

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

TRABALHO PRÁTICO I - PARTE III

A memória cache

Arthur Estevão de Souza Machado
Marcelo Lopes de Macedo Ferreira Cândido

Engenharia de Computação
Laboratório de Arquitetura e Organização de Computadores II
25 de agosto de 2018

INTRODUÇÃO

Ao longo das últimas décadas, a frequência de *clock* dos processadores aumentou rapidamente, além de ter se tornado comum a existência de *multicores*. Contudo, a memória principal não conseguiu acompanhar esse avanço, sendo sua frequência e tempo para acesso relativamente lentos em vista da Unidade Central de Processamento (*CPU*, em inglês). Esse gargalo entre memória e processador foi referenciado ao longo da história como gargalo de Von Neumann (em referência ao matemático húngaro John von Neumann, criador da arquitetura que leva seu nome e que ainda é a base de muitos, senão todos, computadores atualmente). Para amenizar esse problema, surgiu a **memória cache**.

Essa memória, geralmente encontrada no interior de *chips* de processamento, é muito mais rápida que a memória principal. Em contrapartida, ela é bem menor que esta, sendo um dos motivos para isso o alto custo por *byte*.

A cache se baseia no **princípio da localidade**, que se baseia no fato de que dados recentemente usados e os próximos deles tem alta chance de serem reutilizados em breve. A localidade pode ser dividida em dois tipos:

1. espacial: os dados próximos de outros recentemente usados podem vir a ser utilizados também;
2. temporal: os dados recentemente usados podem vir a ser utilizados novamente.

Vale descrever uma propriedade muito importante das caches, parte fundamental dessa prática: a **associatividade**. Para que a cache seja mais eficiente e consiga qualquer endereço de memória possa ser acessado por ela (lembrando que ela não pode conter toda a memória dentro de si por conta do seu tamanho), ela precisa ser mapeada. Esse mapeamento pode ser

- direto: o bloco da memória principal tem um lugar específico na cache. Geralmente definido por uma fórmula matemática
- totalmente associativo: o bloco pode ir para qualquer posição da cache
- associativo por n vias: mescla dos anteriores, sendo constituída por n colunas de m linhas. Cada endereço da memória principal pode ir para uma determinada coluna em qualquer linha (sem considerar a existência de uma política de substituição de blocos, a ser explicada no próximo parágrafo) [3].

Por fim, quando se realiza uma leitura/escrita sobre a cache à procura de um endereço que não está referenciado na mesma, o bloco desse endereço deverá ser trazido da memória principal. Com isso, é bem possível que ele venha a ocupar uma posição já ocupada por outro dado. Essa substituição é um problema que precisa ser tratado e para tal, existem políticas de substituição de blocos. Uma delas é a *Least Recently Used* (*LRU*, a ser utilizada nessa prática), na qual o bloco mais antigo na ordem de uso é substituído [1, 2].

Há muito ainda que se pode falar sobre a memória cache, mas o apresentado é o principal dentro da proposta desse trabalho.

DETALHES DA IMPLEMENTAÇÃO

A fim de efetivar o funcionamento de uma cache associativa por conjunto de duas vias foi utilizada a linguagem de descrição de hardware *Verilog*, em que o uso de registradores que devem armazenar os dados se apresentou constante.

Simulando uma arquitetura de quatro bits e partindo do pressuposto que a cache manipulada não possui mais de uma palavra por bloco (o que tira a necessidade de utilizar um *offset*), recebe um endereço de quatro bits para endereçar seus oito blocos que formam dois conjuntos. Dos quatro bits encaminhados, é necessário utilizar dois para índice e os outros dois para TAG.

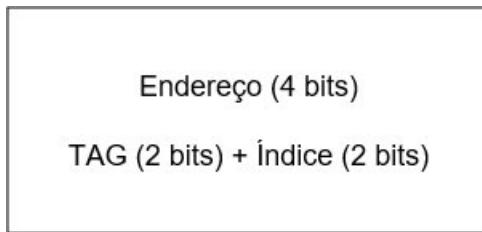


Figura 1: Quebra do endereço

Como uma cache na maioria dos casos atua em conjunto com outros componentes de uma hierarquia de memória é importante implementar os acertos e erros de acesso, ou seja, o acesso ocorreu normalmente ou algo está incoerente com o esperado e uma rotina de tratamento deve ser inicializada. Naturalmente, caso um acesso ocorreu corretamente (Hit = Nível lógico alto na implementação), o erro estará inativo. Dessa maneira, o módulo de implementação da cache recebe apenas um fio para lidar com essa situação.

Na tentativa de ler e enviar um endereço com um índice correto, mas sua TAG inválida é um exemplo de erro de acesso, já que a TAG identifica se o bloco associado corresponde a uma palavra requisitada.

O erro de acesso na cache implementada será decorrente também do bit de validade referente ao índice estar em nível lógico baixo. De fato isso deveria ocorrer, uma vez que essa falha¹ em uma hierarquia acarreta inúmeros erros.

Para simular uma política de substituição, é utilizado o LRU (Least Recently Used) no qual o endereço utilizado mais recentemente irá conter LRU = 0, e o menos recentemente LRU = 1. Essa é uma política que tem relação direta com o princípio da localidade temporal, onde um dado utilizado recentemente possui mais chances de ser reutilizado que um dado utilizado a mais tempo. Isso de fato ocorre, já que variáveis em um programa são reutilizadas inúmeras vezes, e a permanência deles na cache (topo da hierarquia de memória) oferece diversos benefícios na execução de um programa.

O *dirty* na implementação, mostra que uma escrita foi realizada na cache, e o dado deve ser atualizado na memória principal com sua devida rotina de tratamento. Caso o dado foi atualizado somente na memória cache, o *dirty* terá nível lógico alto, e baixo caso contrário.

FUNCIONAMENTO

Para acompanhar o funcionamento do projeto, a dupla apresenta um fluxograma no apêndice do relatório.

Devem ser enviados ao módulo da cache *reset*, *clock*, habilita escrita, endereço e dado de entrada.

