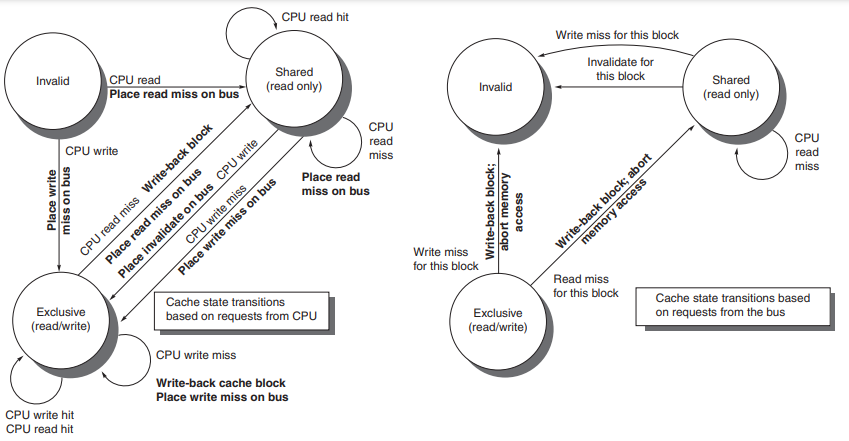
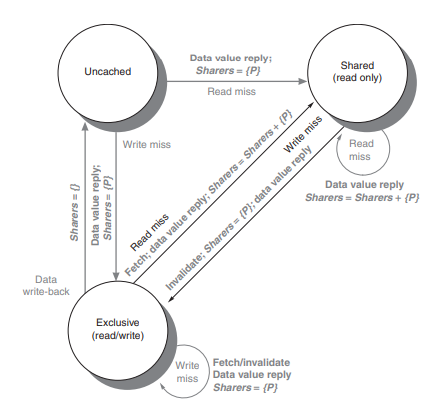
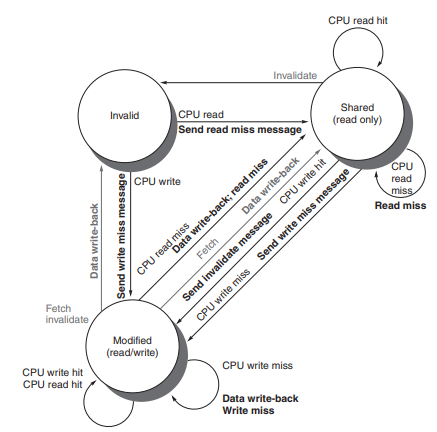
**Snooping**



**Diretório:**

****

Primitivas Básicas: Primitivas capazes de ler e escrever um local de memória de forma indivisível.

Operação básica: atomic exchange (troca atômica)

* Permuta um valor de um registrador por um valor da memória.
  + “0” é usado para indicar que o lock está liberado. “1” é usado para indicar que o lock não está disponível. Retorno: “1” se outro processador já solicitou o acesso e “0” caso contrário. Duas trocas simultâneas serão ordenadas.
* Test-and-set: Testa um valor e marca o valor se ele passou no teste.
* Fetch-and-increment: Retorna um valor de um local de memória e o incrementa.

-> Não pode haver instrução entre a leitura e a escrita.

* Store conditional: retorna “1” se foi bem sucedido e “0” caso contrário. Load linked: retorna o valor inicial

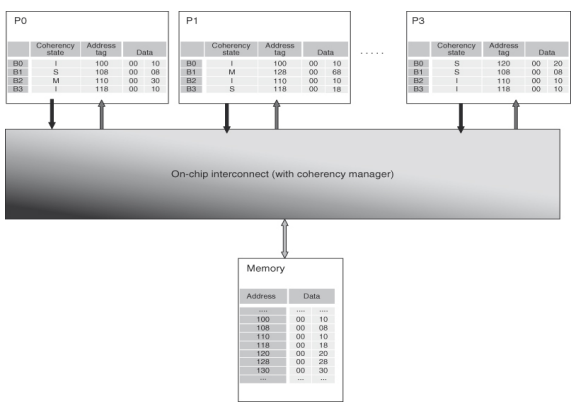
| Troca atômica de R4 (“1”) por R1 (“0”):  try:  MOV R3, R4 ; mov exchange value R4-> R3  LL R2, 0 (R1) ; load linked  SC R3, 0 (R1) ; store conditional  BEQZ R3, try ;branch store fails  MOV R4, R2 ; put load value in R4 | Spin Lock sem coerência:  DADDUI R2,R0,#1  lockit: EXCH R2,0(R1) ;atomic exchange  BNEZ R2,lockit ;already locked? | Spin Lock com coerência:  Lockit: LD R2,0(R1) ;load of lock  BNEZ R2,lockit ;not available-spin  DADDUI R2,R0,#1 ;load locked value  EXCH R2,0(R1) ;swap  BNEZ R2,lockit ;branch if lock wasn’t 0 |
| --- | --- | --- |

Spin lock: locks que um processador tenta adquirir, executando um loop até consegui-lo.

