Relatório pratica 3 – Algoritmo Tomasulo

Arthur Severo de Souza
Victor Le Roy Matos

1. Introdução

Este trabalho apresenta e relata a construção do protocolo MSI Snooping de coerência de cache em Verilog. Foram implementados os módulos necessários para simular a mudança de etiquetas entre as caches L1 de três processadores distintos com quatro blocos de memória diretamente mapeados, cada um contendo estado, tag e dados. As intruções para teste foram explicitadas diretamente no código.

2.1 Desenvolvimento

Para este projeto optamos por utilizar instruções de 9 bits, sendo elas:

2 bits	1 bit	3 bits	3 bits
Proc.	Op (R/W)	Tag	Dados

(Sendo a Read → 0 e Write → 1)

Além disso, tivemos que alterar os valores previsto pelo código de teste para adaptar às condições do projeto.

Blocos

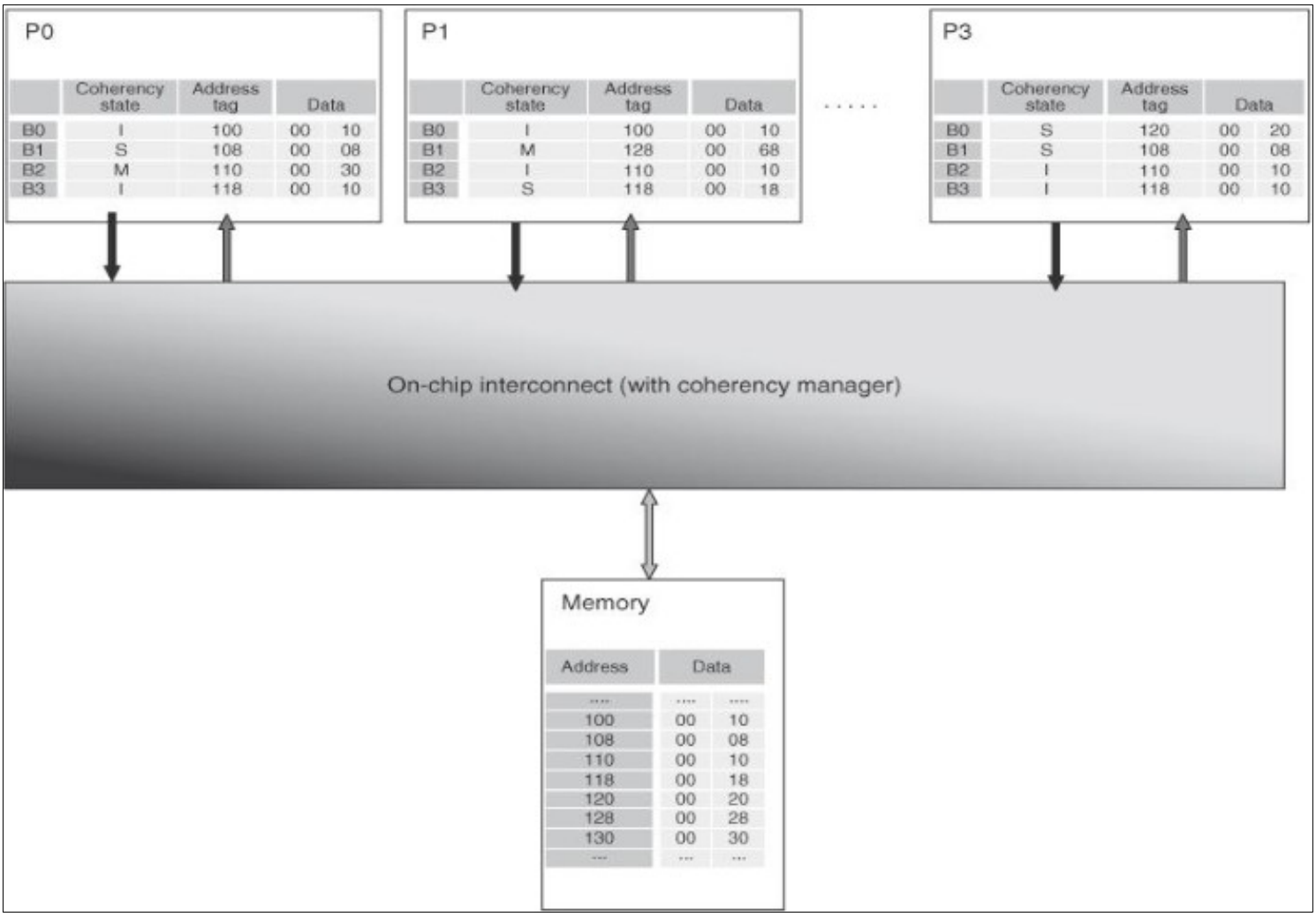
#0 100 → B0 → 000
#1 108 → B1 → 001
#2 110 → B2 → 010
#3 118 → B3 → 011
#4 120 → B0 → 100
#5 128 → B1 → 101
#6 130 → B2 → 110