1. *Clock* foi utilizado para coordenar as devidas manipulações do bloco de memória assim como qualquer circuito síncrono.
2. *Reset* foi necessário para reinicializar os dados contidos na memória.
3. Habilita escrita é necessário para indicar quando se deseja fazer ou leitura ou escrita.
4. Endereço guarda TAG e Índice respectivamente.

¹O bloco apresenta um dado válido em relação a memória principal (Nível lógico 1), ou não (Nível lógico 0).

5. Dado de entrada representa a palavra a ser inserida em caso de escrita.

Ao encaminhar os dados devidamente, o endereço será desmembrado para auxiliar a busca pelo devido índice e em seguida a conferência de TAG. Será identificado caso de leitura ou escrita ao avaliar o habilita escrita.

Caso seja uma condição de escrita (Habilita = 1) são verificadas as vias e comparadas as TAG's. Caso uma das TAG's comparadas seja igual à enviada, *hit*, válido e *dirty* recebem nível lógico 1. Diferente do LRU que agora é o dado mais recente. Então o dado é atribuído à via. Caso a TAG não seja igual à enviada, todos os passos mostrados anteriormente exceto para *hit* são tomados com a utilização do LRU. Isso se dá, pois, representa um caso de cache miss e verificação de *Write Back*.

Em uma situação de leitura são verificadas as vias e comparadas as TAG's. Quando os dados correspondem para uma via, é verificado o bit da validade daquela posição para ver a integridade do dado buscado. Caso o bit de validade seja 0, existe um cache miss e *hit*=0. Caso seja verdadeiro, *hit*=1, o dado de saída é atribuído e LRU é atualizado. Quando a TAG não corresponde em nenhuma das duas vias, *hit* = 0, e o dado não é lido.

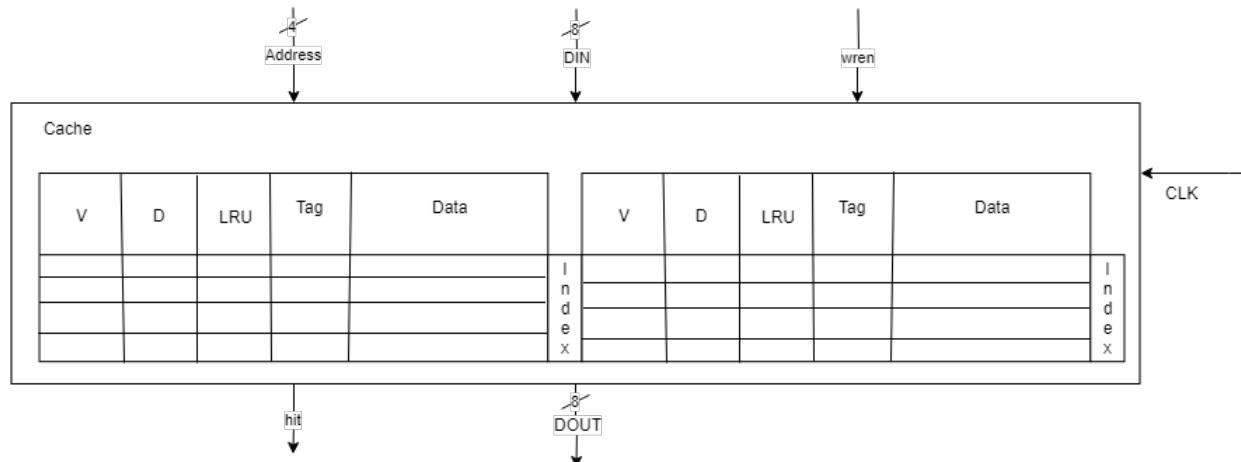


Figura 2: Diagrama esquemático

SIMULAÇÕES

Para simular devidamente a memória cache implementada foi utilizado o software ModelSim-Altera 10.1.d e a placa Altera DE2, em ambos os casos, foram verificados os dados contidos com a

inicialização da memória seguido de escritas e leituras sucessivas. Seguindo esse procedimento, a observação do dados hit, válido e LRU são extremamente importantes como seguem abaixo na simulação do ModelSim.

ModelSim

Utilizando o carregamento prévio, obteve-se as seguintes ondas:

