

Vorlesung Rechnerstrukturen

□ **Kapitel 2: Architektur und Mikroarchitektur von Mikroprozessoren**

- ◆ **2.9 Registerorganisation**
- ◆ **2.10 Cache-Speicher**
- ◆ **2.11 Speicherorganisation**

Vorlesung Rechnerstrukturen

□ **Kapitel 2: Architektur und Mikroarchitektur von Mikroprozessoren**

♦ **2.10 Cache-Speicher**

- ♦ **Organisation von Caches**
- ♦ **Laden von Caches**
- ♦ **Adressieren von Caches**
- ♦ **Aktualisierungsstrategie und**
- ♦ **Cache-Kohärenz: MESI-Protokoll**

Aktualisierungsstrategie und Cache-Kohärenz

□ Kohärenz:

- ◆ Korrektes Voranschreiten des Systemzustandes durch ein abgestimmtes Zusammenwirken der Einzelzustände.

□ Cache-Kohärenz:

- ◆ Prozessor muss auf aktuelle Inhalte im Cache wie im Hauptspeicher zugreifen können;

Aktualisierungsstrategie und Cache-Kohärenz

□ Cache-Kohärenz:

- ♦ Befehls-Caches: Prozessor führt nur Lesezugriffe durch.
- ♦ Daten-Cache: Prozessor führt auch Schreibzugriffe durch.
- ♦ Aktualisieren des Cache-Speichers oder des Hauptspeichers oder beide (write hit):
 - ♦ Aktualisierungsstrategie
- ♦ Lesezugriffe dürfen nicht auf inzwischen veraltete Daten gehen;

□ Ein System ist **konsistent**, wenn alle Kopien eines Datenwortes im, Hauptspeicher und den verschiedenen Cache-Speichern identisch ist.

Aktualisierungsstrategie und Cache-Kohärenz

□ Aktualisierungsstrategien

- ◆ Durchschreibe-Strategie (Write-Through-Strategie)
- ◆ Rückschreibe-Strategie (Copy-Back-Strategie)
 - ◆ gewährleisten Cache-Kohärenz bei nur einem Master im System
 - ◆ unterschiedlich hoher Hardware-Aufwand
 - ◆ manche Prozessoren bieten beide Varianten an
 - ◆ Eintrag in Seitendeskriptoren der Speicherverwaltungseinheit

Aktualisierungsstrategie und Cache-Kohärenz

□ Aktualisierungsstrategie Write-Through

- ◆ bei Schreibzugriffen wird grundsätzlich der Hauptspeicher aktualisiert;
- ◆ bei einem Schreibtreffer (write hit) wird auch der Cache aktualisiert;
- ◆ die Hauptspeicher- und Cache-Speicherinhalte stimmen immer überein, aber der Cache verliert bei Schreibzugriffen seinen Vorteil;

Aktualisierungsstrategie und Cache-Kohärenz

□ Aktualisierungsstrategie Write-Through

◆ Buffered-Write-Through

- ◆ Pufferspeicher zwischen Cache und Hauptspeicher;
- ◆ die Cache-Steuerung kann nächsten Cache-Zugriff starten, bevor das Durchschreiben abgeschlossen ist;
- ◆ Vorteil bei unmittelbar folgenden Lesezugriffen;
- ◆ einfache Realisierung mit geringerem Hardware-Aufwand;
- ◆ Überlappen der Programmausführung und der Hauptspeicheraktualisierung;

Aktualisierungsstrategie und Cache-Kohärenz

□ Aktualisierungsstrategie Write-Through

- ◆ Variante “No-Write-Allocation”

- ◆ bei Write-Miss wird nur der Hauptspeicher und nicht der Cache aktualisiert;

- ◆ Variante “Write-Allocation”

- ◆ bei Write-Miss wird zusätzlich zum Hauptspeicher auch der Cache aktualisiert;

Aktualisierungsstrategie und Cache-Kohärenz

□ Aktualisierungsstrategie Copy-Back (Write-Back)

- ♦ bei Schreibzugriffen wird grundsätzlich der Cache aktualisiert;
- ♦ Aktualisierung des Hauptspeichers erst dann, wenn eine Zeile im Cache durch einen neu zu ladenden Block überschrieben werden muß.
 - ♦ Zurückkopieren ohne Bedingung (simple copy-back)
 - ♦ Zurückkopieren wird von einem für jede Cache-Zeile vorhandenen "Dirty-Bit" abhängig gemacht (flagged copy-back);
 - ♦ **Dirty-Bit** zeigt an, ob der Inhalt einer Zeile verändert worden ist (dirty) oder nicht;

Aktualisierungsstrategie und Cache-Kohärenz

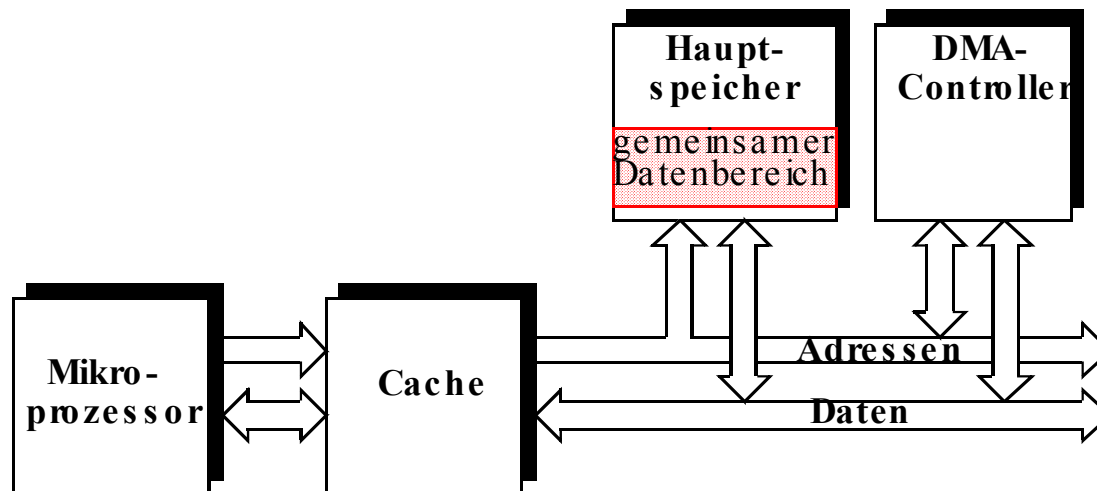
- Aktualisierungsstrategie für Caches mit je einem Valid- und einem Dirty-Bit**

