Prática 1 - parte 1,2 e 3

Erick Henrique
Marina Bernardes

06/2021

—

Laboratório de Arquitetura e Organização de Computadores II

2021.1

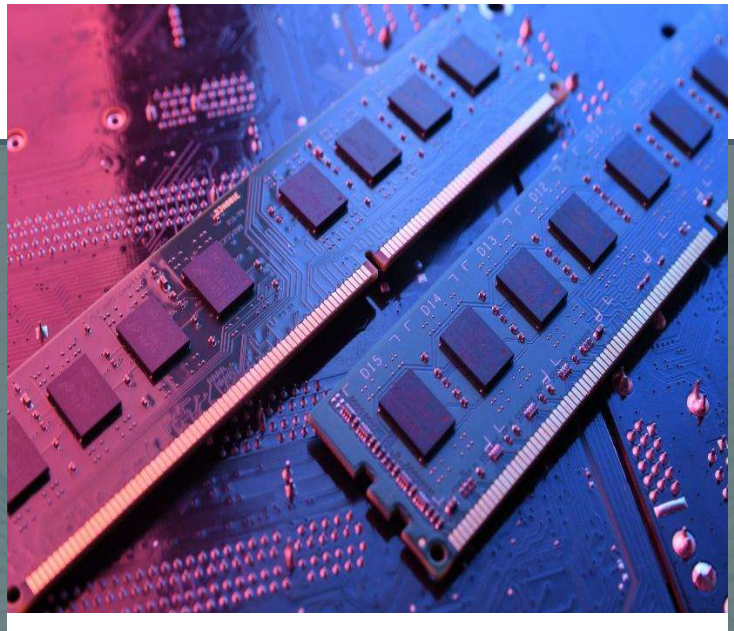
—

Daniela Cristina Cascini Kupsch

Centro Federal de Educação Tecnológica de Minas Gerais

OBJETIVOS

Esta prática tem a finalidade de exercitar os conceitos relacionados à hierarquia de memória.



O PROCESSO

Parte 1

Utilizando os softwares Quartus e ModelSim, implementamos uma memória RAM utilizando a biblioteca LPM.

Essa memória possui 32 palavras de 8 bits, que são acessadas por uma porta com endereçamento de cinco bits.

Para montar a biblioteca LPM utilizamos os passos contidos no arquivo "*Laboratory Exercise 8, Memory Blocks",* disponibilizado pela professora. Criamos um arquivo em Verilog HDL para a chamada dessa biblioteca. O teste foi realizado com o número de ordem de chamada da dupla (Erick "3" e Marina "8"). Esses números foram escritos em posições distintas da memória (primeira e segunda posição), em seguida ocorreu a leitura destas posições.

Para a simulação, geramos ondas para cada input:

• Clock: 50 ps;

• Wren: repeater 1-1-1-0-0-0-0-0-0-0;

• Address: counter;

Algoritmo implementando em Verilog HDL

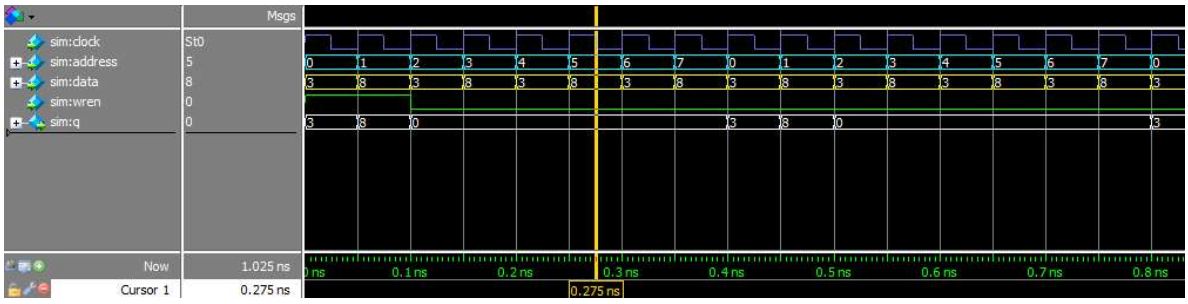
```
module parte1 (address, data, clock, wren, q);

    input     [4:0]  address;
    input     clock, wren;
    input  [7:0]  data;
    output [7:0]  q;

Ram MemoriaRam(address, clock, data, wren, q);

Endmodule
```

Simulação no ModelSim



Como podemos perceber pela simulação, nas duas primeiras posições da memória é feita a escrita das posições da chamada, enquanto wren está habilitado para a escrita e salvo na saída. Ao decorrer da simulação, é feita a leitura destes dados.