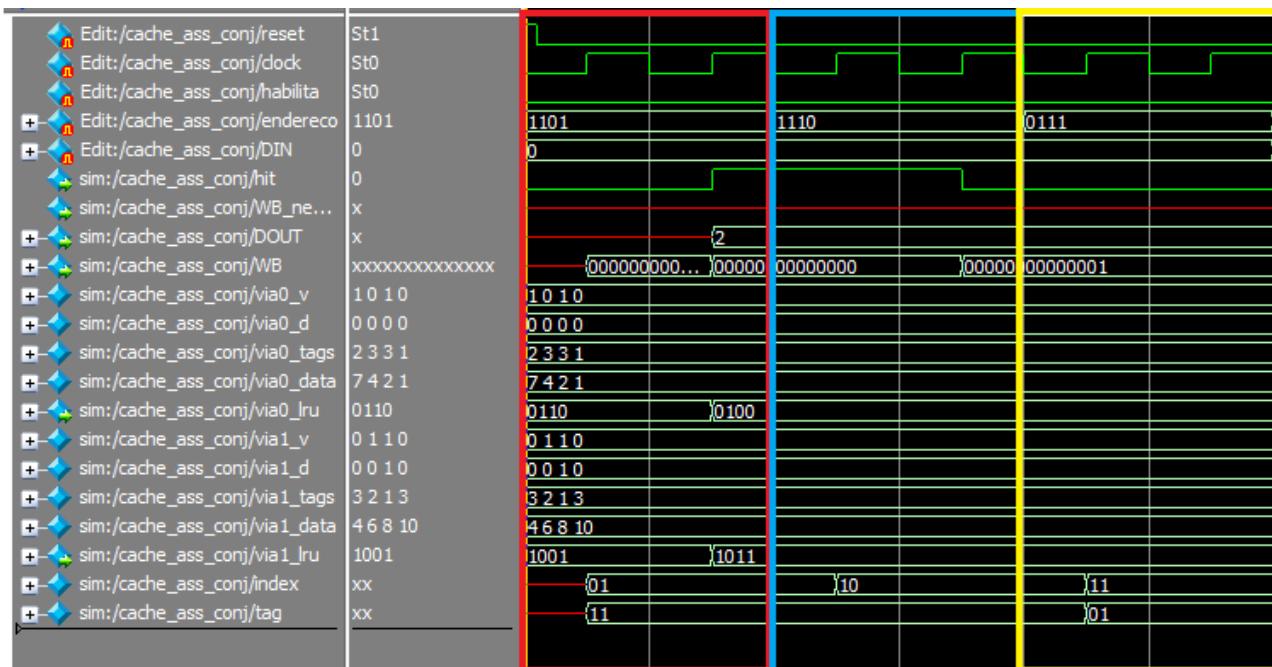


Figura 3: Teste 1

Com a memória inicializada, foi enviado à ela o endereço 1101, correspondente à TAG 11 e índice 01. Juntamente com o endereço desejado, foi encaminhado o habilita escrita em sinal lógico baixo e o dado de entrada que não será utilizado como zero. A primeira verificação após indicar o índice, é a TAG. Para isso compara-se as TAG's das duas vias correspondente ao índice.

VIA 0 - TAG's -2 3 **3 1**

VIA 1 - TAG's -3 2 **1 3**

A TAG da via 0 é igual a TAG enviada, com isso verifica-se se é válido.

VIA 0-Válido -1 0 1 0

Como o bit válido referente está em nível lógico alto, a leitura pode ocorrer os dados são atualizados.

HIT = 1
DOUT = 2
VIA 0 - LRU - 0110 -> 0100
VIA 1 - LRU - 1001 -> 1011

A segunda etapa de teste, correspondente ao retângulo azul, tenta fazer a leitura em uma posição (1110) que o bit de validade se apresenta inválido. De fato, quando a posição não atende a todos os requisitos, a leitura não pode ser concretizada e a atualização dos dados LRU e DOUT não podem ocorrer. Portanto, o único dado a ser alterado é o hit, ficando sem nível lógico baixo, a fim de representar o miss.

A terceira etapa, descrita em amarelo, decorre da tentativa de realizar uma leitura na posição 0111 que representa índice e TAG iguais a 11 e 01 respectivamente. É possível observar que nenhuma das duas vias possui a TAG encaminhada, e portanto, a leitura não pode ser efetivada, mantendo o miss anterior.

Como quarta etapa de teste, será feita a escrita em uma posição de memória destacada em rosa.

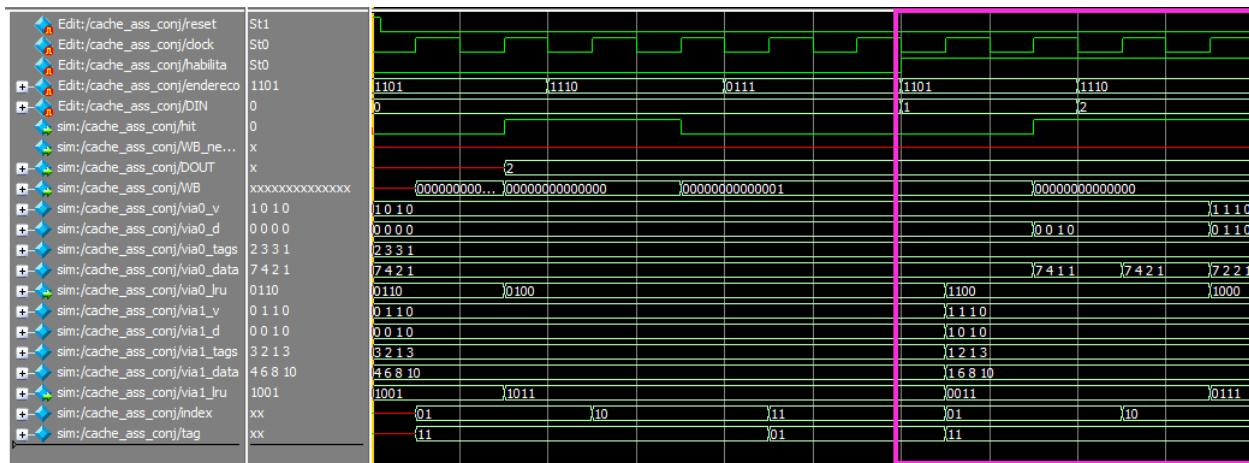


Figura 4: Teste 1

É possível observar que com o dado encaminhado à posição de memória 1101 e 1110, o módulo de memória segue a rotina de dados, atualizando os valores respectivos e armazenando o dado recebido.

A última etapa de testes consiste em ler um dado de um endereço escrito anteriormente, para isso, encaminha-se o endereço desejado e o habilita escrita deve estar em nível lógico baixo. Com a leitura ocorrendo corretamente, os dados são atualizados juntamente com DOUT.

