

# LISTA 4 – INFRAESTRUTURA DE HARDWARE

Aluno: João Marcos Alcântara Vanderley

Professora: Edna Natividade

1)

Endereço da Palavra	Endereço Binário	Tag	Índice	Hit ou Miss
0x03	0000 0011	0	3	M
0xb4	1011 0100	b	4	M
0x2b	0010 1011	2	b	M
0x02	0000 0010	0	2	M
0xbf	1011 1111	b	f	M
0x58	0101 1000	5	g	M
0xbe	1011 1110	b	e	M
0x0e	0000 1110	0	e	M
0xb5	1011 0101	b	5	M
0x2c	0010 1100	2	c	M
0xba	1011 1010	b	a	M
0xfd	1111 1101	f	d	M

Agora vamos checar qual a melhor configuração dentre as três, no caso, qual a que possui a melhor performance.

Temos o seguinte para a Cache 1:

b) Ao montarmos a nossa tabela, obtemos o seguinte:

Endereço da Palavra	Endereço Binário	Tag	Índice	Offset	Hit ou Miss
0x03	0000 0011	0	1	1	M
0xb4	1011 0100	b	2	0	M
0x2b	0010 1011	2	5	1	M
0x02	0000 0010	0	1	0	H
0xbf	1011 1111	b	7	1	M
0x58	0101 1000	5	4	0	M
0xbe	1011 1110	b	6	0	H

Cache 1 miss rate = 100%

Cache 1 total cycles =  $(12 * 25) + (12 * 2) = 324$

Ao analisarmos a Cache 2, chegamos ao seguinte resultado:

Cache 2 miss rate =  $10/12 = 83\%$

Cache 2 total cycles =  $(10 * 25) + (12 * 3) = 286$

Por fim, em relação a Cache 3 temos:

Cache 3 miss rate =  $11/12 = 92\%$

Cache 3 total cycles =  $(11 * 25) + (12 * 5) = 335$

Desse modo podemos observar que a Cache 2 (C2) provêm a melhor performance em comparação com as outras duas caches, logo ela é a melhor configuração dentre as três.

0x0e	0000 1110	0	7	0	M
0xb5	1011 0101	b	2	1	H
0x2c	0010 1100	2	6	0	M
0xba	1011 1010	b	5	0	M
0xfd	1111 1101	f	6	1	M

c) Ao montarmos a nossa tabela, obtemos o seguinte para as três configurações:

			Cache 1	Cache 2	Cache 3
Endereço da Palavra	Endereço Binário	Tag	Índice   Hit ou Miss	Índice   Hit ou Miss	Índice   Hit ou Miss
0x03	0000 0011	0x00	3   M	1   M	0   M
0xb4	1011 0100	0x16	4   M	2   M	1   M
0x2b	0010 1011	0x05	3   M	1   M	0   M
0x02	0000 0010	0x00	2   M	1   M	0   M
0xbf	1011 1111	0x17	7   M	3   M	1   M
0x58	0101 1000	0x0b	0   M	0   M	0   M
0xbe	1011 1110	0x17	6   M	3   H	1   H
0x0e	0000 1110	0x01	6   M	3   M	1   M
0xb5	1011 0101	0x16	5   M	2   H	1   M
0x2c	0010 1100	0x05	4   M	2   M	1   M
0xba	1011 1010	0x17	2   M	1   M	0   M
0xfd	1111 1101	0x1F	5   M	2   M	1   M

2) a)

- Sabemos que a cache contém 64KiB, logo são  $2^{16}$  bytes de dados;

- Sabemos também que cada palavra possui 32 bits, ou seja, 4 bytes.