Cache-Zugriff	Write-Through No-Write Alloc.	Write-Through Write-Alloc.	Copy-Back
Read-Hit	Cache-Datum --> CPU	Cache-Datum --> CPU	Cache-Datum --> CPU
Read-Miss	HS-Block, Tag --> Cache HS-Datum --> CPU 1 --> V	HS-Block, Tag --> Cache HS-Datum --> CPU 1 --> V	Cache-Zeile --> HS HS-Block, Tag --> Cache HS-Datum --> CPU 1 --> V, 0 --> D
Write-Hit	CPU-Datum --> Cache, HS	CPU-Datum --> Cache, HS	CPU-Datum --> Cache 1 --> D
Write-Miss	CPU-Datum --> HS	HS-Block, Tag --> Cache, 1 --> V CPU-Datum --> Cache, HS	Cache-Zeile --> HS HS-Block, Tag --> Cache 1 --> V CPU-Datum --> Cache 1 --> D

Aktualisierungsstrategie und Cache-Kohärenz

□ Datenkonsistenz bei zusätzlichen Masters ohne Cache

- ◆ Beispiel: Mikroprozessorsystem, das neben dem Prozessor mit Cache einen DMA-Controller als weiteren Master aufweist;
 - ◆ zusätzlicher Master hat wie Prozessor Zugriff auf Hauptspeicher
 - ◆ beide teilen sich **gemeinsamen Datenbereich (shared data)**



Aktualisierungsstrategie und Cache-Kohärenz

□ Konsistenzproblem:

- ◆ beim **Write-Through-Verfahren**:
- ◆ DMA-Controller beschreibt eine Speicherzelle, deren Inhalt im Cache als gültig eingetragen war, der Prozessor führt danach einen Lesezugriff mit der Adresse dieser Speicherzelle durch:
- ◆ --> Prozessor liest veraltetes Datum

- ◆ beim **Copy-Back-Verfahren**:
- ◆ der Prozessor führt Schreibzugriff mit der Adresse aus dem gemeinsamen Bereich aus und aktualisiert nur Cache; der DMA-Controller liest anschließend die Speicherzelle mit dieser Adresse:
- ◆ --> der DMA-Controller liest veraltetes Datum (im Hauptspeicher);

Aktualisierungsstrategie und Cache-Kohärenz

□ Lösung des Konsistenzproblems:

♦ **Non-Cachable Data**

- ♦ der vom Prozessor und dem zusätzlichen Master gemeinsam benutzte Speicherbereich wird von der Speicherung im Cache ausgeschlossen;
- ♦ Speicherverwaltung:
 - ♦ Der Adreßbereich wird in seinem für die Speicherverwaltungseinheit bereitgestelltem Deskriptor als „*non-cacheable*“ gekennzeichnet.
 - ♦ Die Cache-Steuerung wird bei Zugriffen auf den so gekennzeichneten Bereich nicht aktiv.
- ♦ Es werden auch die für Schnittstellen und Controller reservierten Adreßbereiche als „*non-cacheable*“ gekennzeichnet, um den direkten Zugriff auf deren Daten-, Steuer- und Statusregister zu gewährleisten.

Aktualisierungsstrategie und Cache-Kohärenz

□ Lösung des Konsistenzproblems:

♦ **Cache-Clear, Cache-Flush**

- ♦ Die Zugriffe von Prozessor und DMA-Controller auf den gemeinsamen Datenbereich werden von zwei unterschiedlichen Tasks ausgeführt;
- ♦ In diesem Fall kann die Task, die den DMA-Vorgang auslöst, dafür sorgen, daß der Cache gelöscht wird, d.h nachfolgende Prozessorzugriffe führen zu einem Neuladen des Cache;
 - ♦ Write-Through: Cache-Clear:
 - ♦ Die Cache-Einträge werden auf ungültig gesetzt.
 - ♦ Copy-Back: Cache-Flush:
 - ♦ Alle mit „dirty“ gekennzeichneten Einträge im Cache werden in den Hauptspeicher zurückgeschrieben, danach werden Cache-Einträge auf ungültig gesetzt.

Aktualisierungsstrategie und Cache-Kohärenz

□ Lösung des Konsistenzproblems:

♦ **Bus-Snooping**

- ♦ Die Cache-Steuerung beobachtet den Bus hinsichtlich der Speicherzugriffe anderer Master:

- ♦ Write-Through

- ♦ bei Write-Hit des anderen Masters wird der entsprechende Cache-Eintrag ungültig gesetzt.

