Lista IV

Nome: Gustavo de Assis Xavier

1.a) P0: read 120
P0: read 128
P0: read 130
=> P0: read 120 (read-miss)
P0.130 (S, 120 0020) -> memória
=> P0: read 128 (read-miss)
P0.133 (S, 128 00 (08) -> cache de P1
=> P0.read 130 (read miss) -> memória, WB 110

I1: Logo, temos 100 (memória) + 40 (cache) + 10 + 100 (memória) + 10 (WB) = 260 ciclos de stall. (Escrever o bloco de volta a memória)

Implementação 2: 100 + 130 + 10 + 100 + 10 = 350 stall's.

b) P0: read 100 (read-miss) -> memória
P0: write 108 ← 48 (write-hit) -> invalidação
P0: write 130 ← 78 (write-miss) -> memória - WB 110

Implementação 1: 100 + 15 + 100 + 10 = 22 stall's.
Implementação 2: 100 + 15 + 100 + 10 = 225 stall's.

c) P1: read 120 (read-miss) -> memória
P1: read 128 (read-hit)
P1: read 130 (read-hit) -> memória
I1: 100 + 100 = 200 stall's.
I2: " = 200 stall's.

d) P1: read 100 (read-miss) -> memória
P1: write 108 ← 48 (write-miss) -> memória, WB 128
P1: write 130 ← 78 (write-miss) -> memória
I1 = I2 = 100 + 100 + 10 + 100 = 310 stall's

2) P0,0 : write 100 ← 80
Não afeta outros.

b) P0,0 : write 108 ← 88
P0,0 . B1 fica modificado
P3,1 recebe invalidação

c) P0,0 : write 118 ← 90
P0,0 write miss
P0,1 recebe invalidação.

d) P0,1 : write 128 ← 98
P0,1 write miss

3) P0 adiquire o lock → busca na memória → 100
seção crítica → 1000
P0 escreve → WB → → 10
→ P1 espera
1150 x stal

P1 adiquire o lock → cache → 40
seção crítica → 1000
P1 escreve → WB → 10

P3 espera
1150 + 1000 + 10 + 4
= 2200 x stall

P3 adiquire o lock → cache → 40
seção crítica → 1000

D S T Q Q S S

4) O while executa depois que A=2000, logo, C=2000.

b) Se read A for mais rápido que read Flag ou se write Flag chegar em P2 mais rápido que a escrita de A.

c) Garantir que P1 seja executada em ordem. Para isso, podemos usar lock com coerência.

5) O problema é que para economizar energia se deve colocar o sistema em modo ocioso, mas isso pode gerar problemas em picos repentinos de demanda.