Ademais, cada

bloco contém 2 palavras, logo teremos 8 bytes por bloco, que são  $2^3$  bytes;

- Logo,  $2^{16} / 2^3 = 2^{13}$  linhas de dados;

- Como cada endereço possui 32 bits, teremos que a divisão será da seguinte forma:

2 bits de offset de palavras

1 bit de bloco de offset ( $2^1$  palavras por bloco)

13 bits de índice (pois são  $2^{13}$  linhas de dados)

16 bits de tag ( $32 - 13 - 2 - 1 = 16$ )

- Logo, o total de bits necessário para implementar essa cache é o seguinte:

$(2^{16} * 4 \text{ bits de dados}) + (2^{13} * 16 \text{ bits de tag}) + (2^{13} * 1 \text{ bit de validade}) =$

$2^{13}(2^3 * 4 + 16 + 1) =$

$2^{13}(32 + 16 + 1) =$

$49 * 2^{13} \text{ bits}$

b)

- Cada palavra tem 4 bytes e cada bloco contém 32 palavras. Logo, cada bloco contém  $2^7$  bytes

- A cache contém 128KiB =  $2^{17}$  bytes de dados. Logo temos  $2^{17} / 2^7 = 2^{10}$  linhas de dados

- Como cada endereço possui 32 bits, teremos que a divisão será da seguinte forma:

2 bits de offset de palavras

5 bit de bloco de offset ( $2^5$  palavras por bloco)

10 bits de índice (pois são  $2^{10}$  linhas de dados)

15 bits de tag ( $32 - 10 - 5 - 2 = 15$ )

- Logo, o total de bits necessário para implementar essa cache é o seguinte:

$(2^{16} * 4 \text{ bits de dados}) + (2^{10} * 15 \text{ bits de tag}) + (2^{10} * 1 \text{ bit de validade}) =$

$2^{10}(2^6 * 4 + 15 + 1) =$

$2^{10}(256 + 16 + 1) =$

$273 * 2^{10} \text{ bits}$

Logo,  $(273 * 2^{10}) / (49 * 2^{13} \text{ bits}) = 0,70$ , então tivemos uma diminuição de 30%

c) O fato do tamanho do bloco ser maior pode ocasionar em uma penalidade de erro de maior tempo assim como um tempo de acerto maior do que o da primeira cache. Logo, isso pode acarretar em um desempenho mais lento do que a primeira cache.

3)

a) Temos 5 bits de offset, se a palavra tem 32 bits, 2 bits do offset deles são bits de palavras e os outros 3 são bits de bloco, o que nos leva a 8 palavras por bloco.

b) Existem 6 bits de índice, sendo  $2^6 = 64$  blocos.

c) Essa cache armazena um total de  $32 * 8$  palavras por bloco \* 8 bytes por palavra = 2048 bytes = 16,384 bits. Cada linha de dados contém 53 bits de tag e 1 bit de validade, logo o número de bits necessários é o seguinte:

$$16,384 + (53 * 64) + (1 * 64) = 19,840 \text{ bits}$$

Dessa forma, a razão entre bits totais necessários para a implementação da cache sobre os bits de armazenamento de dados

é igual a 1.21

d)

Endereço Decimal	Endereço Binário	Tag	Índice	Offset	Hit ou Miss	Slots substituídos
0	0000 0000 0000	0	000 000	00000	M	
4	0000 0000 0100	0	000 000	00100	H	
16	0000 0001 0000	0	000 000	10000	H	
132	0000 1000 0100	0	000 100	00100	M	
232	0000 1110 1000	0	000 111	01000	M	
160	0000 1010 0000	0	000 101	00000	M	
1024	0100 0000 0000	0	100 000	00000	M	
30	0000 0001 1110	0	000 000	11110	H	
140	0000 1000 1100	0	000 100	01100	H	
3100	1100 0001 1100	1	100 000	11100	M	1024
180	0000 1011 0100	0	000 101	10100	H	
2180	1000 1000 0100	1	000 100	00100	M	140 e 132

e)  
A  
razão  
de hits  
é a

seguinte:  $5/12 = 0.42 = 42\%$

f)

Temos o seguinte no estado final da cache:

< index, tag, data >

< 0, 3, Mem[0xC00] – Mem[0xC1F] >

< 4, 2, Mem[0x880] – Mem[0x89f] >

< 5, 0, Mem[0x0A0] – Mem[0x0Bf] >

< 7, 0, Mem[0x0e0] – Mem[0x0ff] >

5)

a)

