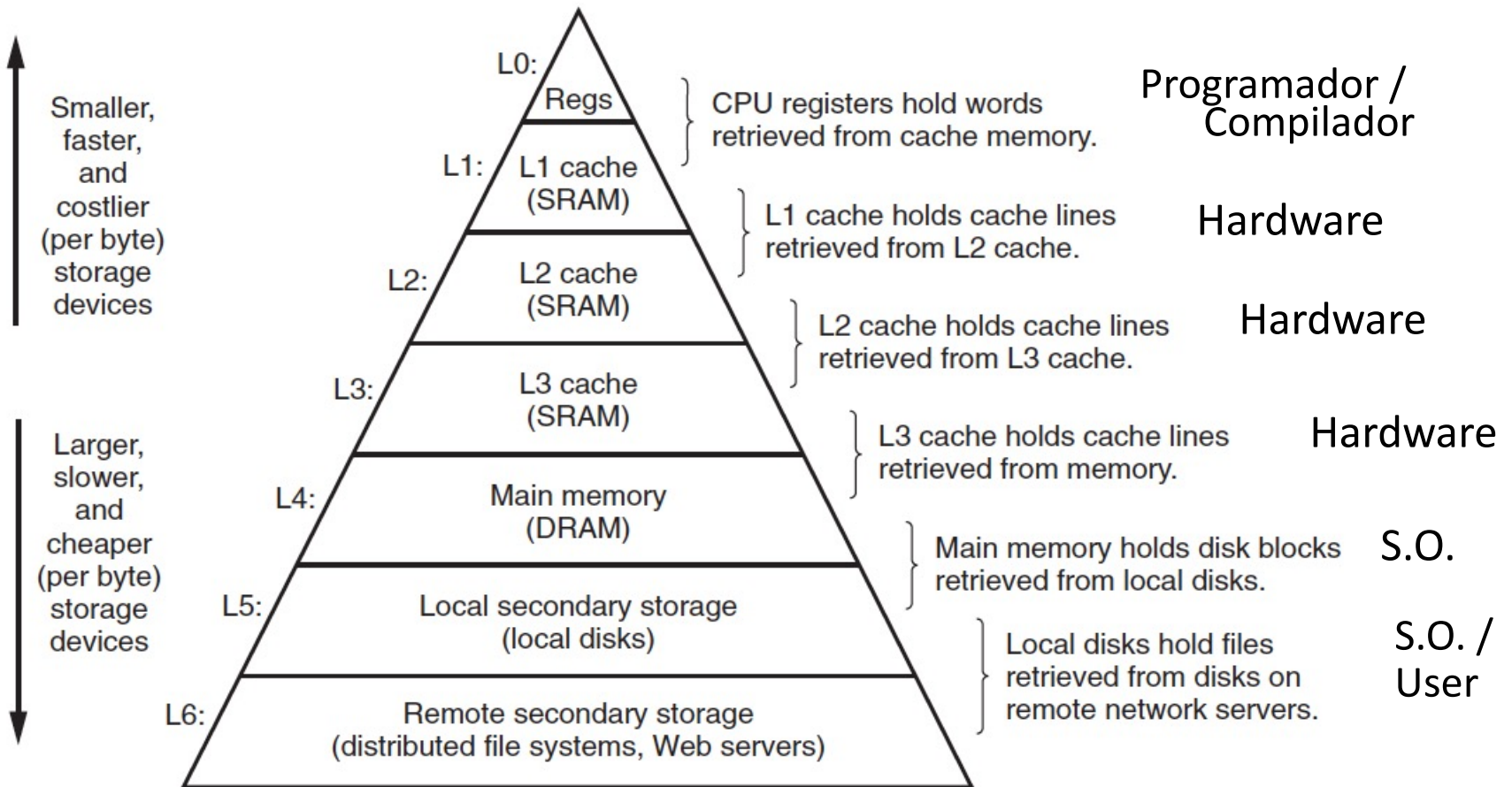
Cada nível contém uma cópia do código e dados mais usados em cada instante, explorando a localidade.

# Hierarquia de Memória

- Abandonamos o modelo de memória linear:  
“a memória é um vector (linear) com um tempo de acesso constante para cada *byte*”
- modelo de memória hierárquico:  
“a memória é uma estrutura hierárquica com um tempo de acesso a cada *byte* variável e dependente da distância a que se encontra do CPU”



# Hierarquia de Memória



[Computers Systems: A Programmers' Perspective; Bryant & Hallaron; Pearson, 2nd ed.; 2011]

# Hierarquia de Memória: Inclusão

Memória Central	
ADDR	DATA
00000000	0xhh
00000001	0xhh
00000010	0xhh
00000011	0xhh
00000100	0xhh
00000101	0xhh
00000110	0xhh
...	...
11111001	0xhh
11111010	0xhh
11111011	0xhh
11111100	0xhh
11111101	0xhh
11111110	0xhh
11111111	0xhh

Os dados contidos num nível da cache mais próximo do processador são um sub-conjunto dos dados contidos no nível anterior.

