

Zentralübung Rechnerstrukturen

Lösungsblatt 7: Vektorrechner und Cache-Kohärenz

1 Vektorrechner

```
1 LV   V1, Ra           # int a[n] in V1 laden
2 LV   V2, Rb           # int b[n] in V2 laden
3 mov  R1, 1            # R1 mit 1 initialisieren
4 CVI  V3, R1           # Create Vektor Index
5                                     # 0, R1 * 1, R1 * 2, ...
6                                     # entspricht for (i=0;i<n,i++)
7
8 SLTV.D V1, V2         # compare elements with 'Less Than'
9                                     # if true 1 in Vektor Mask Register
10                                    # else 0 in Vektor Mask Register
11                                    # if (a[i] < b[i])
12
13 SUBV.D V2, V2         # alle Stellen mit 1 im VMR
14                                     # auf 0 setzen
15 ADDV.D V2, V3         # und dann mit den Werten aus V3 auffuellen
16                                     # B[i] = i;
17
18 CVM                    # Clear Vektor Mask
19                                     # Alle Eintraege im VMR auf 1 setzen
20
21 SV   Rb, V2           # V2 speichern
```

2 Cache

2.1 Cacheleistung I

$$a) t_a = t_{L1} * r_{H1} + (1 - r_{H1}) * (t_{L2} * r_{H2} + t_{HS} * (1 - r_{H2}))$$

$$b) \text{ Alternative A: } t_a = 10ns * 70\% + (1 - 70\%) * (30ns * 40\% + 100ns * (1 - 40\%))$$

$$t_a = 7ns + 0.3 * (12ns + 60ns) = 28.6ns$$

$$\text{ Alternative B: } t_a = 12ns * 75\% + (1 - 75\%) * (25ns * 35\% + 100ns * (1 - 35\%))$$

$$t_a = 9ns + 0.25 * (8.75ns + 65ns) = 27,4375ns$$

Die durchschnittliche Zugriffszeit in Entwurfsalternative B ist geringer, somit ist diese Entwurfsalternative zu wählen.

2.2 Cacheleistung II

Alternative A:

$$t_A = 0.8 * 2.5 ns + 0.2 * (0.9 * 12.5 ns + 0.1 * 112.5 ns)$$

$$t_A = 2 ns + 0.2 * (11.25 ns + 11.25 ns) = 2 ns + 0.2 * 22.5 ns$$

$$t_A = 2 ns + 4.5 ns = 6.5 ns$$

Alternative B:

$$t_B = 0.7 * 2 ns + 0.3 * (0.9 * 10 ns + 0.1 * 100 ns)$$

$$t_B = 1.4 ns + 0.3 * (9 ns + 10 ns) = 1.4 ns + 0.3 * 19 ns$$

$$t_B = 1.4 ns + 5.7 ns = 7.1 ns$$

Die durchschnittliche Zugriffszeit in Entwurfsalternative A ist geringer, somit ist diese Entwurfsalternative zu wählen.

2.3 Beweise

- a) Gegeben Sei ein 2-fach assoziativer Cache mit 2 Blöcken, sowie ein 4-fach assoziativer Cache mit 1 Block. Beide Caches haben also eine Größe von 4 Einträgen.

Weiterhin seien sechs Datenwörter mit den folgenden Adressen gegeben: A, B, C, D, E, F, wobei A, B, E, F auf den ersten Block des 2-fach assoziativen Cache, C und D auf den zweiten Block abgebildet werden. Alle 6 Adressen werden auf den Block des 4-fach assoziativen Cache abgebildet.

Folgende Zugriffsfolge erzeugt bei 4-fach assoziativen Cache mehr Misses, als beim 2-fach assoziativen Cache: $ABCDABEFCD(ABEFCD)^n$

- b) Gegeben Sei ein Direkt-Mapped-Cache, sowie ein vollassoziativer Cache, beide haben eine Kapazität von 4 Cachezeilen. Weiterhin seien 5 Datenwörter mit den folgenden Adressen: A, B, C, D und E, wobei A auf die erste Cachezeile, B auf die zweite Cachezeile, C auf die dritte Cachezeile und D und E auf die vierte Cachezeile abgebildet werden.

Folgende Zugriffsfolge erzeugt bei vollassoziativen Cache mehr Misses, als beim direkt abgebildeten Cache: $ABCDE(ABCDE)^n$

2.4 Verständnisfragen

- a) Räumliche und Zeitliche Lokalität (90 / 10-Regel)
- b) *Conflict-, Capacity-, Compulsory-Misses*
Im Fall von Shared-Memory-Systemen existiert noch der *Coherence-Miss*, welcher wiederum in *True-Sharing-Miss* und *False-Sharing-Miss* unterteilt werden kann.
- c) SRAM-Zellen benötigen viel Chipfläche, deswegen ist der Aufbau des Hauptspeichers aus SRAM-Zellen entweder zu teuer oder man könnte nur geringe Kapazitäten anbieten.