Valores

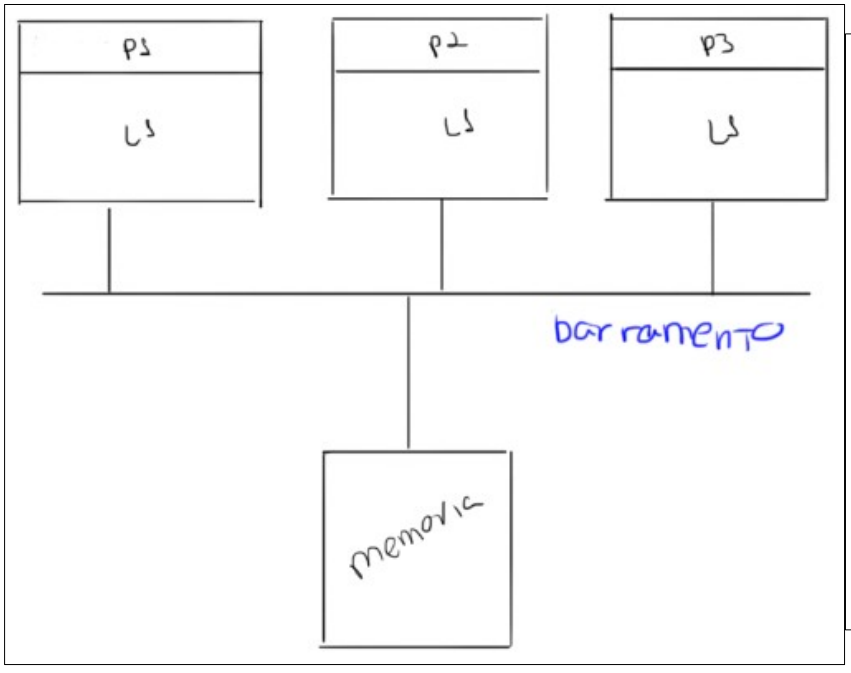
#0 08/68 → 000
#1 10 → 001
#2 18 → 010
#3 20/28 → 011
#4 30 → 100
#5 48 → 101
#6 78 → 110
#7 80 → 111

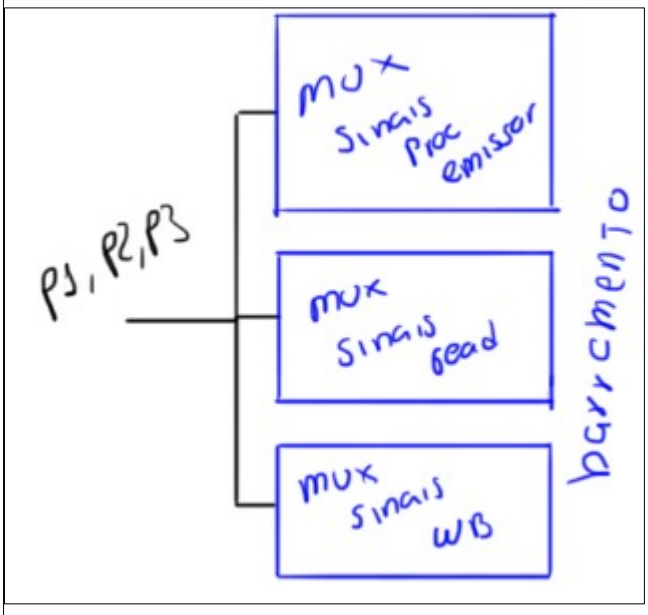
2.2 Módulos

Foi criado um diagrama a partir do esquema do algoritmo disponibilizado no livro da disciplina (Referência 1), além do arquivo do código teste.