Encontrou?

Não, procura nível abaixo

Envia para cima

Ler endereço 5

Cache	
TAG	DATA
00000100	0xhh
00000101	0xhh
00000110	0xhh
00000111	0xhh

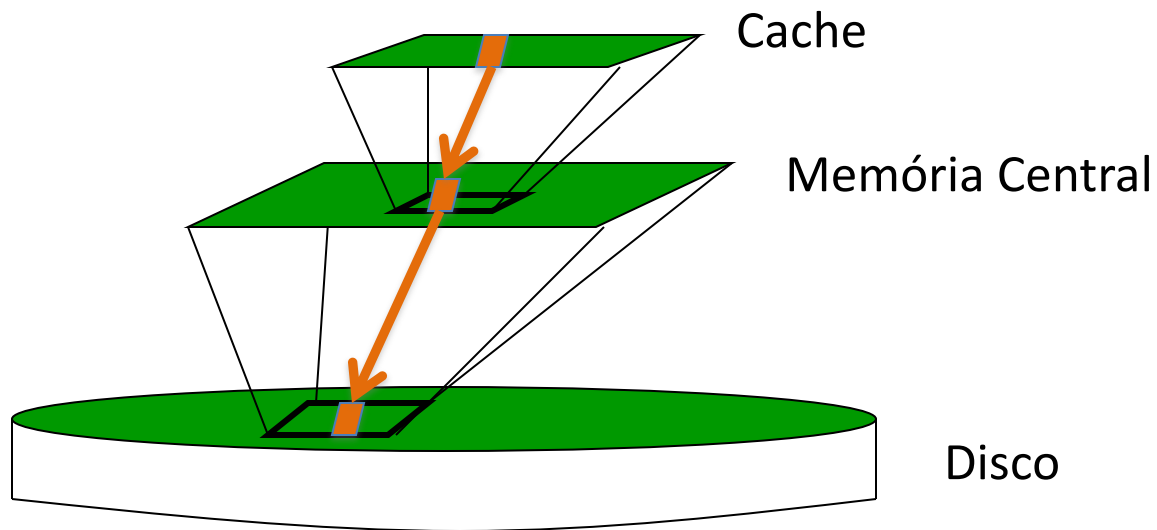
Ler endereço 7

Encontrou?