O CPI será dado por:

$$\text{Miss Penalty} = 100\text{ns}/0.5\text{ns} = 200 \text{ ciclos}$$

$$\text{CPI} = 1.5 + 0.08 * 200 = 17.5$$

b)

$$\text{CPI} = 1.5 + (0.08 * 15) + (0.032 * 200) = 9.1$$

c)

$$\text{CPI} = 1.5 + (0.08 * 32) + (0.016 * 200) = 7.26$$

d)

A nova latência será de 400 ciclos, então os novos CPI's serão dados por:

$$\text{a) CPI} = 1.5 + (0.08 * 400) = 33.5$$

$$\text{b) CPI} = 1.5 + (0.08 * 15) + (0.032 * 400) = 15.50$$

$$\text{c) CPI} = 1.5 + (0.08 * 32) + (0.016 * 400) = 10.46$$

6)

a)

a)

Endereço Decimal	Endereço Hexadecimal	Página Virtual	TLB Hit ou Miss
4669	0x123d	0001	TLB Miss, TP hit, PF
2227	0x08b3	0000	TLB Miss, TP hit
13916	0x365c	0011	TLB hit
34587	0x871b	1000	TLB Miss, TP hit, PF
48870	0xbec6	1011	TLB Miss, TP hit
12608	0x3140	0011	TLB hit
49225	0xc049	1100	TLB Miss, TP Miss

TLB				
-----	--	--	--	--

	Validade	Tag	Endereço Físico	Último Acesso
0x123d	1	0xb	12	5
	1	0x7	4	2
	1	0x3	6	4
	1	0x1	13	0
0x08b3	1	0x0	5	0
	1	0x7	4	3
	1	0x3	6	5
	1	0x1	13	1
0x365c	1	0x0	5	1
	1	0x7	4	4
	1	0x3	6	0
	1	0x1	13	2
0x871b	1	0x0	5	2
	1	0x8	14	0
	1	0x3	6	1
	1	0x1	13	3
0xbee6	1	0x0	5	3
	1	0x8	14	1
	1	0x3	6	2
	1	0xb	12	0
0x3140	1	0x0	5	4
	1	0x8	14	2
	1	0x3	6	0
	1	0xb	12	1
0xc049	1	0xc	15	0
	1	0x8	14	3
	1	0x3	6	1
	1	0xb	12	2

b)

h)

Endereço Decimal	Endereço Hexadecimal	Página Virtual	TLB Hit ou Miss
4669	0x123d	00	TLB Miss, TP hit, PF
2227	0x08b3	00	TLB hit
13916	0x365c	00	TLB hit
34587	0x871b	10	TLB Miss, TP hit, PF
48870	0xbee6	10	TLB hit
12608	0x3140	00	TLB hit
49225	0xc049	11	TLB hit

	Validade	Tag	Endereço Físico	Último Acesso
0x123d	1	0xb	12	5
	1	0x7	4	2
	1	0x3	6	4
	1	0x0	13	0
0x08b3	1	0xb	12	6
	1	0x7	4	3
	1	0x3	6	5
	1	0x0	13	0
0x365c	1	0xb	12	7
	1	0x7	4	4
	1	0x3	6	6
	1	0x0	13	0
0x871b	1	0x2	14	0
	1	0x7	4	5
	1	0x3	6	7
	1	0x0	13	1
0xbec6	1	0x2	14	0
	1	0x7	4	6
	1	0x3	6	8
	1	0x0	13	2
0x3140	1	0x2	14	1
	1	0x7	4	7
	1	0x3	6	9
	1	0x0	13	0
0xc049	1	0x2	14	2
	1	0x7	4	8
	1	0x3	6	0
	1	0x0	13	1

Uma página de tamanho maior leva a uma redução na quantidade de **TLB Miss**, mas pode levar a uma menor utilização da memória física e a uma maior segmentação.