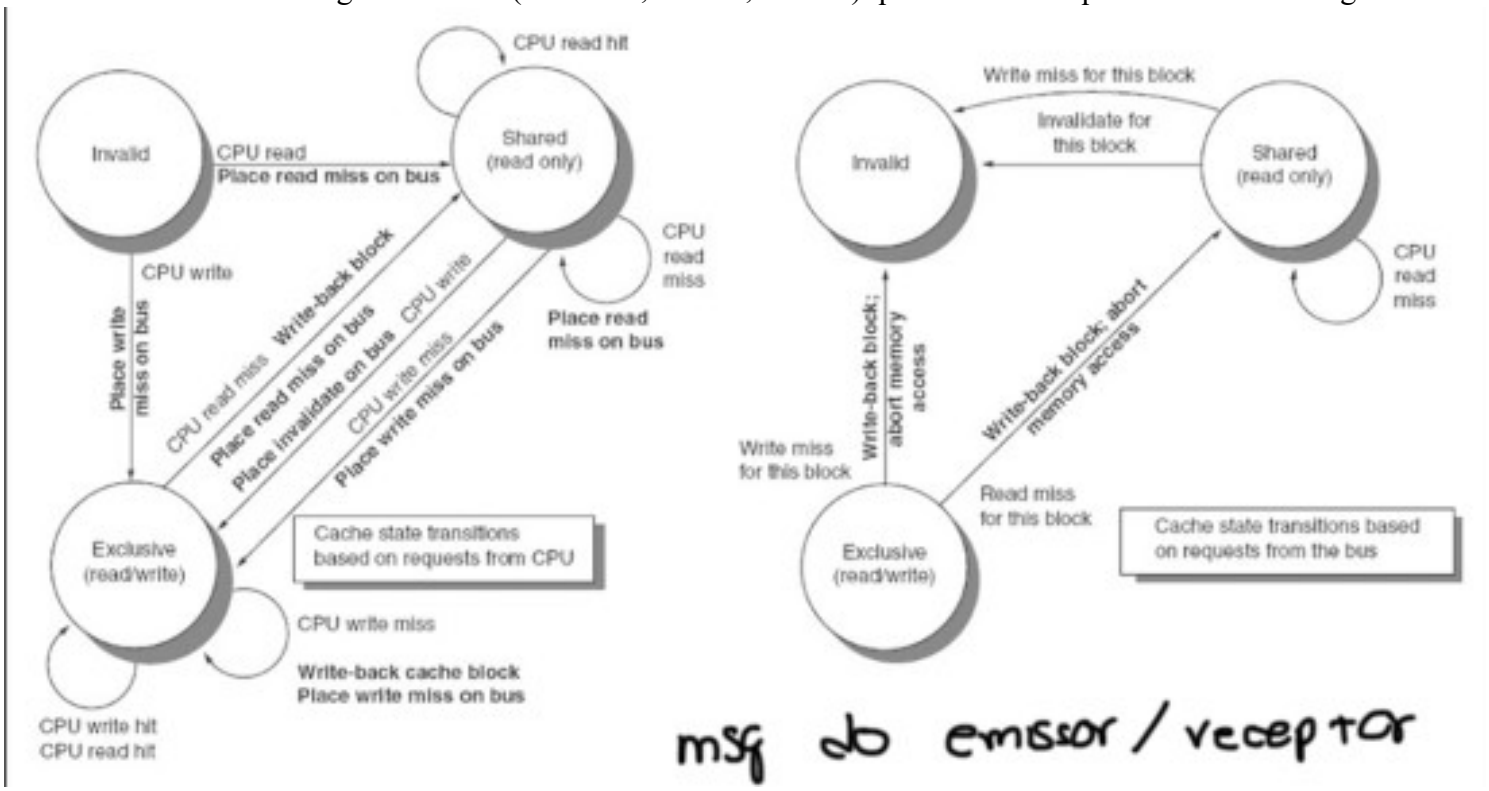
A partir do esquema, foram definidos os seguinte diagramas





- Processadores:
  Realiza operações de *read* e *write*, armazena os blocos da cache e manipula as máquinas de estado.
- Máquinas de estado (Emissor e receptor)
  A partir de valores recebidos, retorna as mudanças necessárias em um bloco de cache para preservar a coerência entre as caches e a memória.
- Memória
  Armazena os valores à depender de suas tags.
- Barramento (Bus)
  Escolhe o processador emissor e distribui suas saídas.
- Mux de sinais (WB/Read)
  Escolhe o processador que vai utilizar a memória.