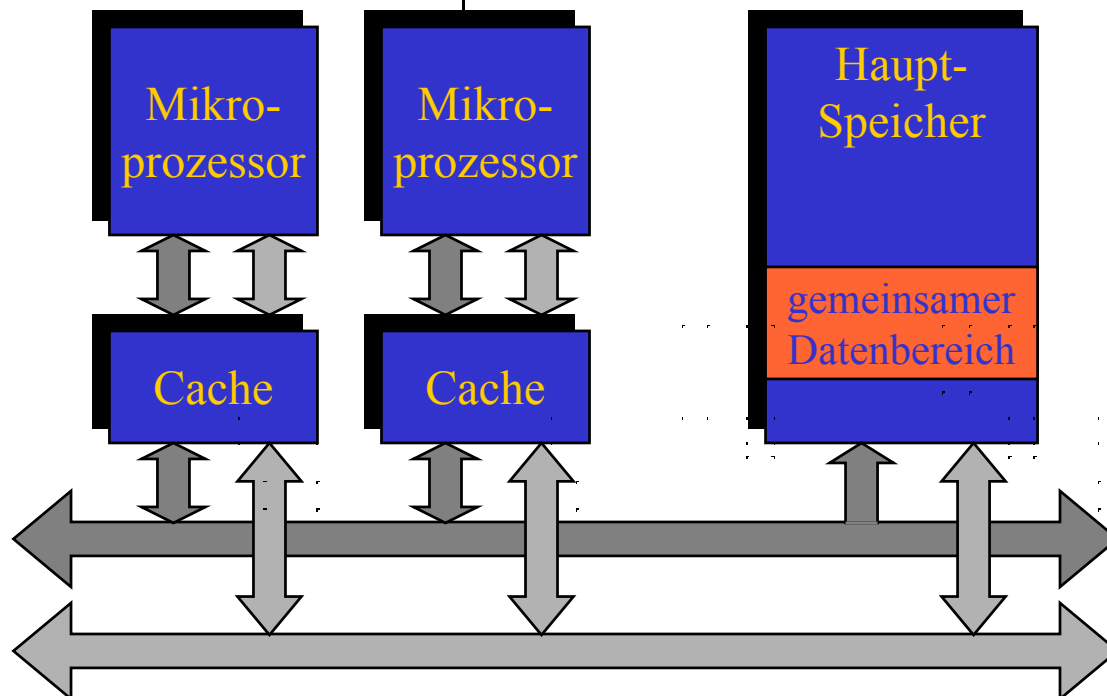
- ♦ Copy-Back

- ♦ Der Cache-Eintrag wird bei Write-Hit entweder als ungültig gekennzeichnet (Write Invalidate) oder er wird aktualisiert und als „dirty“ gekennzeichnet (Write Update);

Aktualisierungsstrategie und Cache-Kohärenz

□ Lösung des Konsistenzproblems:

- ♦ Aufwendige Techniken bei Systemen mit mehreren Mastern mit Caches
- ♦ Beispiel: Mikroprozessorsystem mit 2 Prozessoren mit Cache. Zugriff auf gemeinsame Daten im Hauptspeicher.



Aktualisierungsstrategie und Cache-Kohärenz

□ MESI-Kohärenzprotokoll

- ◆ Aktualisierungsstrategie:
 - ◆ Copy-Back-Verfahren
 - ◆ Write-Through in abgemagerter Form
 - ◆ Festlegung in Datenstrukturen der Speicherverwaltungseinheiten
- ◆ Jeder Cache verfügt über Snoop-Logik und Steuersignale:
 - ◆ Invalidate-Signal: Invalidieren von Einträgen in den Caches anderer Prozessoren.
 - ◆ Shared-Signal: Anzeige, ob ein zu ladender Block bereits als Kopie vorhanden ist.
 - ◆ Retry-Signal: Aufforderung für einen Prozessor, das Laden eines Blockes abzubrechen. Das Laden wird dann wieder aufgenommen, wenn ein anderer Prozessor aus dem Cache in den Hauptspeicher zurückgeschrieben hat.

Aktualisierungsstrategie und Cache-Kohärenz

□ MESI-Kohärenzprotokoll

- ◆ Jede Cache-Zeile ist um zwei Statusbits erweitert:
 - ◆ Zeigen Protokollzustände an:
 - Invalid (*I*)
 - Shared (*S*)
 - Exclusive (*E*)
 - Modified (*M*)

Aktualisierungsstrategie und Cache-Kohärenz

□ MESI-Kohärenzprotokoll

♦ Statusbits:

♦ Invalid (*I*): Die betrachtete Cache-Zeile ist ungültig.

- ♦ Lese- und Schreibzugriff aus diese Zeile veranlassen den Cache, den Speicherblock in die Cache-Zeile zu laden.
- ♦ Die anderen Caches, die den Bus beobachten, zeigen mit Hilfe des Shared-Bits an, ob dieser Block gespeichert ist (Shared Read Miss) oder nicht (Exclusive Read Miss).
- ♦ Es erfolgt der Übergang in den Zustand *S* oder *E*.
- ♦ Beim Write-Miss erfolgt der Übergang in den Zustand *M*. Der Prozessor gibt dabei wegen der Änderung das Invalidate-Signal aus, das von den anderen Caches ausgewertet wird.

Aktualisierungsstrategie und Cache-Kohärenz

□ MESI-Kohärenzprotokoll

- ◆ Statusbits:
 - ◆ Shared (*S*), Shared Unmodified: Der Speicherblock existiert als Kopie in der Zeile des betrachteten Caches sowie gegebenenfalls in anderen Caches.
 - ◆ Lesezugriff auf die Cache-Zeile (Read-Hit):
 - ◆ Der Zustand wird nicht verändert.
 - ◆ Schreibzugriff auf die Cache-Zeile (Write-Hit):
 - ◆ Die Cache-Zeile wird geändert und geht in den Zustand *M* über.
 - ◆ Ausgeben des Invalidate-Signals, woraufhin die Caches, bei denen diese Cache-Zeile ebenfalls im Zustand *S* ist, diese als ungültig kennzeichnen (Zustand *I*).

Aktualisierungsstrategie und Cache-Kohärenz

□ MESI-Kohärenzprotokoll

◆ Statusbits:

- ◆ Exclusive (*E*), Exclusive Unmodified: Der Speicherblock existiert als Kopie nur in der Zeile des betrachteten Caches.
 - ◆ Der Prozessor kann lesend und schreiben zugreifen, ohne den Bus benützen zu müssen.
 - ◆ Schreibzugriff: Wechseln in den Zustand M.
 - ◆ Andere Caches sind nicht betroffen.