c)

Endereço Decimal	Endereço Hexadecimal	Página Virtual	Índice	TLB Hit ou Miss
4669	0x123d	0001	1	TLB Miss, TP hit, PF
2227	0x08b3	0000	0	TLB Miss, TP hit
13916	0x365c	0011	1	TLB hit
34587	0x871b	1000	0	TLB Miss, TP hit, PF
48870	0xbec6	1011	1	TLB Miss, TP hit
12608	0x3140	0011	1	TLB hit

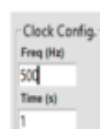
TLB					
	Validade	Tag	Endereço Físico	Índice	Último Acesso
0x123d	1	0xb	12	0	5
	1	0x7	4	1	2
	1	0x3	6	0	4
	1	0x1	13	1	0
0x08b3	1	0x0	5	0	0
	1	0x7	4	1	3
	1	0x3	6	0	5
	1	0x1	13	1	1
0x365c	1	0x0	5	0	1
	1	0x7	4	1	4
	1	0x3	6	0	0
	1	0x1	13	1	2
0x871b	1	0x8	14	0	0
	1	0x7	4	1	5
	1	0x3	6	0	1
	1	0x1	13	1	3
0xbec6	1	0x8	14	0	1
	1	0x7	4	1	6
	1	0x3	6	0	2
	1	0xb	12	1	0
0x3140	1	0x8	14	0	2
	1	0x7	4	1	7
	1	0x3	6	0	0
	1	0xb	12	1	1
0xc049	1	0xc	15	0	0
	1	0x7	4	1	8
	1	0x3	6	0	1
	1	0xb	12	1	2

7) As técnicas de detecção e correção de erros em diferentes níveis de hierarquia de memória visam descobrir se durante a cópia de um dado de um nível para outro, por exemplo, da memória principal para a cache, houve algum erro na transmissão do dado, ou seja, algum bit foi invertido. Isso pode acontecer principalmente, pois se pensarmos na placa mãe, a informação é copiada da memória para cache através de um barramento, e com o aumento da taxa de clock e o aumento de fios nesse barramento pode acontecer a mudança de um bit. Então, é importante que seja detectado se

houve algum erro na transmissão. O código de Hamming com distância 3 fornece correção de erro no caso de 1 erro, e detecção de erros em 2 bits. Para calcular o código de Hamming, temos que começar numerando os bits de 1 começando à esquerda, todas as posições de bits que são uma potência de 2 são bits de paridade (posições 1, 2, 4, 8) e cada bit de paridade verifica certos bits de dados. No final, o valor dos bits de paridade indica quais bits estão com erro, por exemplo, se temos os bits de paridade = 0000, não temos nenhum erro, mas caso tenhamos os bits de paridade = 1010, percebemos que o bit 10 foi invertido. A posição 1 verifica os bits 1, 3, 5, 7, 9; a posição 2 verifica os bits 2, 3, 6, 7, 10, 11; a posição 4 verifica os bits 4, 5, 6, 7, 12; e por fim, a posição 8 verifica os bits 8, 9, 10, 11, 12. E assim, terminamos o processo verificando a paridade do bits de acordo com as posições, e conseguimos descobrir se houve erro.

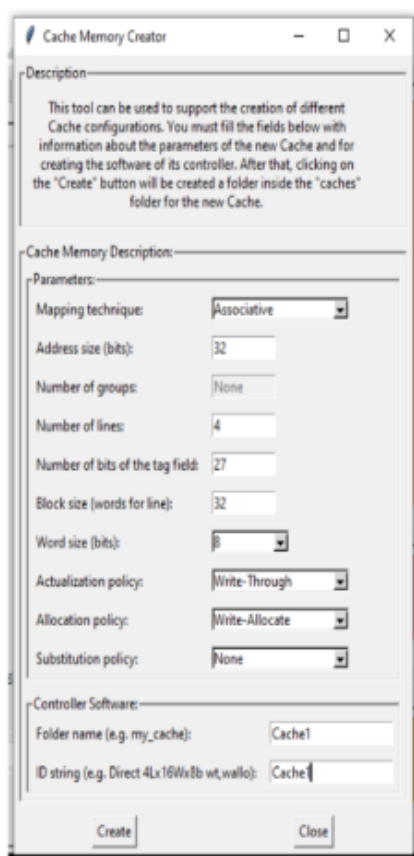
8)