Foi utilizado o algoritmo MSI (Modified, Shared, Invalid) que utiliza a máquina de estados a seguir



2.3 Funcionamento

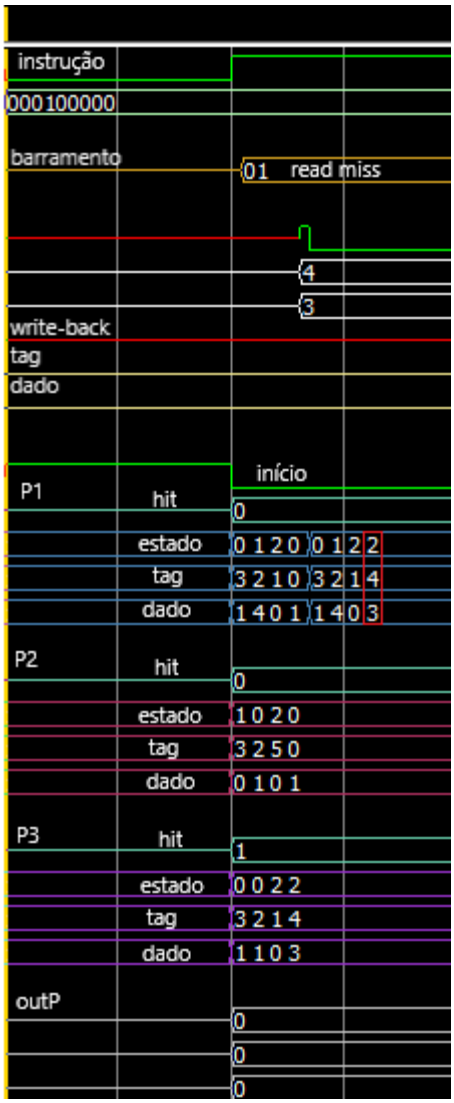
Enviamos manualmente uma instrução ao programa, em seguida, o processador escolhido "estuda" a instrução e define se ela é *read* ou *write*. Caso encontre a tag em sua cache, ele verifica a validade do bloco e realiza a operação requisitada. A seguir, a máquina emissora é chamada e são realizadas as operações de transição de estados e o resultado é distribuído para o resto dos módulos. Os receptores dos sinais verificam se existe a tag desejada e realiza as operações a partir da saída da máquina receptora.

3. Simulações

Utilizamos a sequência de instruções disponibilizadas pela professora.
A primeira linha de cada item representa a instrução e as linhas seguinte são o resultado esperado.
As mudanças significativas estão marcadas entre bordas vermelhas.

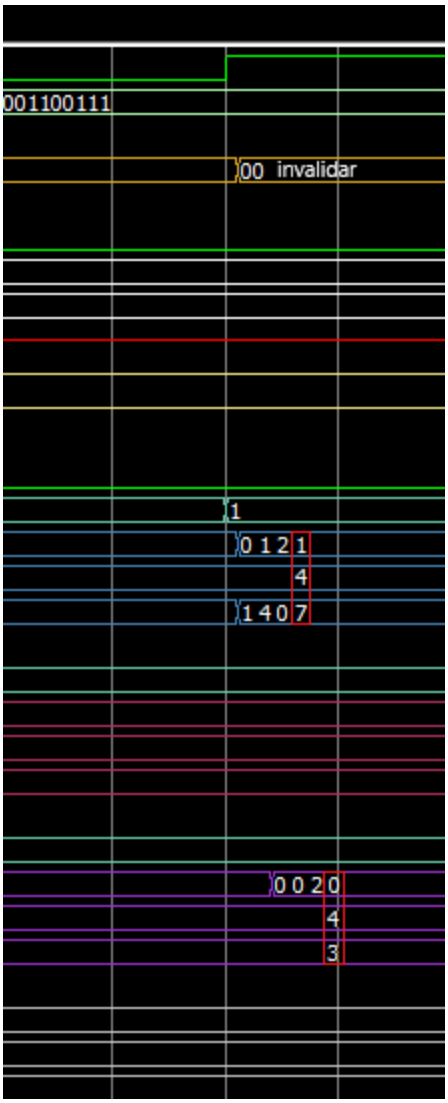
Clock → borda de descida, 200 ps, 50 duty

a) P0: read 120
P1.0(2, 4, 3)