Sim, envia para cima

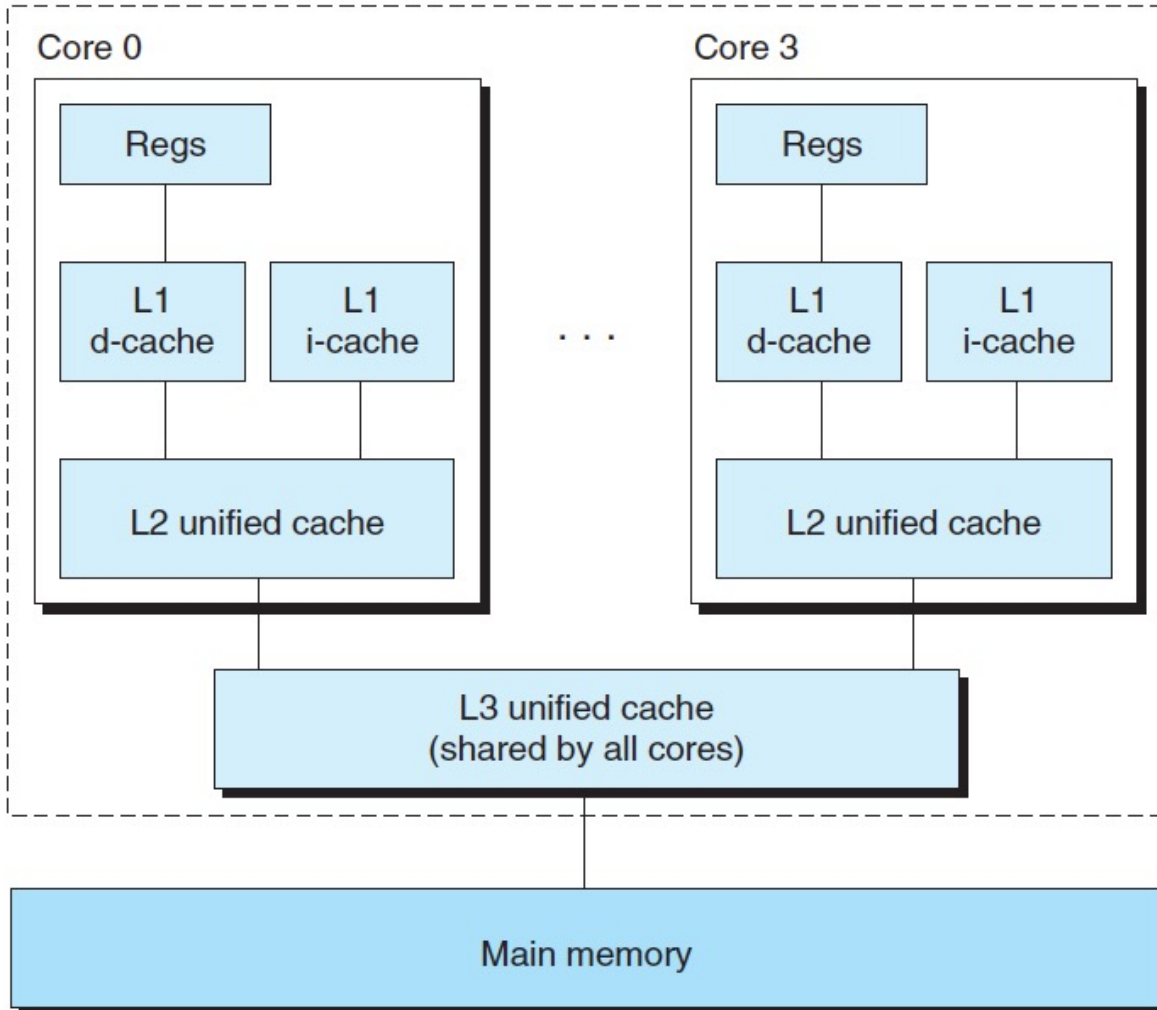
# Hierarquia de Memória: Escrita

Uma escrita num nível superior deve (eventualmente) ser propagada para os níveis inferiores (ver políticas de escrita na *cache*)



# Intel Core i7 : Hierarquia da memória

Processor package



cache	access (cc)	size
L1-d	4	32 KiB
L1-i	4	32 KiB
L2	11	256 KiB
L3	30-40	8 MiB

[Computers Systems: A Programmers' Perspective; Bryant & Hallaron; Pearson, 2nd ed.; 2011]

# Hierarquia de Memória: Terminologia

**Bloco** – Quantidade de informação que é transferida de cada vez do nível imediatamente acima da hierarquia para o nível actual.

**Linha** – a cache está dividida em linhas. Cada linha tem o seu endereço (índice) e tem a capacidade de um bloco

**Hit** – Diz-se que ocorreu um *hit* quando o elemento de memória acedido pelo CPU se encontra na cache.

**Miss** – Diz-se que ocorreu um *miss* quando o elemento de memória acedido pelo CPU não se encontra na cache, sendo necessário lê-lo do nível inferior da hierarquia.

Cache	
	000
	001
	010
	011
	100
	101
	110
	111

# Hierarquia de Memória: Terminologia

**Hit rate** – Percentagem de *hits* ocorridos relativamente ao total de acessos à memória.  $Hit\ rate = \#hits / \#acessos$

**Miss rate** – Percentagem de *misses* ocorridos relativamente ao total de acessos à memória.  $Miss\ rate = (1 - hit\ rate)$

**Hit time** – Tempo necessário para aceder à cache, incluindo o tempo necessário para determinar se o elemento a que o CPU está a aceder se encontra ou não na cache.

**Miss penalty** – Penalização incorrida para aceder a um bloco dos níveis superiores da hierarquia, ocorre um *miss*.

# Hierarquia de Memória e Localidade

- Localidade Temporal

A primeira vez que um endereço de memória é acedido, é carregado do nível de memória inferior para a *cache* – **cold miss**

O próximo acesso a esse endereço encontra os dados na *cache* – **hit**

(excepto se entretanto foram removidos devido a uma **colisão**, resultando nesse caso numa **miss**)

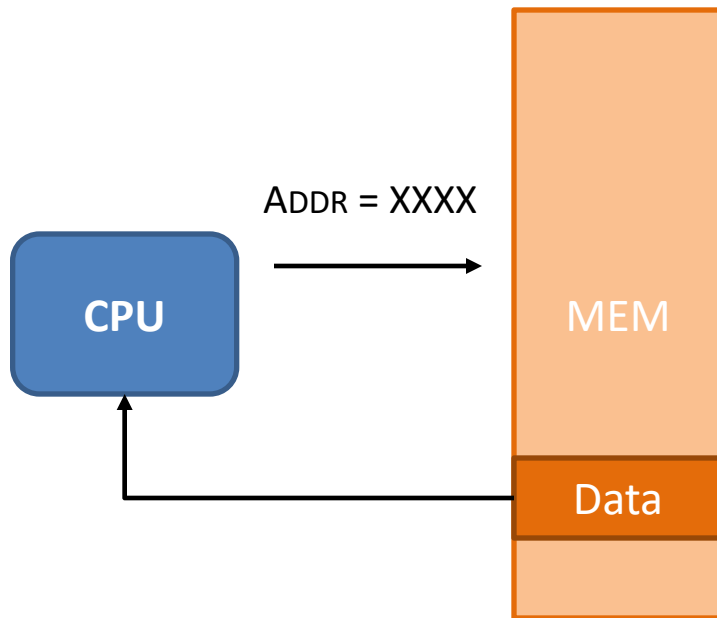
- Localidade Espacial

Quando um endereço é carregado para a *cache*, é carregado um bloco de endereços consecutivos