Parte 2

Utilizando os softwares Quartus e ModelSim, inicializamos a memória RAM utilizando um arquivo MIF (*"Memory Initialization File"*).

Para montar esse arquivo seguimos os passos contidos no arquivo "*Laboratory Exercise 8, Memory Blocks",* disponibilizado pela professora. Registramos as duas primeiras posições da memória com o número da nossa chamada e as demais posições com números sequenciais ao maior número (8). Linkamos na biblioteca LMP, o ***ramlpm.mif*** gerado. Por fim, criamos um arquivo em Verilog HDL para a chamada dessa biblioteca.

Para a simulação, geramos ondas para cada input

• Clock: 100 ps;

• Wren: constant 0;

• Address: counter;

Algoritmo implementando em Verilog HDL

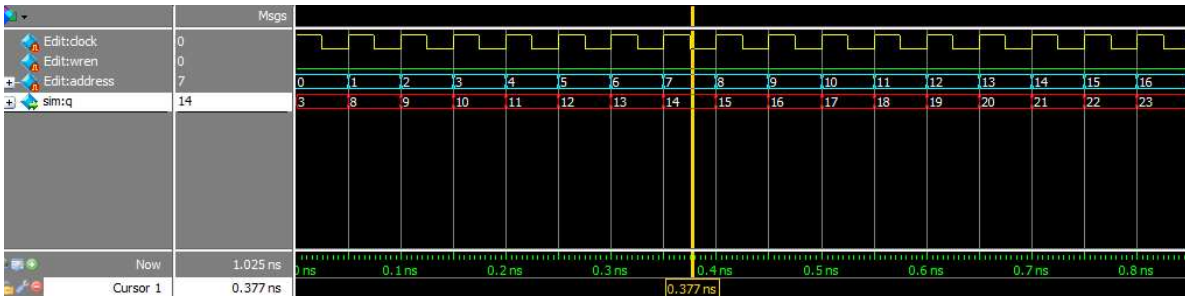
```
    module parte2 (address, data, clock, wren, q);

        input    [4:0]  address;
        input    clock, wren;
        input [7:0]  data;
        output [7:0]  q;

Ramlpm RAMramlpm(address, clock, data, wren, q);

endmodule
```

Simulação no ModelSim



Como podemos perceber pela simulação, nas duas primeiras posições da memória é feita a escrita das posições da chamada, e nas outras posições, é salvo valores maiores que 8.

Parte 3

Implementamos uma hierarquia de memória, organizada em uma cache e uma memória principal.

Cache é o nível mais alto ou primeiro nível de hierarquia da memória. Para a última parte da prática, iremos usar a cache L1 totalmente associativa, ou seja, uma estrutura de cache em que um bloco pode ser posicionado em qualquer local da cache.

A memória principal é diretamente mapeada, cada local da memória é mapeado exatamente para um local na cache, e foi criada a partir da biblioteca LPM. A atualização da memória ocorre utilizando a política de Write-Back, onde inicialmente a escrita de um valor é realizada só no bloco da cache, e escrito na RAM depois que o valor é substituído por outro.

Tanto a memória principal quanto a cache foram inicializadas previamente. Para a memória, inicializamos um arquivo.mif com os seguintes valores:

Endereço	Valor
100	5
101	3
102	1
103	0
104	1
105	8
106	3
107	4
108	9

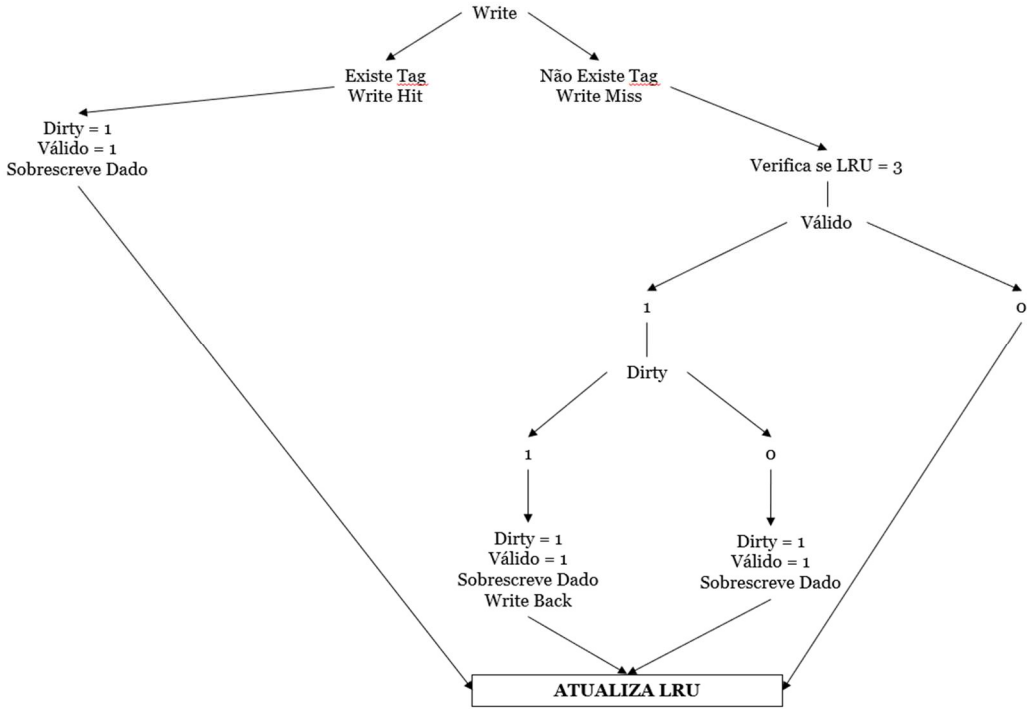
Já a cache, preferimos iniciá-la no *module cache*, implementado na linguagem Verilog, com os seguintes valores:

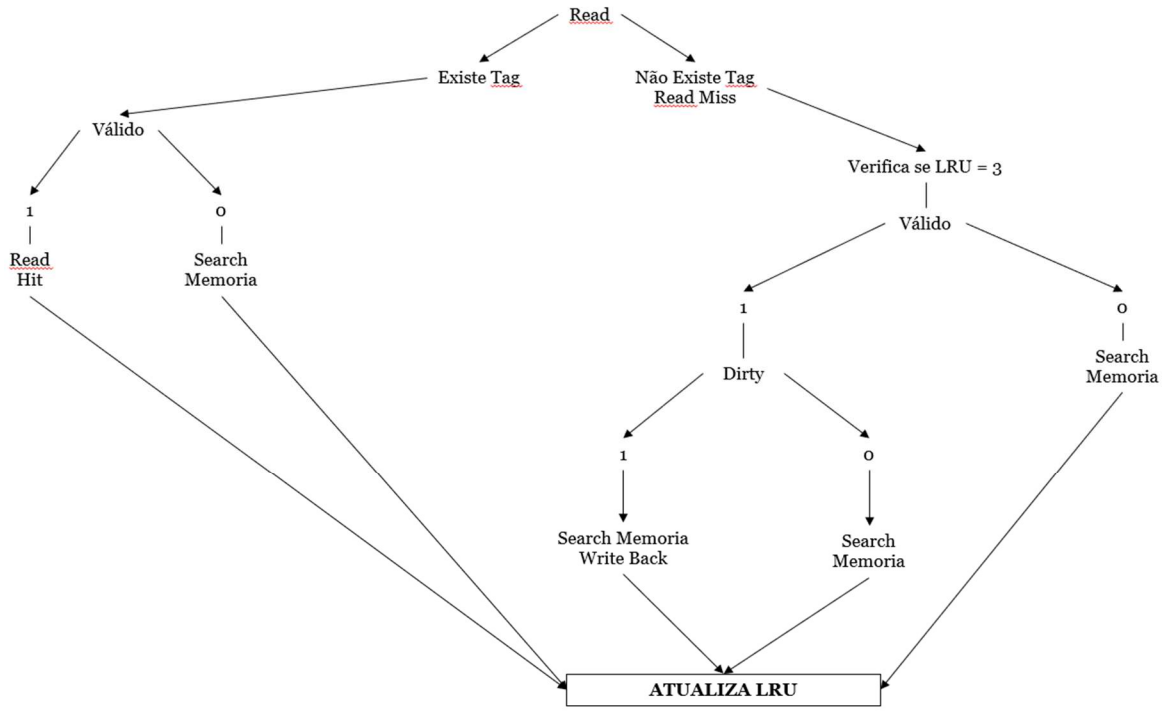
Válido	Dirty	LRU	Tag	Valor
1	0	0	100	5
1	0	1	102	1
0	0	3	105	5
1	0	2	101	3

O conteúdo mais antigo da cache é substituído na ausência de espaço ou na ausência do campo que contém as informações de endereço (tag) utilizando a política de LRU (least recently used). Ela é sempre atualizada.