Aktualisierungsstrategie und Cache-Kohärenz

□ MESI-Kohärenzprotokoll

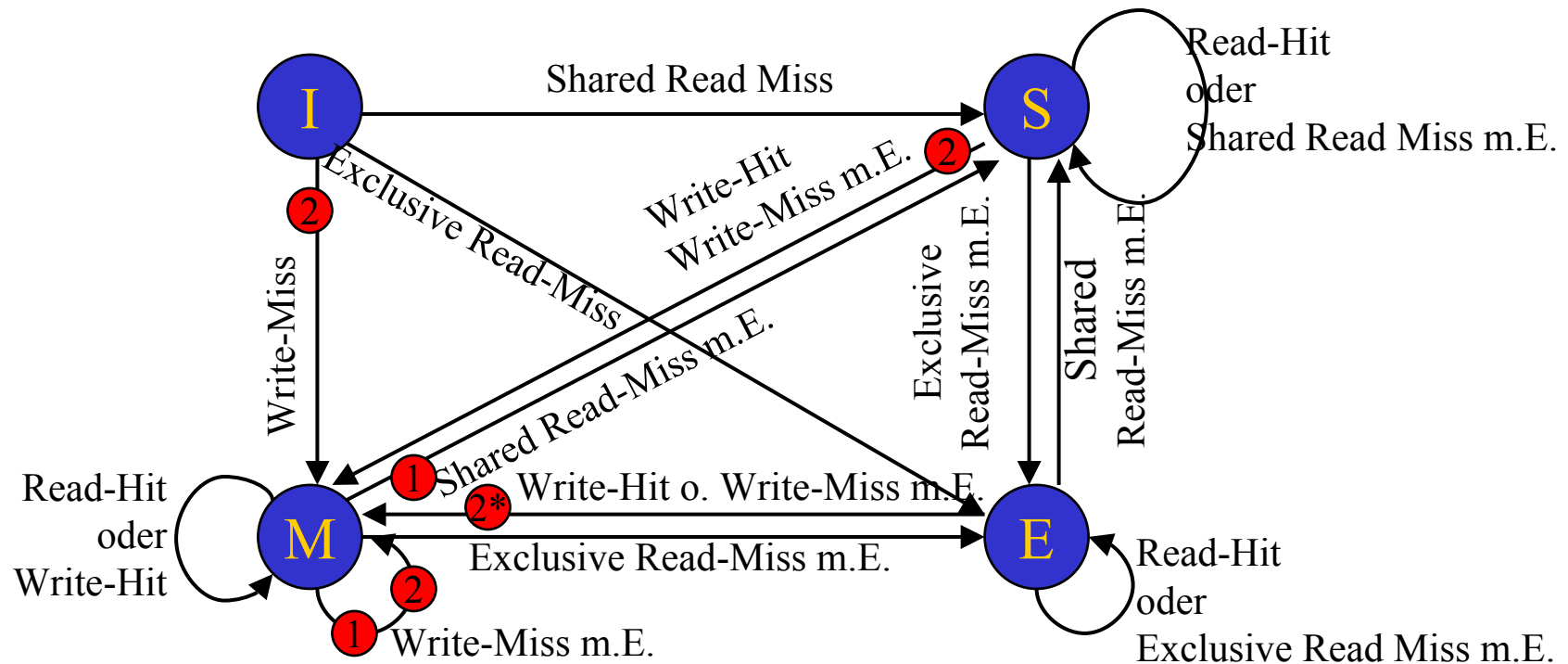
◆ Statusbits:

- ◆ Modified (*M*), Exclusive Modified: Der Speicherblock existiert als Kopie nur in der Zeile des betrachteten Caches. Er wurde nach dem Laden verändert.
 - ◆ Der Prozessor kann lesend und schreiben zugreifen, ohne den Bus benützen zu müssen.
 - ◆ Bei einem Lese- oder Schreibzugriff eines anderen Prozessors auf diesen Block (Snoop-Hit) muss dieser in den Hauptspeicher zurückkopiert werden. Es erfolgt ein Zustandübergang nach S oder nach I.
 - ◆ Der Prozessor, der diesen Block aus dem Hauptspeicher holen will, wird mit Hilfe des Retry-Signals darüber informiert, dass zunächst ein Zurückschreiben erforderlich ist.

Aktualisierungsstrategie und Cache-Kohärenz

□ Zustandsgraph des MESI-Protokolls

- ◆ Zustandsübergänge, die durch die lokalen Schreib- und Lesezugriffe ausgelöst werden (Zugriffe des eigenen Prozessors)