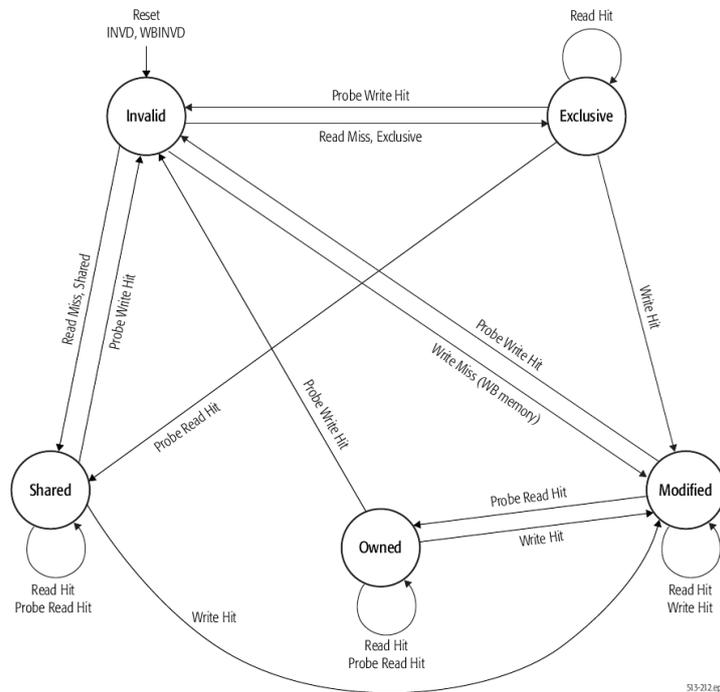
3 Cache-Kohärenzprotokolle

3.1 MESI

| Prozessor | Aktion | Prozessor 1 | | Prozessor 2 | | Prozessor 3 | |
|-----------|--------|-------------|--------|-------------|--------|-------------|--------|
| | | Line 1 | Line 2 | Line 1 | Line 2 | Line 1 | Line 2 |
| | init | - | - | - | - | - | - |
| 1 | rd 6 | 6/E | | | | | |
| 2 | rd 2 | | | 2/E | | | |
| 1 | rd 4 | | 4/E | | | | |
| 3 | rd 4 | | 4/S | | | 4/S | |
| 2 | rd 3 | | | | 3/E | | |
| 3 | wr 7 | | | | | | 7/M |
| 1 | wr 4 | | 4/M | | | 4/I | |
| 2 | rd 7 | | | 7/S | | | 7/S |
| 3 | wr 5 | | | | | 5/M | |
| 1 | rd 3 | 3/S | | | 3/S | | |
| 3 | wr 3 | 3/I | | | 3/I | | 3/M |
| 2 | wr 7 | | | 7/M | | | |

3.2 MOESI

a)



| | Prozessor | Aktion | Prozessor 1 | | Prozessor 2 | | Prozessor 3 | |
|----|-----------|--------|-------------|--------|-------------|--------|-------------|--------|
| | | | Line 1 | Line 2 | Line 1 | Line 2 | Line 1 | Line 2 |
| | | init | - | - | - | - | - | - |
| | 1 | rd 4 | 4/E | | | | | |
| | 3 | rd 4 | 4/S | | | | 4/S | |
| | 2 | rd 3 | | | 3/E | | | |
| | 2 | wr 5 | | | | 5/M | | |
| b) | 1 | rd 2 | | 2/E | | | | |
| | 2 | wr 4 | 4/I | | 4/M | | 4/I | |
| | 2 | rd 1 | | | | 1/E | | |
| | 3 | rd 4 | | | 4/O | | 4/S | |
| | 3 | rd 3 | | | | | | 3/E |
| | 1 | wr 1 | 1/M | | | 1/I | | |
| | 3 | rd 1 | 1/O | | | | 1/S | |

c) MOESI:

Lesend: 6 Zugriffe

Schreibend: 1 Zugriff

MESI:

Lesend: 8 Zugriffe

Schreibend: 3 Zugriffe

Es würden zwei Lesezugriffe und zwei Schreibzugriffe eingespart → Leistungssteigerung

vorhanden

3.3 Verständnisfragen

a) Ein Zustandsübergang von O nach S kann es in einem Write-Back Invalidate-Protokoll nicht geben. Gemäß einem Write-Back Invalidate-Protokoll werden bei Veränderung eines geteilten Datums, die Datums in den entfernten Caches invalidiert und demzufolge dem Datum den Zustand M zugewiesen.

Der Wechsel von Zustand S in den Zustand O , müsste eine Aktualisierung des Datum in den entfernten Caches nach sich ziehen. Diese Vorgehensweise entspricht einem Write-Back Update-Protokoll

b) In DSM-Systemen existiert kein gemeinsamer Speicherbus, den eine Snooping-Logik überwachen könnte.

c) In DSM-Systemen kommen verzeichnisbasierte Cache-Kohärenzprotokolle zum Einsatz.