Há duas operações na memória, escrita e leitura. Para ambas sempre é verificado a validade e a existência da tag. Esse processo retorna miss (os dados não estão presentes na cache, como a tag ou estão presentes, mas não estão válidos) ou hit (dados estão presentes na cache e estão válidos). No caso da operação de escrita, é necessário um bit chamado dirty, que indica se o bloco foi alterado na cache.

A seguir, apresentamos um esquema que nos auxiliou na implementação do algoritmo:




Read
Existe Tag
Não Existe Tag
Read Miss
Válido
1
0
Read
Hit
Search
Memoria
Verifica se LRU = 3
Válido
1
0
Search
Memoria
Dirty
1
0
Search Memoria
Write Back
Search
Memoria
ATUALIZA LRU

Algoritmo implementado em Verilog HDL

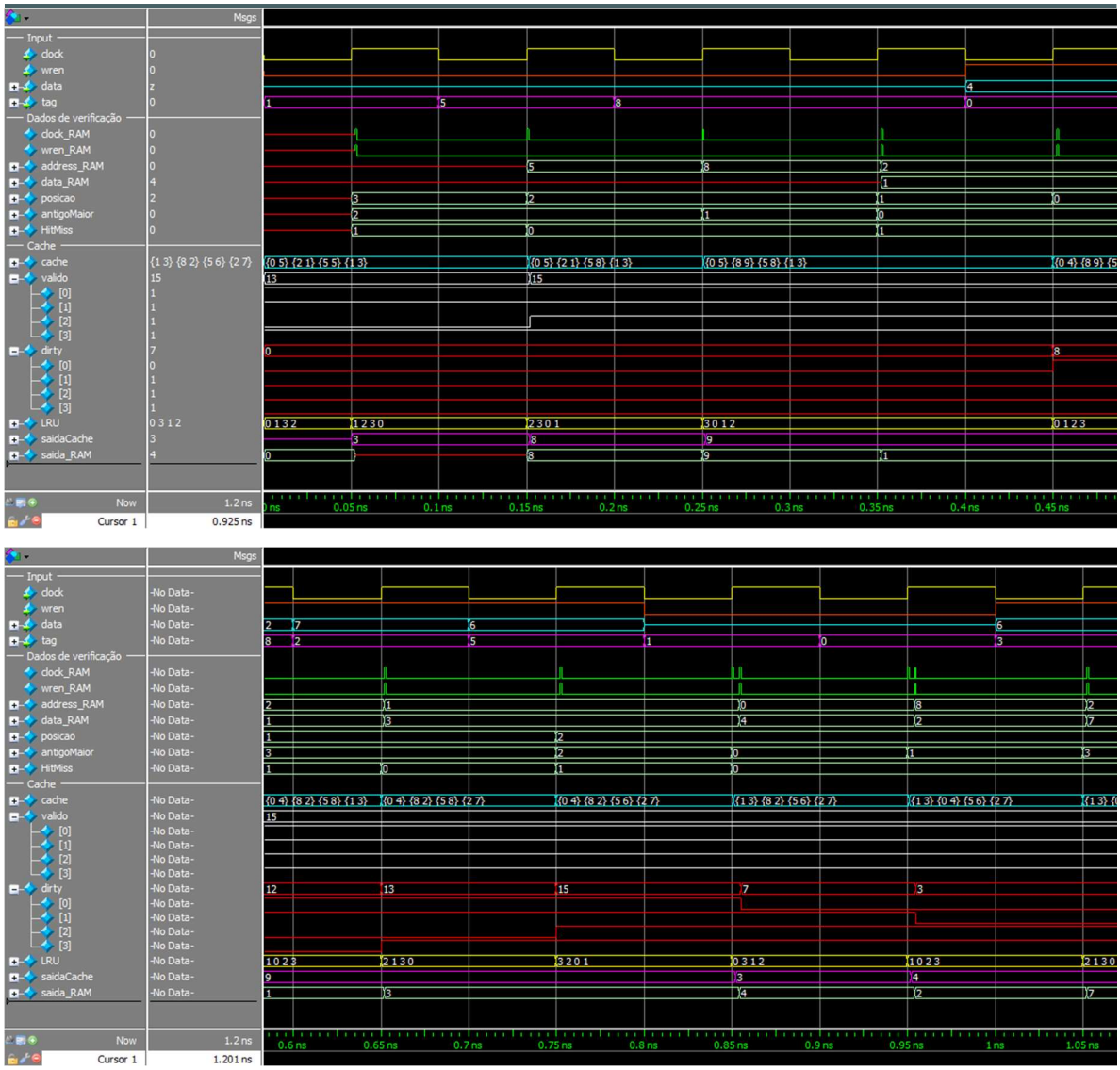
O algoritmo produzido está disponível em <https://pt.textbin.net/fnrozjrdxj>.