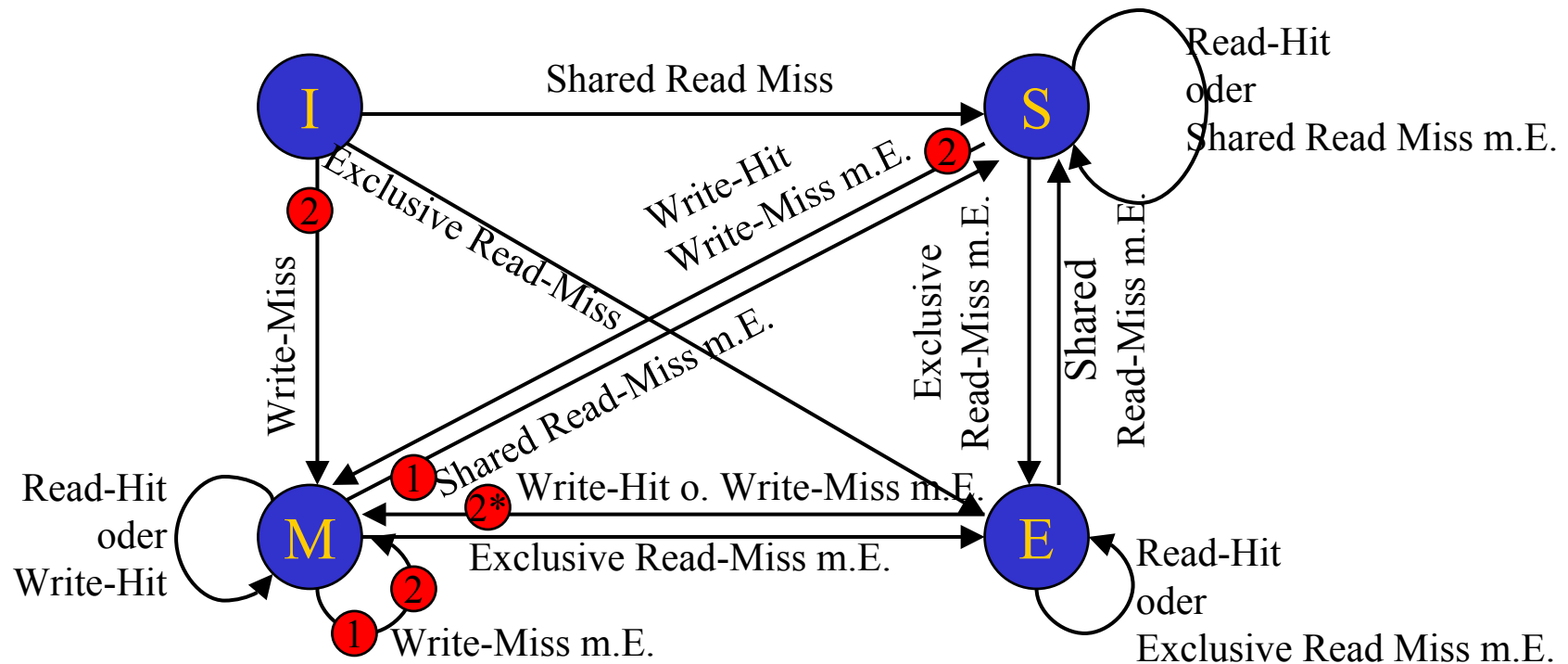


m.E.: mit Ersetzung

Aktualisierungsstrategie und Cache-Kohärenz

□ Zustandsgraph des MESI-Protokolls

- ◆ Zustandsübergänge, die durch die lokalen Schreib- und Lesezugriffe ausgelöst werden (Zugriffe des eigenen Prozessors)

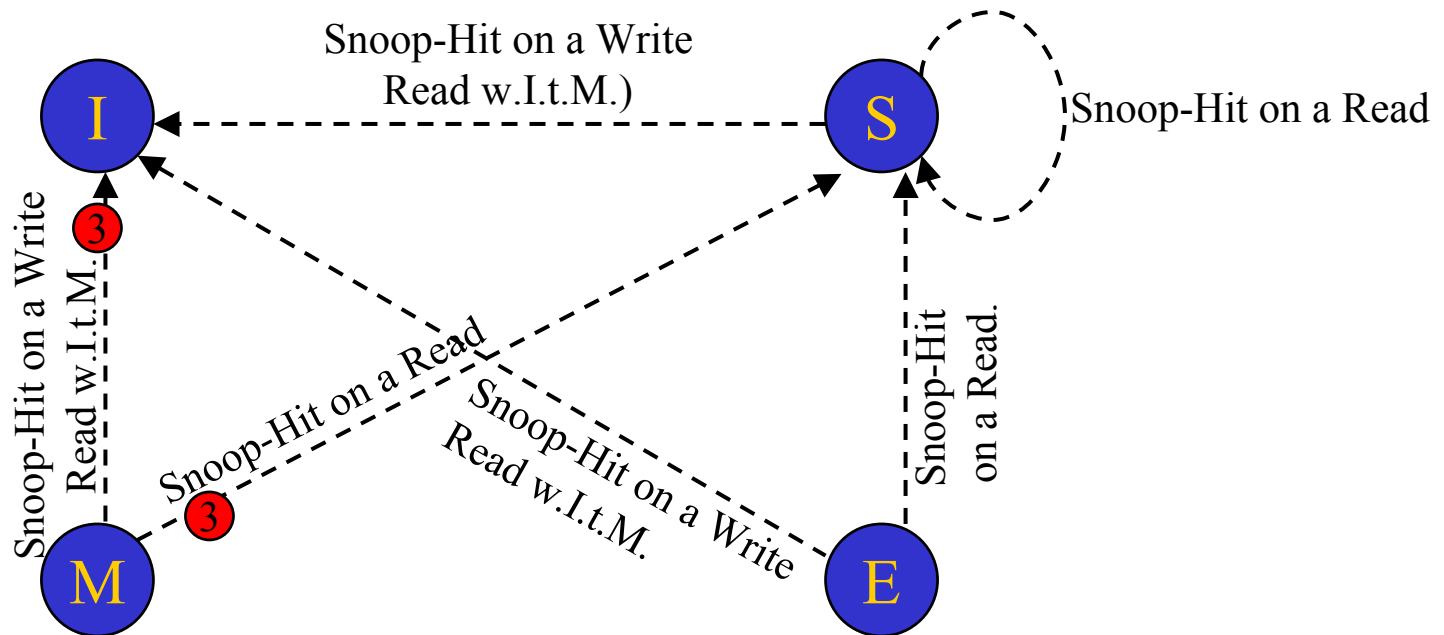


m.E.: mit Ersetzung

Aktualisierungsstrategie und Cache-Kohärenz

□ Zustandsgraph des MESI-Protokolls

- ◆ Zustandsübergänge, die durch Aktionen, die auf dem Bus zu beobachten sind, ausgelöst werden.