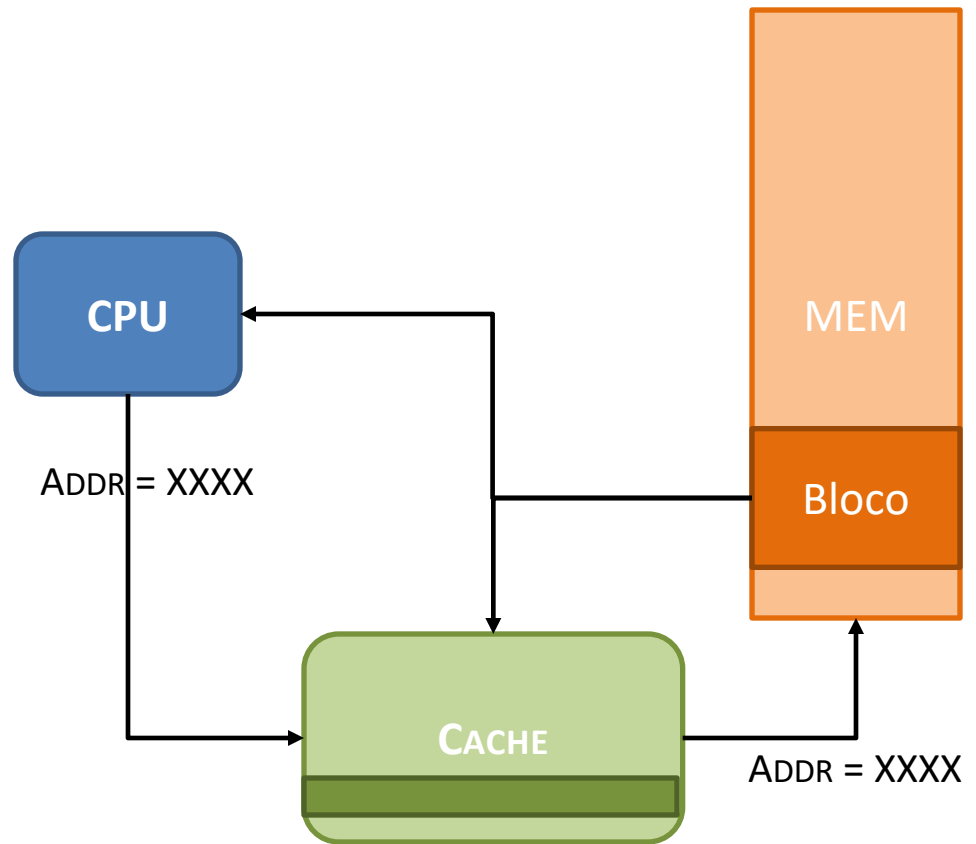
O acesso seguinte a um endereço na vizinhança resulta num **hit**

# Hierarquia de Memória – 1º acesso

$$\text{Penalização} = T_{\text{MEM\_ACCESS}}$$



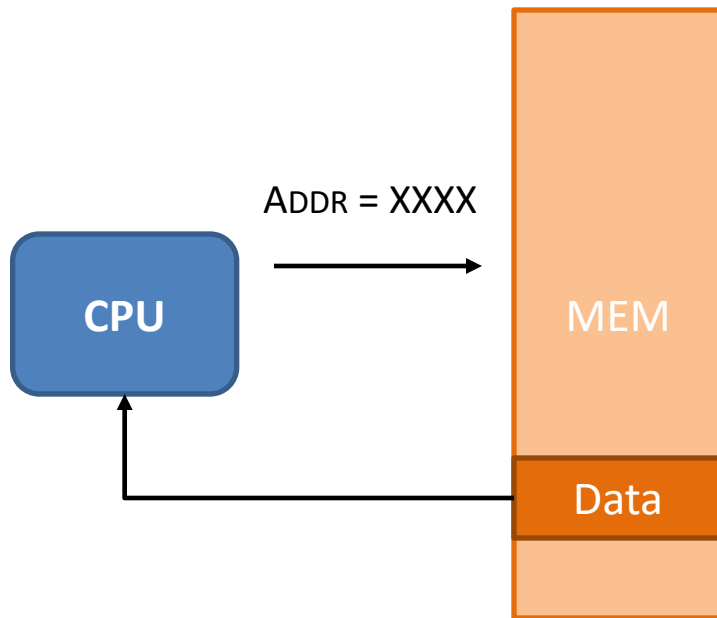
$$\begin{aligned}\text{Penalização} &= \text{HIT\_TIME} + T_{\text{MEM\_ACCESS}} \\ &= \text{HIT\_TIME} + \text{MISS\_PENALTY}\end{aligned}$$