Simulação no ModelSim

Para simular o funcionamento do nosso algoritmo, implementamos no ModelSim alguns casos de testes:

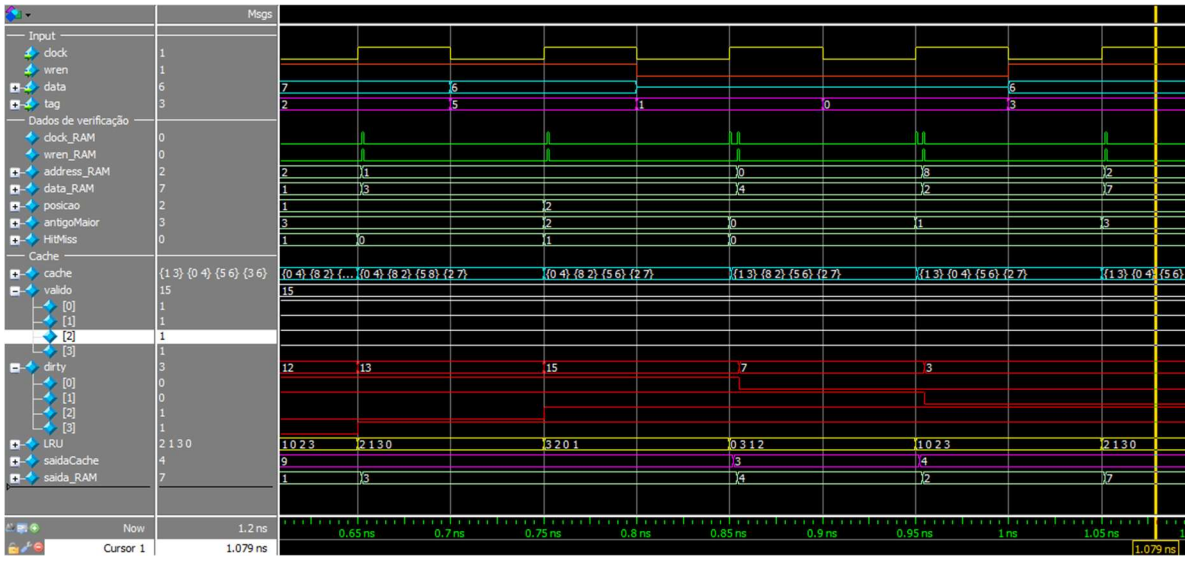
1. Operação Read Tag 101. Saída esperada: Read Hit. Saída de cache esperado: 3.
2. Operação Read Tag 105. Saída esperada: Read Miss. Saída de cache esperado: 8.
3. Operação Read Tag 108. Saída esperada: Read Miss. Saída de cache esperado: 9.
4. Operação Read Tag 108. Saída esperada: Read Hit. Saída de cache esperado: 9.
5. Operação Write Tag 100. Dado enviado: 4. Saída esperada: Write Hit.
6. Operação Write Tag 108. Dado enviado: 2. Saída esperada: Write Hit.
7. Operação Write Tag 102. Dado enviado: 7. Saída esperada: Write Miss.
8. Operação Write Tag 105. Dado enviado: 6. Saída esperada: Write Hit.
9. Operação Read Tag 101. Saída esperada: Read Miss. Write Back. Saída de cache esperado: 3.
10. Operação Read Tag 100. Saída esperada: Read Miss. Write Back. Saída de cache esperado: 4.
11. Operação Write Tag 103. Dado enviado: 6. Saída esperada: Write Miss. Write Back.
12. Operação Read Tag 108. Saída esperada: Read Miss. Write Back. Saída de cache esperado: 2.

Para fins de simplificação, adaptamos a Tag para ser referenciada pelo seu bit menos significativo. Sendo assim, aplicamos tais casos e obtemos os seguintes resultados:



A fins de comparação de sucesso, analisamos a operação 11: Operação Write Tag 103. Dado enviado: 6. Saída esperada: Write Miss. Write Back.

Resultado obtido:



Analisando os dados, temos:

Write (wren =1); Tag (tag = 3); Dado enviado (6); Hit ou Miss? Miss (HitMiss = 0). Sendo assim, a cache deve representar:

Válido	Dirty	LRU	Tag	Valor
1	0	2	101	3
1	0	1	100	4
1	1	3	105	6
1	1	0	103	6

Sendo assim, concluímos que a saída representada pela simulação corresponde com a saída esperada.