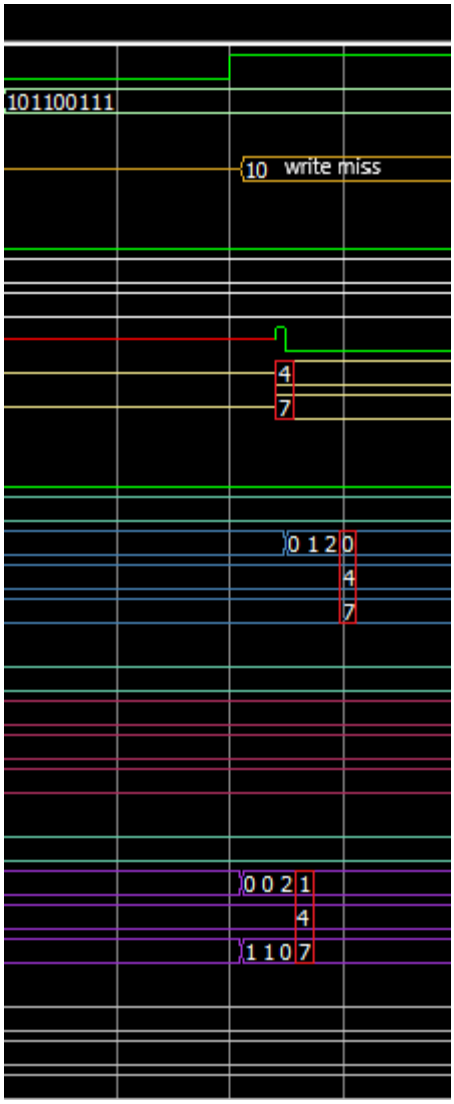
P1 → Read miss no barramento, busca o bloco 120 (#4) com valor 20 (#3) da memória e aloca no bloco B0, o estado passa a ser compartilhado (#2)

b) P0: write 120 <- - 80
P1.0(1, 4, 7)
P3.0(0, 4, 3)



P1 → Write hit, invalidar no barramento, escreve no B0 a tag 120 (#4) e o valor 80 (#7), o estado passa a ser modificado (#1)
P3 → Recebe invalidar, o bloco B0 de tag 120 (#4) passa a ser invalido (#0)

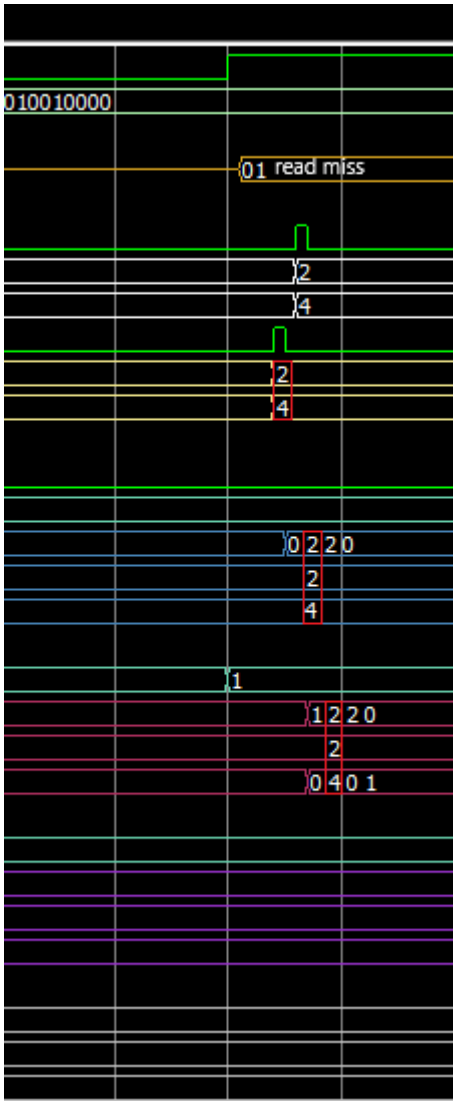
c) P3: write 120 <--80
P1.0(0, 4, 7)
P3.0(1, 4, 7)
M(4, 7)



