UNIVERSIDADE FEDERAL DO PARANÁ

Carlos Iago Bueno, Gabriel Lüders e Richard Fernando Heise Ferreira

IMPLEMENTAÇÃO DE MEMÓRIA *CACHE* ORGANIZADA POR MAPEAMENTO DIRETO E ESCRITA CONTROLADA PELA POLÍTICA *WRITE-THROUGH*

CURITIBA
2021

RESUMO

O objetivo deste trabalho é implementar uma memória de acesso rápido (*cache*) em simulador de circuitos. A organização escolhida para a memória *cache* foi a de mapeamento direto e a política de escrita adotada foi a de *write-through*.

INTRODUÇÃO

As memórias *cache* são memórias de espaço reduzido com rápida velocidade de acesso. Ela armazena os dados usados com mais frequência pela unidade central de processamento. Quando a *CPU* precisa buscar informações, ao invés de buscar sempre na memória principal, muito mais lenta, ela busca primeiramente na *cache*, tornando os processos dentro do sistema mais eficientes.

As memórias são organizadas através de blocos ou linhas de informações que podem ser organizadas em conjuntos. O mapeamento direto é uma organização de simples implementação, porém há apenas uma posição na *cache* onde este dado pode ser alocado, diferente de outros tipos de mapeamento que permitem outras configurações de dados.

Nesta forma de organização, os blocos da memória principal são chamados de linha de *cache*. Esta linha é dividida nos seguintes componentes: *index*, *bit* de validade, *tag* e os dados. O *bit* de validade serve para indicar se os dados que estão naquela posição são válidos. A *tag* serve para diferenciar dados cujos endereços têm o mesmo índice na memória *cache*.

As memórias necessitam também de uma política de escrita. O *write-through* é uma política de simples implementação que mantém consistência interna, já que nunca está dessincronizada com a memória principal. Este método escreve tanto na linha específica do *cache* quanto na zona de memória ao mesmo tempo.

METODOLOGIA

1. Ambiente de simulação

A construção da memória foi feita através de uma ferramenta educacional para a concepção e a simulação digital de circuitos lógicos, o *Logisim*. *O Logisim* é usado por acadêmicos e profissionais para projetar e simular *CPUs* completas para fins educacionais.

2. Arquitetura

Este trabalho foi modelado a partir de uma implementação aberta do *MIPS pipeline* de 5 estágios feita por Singh, J; Shafiq, M. (2015). A implementação pode ser encontrada rodando o *Logisim* e abrindo o arquivo PipelinedMips.circ. O único arquivo modificado foi o main, os adicionados foram *cache*, *cache-entry* e *memory*. Os testes estão na pasta de testes. Cada teste possui uma versão para ser carregada na memória de instruções e uma versão para facilitar o entendimento.

IMPLEMENTAÇÃO

Com base nas instruções de construção de *cache* de Davis, J e Weinman, J. (2013) foram implementadas quatro entradas da memória *cache*. Cada uma possui 4 registradores de 32 *bits* para os dados, 1 registrador de 1 *bit* para válido e 1 registrador de 4 *bits* para *tag*. Além dos registradores, temos ainda as seguintes entradas: *offset*, *tag*, *memWrite*, *memRead*, *clock*, *reset* e *data_in* e as seguintes saídas: *miss* e *data_out*.

O registrador de válido só torna a entrada válida quando se quer escrever na *cache-entry* (*memWrite* ou quando há um *miss*). O registrador de *tag* guarda a *tag* do bloco selecionado que será, então, comparada com a *tag* de entrada. Em caso de o válido estar em 1 e as *tags* não baterem ou não ser válido para escrita ocorrerá um *miss*, contudo, esse sinal de *miss* só será validado se estivermos em uma situação de leitura, isto é, *memRead* ativo.