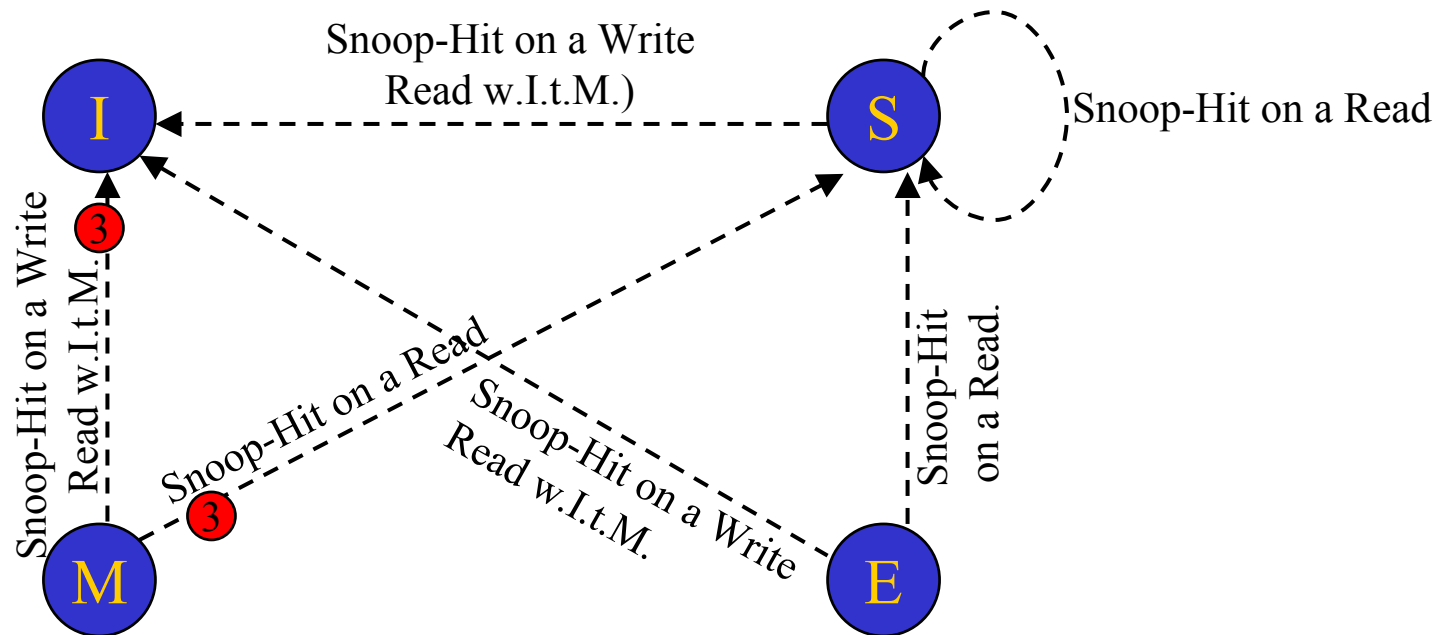


- 1 Retry-Signal wird aktiviert und danach wird die Cache-Zeile in den Hauptspeicher kopiert.

Aktualisierungsstrategie und Cache-Kohärenz

□ Zustandsgraph des MESI-Protokolls

- ♦ Zustandsübergänge, die durch Aktionen, die auf dem Bus zu beobachten sind, ausgelöst werden.

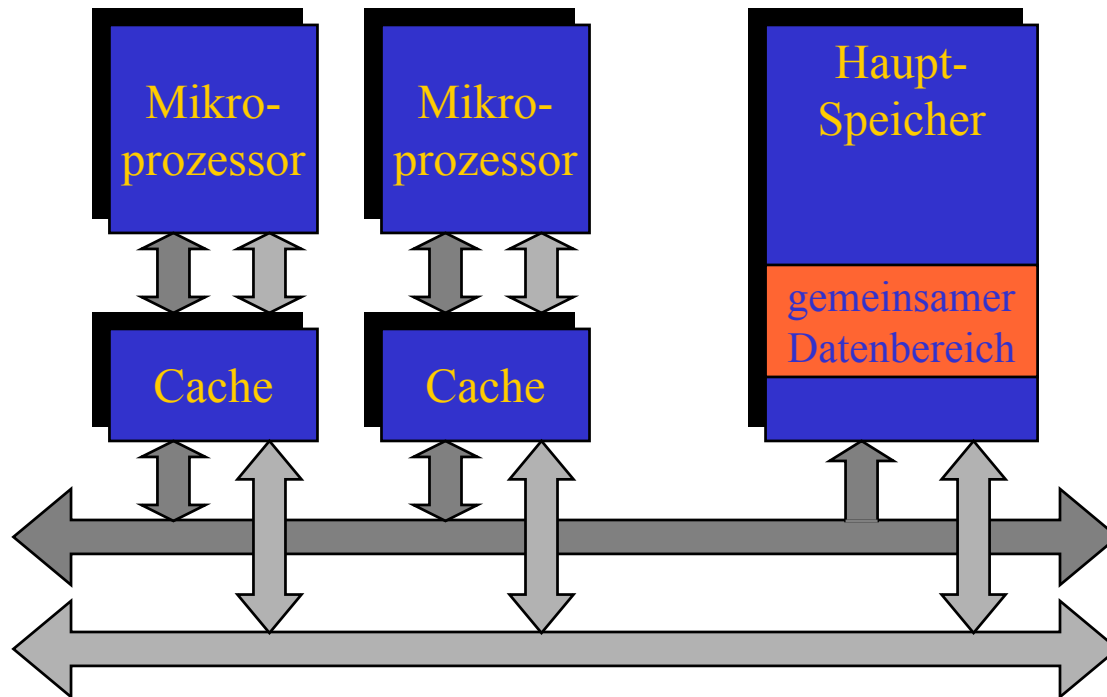


w.I.t.M.: With Intend to Modify

- ③ Retry-Signal wird aktiviert und danach wird die Cache-Zeile in den Hauptspeicher kopiert.

Aktualisierungsstrategie und Cache-Kohärenz

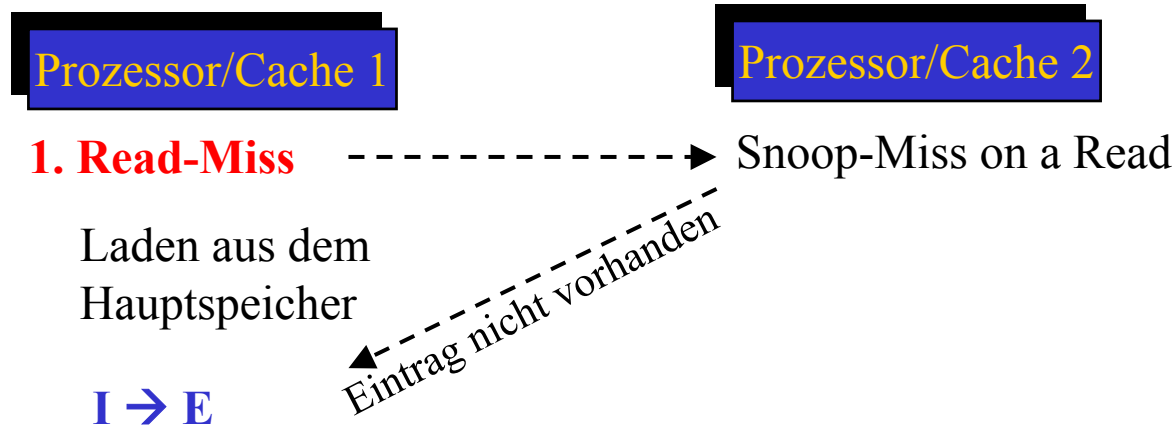
- Wirkungsweise des MESI-Protokolls
 - ◆ Beispiel: Mikroprozessorsystem mit zwei Prozessoren