P3 → Write miss no barramento, busca a tag 120 (#4) na cache de P1, aloca no bloco B0 e escreve o valor 80 (#7), o estado passa a ser modificado (#1)
P1 → Recebe invalidar e write-back, o bloco B0 de tag 120 (#4) passa a ser invalido (#0)
M → Sofre write-back na tag 120 (#4) com o valor 80 (#7)

d) P1: read 110

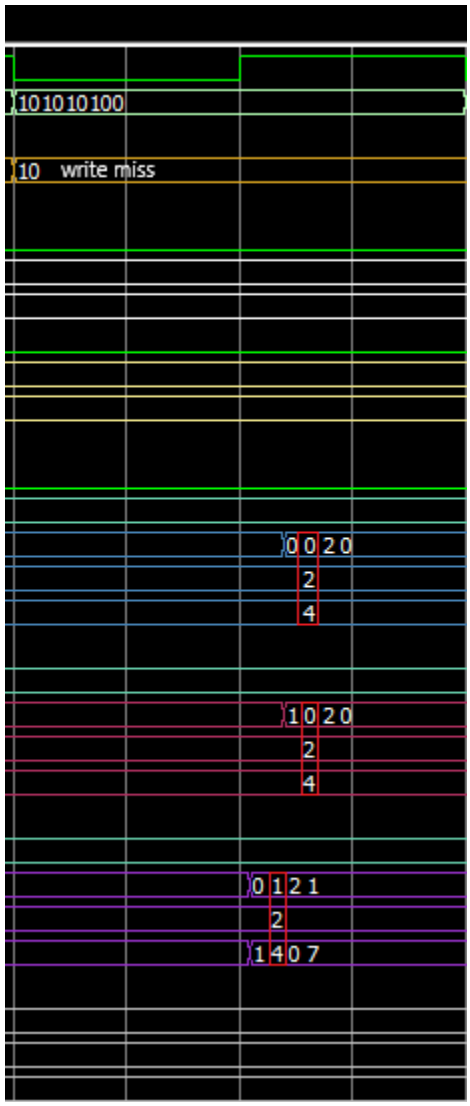
P1.2(2, 2, 4)
P2.2(2, 2, 4)
M(2, 4)



P2 → Read miss no barramento, busca a tag 110 (#2) na cache de P1 e aloca no bloco B2 com o valor 30 (#4), o estado passa a ser compartilhado (#2)
P1 → Recebe read miss e write-back, o estado do bloco B2 de tag 110 (#2) passa a ser compartilhado (#2)
M → Sofre write-back na tag 110 (#2) com o valor 30 (#4)

e) P3: write 110 <-- 30

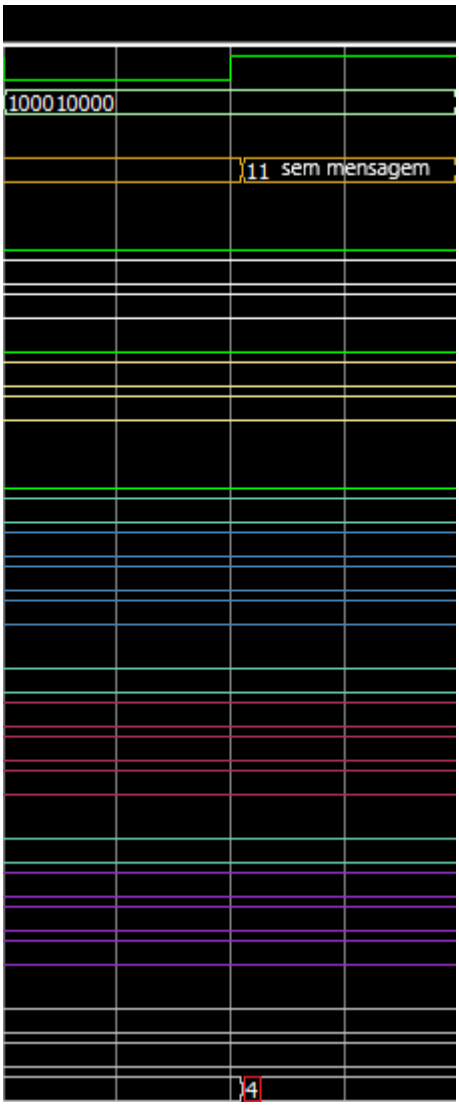
P1.2(0, 2, 4)
P2.2(0, 2, 4)
P3.2(1, 2, 4)