* Sem coerência: manter as variáveis de lock na memória Tentar adquirir o lock realizando uma operação atômica. Testar se a troca retorna o lock como liberado Para liberar, armazena “0” no lock
* Com coerência: colocar os locks na cache e usar os mecanismos de coerência para manter os locks coerentes
  + Realiza leituras em uma cópia local do lock, até que veja que o lock está disponível.
    - O processador corre para conseguir o lock
    - Ele tenta adquirir o lock fazendo o “swap”(troca)
    - Lê o valor antigo e armazenar 1 na variável de lock. O processador vencedor vê “0” na variável de lock e o perdedor vê “1”. O vencedor executa o código e depois coloca “0” na variável de lock. Isto reinicia a “race condition” (condição de corrida).

| Lockit usando coerência lockit: LL R2,0(R1) ;load linked  BNEZ R2,lockit ;not available-spin  DADDUI R2,R0,#1 ;load locked value  SC R2,0(R1) ;store  BEQZ R2,lockit ;branch if lock wasn’t 0 |  | 2 DESVANTAGENS de se aumentar o paralelismo em nível de requisição em um WSC:  1. Sobrecarga de gerenciamento: À medida que o número de requisições paralelas aumenta, também aumenta a complexidade do gerenciamento dessas requisições. Isso pode incluir o agendamento de tarefas, a sincronização de dados e a resolução de conflitos. Essa sobrecarga adicional pode levar a uma diminuição no desempenho geral do sistema.  2. Aumento do consumo de energia: O processamento paralelo pode levar a um aumento no consumo de energia, pois mais unidades de processamento estão ativas ao mesmo tempo. Isso pode aumentar os custos operacionais e ter um impacto negativo no meio ambiente |
| --- | --- | --- |

Desafios enfrentados para reduzir consideravelmente o uso de energia? 1. Melhorar a eficiência energética de alimentação do WSC 2. Carga dos servidores em comparação com a energia necessária 3. Melhorar o gasto energético com refrigeração



**a) P0: read 120** (read hit)

P0.B0 (S, 120 0020)

**b) P0: write 120 <- - 80 (escreve 80 na posição 120)**

P0.B0 (M, 120, 0080)

P3.B0 (I, 120 0020)

**c) P3: write 120 <--90** (write-miss)

P3.B0 (M, 120 0090)

P0.B0(I, 120 0080)

Mem(120, 0080)

**d) P1: read 110**

P1.B2 (S, 110 0030)

P0.B2 (I, 110, 0030)

M(110, 0030)

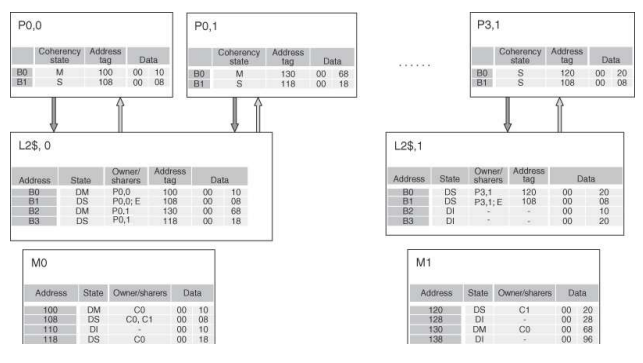
**e) P3: write 110 <-- 30**

P3.B2 (M, 110, 0030)

**f) P3: read 110**

Read Hit: Retorna valor para processador.

| **g) P0: write 108 < -- 48 (Write hit)**  P0.B1 (M, 108 0048)  P3.B1: (I, 108 0008) | **h) P0: write 130 <--78:** P0.B2(M, 130 0078) | **i) P3: write 130 <--88**  Mem (110, 0030) -> WB 110  P3.B2 (M, 130 0078)  P0.B2 (I, 130 0078)  Mem (130, 0078) -> WB P0 |
| --- | --- | --- |



**a) P0,0: read 100 (**read hit)

Retorna dado para o processador.

**b) P0,0: read 128**

P0,0.B2 (S, 128 0028)

M0 (108, DS, C1 0008)

L2,1.B1 (DS, P3,1, 108 0008)

L2,0.B1 (DS, P0,0, 128 0028)

M1 (128, DS C0 0028)

**c) P0,0: write 128 < -- 78** (write hit)

P0,0.B1 (M, 128, 0078)

L2,0.B1 ( DM, P0,0, 120 0078)

M0 (128, DM, C0 0078)

**d) P0,0: read 120** (read miss)

P0,0.B0 (S, 120, 0020)

WB M (100 0030)

M0(100 DI, 0030)

L2,0.B0(DS, 120, P0,0 E, 0020)

L2,1.B0 ()DS, 120, P3,1 E, 0020

M1 (120, C0,C1, 0020)

| **e) P0,1: read 120**  P0,1.B0(S, 120, 0020)  L2,0.B2(DI, -, 130 0068)  L2,0.B0 (DS P0,0, P0,1 E, 120 0020 )  M1 (DI, -, 130 0068) | **f) P0,1: write 120 <-- 80**  P0,0.B0(I, 120 0020)  P3,1.B0(I, 120 0020)  P0,1.B0(M, 120 0080)  L2.0.0B0(DM, P0,1, 120 0080)  L2.1(DI,-,120 0020) | M1 (120, DM, C0, 0080) | **g) P0,0: write 120 < -- 90 (wm)**  P0,0 .B0(M,120,0090) P0,1 (I, 120 0080) L2.B0(DM, P0,0, 120 0090) WB (120 0080) M1(120 DM C0 0090) | **h) P0,1: read 120**  WB (120 0090)  P0,0 B0(S, 120 0090)  P0,1.B0(S,.120 0090)  L2.0.B0 (DS, P0,0, P0,1, 120 0090 )  M1 (120 DS C0 120 0090 |
| --- | --- | --- | --- |