**Caso 1)** Configuração do **clock** utilizada na simulação de ambas as caches:

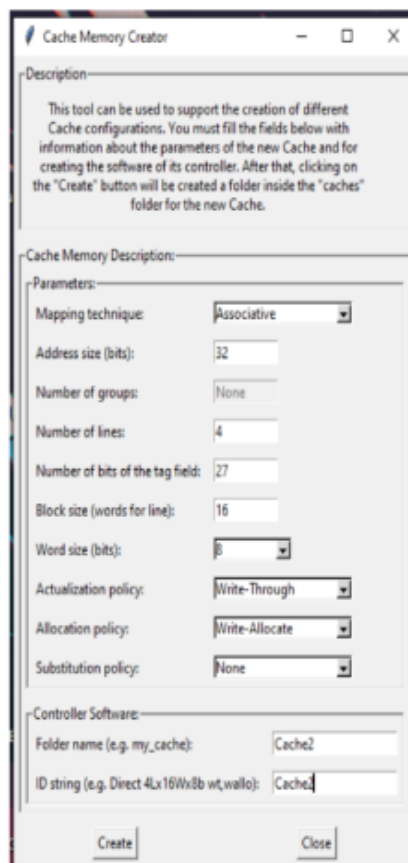


A small dialog box titled "Clock Config." with two input fields. The first field is labeled "Freq (Hz)" and contains the value "500". The second field is labeled "Time (s)" and contains the value "1".

Configuração da **cache 1** (32 palavras) e da **cache 2** (16 palavras):



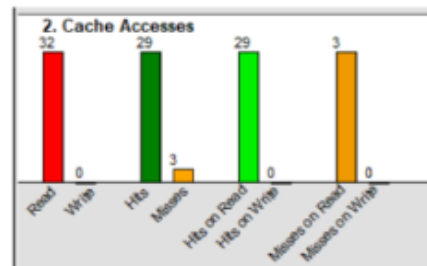
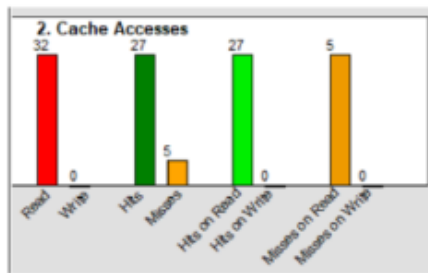
A screenshot of the "Cache Memory Creator" dialog box. The "Description" section contains a paragraph of text. The "Cache Memory Description" section has a "Parameters" group with the following settings: Mapping technique: Associative; Address size (bits): 32; Number of groups: None; Number of lines: 4; Number of bits of the tag field: 27; Block size (words for line): 32; Word size (bits): 8; Actualization policy: Write-Through; Allocation policy: Write-Allocate; Substitution policy: None. The "Controller Software" section has "Folder name (e.g. my\_cache): Cache1" and "ID string (e.g. Direct 4x16Wx8b wt,wallo): Cache1". At the bottom are "Create" and "Close" buttons.



A screenshot of the "Cache Memory Creator" dialog box. The "Description" section contains a paragraph of text. The "Cache Memory Description" section has a "Parameters" group with the following settings: Mapping technique: Associative; Address size (bits): 32; Number of groups: None; Number of lines: 4; Number of bits of the tag field: 27; Block size (words for line): 16; Word size (bits): 8; Actualization policy: Write-Through; Allocation policy: Write-Allocate; Substitution policy: None. The "Controller Software" section has "Folder name (e.g. my\_cache): Cache2" and "ID string (e.g. Direct 4x16Wx8b wt,wallo): Cache2". At the bottom are "Create" and "Close" buttons.