P3 → Write miss no barramento, busca a tag 110 (#2) na memória, aloca no bloco B2 e escreve o valor 30 (#4), o estado passa a ser modificado (#1)
P1 → Recebe invalidar, o bloco B2 de tag 110 (#2) passa a ser invalido (#0)
P2 → Recebe invalidar, o bloco B2 de tag 110 (#2) passa a ser invalido (#0)

f) P3: read 110

outp3(4)

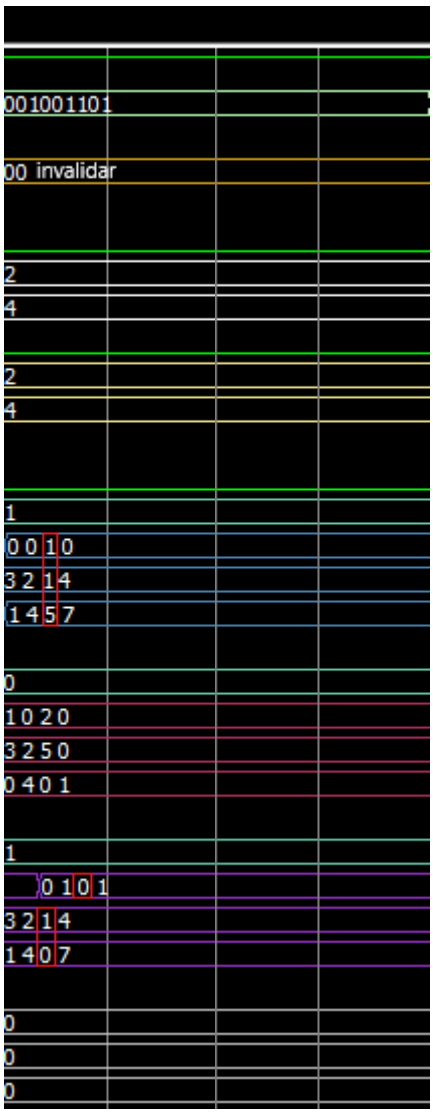


P3 → Read hit, retorna o valor 30 (#4) para o processador

g) P0: write 108 < -- 48

P1.1(1, 1, 5)

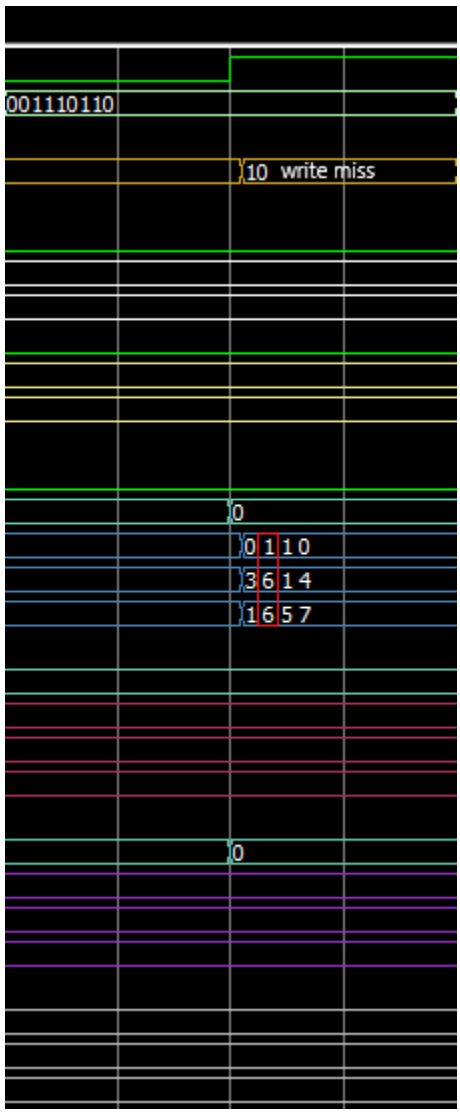
P3.1(0, 1, 0)



