Pontifícia Universidade Católica do Rio Grande do Sul

Arquitetura de Computadores II – FACIN

Augusto Bergamin

Gabriel F. Susin

Alejandro Romero Padilla

TRABALHO PRÁTICO 2 –

HIERARQUIA DE MEMÓRIA

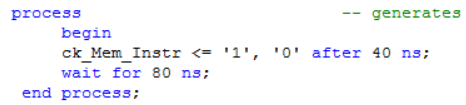
10 DE JULHO DE 2016, PORTO ALEGRE.

1. INTRODUÇÃO

Neste documento será apresentada a forma como foi desenvolvido o trabalho 3 da disciplina de arquitetura de computadores II. Foi realizada a implementação de uma hierarquia de memória na parte de instruções do processador. Para isso, criaram-se duas caches (L1 e L2) utilizando dois métodos diferentes, direto e associativo, respectivamente. Inicialmente temos uma cache L1, menor e mais rápida, utilizando o mesmo clock do processador e com tamanho para 512 bits de dados. Já a L2, por sua vez, constitui-se de uma memória mais lenta, porém maior, tendo 2048 bits para dados.

1. DESENVOLVIMENTO
   1. CPU

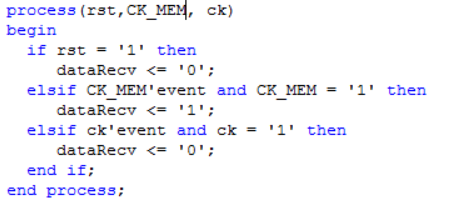
Inicialmente foi feita a implementação de um atraso na memória, para isso foi necessário criar um clock novo que fosse de 4x mais lento que o da CPU.



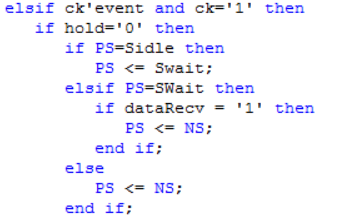
Era necessário que a CPU esperasse os dados chegarem, uma vez que o clock da memória era mais lento. Para isso criou-se um estado a mais na máquina de estados da CPU, chamado Swait.



Foi adicionado um processa que controlará a passagem deste estado para os próximos, travando assim a CPU quando necessário.

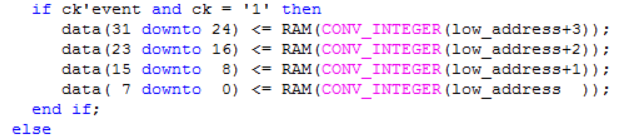


Como pode ser visto, quando tivermos um clock da memória significa que um dado foi transmitido então ativamos um sinal de dataRecv, no próximo clock da CPU ele é baixado. Este sinal é utilizado no process de controle da máquina de estado, como pode ser visto a seguir:



Agora uma vez que entramos no estado de espera, só sairemos quando um dado for recebido.

Por final foi colocado o clock na memória de instruções:



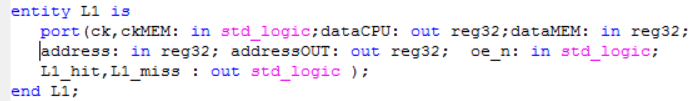
* 1. CACHE L1

A implementação da cache L1 é feita no formato direto, com 1 bit de validade, 26 bits de tag e 4 palavras de 32 bits, somando 155 bits por linha. Isso foi implementado na entidade de funções auxiliares.



Temos assim um tipo chamado CACHE\_L1, com 4 linhas de 155 bits.

Como portas das entidades teremos:



* Ck e ckMEM: São os clocks da L1(igual ao da CPU) e da memória principal;
* dataCPU: registrador de saída para a CPU;
* dataMEM: registrador de entrada da memória;
* address: endereço vindo do processador;
* addressOUT: endereço de saída para a memória;
* oe\_n: sinal correspondente ao reset da CPU;
* L1\_hit, L1\_miss: sinais de saída indicando hit ou miss da cache.

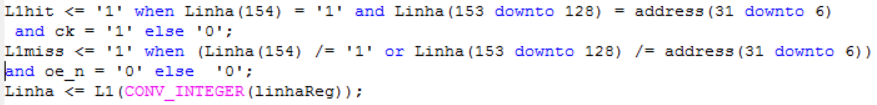
O endereço de entrada é dividido da seguinte forma:

|  |  |  |  |
| --- | --- | --- | --- |
| TAG (26 BITS) | LINHA (2 BITS) | PALAVRA (2 BITS) | 00 |

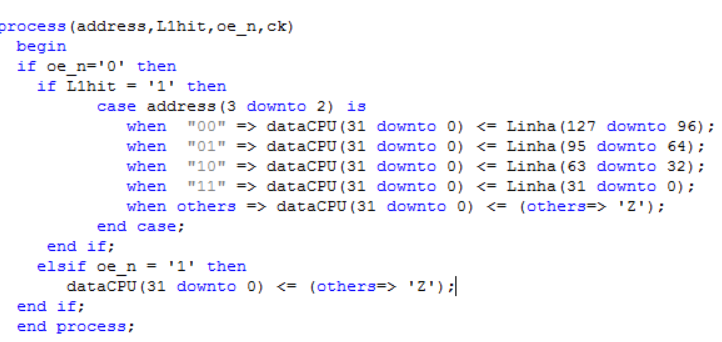
A linha é colocada em um sinal ainda na declaração dos mesmos:



Temos um trecho de código combinacional que a salva a linha escolhida em um std\_logic\_vector de 155 bits. Esse sinal tem seu bit de validade comparado com 1 assim como os bits correspondentes à TAG para verificar se temos um HIT ou um MISS.