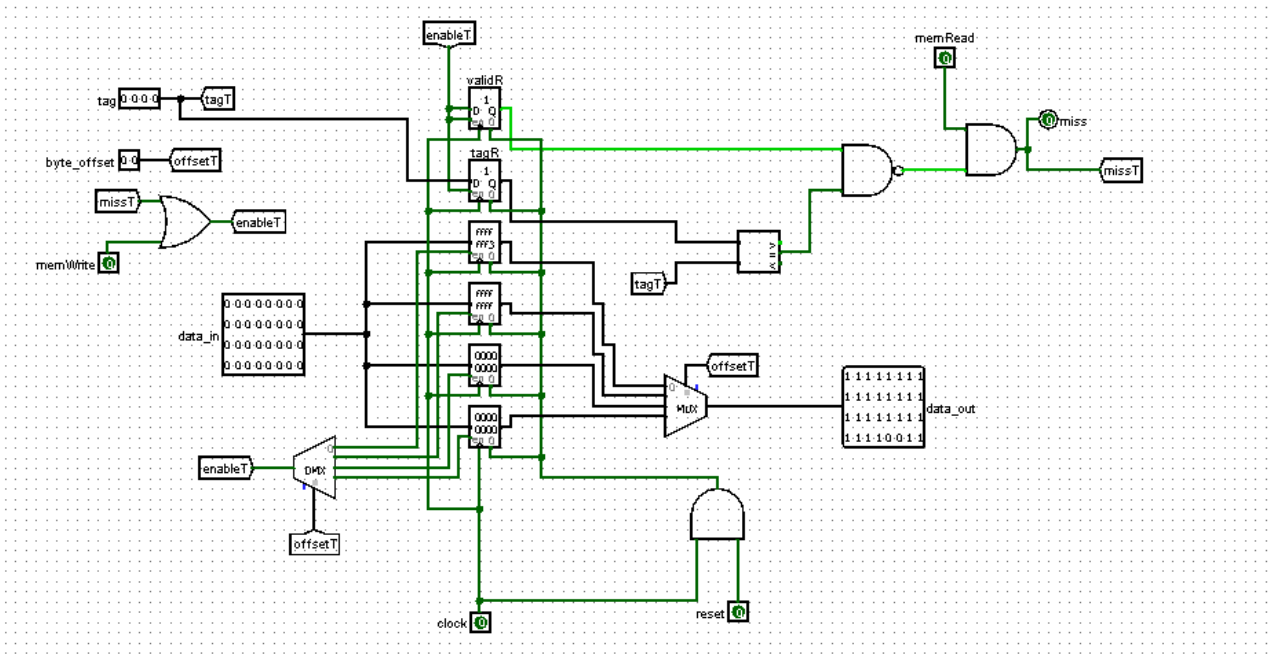

Imagem 1. Cache-entry

Em caso de leitura, o sinal de *read* é ligado, a *entry* é testada para *miss*, se for um *miss*, precisamos habilitar a escrita na *cache*, em caso de *hit*, o dado armazenado em um registrador simplesmente é selecionado usando o *byte* de *offset* e redirecionado para a saída. Em caso de escrita, o sinal de *memWrite* vem ativo e simplesmente escreve no registrador selecionado pelo *byte* de *offset*.

Subindo um nível de abstração há 4 entradas de *cache* (*four-way cache*, com 4 *cache-entry*) e uma lógica de seleção específica. As entradas são o endereço de escrita ou de leitura, *addr*; o dado a ser escrito, *data-in*; os sinais de memWrite e *memRead* que controlam a leitura ou escrita na *cache* e, nas saídas, temos o sinal de *miss* e o *data-out*.