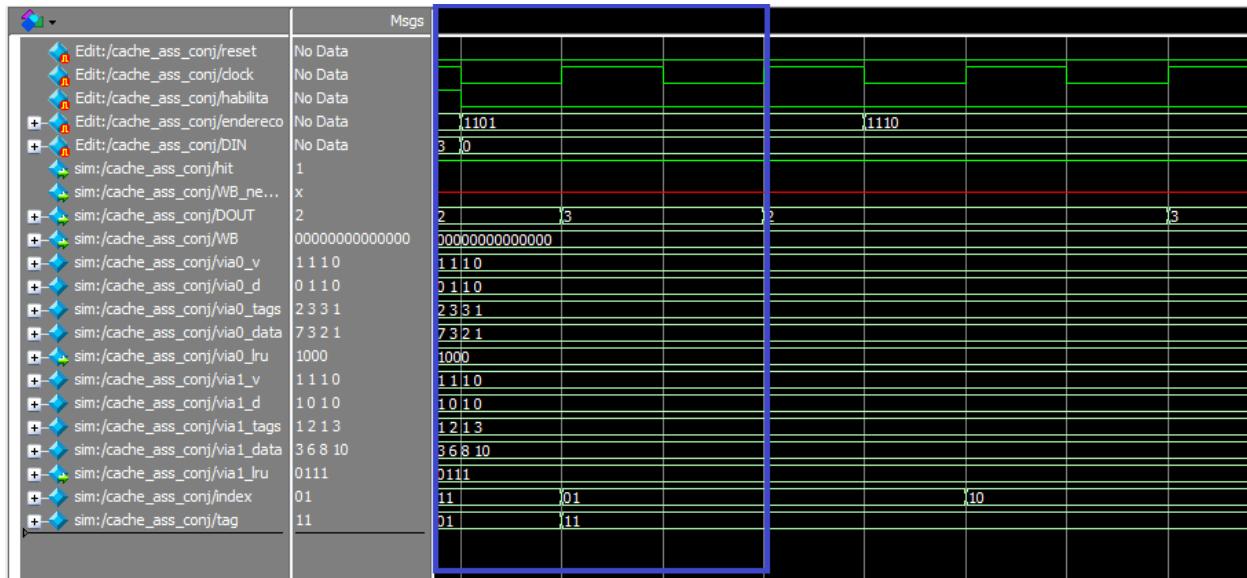


Figura 5: Teste 1

Placa Altera

A simulação seguinte apresenta o mesmo padrão da anterior, porém utilizando as chaves SW para entrada e os LED's aliados ao *display* para observar a saída.

Chaves utilizadas :

Entradas:

$$\text{Clock} = \text{SW}[17]$$

$$\text{Habilita escrita} = \text{SW}[15]$$

$$\text{Reset} = \text{SW}[14]$$

HEX2 = Endereço = SW[3:0]

HEX7 = TAG do endereço desfragmentado

HEX6 = Índice do endereço desfragmentado

HEX4 = Dado de entrada = SW[13:6]

Saídas:

LEDG[0:0] = Hit

LEDR[0:0] = Miss

LEDR[9:2] = LRU's

HEX0 = Dado lido da memória

O teste iniciou com a memória carregada previamente, apresentando o ponto de partida dos LRU's (LEDR[5:2] para via 0 e LEDR[9:6] para via 1).

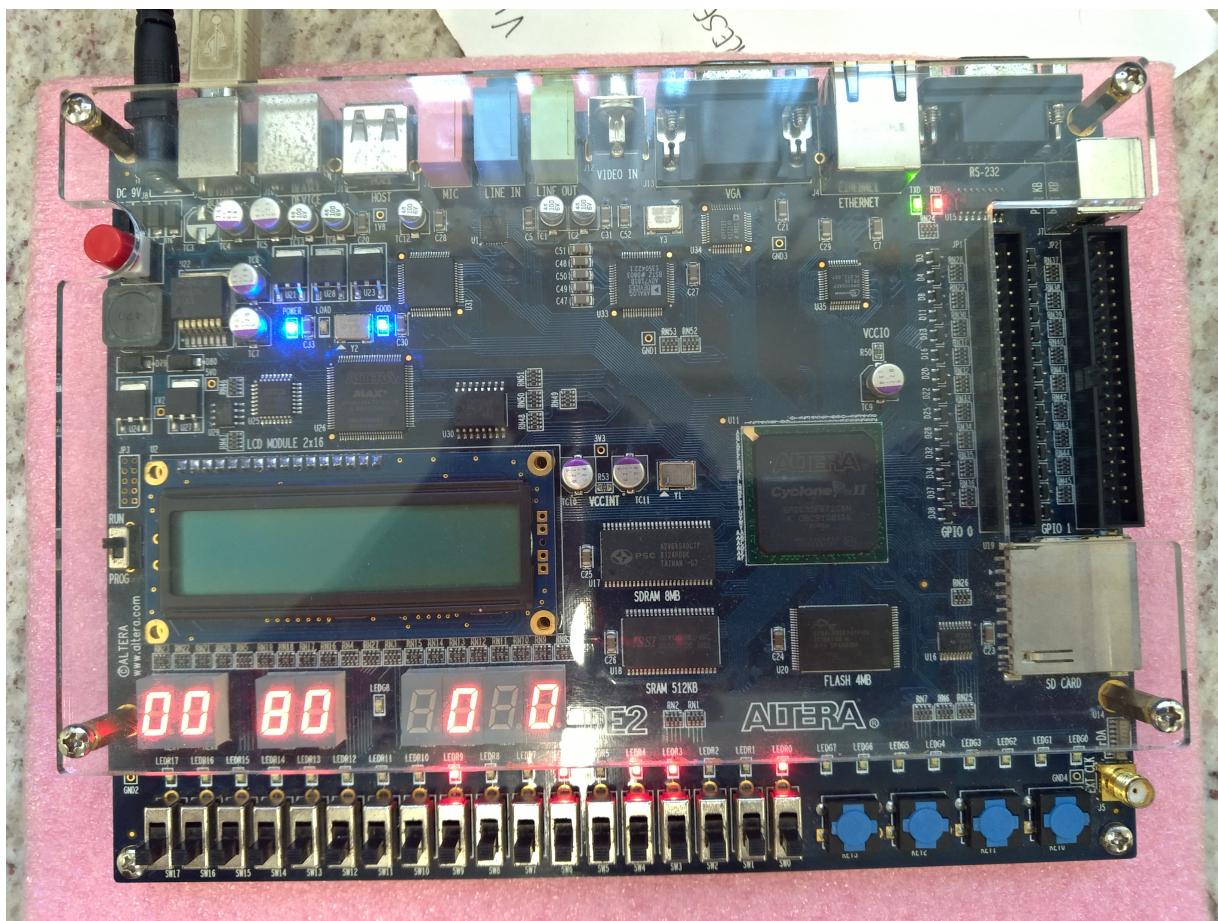


Figura 6: Teste 1

Em seguida foi testada uma leitura com acerto no endereço 1101, correspondente ao índice 1 e TAG 3. É possível observar a atualização do LRU dO LEDR 3 E 7, em que a via 0 recebe 0 (Mais recente) e a via 1 recebe 1. O dado retornado corresponde ao dado inicial esperado (2) ocorrendo um cache hit (Representado pelo LEDG 0).