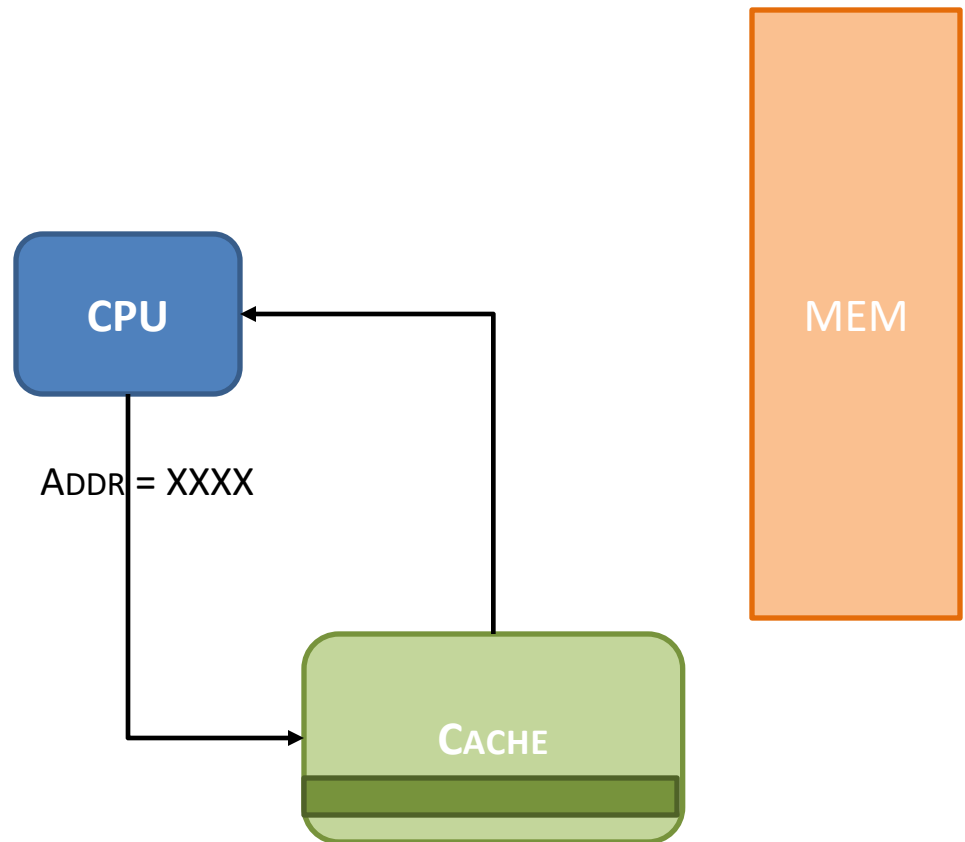


# Hierarquia de Memória – localidade temporal

Penalização =  $T_{\text{MEM\_ACCESS}}$

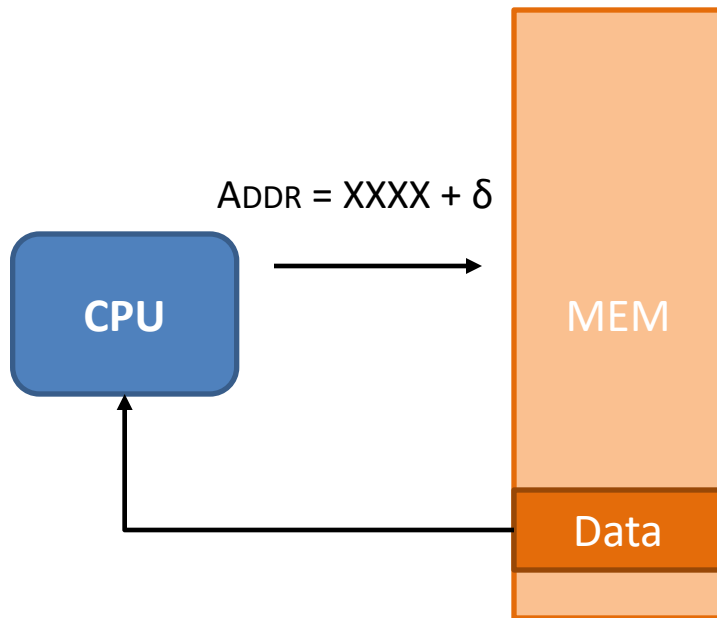


Penalização = HIT\_TIME

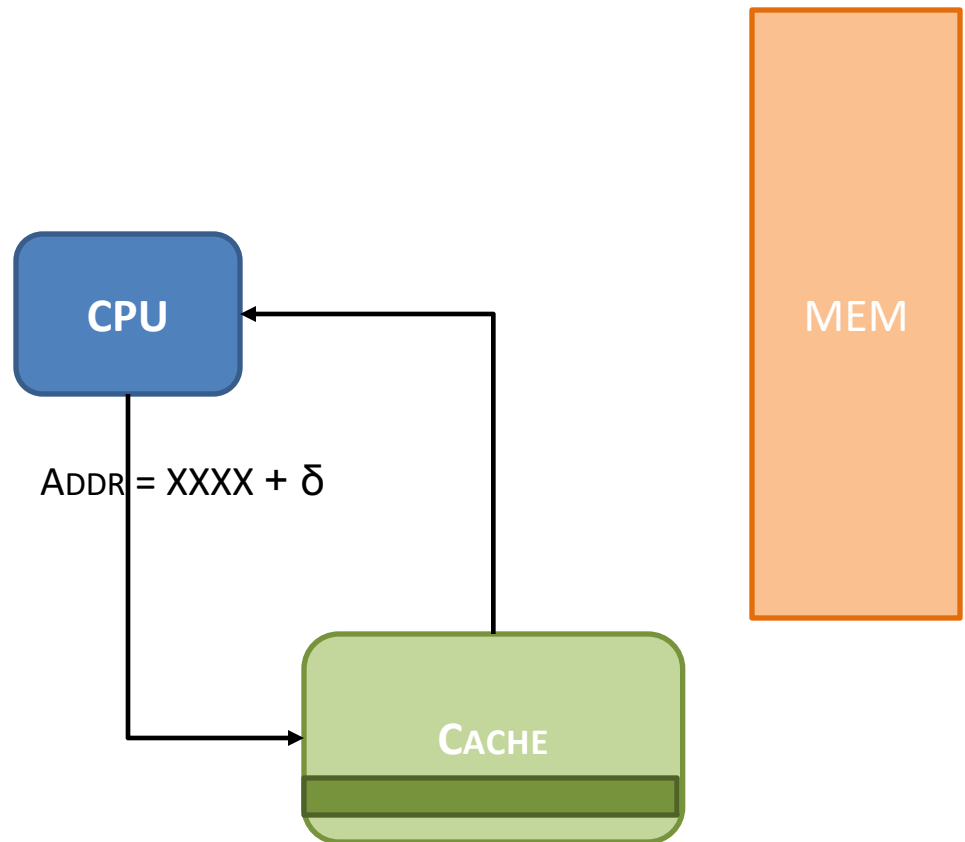


# Hierarquia de Memória – localidade espacial

Penalização =  $T_{\text{MEM\_ACCESS}}$



Penalização = HIT\_TIME



# Hierarquia da memória - Desempenho

$$T_{exec} = \#I * CPI * T_{cc}$$

Como é que a hierarquia de memória influencia  $T_{exec}$ ?

$\#I$  – O número de instruções a executar depende do algoritmo, do conjunto de instruções e do compilador.

$T_{cc}$  – é fixo para cada máquina. Não é alterado modificando a organização da memória.

# Hierarquia da memória - Desempenho

$$T_{exec} = \#I * CPI * T_{cc}$$

$$CPI = CPI_{CPU} + CPI_{MEM}$$

**CPI<sub>CPU</sub>** – nº de ciclos que o processador necessita, em média, para executar cada instrução;

O *hit time* considera-se incluído no CPI<sub>CPU</sub>

**CPI<sub>MEM</sub>** – nº de ciclos que o processador espera, em média, por dados da memória central, por que não encontrou estes dados na cache.

Estes são vulgarmente designados por ***memory stall cycles*** ou ***wait states***.

$$T_{exec} = \#I * (CPI_{CPU} + CPI_{MEM}) * T_{cc}$$

# Hierarquia da memória - Desempenho

$$CPI_{MEM} = CPI_{MEM\_I} + CPI_{MEM\_D}$$