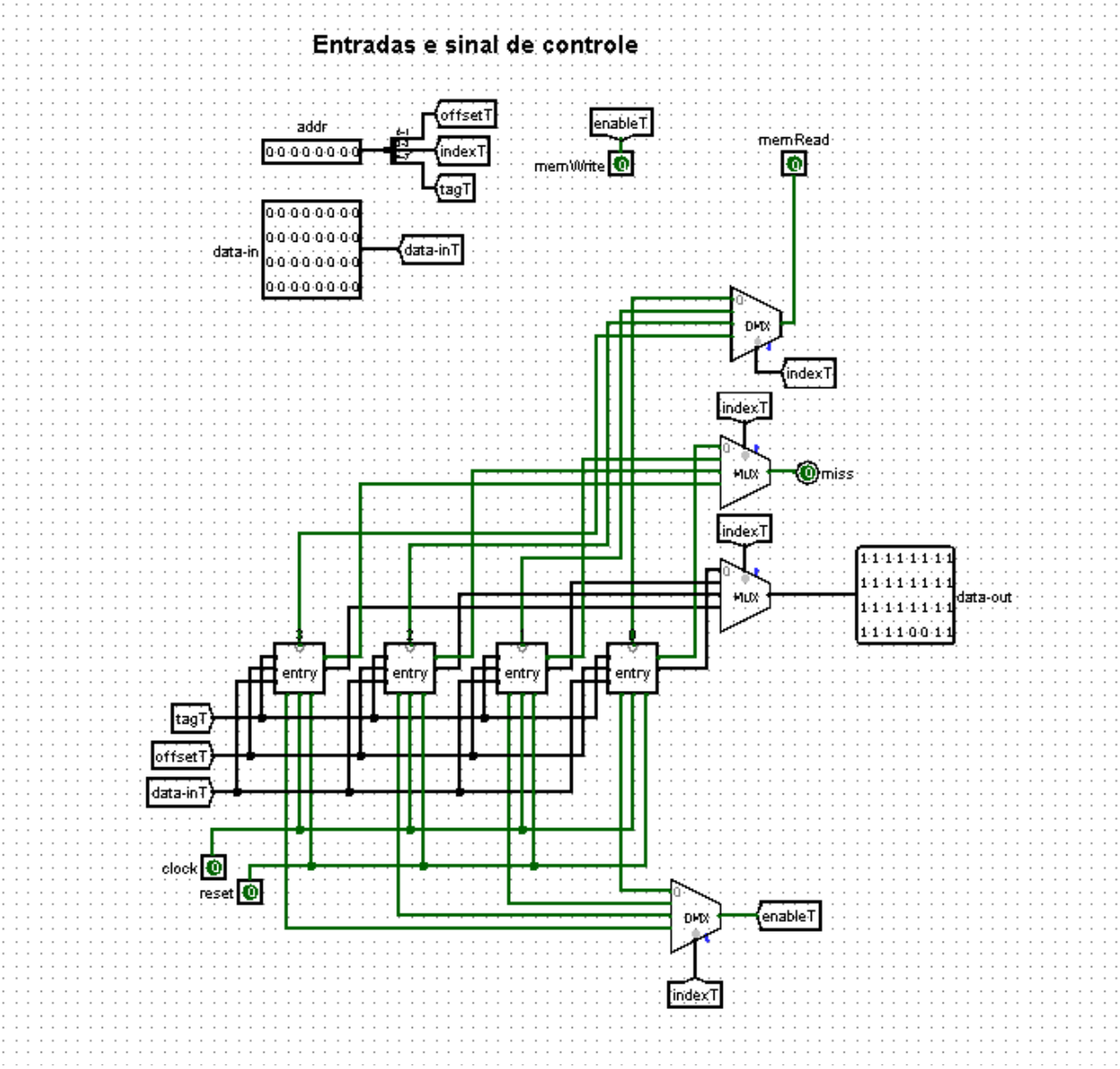

Imagem 2. *Four-Way Cache*

O endereço que vem através da entrada *addr* é quebrado em 4 *bits* de *tag*, 2 *bits* de *index* e 2 *bits* de *offset*. *Tag* e *offset*, bem como *data-in*, *memRead* e *memWrite*, são repassados para cada entrada, porém, somente a entrada habilitada a ser lida/escrita efetivamente gerenciará os dados em uma dada interação. Essa entrada é habilitada através dos *bits* de *index*.

Com a *cache* pronta, basta agora que o circuito assuma seu papel de L1 propriamente dita, para tanto o circuito até aqui foi compartimentalizado e associado à uma memória *RAM*. Nessa etapa, as entradas são as mesmas do circuito anterior, mas a lógica é bem diferente. Primeiramente todos os sinais de controle e o endereço recebido da *CPU* vão diretamente para a *cache* para serem lidados internamente, já os dados devem ser selecionados da seguinte maneira: se o sinal *memWrite* está ligado,

independente de um *miss* ser detectado ou não, deve-se pegar o valor que vem da entrada do processador, ou seja, o *data-in*. Já se o sinal de escrita está desligado o *miss* é quem ditará que a escrita na cache pelos dados atualizados da *RAM*.