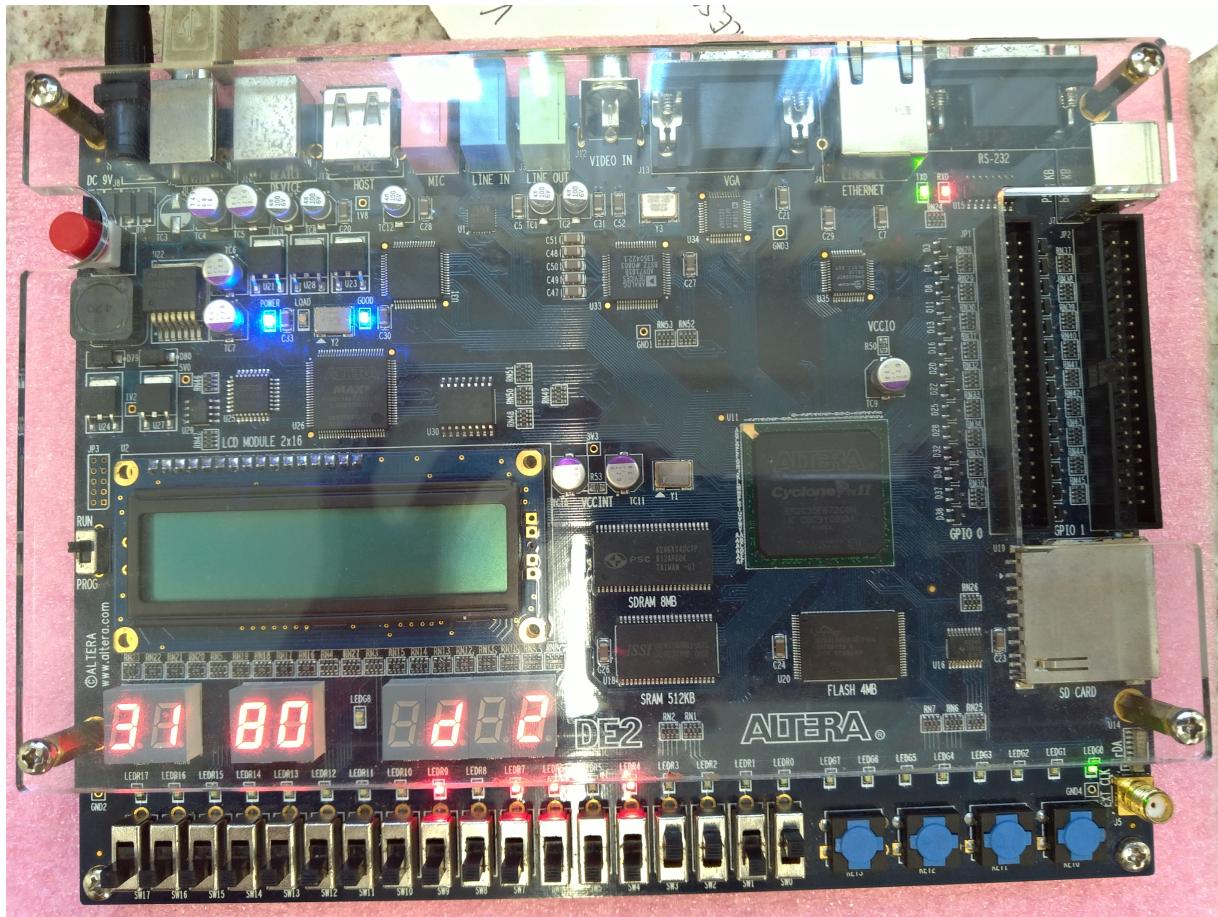


Figura 7: Teste 2

Como terceira etapa do teste, foi testado uma leitura com falha no endereço 0100, correspondente ao índice 0 e TAG 1. O destaque fica para o LEDR 0 correspondente ao cache miss.