Aktualisierungsstrategie und Cache-Kohärenz

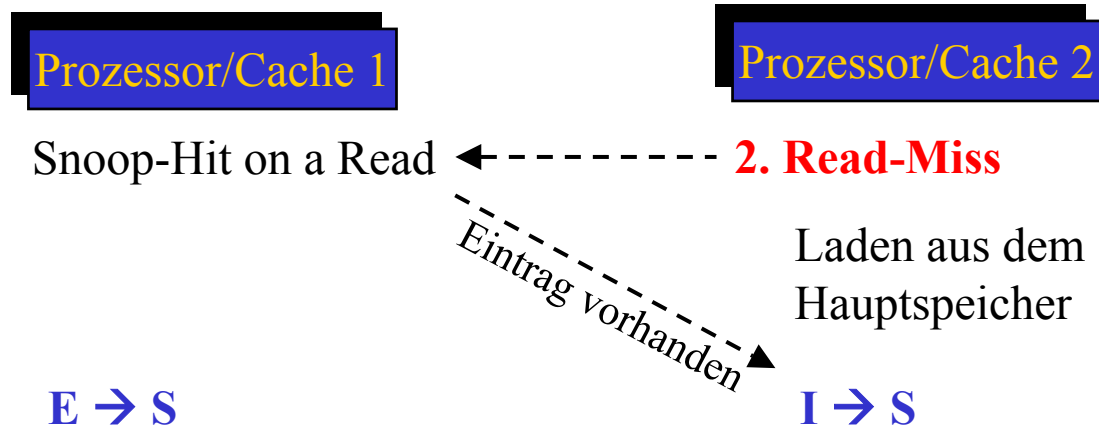
□ Wirkungsweise des MESI-Protokolls

- ◆ Beispiel: Mikroprozessorsystem mit zwei Prozessoren
 - ◆ Vier aufeinanderfolgende Zugriffe auf ein und denselben Speicherblock



Aktualisierungsstrategie und Cache-Kohärenz

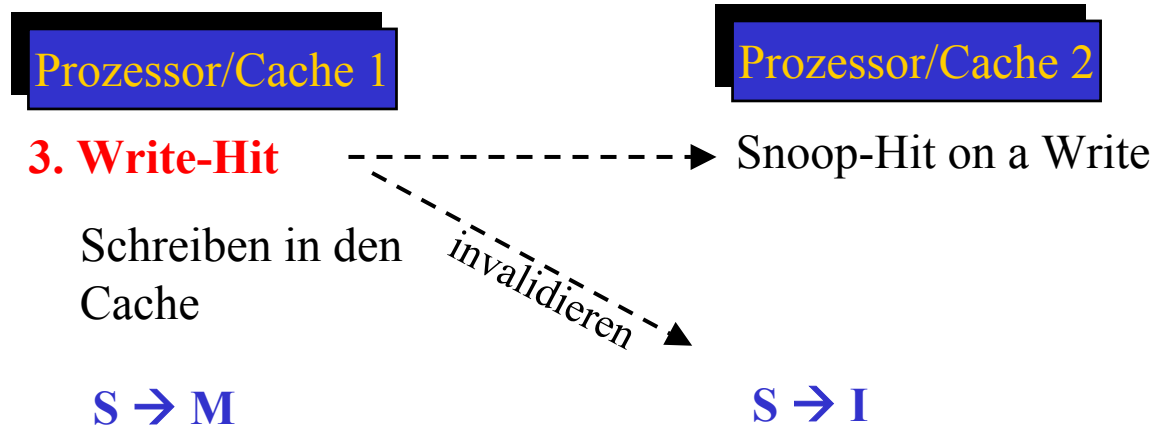
- Wirkungsweise des MESI-Protokolls
 - ◆ Beispiel: Mikroprozessorsystem mit zwei Prozessoren
 - ◆ Vier aufeinanderfolgende Zugriffe auf ein und denselben Speicherblock



Aktualisierungsstrategie und Cache-Kohärenz

□ Wirkungsweise des MESI-Protokolls

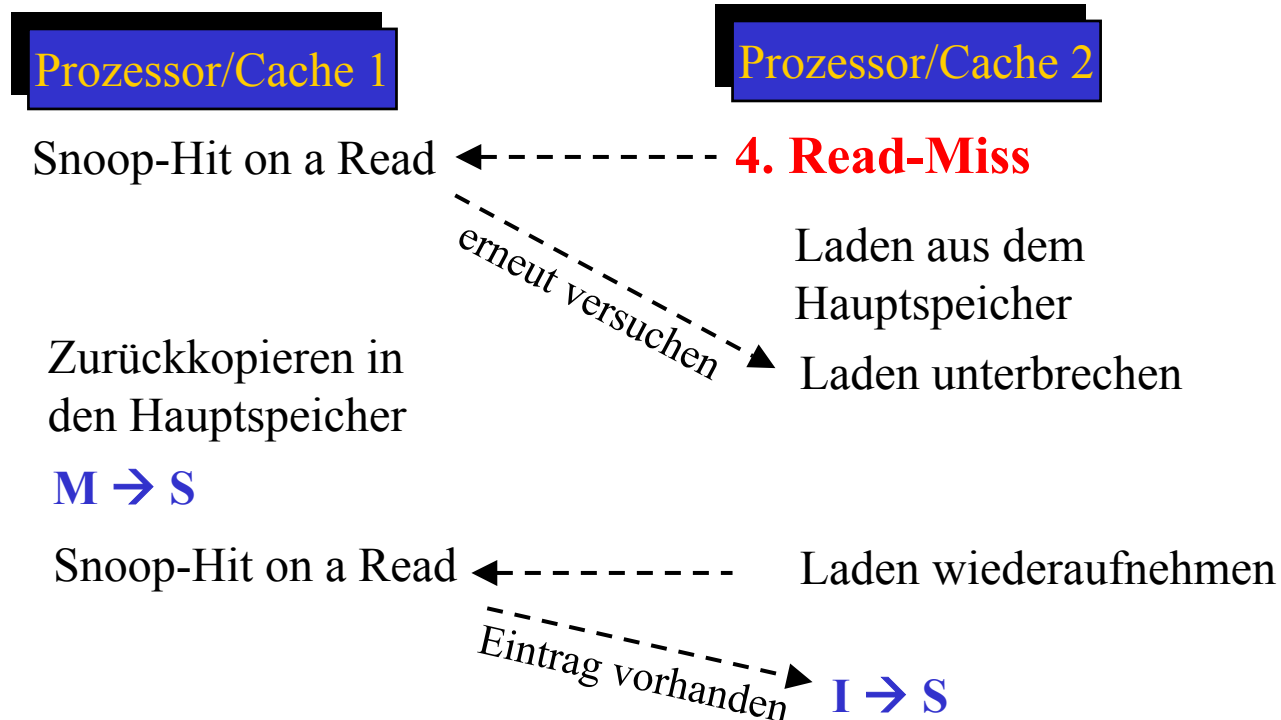
- ◆ Beispiel: Mikroprozessorsystem mit zwei Prozessoren
 - ◆ Vier aufeinanderfolgende Zugriffe auf ein und denselben Speicherblock