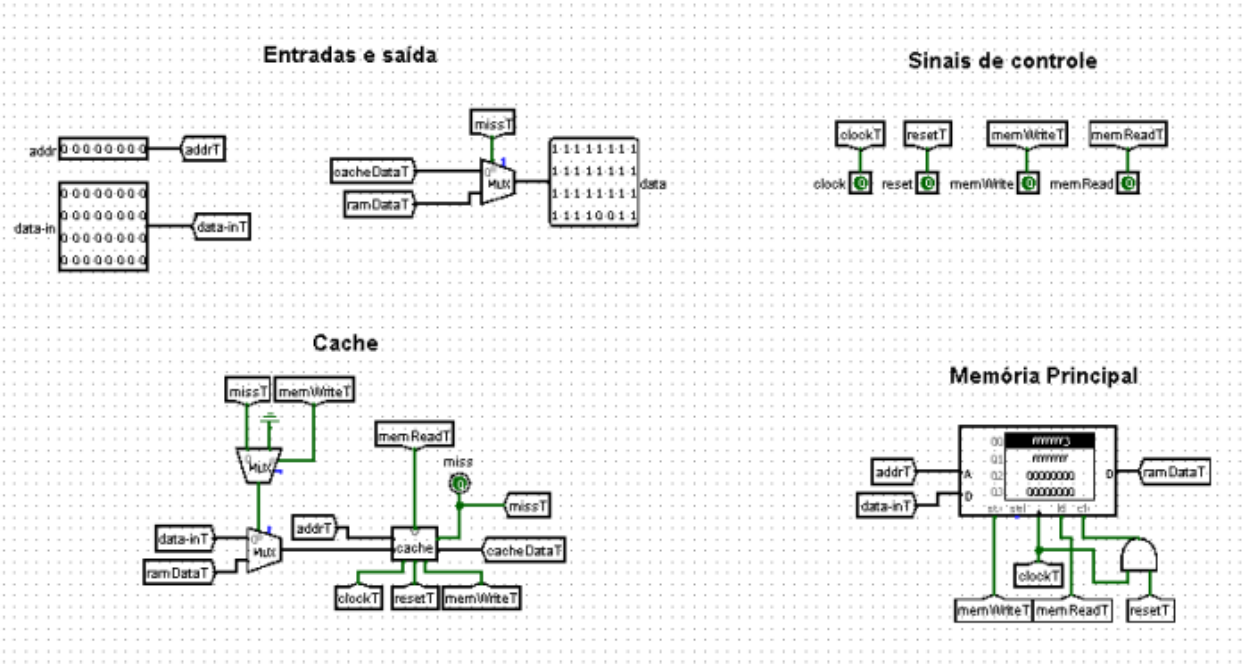

Imagem 3. Visão Geral da memória

Ainda, foi implementado uma lógica de contadores para efetivamente apresentar a quantidade de acessos à *Cache* e *RAM* separadamente, além de um contador para a quantidade de *misses* detectados. Se há uma leitura, e não há um *miss*, apenas o *Cache Accesses* é acrescido de um. Nos outros casos, ambos os contadores são acrescidos em um.