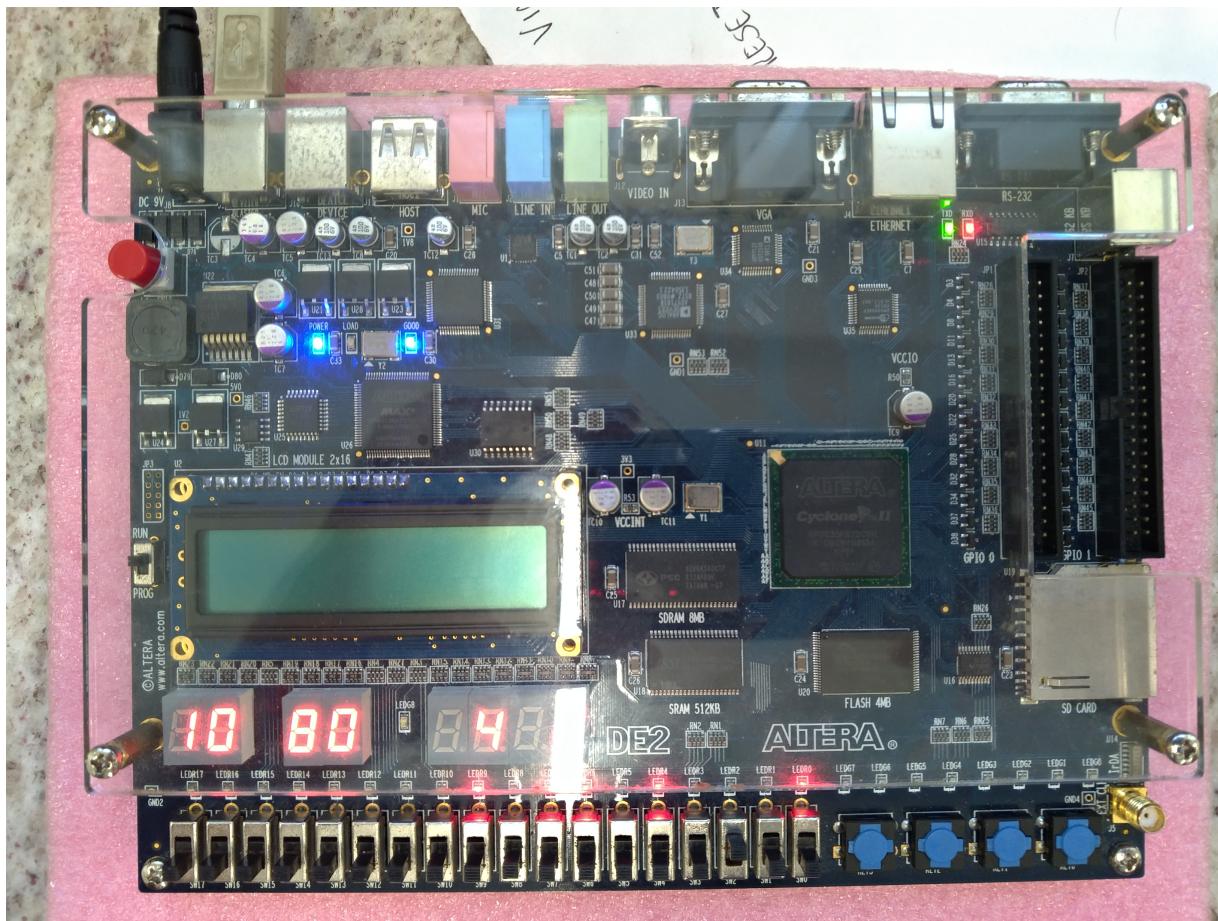


Figura 8: Teste 3

O quarto caso de teste, trata uma escrita com dado de entrada 15 (F em hexadecimal) no endereço 1110, correspondente ao índice 2 e TAG 3. A TAG é válida para a via 0, então ocorre um hit (LEDG 0) e além disso, temos o LRU atualizado, LEDR 4 em nível lógico baixo e LEDR 8 em nível lógico alto (Menos recente).

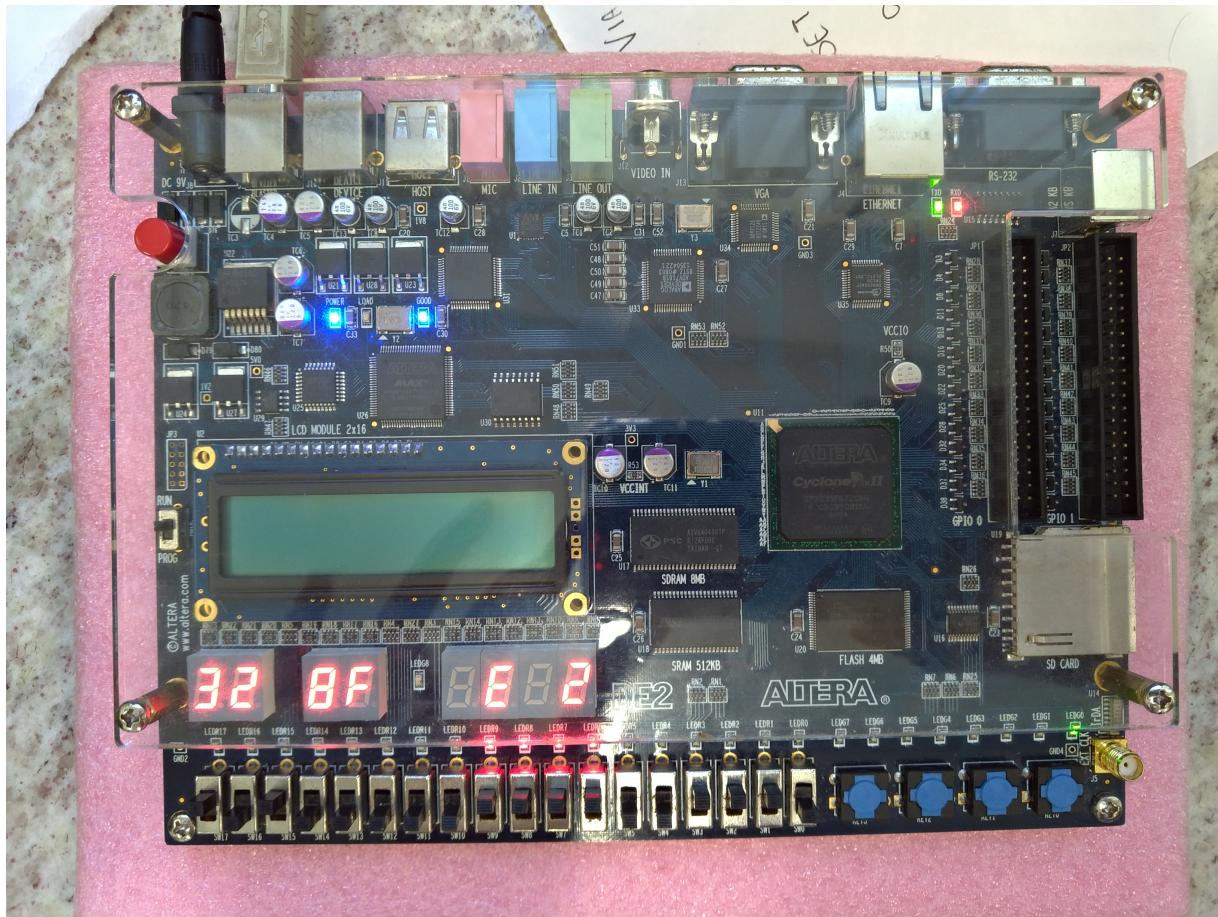


Figura 9: Teste 4

O quinto caso de teste, fará uma escrita em uma posição na qual a TAG não corresponde à enviada. O dado encaminhado é 14 (E em hexadecimal) e o endereço 0000, onde TAG e índice são 0, o que não bate com nenhuma das vias, gerando um cache miss (LEDR 0 ativo). Utilizando a política de LRU, o dado utilizado a mais tempo pertence à via 1, uma vez que o LEDR 6 está ativo e portanto, é o menos recente. Com isso, a escrita é realizada, e o LRU atualizado.