Resultado obtidos para a **cache 1** e **cache 2**, respectivamente:





$$Miss Rate_{cache1} = \frac{5}{32} \approx 15,6\%$$

$$Miss Rate_{cache2} = \frac{3}{32} \approx 9,4\%$$

Como podemos observar, o *Miss Rate* da **cache 1** foi maior do que o da **cache 2**, isso provavelmente se deve ao fato da tamanho da **cache 1** ser maior, o que acaba ocasionando mais facilmente em erros de leitura (o código fornecido na questão só possui instruções de leitura, por isso só vemos **Misses/Hits on Read** presentes nos gráficos acima).

**Caso 2)** Configuração do **clock** utilizada na simulação das três as caches:

Clock Config.

Freq (Hz)

500

Time (s)

1

Configuração da **cache 1** (Associativa por conjunto 2-way) e seu resultado obtido:

Cache Memory Creator

Description:

This tool can be used to support the creation of different Cache configurations. You must fill the fields below with information about the parameters of the new Cache and for creating the software of its controller. After that, clicking on the "Create" button will be created a folder inside the "caches" folder for the new Cache.

Cache Memory Description:

Parameters:

Mapping technique: Associative

Address size (bits): 32

Number of groups: None

Number of lines: 2

Number of bits of the tag field: 29

Block size (words for line): 16

Word size (bits): 8

Actualization policy: Write-Through

Allocation policy: Write-Allocate

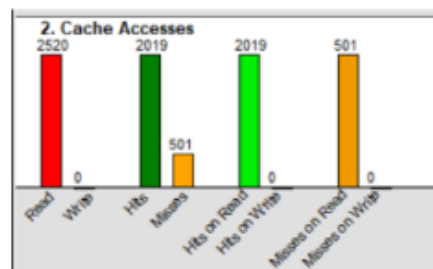
Substitution policy: None

Controller Software:

Folder name (e.g. my\_cache): 4way

ID string (e.g. Direct 4Lx16Wx8b wt,wallo): 2way

Create Close



Configuração da **cache 2** (Mapeamento direto) e seu resultado obtido:

**Cache Memory Creator**

**Description:**

This tool can be used to support the creation of different Cache configurations. You must fill the fields below with information about the parameters of the new Cache and for creating the software of its controller. After that, clicking on the "Create" button will be created a folder inside the "caches" folder for the new Cache.

**Cache Memory Description:**

**Parameters:**

Mapping technique: Direct

Address size (bits): 32

Number of groups: None

Number of lines: 4

Number of bits of the tag field: 27

Block size (words for line): 16

Word size (bits): 8

Actualization policy: Write-Through

Allocation policy: Write-Allocate

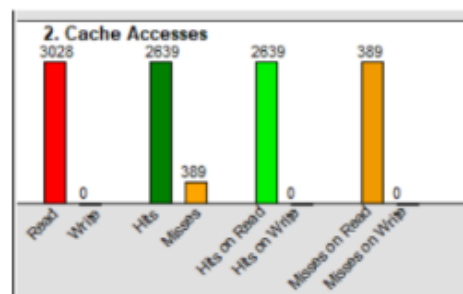
Substitution policy: None

**Controller Software:**

Folder name (e.g. my\_cache): direct

ID string (e.g. Direct 4Lx16Wx8b wt,wallo): Direct 4Lx16Wx8b wt,w

Create Close



Configuração da **cache 3** (Completamente associativo) e seu resultado obtido:

Clock Config.

Freq (Hz)

500

Time (s)

1

Configuração da **cache 1** (32 palavras com blocos de 4 palavras) e seu resultado obtido:

Cache Memory Creator

Description

This tool can be used to support the creation of different Cache configurations. You must fill the fields below with information about the parameters of the new Cache and for creating the software of its controller. After that, clicking on the "Create" button will be created a folder inside the "caches" folder for the new Cache.