$$CPI_{MEM\_XX} = \%Mem_{XX} * mr_{XX} * mp, XX \in \{I, D\}$$

$$CPI_{MEM} = (\%Mem_I * mr_I + \%Mem_D * mr_D) * mp$$

$\%Mem$  : % acesso à memória

$mr$  : miss rate

$mp$  : miss penalty

**Instruções** – Todas as instruções são lidas da memória, logo a  $\%Mem_I = 100\%$ .

$mr_I$  refere-se ao acesso às instruções.  $< mr_D$  devido à localidade espacial.

**Dados** – Apenas uma determinada percentagem de instruções acede à memória ( $\%Mem_D$ ).  $mr_D$  refere-se ao acesso a dados.

$$CPI_{MEM} = (mr_I + \%Mem_D * mr_D) * mp$$

# Hierarquia da memória - Desempenho

$$T_{exec} = \#I * (CPI_{CPU} + CPI_{MEM}) * T_{cc}$$

$$CPI_{MEM} = (mr_I + \%Mem * mr_D) * mp$$

substituindo

$$T_{exec} = \#I * [CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp] * T_{cc}$$

**NOTA:** A *miss penalty* (*mp*) tem que ser expressa em ciclos do *clock*.

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200

## REGS

IP	500
R1	0
R2	0
R3	0

## CACHE

TAG	W1	W2

#I	0	
#I_MEM	0	
	I	D
MEM	0	0
HITS	0	0
MISSES	0	0

**Executar 1ª  
instrução!**

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200

## REGS

IP	501
R1	10
R2	0
R3	0

## CACHE

TAG	W1	W2
500	I1	I2
1000	10	100

#I	1	
#I_MEM	1	
	I	D
MEM	1	1
HITS	0	0
MISSES	1	1

Executar 2ª  
instrução!



# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200

## REGS

IP	502
R1	10
R2	2
R3	0

## CACHE

TAG	W1	W2
500	I1	I2
1000	10	100

#I	2	
#I_MEM	1	
	I	D
MEM	2	1
HITS	1	0
MISSES	1	1

Executar 3ª  
instrução!

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	503
R1	10
R2	2
R3	200

## CACHE

TAG	W1	W2
500	I1	I2
1000	10	100
502	I3	I4
1002	200	XXX

#I	3	
#I_MEM	2	
	I	D
MEM	3	2
HITS	1	0
MISSSES	2	2

Executar 4<sup>a</sup>  
instrução!

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	504
R1	210
R2	2
R3	200

## CACHE

TAG	W1	W2
500	I1	I2
1000	10	100
502	I3	I4
1002	200	XXX

#I	4	
#I_MEM	2	
	I	D
MEM	4	2
HITS	2	0
MISSES	2	2

**Executar 5<sup>a</sup>  
instrução!**

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	505
R1	210
R2	1
R3	200

## CACHE

TAG	W1	W2
504	15	16
1000	10	100
502	13	14
1002	200	XXX

#I	5	
#I_MEM	2	
	I	D
MEM	5	2
HITS	2	0
MISSES	3	2

Executar 6<sup>a</sup> instrução!

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	502
R1	210
R2	1
R3	200

## CACHE

TAG	W1	W2
504	15	16
1000	10	100
502	13	14
1002	200	XXX

#I	6	
#I_MEM	2	
	I	D
MEM	6	2
HITS	3	0
MISSES	3	2

Executar 7ª  
instrução!

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	503
R1	210
R2	1
R3	100

## CACHE

TAG	W1	W2
504	15	16
1000	10	100
502	13	14
1002	200	XXX

#I	7	
#I_MEM	3	
	I	D
MEM	7	3
HITS	4	1
MISSES	3	2

Executar 8ª  
instrução!

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	504
R1	310
R2	1
R3	100

## CACHE

TAG	W1	W2
504	15	16
1000	10	100
502	13	14
1002	200	XXX

#I	8	
#I_MEM	3	
	I	D
MEM	8	3
HITS	5	1
MISSES	3	2

Executar 9ª  
instrução!

# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	505
R1	310
R2	0
R3	100

## CACHE

TAG	W1	W2
504	15	16
1000	10	100
502	13	14
1002	200	XXX

#I	9	
#I_MEM	3	
	I	D
MEM	9	3
HITS	6	1
MISSES	3	2

Executar 10<sup>a</sup> instrução!



# Hierarquia da memória - Desempenho

## Memória

500	LD [1000], R1
501	MV \$2, R2
502	LD [1000+R2], R3
503	ADD R3, R1
504	SUB \$1, R2
505	JNZ 502

1000	10
1001	100
1002	200
	XXX

## REGS

IP	506
R1	310
R2	0
R3	100

## CACHE

TAG	W1	W2
504	I5	I6
1000	10	100
502	I3	I4
1002	200	XXX

#I	10	
#I_MEM	3	
	I	D
MEM	10	3
HITS	7	1
MISSES	3	2

# Hierarquia da memória - Desempenho

$$CPI_{MEM} = (mr_I + \%Mem * mr_D) * mp$$

$$mr_I = 3 / 10 = 0.3$$

$$mr_D = 2 / 3 = 0.67$$

$$\%Mem = 3 / 10 = 0.3$$

$$CPI_{MEM} = (0.3 + 0.67 * 0.3) * mp = 0.5 * mp$$

#I	10	
#I_MEM	3	
	I	D
MEM	10	3
HITS	7	1
MISSES	3	2

Sem cache:  $mr_I=1$ ,  $mr_D=1$ ,  $\%Mem=0.3$ ,  $CPI_{MEM}=1.3*mp$

# Hierarquia da memória - Desempenho

Considere uma máquina com uma frequência do relógio de 2 GHz, *miss rate* de 4% para instruções, 5% para dados e uma *miss penalty* de 25 ns. Assuma ainda que 40% das instruções são *loads* ou *stores*, e que o  $CPI_{CPU}$  é 1. Qual o CPI?

$$CPI = CPI_{CPU} + CPI_{MEM} = CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp$$

$$mp_{ciclos} = mp_{tempo} * f = 25 * 10^{-9} * 2 * 10^9 = 50 \text{ ciclos}$$

$$CPI = 1 + (0.04 + 0.4 * 0.05) * 50 = 1 + 3 = 4$$

Se o programa executar  $10^9$  instruções qual o tempo de execução?

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 4 * \frac{1}{2 * 10^9} = 2s$$

# Hierarquia da memória - Desempenho

Considere um programa com as características apresentadas na tabela, a executar numa máquina **ideal** com memória de tempo de acesso 0. Se a frequência do processador for 2 GHz, qual o CPI médio e o tempo de execução?