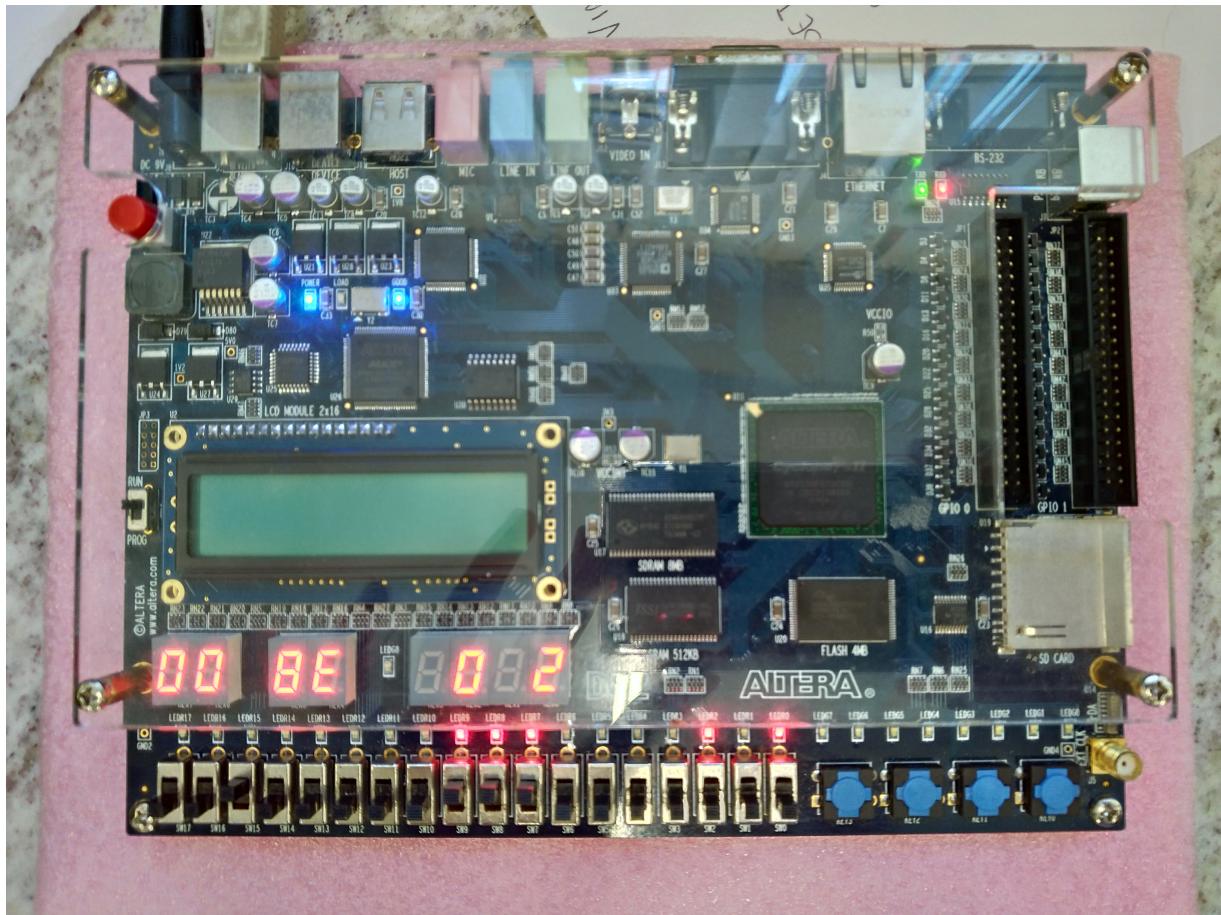


Figura 10: Teste 5

Como sexta etapa, será realizada a leitura do último endereço escrito. Para isso, será enviado o endereço 0000, e como o dado foi escrito anteriormente, certamente estará válido ocorrendo assim um cache hit (LEDG 0) e retornando 15 (E) como dado de saída (HEX0).