Cache Memory Description:

Parameters:

Mapping technique: Direct

Address size (bits): 32

Number of groups: None

Number of lines: 4

Number of bits of the tag field: 27

Block size (words for line): 32

Word size (bits): 8

Actualization policy: Write-Through

Allocation policy: Write-Allocate

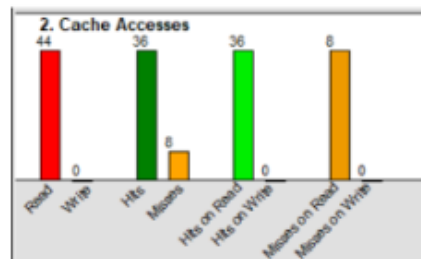
Substitution policy: None

Controller Software:

Folder name (e.g. my\_cache): 32p4bloco

ID string (e.g. Direct 4Lx16Wx8b wt, wallo): 32p4bloco

Create Close



**Cache Memory Creator**

**Description**

This tool can be used to support the creation of different Cache configurations. You must fill the fields below with information about the parameters of the new Cache and for creating the software of its controller. After that, clicking on the "Create" button will be created a folder inside the "caches" folder for the new Cache.

**Cache Memory Description:**

**Parameters:**

Mapping technique: Set Associative

Address size (bits): 32

Number of groups: 2

Number of lines: 2

Number of bits of the tag field: 26

Block size (words for line): 16

Word size (bits): 8

Actualization policy: Write-Through

Allocation policy: Write-Allocate

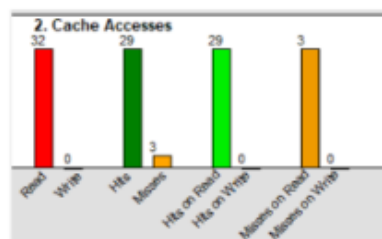
Substitution policy: None

**Controller Software:**

Folder name (e.g. my\_cache): setassociative

ID string (e.g. Direct 4Lx16Wx8b wt,wallo): setassociative

Create Close



$$Miss\ Rate_{cache1} = \frac{501}{2520} \approx 19,9\%$$

$$Miss\ Rate_{cache2} = \frac{389}{3028} \approx 12,8\%$$

$$Miss\ Rate_{cache3} = \frac{3}{32} \approx 9,4\%$$

Podemos ver que a cache mais eficiente é a **cache 3**, a completamente associativa de grau 2.

**Caso 3)** Configuração do **clock** utilizada na simulação das três as caches:

**Cache Memory Creator**

Description:

This tool can be used to support the creation of different Cache configurations. You must fill the fields below with information about the parameters of the new Cache and for creating the software of its controller. After that, clicking on the "Create" button will be created a folder inside the "caches" folder for the new Cache.

Cache Memory Description:

Parameters:

Mapping technique: Direct

Address size (bits): 32

Number of groups: None

Number of lines: 2

Number of bits of the tag field: 29

Block size (words for line): 32

Word size (bits): 8

Actualization policy: Write-Through

Allocation policy: Write-Allocate

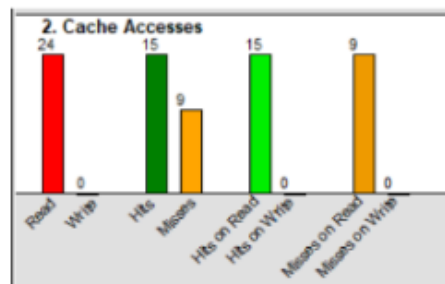
Substitution policy: None

Controller Software:

Folder name (e.g. my\_cache): 32p2bloco

ID string (e.g. Direct 4Lx16Wx8b wt,wallo): 32p2bloco

Create Close



$$Miss\ Rate_{cache1} = \frac{8}{44} \approx 18,2\%$$

$$Miss\ Rate_{cache2} = \frac{9}{24} \approx 37,5\%$$

Logo, a cache mais eficiente é a **cache 1**, uma cache de mapeamento direto de 32 palavras com blocos de 4 palavras.