Instrução	Nº Instruções	CPI <sub>CPU</sub>
Cálculo	3*10 <sup>8</sup>	1,1
Acesso à Mem.	6*10 <sup>8</sup>	2,5
Salto	1*10 <sup>8</sup>	1,7
<b>TOTAL:</b>	10 <sup>9</sup>	

$$CPI = CPI_{CPU} + CPI_{MEM} = (3 * 1.1 + 6 * 2.5 + 1 * 1.7) / 10 + 0 = 2$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 2 * \frac{1}{2 * 10^9} = 1s$$

# Hierarquia da memória - Desempenho

Considere o mesmo programa e máquina do acetato anterior, mas agora com um tempo de acesso à memória de 10 ns (por palavra ou instrução). Suponha ainda que esta máquina não tem cache. Qual o CPI efectivo e  $T_{exec}$ ?

$$CPI = CPI_{CPU} + CPI_{MEM} = CPI_{CPU} + (mr_I + \%Mem * mr_D) * mp$$

Se a máquina não tem cache, então  $mr_I = mr_D = 100\%$ .

Da tabela tiramos que  $\%Mem = 60\%$ .

$mp$  expresso em ciclos do relógio é  $10 * 2 = 20$  ciclos ( $f=2$  GHz)

$$CPI = CPI_{CPU} + CPI_{MEM} = 2 + (1 + 0.6 * 1) * 20 = 2 + 32 = 34$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 34 * \frac{1}{2 * 10^9} = 17s$$

# Hierarquia da memória - Desempenho

Considere agora que existe uma *cache* com linhas de 4 palavras; a *miss rate* de acesso às instruções é de 6% e de acesso aos dados é de 10%; o tempo de acesso à memória central é constituído por uma latência de 40 ns mais 10 ns por palavra. Qual o CPI médio e o tempo de execução?

$$mp = 40 + 10 * 4 = 80 \text{ ns} ; \text{ em ciclos } mp = 80 * 2 = 160 \text{ ciclos}$$

$$CPI = CPI_{CPU} + CPI_{MEM} = 2 + (0.06 + 0.6 * 0.1) * 160 = 2 + 19.2 = 21.2$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 21.2 * \frac{1}{2 * 10^9} = 10.6s$$

# Hierarquia da memória - Desempenho

Suponha que a capacidade da *cache* é aumentada para o dobro, ficando a *cache* com o dobro das linhas e resultando numa *miss rate* de acesso às instruções de 3.2% e acesso aos dados de 8%. No entanto, o tempo de acesso à cache (*hit time*) também aumenta, resultando num  $CPI_{CPU}$  de 2.5 . Qual o CPI médio e o tempo de execução?

$$\begin{aligned} CPI &= CPI_{CPU} + CPI_{MEM} = \\ &= 2.5 + (0.032 + 0.6 * 0.08) * 160 = 2.5 + 12.8 = 15.3 \end{aligned}$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 15.3 * \frac{1}{2 * 10^9} = 7.65s$$

# Hierarquia da memória - Desempenho

Para tirar maior partido da localidade espacial aumentou-se o número de palavras por linha de 4 para 8, reduzindo a *miss rate* de instruções para 1% e de dados para 6%. O tempo de acesso à memória central é composto por uma latência de 40 ns mais 10 ns por palavra. Qual o CPI médio e o tempo de execução?

$$mp = 40 + 10 * 8 = 120 \text{ ns} ; \text{ em ciclos } mp = 120 * 2 = 240 \text{ ciclos}$$

$$CPI = CPI_{CPU} + CPI_{MEM} = 2.5 + (0.01 + 0.6 * 0.06) * 240 = 2.5 + 11.04 = 13.52$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 13.52 * \frac{1}{2 * 10^9} = 6.76s$$



# Hierarquia da memória - Desempenho

Para reduzir a *miss penalty* a memória central foi substituída por outra com uma latência de 40 ns e 5 ns por palavra. Qual o CPI médio e o tempo de execução?

$$mp = 40 + 5 * 8 = 80 \text{ ns} ; \text{ em ciclos } mp = 80 * 2 = 160 \text{ ciclos}$$

$$CPI = CPI_{CPU} + CPI_{MEM} = 2.5 + (0.01 + 0.6 * 0.06) * 160 = 2.5 + 7.36 = 9.86$$

$$T_{exec} = \#I * CPI * T_{cc} = 10^9 * 9.86 * \frac{1}{2 * 10^9} = 4.93s$$

# Hierarquia da memória - Desempenho

O processador foi substituído por outro com uma frequência de 3 GHz, sem que a memória tenha sofrido qualquer alteração. Qual o CPI médio e o tempo de execução?

O ciclo do relógio é agora de 0.33 ns, logo  $mp = 80 \cdot 3 = 240$  ciclos

$$CPI = CPI_{CPU} + CPI_{MEM} = 2.5 + (0.01 + 0.6 \cdot 0.06) \cdot 240 = 2.5 + 11.04 = 13.54$$

$$T_{exec} = \#I \cdot CPI \cdot T_{cc} = 10^9 \cdot 13.54 \cdot \frac{1}{3 \cdot 10^9} = 4.513s$$