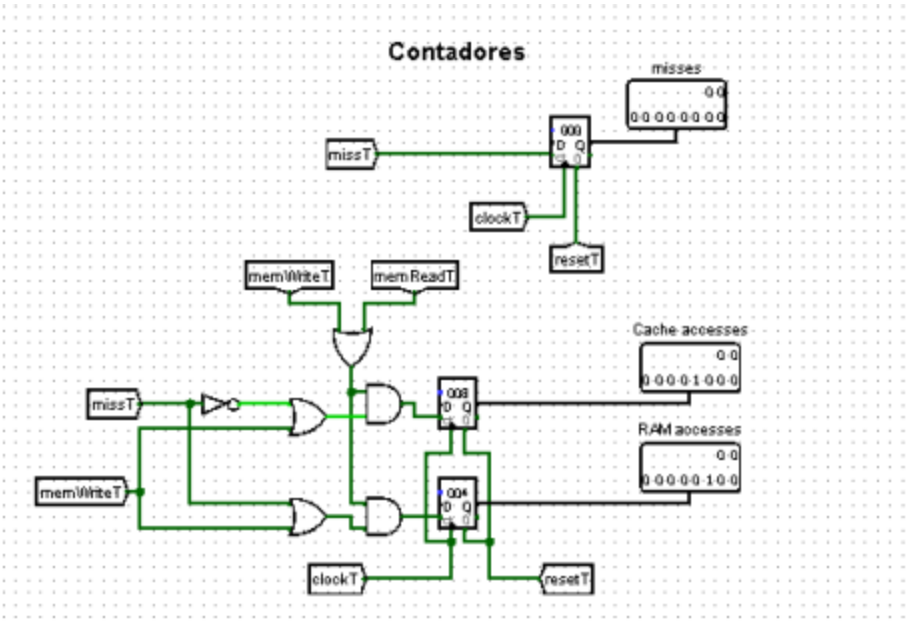

Imagem 4. Contadores

Por fim, os circuitos desenvolvidos foram adicionados ao principal, de forma a integrar a *cache* na unidade central de processamento. Foram acrescentadas à saída da memória: Uma saída para *miss*, uma para visualizar a quantidade de *misses*, uma para quantidade de acessos à *RAM* e outra quantidade de acessos à *cache*. E à entrada da memória foi adicionado um *bit* de reset para fins de teste.

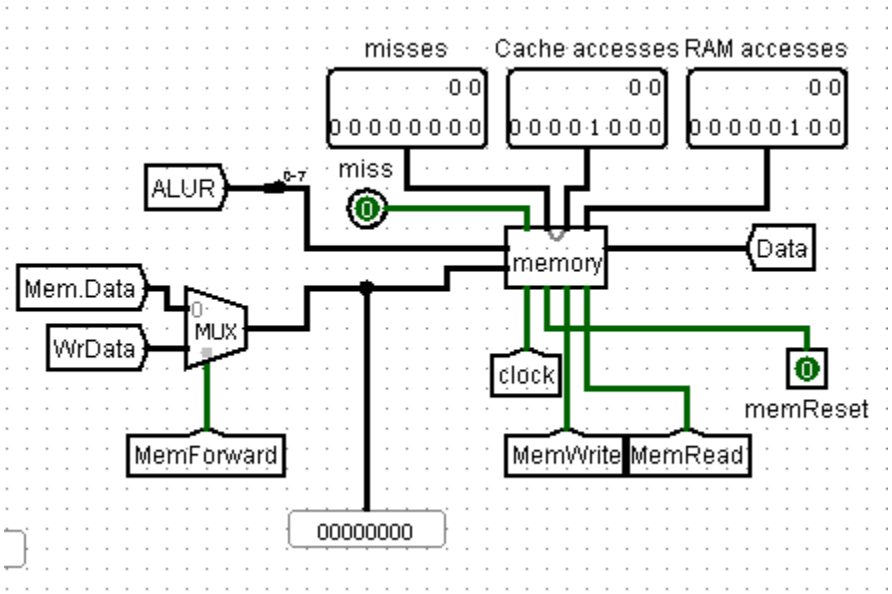

Imagem 5. Memória integrada à *CPU*

DISCUSSÃO E CONCLUSÃO

Foi possível reproduzir uma implementação satisfatória de uma memória de acesso rápido de 4 vias no *Logisim*, com uma organização de mapeamento direto e política de escrita de *write-through*. É importante dizer que essa memória *cache* não é exatamente a mesma proposta inicialmente: a política de *write-through* foi implementada, a escrita na *entry* específica utilizando o *miss* também. Os contadores também foram adicionados a fim de garantir um maior controle do sistema bem como formas de depuração. Ainda, executamos testes que podem ser conferidos na pasta de testes em um arquivo a ser executado no *Logisim* e outro que explica, de forma didática, o propósito de cada teste.

REFERÊNCIAS BIBLIOGRÁFICAS

Wikipédia, a enciclopédia livre. Cache. **2021**. Disponível em: <https://pt.wikipedia.org/wiki/Cache> Acesso em 12 de Dezembro de 2021.

Abayomi, A. O.; Olukayode, A. A.; Olakunle, G. O. An Overview of Cache Memory in Memory Management. *Automation, Control and Intelligent Systems,* **2020**; *8(3)*: 24-28. doi: 10.11648/j.acis.20200803.11

Singh, J; Shafiq, M. MIPS-Logisim. **2015**. Disponível em: <https://github.com/jsingh07/MIPS-Logisim> Acesso em 12 de Dezembro de 2021.

Davis, J; Weinman, J. Lab 11a - Build a Cache. **2013**. Disponível em: <https://curtsinger.cs.grinnell.edu/teaching/2015F/CSC211L/labs/lab11/> Acesso em 12 de Dezembro de 2021.

Bueno, C. I.; Lüders, G.; Ferreira, R. F. H. **2021**. Disponível em: <https://github.com/RichardHeise/MIPS-Pipeline-LOGISIM> Acesso em 12 de Dezembro de 2021.