P0 → Write hit, escreve no bloco B1 de tag 108 (#1) o valor 48 (#5)
P3 → Recebe invalidar, o bloco B1 de tag 108 (#1) passa a ser invalido (#0)

h) P0: write 130 <--78

P1.2(1, 6, 6)



P0 → Write miss no barramento, busca a tag 130 (#6) na memória, aloca no bloco B2 e escreve o valor 78 (#6), o estado passa a ser modificado (#1)

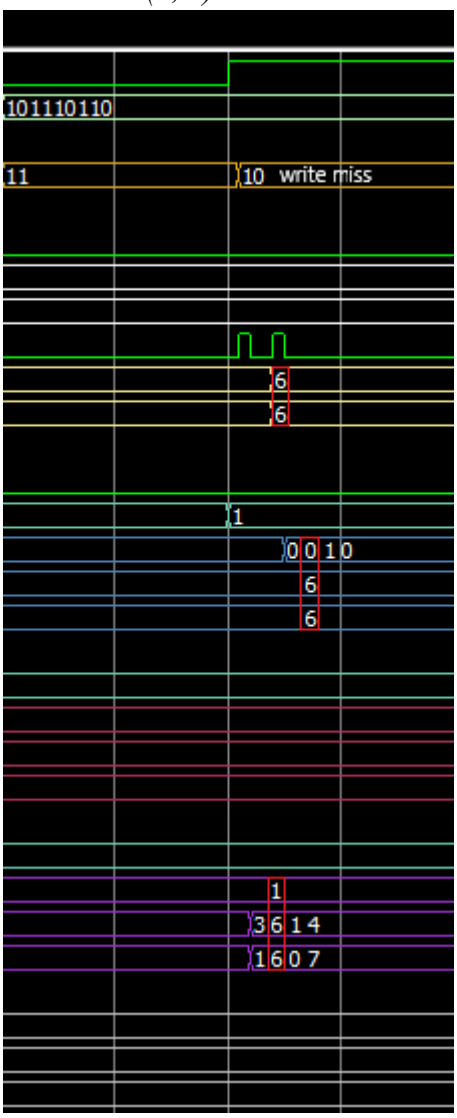
i) P3: write 130 <--78

M(6, 6)

P1.2(0, 6, 6)

P3.2(1, 6, 6)

M(2, 1)



P3 → Write miss no barramento, busca a tag 130 (#6) na cache de P0, aloca no bloco B2 e escreve o valor 78 (#6), o estado passa a ser modificado (#1)
P0 → Recebe invalidar e write-back, o bloco B2 de tag 130 (#6) passa a ser invalido (#0)
M → Sofre write-back na tag 110 (#2) com o valor 10 (#1) **(Vindo da cache de P3)**, então, sofre write-back na tag 130 (#6) com o valor 78 (#6) **(Vindo da cache de P0)**

4. Dificuldades

Inicialmente planejamos estabelecer apenas um módulo representante de ambas máquinas de estados, porém, devido à um *bug* não identificado decidimos separar o emissor do receptor. Além disso, tivemos dificuldades para definir delays adequados e enviar sinais corretos aos processadores (write-back, sinal de máquina, etc…). Para este último, resolvemos usando *muxes* que controlam a entrada de valores na memória principal.

5. Sugestões

No geral, acreditamos que a maior das complicações encontradas nas práticas é o uso do Quartus e do ModelSim como principais ferramentas de trabalho, assim como o entendimento da lógica necessária para o uso do Verilog. Uma sugestão seria a procura de um novo conjunto de softwares capazes de simular projetos em linguagem de descrição de hardware e algumas aulas dedicadas à linguagem Verilog em si.

6. Comentários adicionais

Apesar dos desafios e a dificuldade de implementação, este projeto, em conjunto com os estudos do livro, nos auxiliou a compreender melhor o algoritmo de coerência de cache por *snooping*.