Feito isso temos um process de envio dos dados para a CPU:

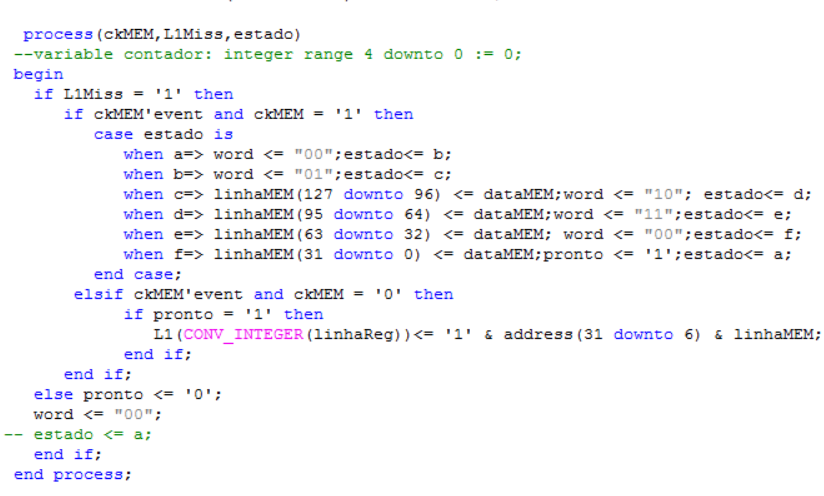


Nele, caso tenhamos o oe\_n igual a ‘1’, ou seja, com reset da CPU, manteremos o dataCPU em Z. Caso contrario, verificaremos se temos um hit, dessa forma, quando tivermos, verificaremos a palavra presente no endereço para poder enviar o dado certo para a CPU.

O endereço que vai para memória é escrito de maneira combinacional, concatenando os 28 bits mais altos do address com um sinal word e dois 00. Esse sinal word é variante, para permitir pegar todo o conteúdo correto da memória.



Por fim, temos o process que controla a busca na memória. Esse process possui uma máquina de estados que muda o word, mudando assim o endereço enviado, e também salva os dados recebidos da memória.

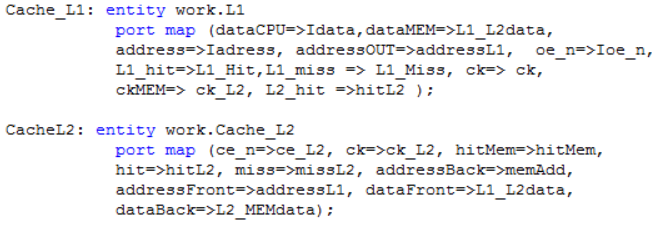


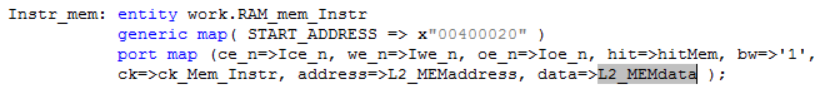
Os dois primeiros estados fazem apenas variar o endereço. No terceiro estado já teremos dados corretos a serem recebidos no barramento de entrada, assim capturamos esses dados e salvamos em um std\_logic\_vector auxiliar. Quando chegamos ao último estado, levantamos uma flag de pronto, na próxima borda de decida do clock da memória, salvaremos esse sinal auxiliar na cache. A máquina de estados é movida usando a borda de subida do clock da memória e só está ativa quando tivermos miss. Por fim, quando tivermos o hit, baixamos o sinal de pronto.

No testbench foi feito o portmap da L1, colocando o MISS da mesma como ce da memória e ligando as portas da memória à L1 quando tivermos o resetCPU em ‘0’, caso contrário as portas que alimentam a memória são as mesmas de sempre, para permitir o preenchimento dela quando inicializado.

* 1. CACHE L2
  2. CACHE L1 E L2

Uma vez implementadas as duas caches, foi necessário junta-las.





Foi feito o portmap dos das duas caches, para conectadas utiliza-se de sinais intermediários. Conectamos a entrada de endereços da L2 com a saída de endereços da L1, a saída de dados da L2 com a entrada de dados da L1. A saída de endereços da L2 vai para a entrada da RAM de instruções.

A saída de endereços da L2 vai para um mux, onde se escolhe entre a saída da L2 ou o endereço gerado pela CPU, caso tenhamos reset.



A entrada de dados da RAM também pode vir da CPU, em caso de reset.



O ce da RAM vira o sinal de miss da L2 negado. Isso pois a memória só será ativada quando tivermos um miss na cache. O ce da L2 será então o miss da L1.



1. ANÁLISE DOS RESULTADOS
   1. PROG1
      1. MIPS COM ATRASO NA MEMORIA
      2. MIPS COM CACHE L1
      3. MIPS COM HIERARQUIA COMPLETA
   2. PROG2
      1. MIPS COM ATRASO NA MEMORIA
      2. MIPS COM CACHE L1
      3. MIPS COM HIERARQUIA COMPLETA
   3. PROG3
      1. MIPS COM ATRASO NA MEMORIA
      2. MIPS COM CACHE L1
      3. MIPS COM HIERARQUIA COMPLETA
   4. PROG4
      1. MIPS COM ATRASO NA MEMORIA
      2. MIPS COM CACHE L1
      3. MIPS COM HIERARQUIA COMPLETA