Aktualisierungsstrategie und Cache-Kohärenz

□ Wirkungsweise des MESI-Protokolls

- ◆ Beispiel: Mikroprozessorsystem mit zwei Prozessoren
 - ◆ Vier aufeinanderfolgende Zugriffe auf ein und denselben Speicherblock



Fallbeispiele

❑ Fallstudie Intel Pentium III

◆ On-Chip L1-Caches

- ◆ zwei getrennte 4-fach mengenassoziative Cache-Speicher (4-way set associative Cache) für Befehle und Daten
- ◆ 128 Sätze zu je 4 Zeilen zu je 32 Bytes: --> 16KBytes
- ◆ Anzahl Zeilen $m=512$
- ◆ Anzahl Wörter $b=8$,
- ◆ Anzahl Sätze $v=512/4 = m/k=128$
- ◆ Aktualisierungsstrategie: *Write-Through, Copy-Back (Write Back) und weitere Verfahren*
- ◆ MESI-Cache-Kohärenzprotokoll für Daten-Cache

Fallbeispiele

❑ Fallstudie Intel Pentium III

- ◆ On-Chip L2-Cache (Pentium III, Coppermine)
 - ◆ ein gemeinsamer 8-fach mengenassoziativer Cache-Speicher der zweiten Stufe (Level 2) für Befehle und Daten
 - ◆ 128 Sätze zu je 8 Zeilen zu je 32 Bytes → 256 KBytes
- ◆ Off-Chip L2 Cache (Pentium III, Vorgänger Katmai)
 - ◆ gemeinsames Gehäuse
 - ◆ ein gemeinsamer 4-fach mengenassoziativer Cache-Speicher der zweiten Stufe
 - ◆ 1K Sätze zu je 4 Zeilen zu je 32 Bytes → 512 KBytes

Fallbeispiele

□ Aktuelle Mikroprozessoren

Prozessor	Taktfrequenz	Cache (B/D/L2)	Bandbreite
Alpha 21264B	833 MHz	64K/64K	2,66 GB/s
AMD Athlon	1,33 GHz	64K/64K/256K	2,1 GB/s
HP PA8600	552 MHz	512K/1M	1,54 GB/s
IBM Power3-II	450 MHz	32K/64K	1,6 GB/s
Intel Pentium 4	1,7 GHz	12K/8K/256K	1,06 GB/s
MIPS R12000	400 MHz	32K/32K	539 MB/s
Sun Ultra II	480 MHz	16K/16K	1,9 GB/s
Sun Ultra III	900 MHz	32K/64K	4,8 GB/s

Zusammenfassung

- Einführung in die Organisation von Cache-Speichern
 - ◆ Einsatz als Pufferspeicher zwischen Prozessor und Hauptspeicher

- Weitere Einsatzgebiete von Cache-Speichern
 - ◆ in Speicherverwaltungseinheiten zur Pufferung von Adressinformationen, Deskriptoren (Translation Lookaside Buffer, TLB)
 - ◆ Sprungvorhersage (Branch Prediction Cache)
 - ◆ Sprungzielvorhersage (Branch Target Cache)

Zusammenfassung

- Cache-Kohärenzverfahren in Multiprozessorsystemen
 - ◆ MESI-Protokoll
 - ◆ Tabellenorientierte Konsistenzprotokolle (directory based cache coherence)

- Forschungsthema: Datenlokalität durch Code-Transformationen

Vorlesung Rechnerstrukturen

□ **Kapitel 2: Architektur und Mikroarchitektur von Mikroprozessoren**

- ♦ **2.9 Registerorganisation**
- ♦ **2.10 Cache-Speicher**
- ♦ **2.11 Speicherorganisation**
 - ♦ Speicherverschränkung
 - ♦ **Virtuelle Speicherverwaltung**

Speicherverschränkung

- Speicherverschränkung (Memory Interleaving)
 - ◆ Technik, um die Zugriffsgeschwindigkeit auf den Hauptspeicher stärker an die Verarbeitungsgeschwindigkeit der CPU anzupassen.
 - ◆ Parallelisierung der Speicherzugriffe
 - ◆ Speicher wird in n **Speichermodule** (oder **Speicherbänke**) M_0, \dots, M_{n-1} unterteilt und jede Speicherbank mit einer eigenen Adressierlogik versehen.
 - ◆ Verteilung auf n Speicherbänke nennt man **n -fache Verschränkung**.
 - ◆ Der Zugriff auf die Speicherplätze erfolgt zeitlich **verschränkt**.

Speicherverschränkung

- Speicherverschränkung (Memory Interleaving)
 - ◆ Verschränkungsregel
 - ◆ Verteilung der Speicherplätze auf Speicherbänke
 - ◆ Speicherplatz A_i wird in der Speicherbank M_j gespeichert *genau dann wenn* $j=i \bmod n$
 - ◆ Zuteilung
 - ◆ der Adressen A_0, A_n, A_{2n}, \dots auf die Speicherbank M_0 ,
 - ◆ der Adressen $A_1, A_{n+1}, A_{2n+1}, \dots$ auf die Speicherbank M_1 ,
 - ◆ etc.
 - ◆ n -fache Verschränkung: nach einer gewissen Anlaufzeit werden in jedem Speicherzyklus n Speicherworte geliefert.