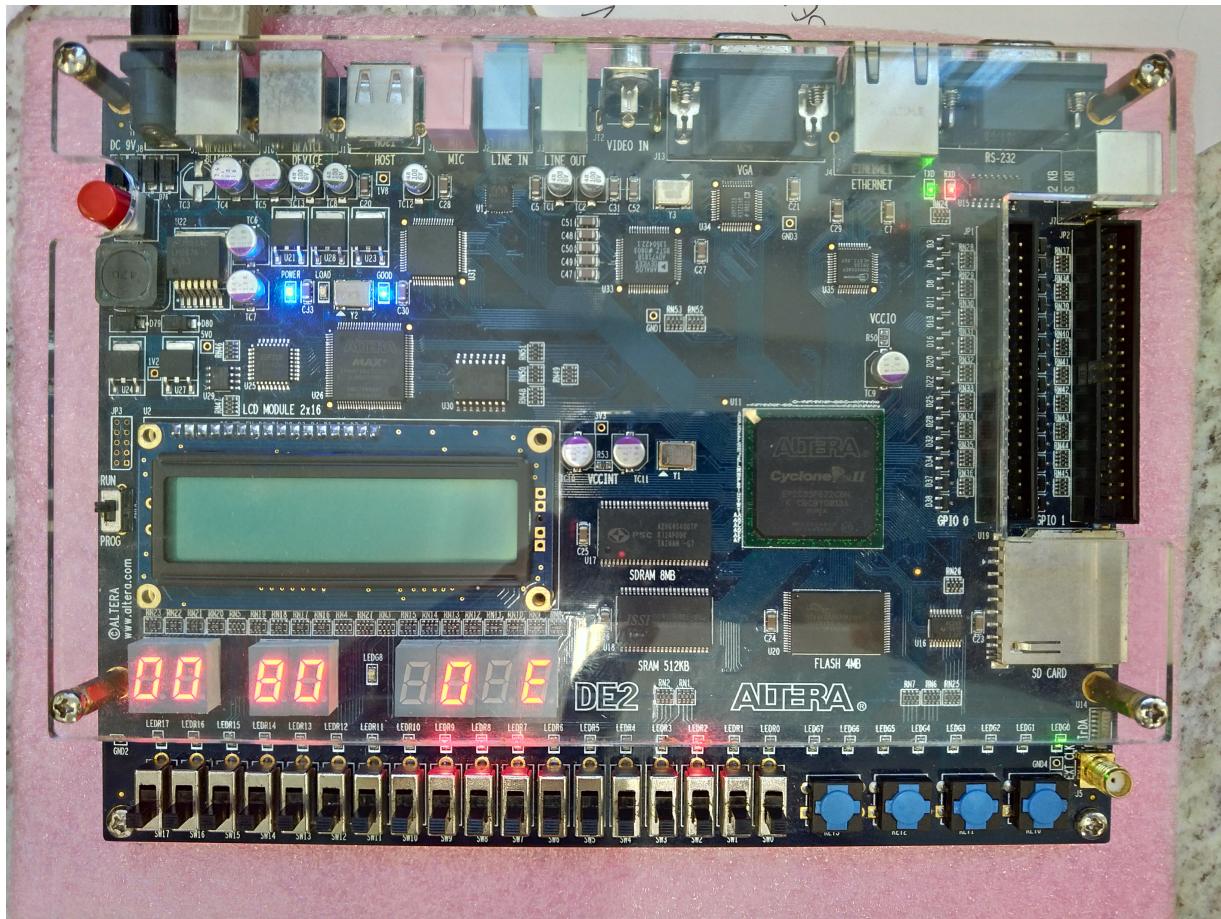


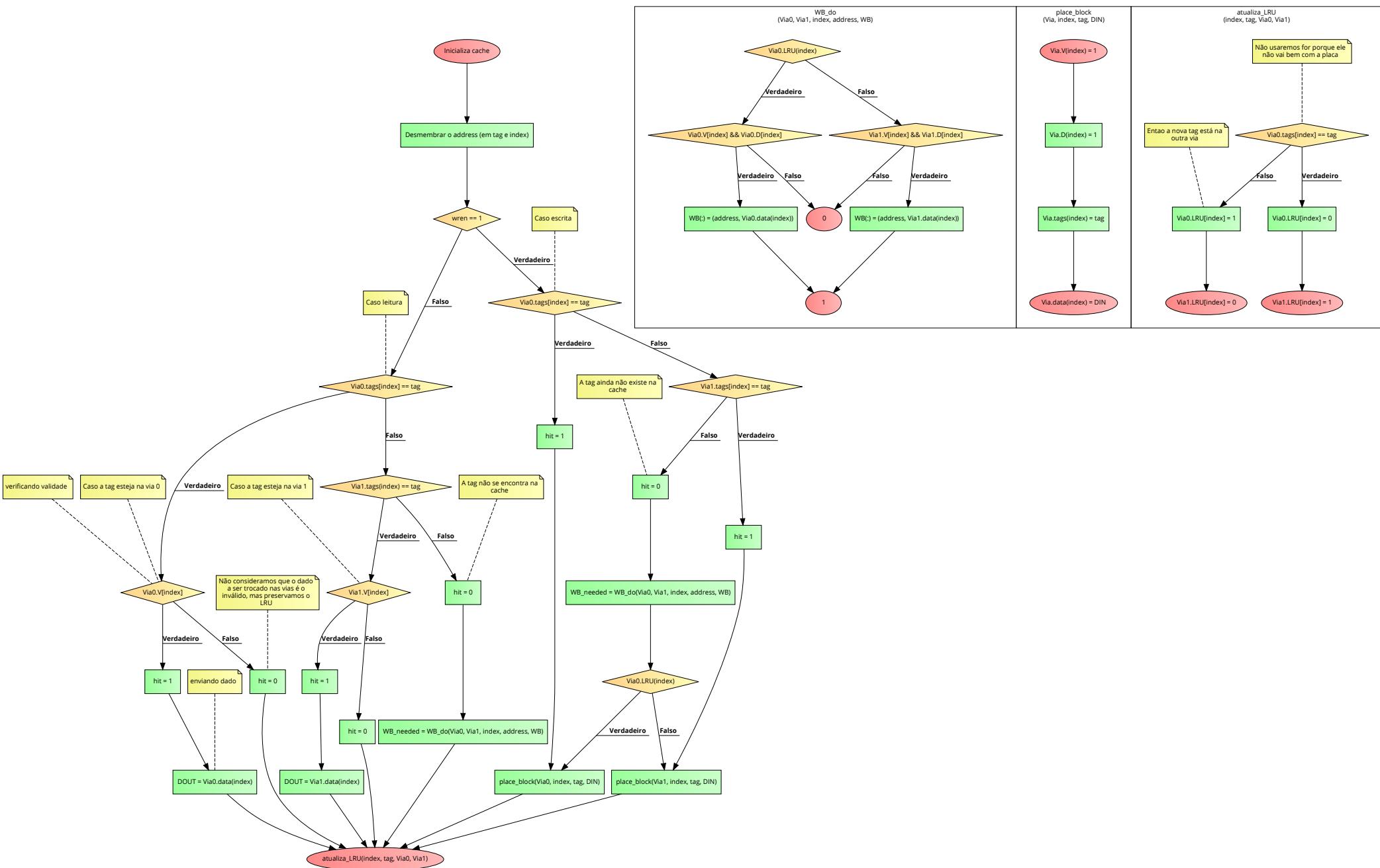
Figura 11: Teste 6

DIFICULDADES E SUGESTÕES

Sem dúvidas este trabalho requer um grande estudo prévio sobre hierarquia de memória e principalmente sobre memórias cache, em especial a associativa por conjunto de duas vias, tratada no presente relatório. A complexidade do assunto é elevada, mas a experiência de implementação extremamente enriquecedora.

CONSIDERAÇÕES FINAIS

A cache, de fato, pode assumir um papel fundamental na hierarquia de memória retornando dados mais rapidamente que a memória principal. Além disso, reduz a troca de dados entre processador e memória principal reduzindo o gargalo de Von Neumann. Por conta deste, a ausência de uma cache iria prejudicar o processamento já que a taxa de transferência que o processador consegue trabalhar é muito maior que a taxa trabalhada pela memória.



REFERÊNCIAS

- [1] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A quantitative approach*. Elsevier, 2011.
- [2] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design*. Elsevier, 2013.
- [3] Wikipédia. Cpu cache — wikipédia, a enciclopédia livre, 2018. [Online; accessed 29-maio-2018].