Caches und TLBs (nur zur Information)

Caches
Der transparente Cache ist vom Prinzip her eine statistische Angelegenheit. Er bringt nur deshalb die gewünschte Wirkung (Zugriffsbeschleunigung) hervor, weil die weitaus meisten Anwendungen ein "lokales Adressierungsverhalten" aufweisen, das heißt, die Befehls- und Datenadressen verändern sich nicht vollkommen regellos, sondern es wird vorwiegend auf aufeinanderfolgende Adressen*) bzw. auf vergleichsweise kleine Adreßbereiche**) zugegriffen (Lokalität).

*): Beispiele: Befehlslesen, Verarbeitung von Datenfeldern und Bildspeicherinhalten.

**): Beispiele: Stackzugriffe, Zugriffe auf die lokalen Variablen des laufenden Programms.

Die Arbeitsprinzipien der Caches sind auf Grundlage von Messungen und Erfahrungen optimiert worden. Solche Caches sind deshalb - statistisch gesehen - recht wirksam. Im Einzelfall ist das Zeitverhalten aber kaum vorhersagbar, und Anwendungen, deren Adressierungsverhalten vom statistischen Durchschnitt beträchtlich abweicht, können bei Nutzung eines Caches unter Umständen langsamer laufen als ohne. Für die *Funktionsfähigkeit* von Software ist dies aber bedeutungslos; selbst Systemprogrammierer brauchen sich, zumindest was den Einzelprozessorbetrieb angeht, aus funktioneller Sicht nicht um transparente Caches zu kümmern.

Adreßumsetzungspuffer (TLBs)
Adreßumsetzungspuffer sind unbedingt notwendig, um das Prinzip des seitenorientierten virtuellen Speichers praktisch verwirklichen zu können (da ohne Hardware-Unterstützung jedem eigentlichen Speicherzugriff mehrere Zugriffe auf Tabellenstrukturen vorausgehen müßten).

Ein Adreßumsetzungspuffer (Translation Lookaside Buffer, TLB*) muß praktisch das gleiche leisten wie ein transparenter Cache: er muß auf eine angebotene Adresse hin eine Angabe liefern (TLB Hit), oder sofern diese Adresse nicht eingetragen ist (TLB Miss), veranlassen, daß die Hardware den Eintrag aufbaut (hierzu muß der Prozessor tatsächlich auf die Tabellenstrukturen zugreifen).

*): sprich: Trännsleeschn Luckäseid Baffer, Tie-Ell-Bie.

Ein Cache-Eintrag enthält Daten, ein TLB hingegen eine *physische Adresse*. Caches und TLBs unterscheiden sich somit nicht im Wirkprinzip, sondern in der Nutzung und demzufolge in Formatierung und Länge der betreffenden Informationsstrukturen. Wir können deshalb Caches und TLBs gemeinsam behandeln. Von besonderer Bedeutung sind die Steuerungs- und Verwaltungsprinzipien. Diese werden im folgenden zunächst allgemein beschrieben, und zwar allein aus Sicht des Caches (der TLB ist demgegenüber eher eine Vereinfachung).

Aufgaben der Cache-Steuerung und -Verwaltung
Die Steuer- und Verwaltungsvorkehrungen eines transparenten Caches haben folgende Aufgaben zu erfüllen:

1. bei jedem Zugriff zu prüfen, ob die betreffende Adresse ein Cache Hit ist oder ein Cache Miss, d. h. ob sich der Speicherinhalt, der zur betreffenden Adresse gehört, im Cache befindet oder nicht,
2. im Falle eines Cache Hit bei einem Lesezugriff die Daten vom Cache zum Prozessor zu liefern und bei einem Schreibzugriff die Daten wenigstens in den Cache einzutragen,
3. im Falle eines Cache Miss die Daten aus dem Arbeitsspeicher heranzuschaffen bzw. in diesen zu schreiben. Zudem ist typischerweise ein neuer Cache-Eintrag (Zugriffsadresse + zugehörige Daten) aufzubauen. Hierzu muß unter Umständen (wenn ansonsten im Cache kein Platz mehr ist) ein anderer Cache-Eintrag aufgegeben werden.

Die funktionellen Anforderungen betreffen im einzelnen:

1. das Abbildungsverfahren (Cache Organization),
2. das Schreibverfahren (Write Policy),
3. das Umgehen des Cache (Cache Policy),
4. das Einlagerungsverfahren (Allocation Policy),
5. das Auslagerungsverfahren (Replacement Policy),
6. die Kennzeichnung der Gültigkeit von Cache-Einträgen (Validity),
7. die Cache-Kohärenz (Cache Coherency),
8. das Abbildungsvermögen (Cacheability).

Hinweis:
In einem System mit mehreren Caches (externen und internen) sind *alle* vorhandenen Caches zu berücksichtigen (so daß aus Sicht der Software wirklich die erforderliche “Transparenz” gewährleistet ist).

Abbildungsverfahren (Cache Organization)
Es ist zweckmäßig, nicht nur einzelne Bytes aus dem Arbeitsspeicher in den Cache abzubilden. Erfahrungsgemäß haben sich Bereiche von 4 bis 64 Bytes als sinnvoll erwiesen. Ein solcher Bereich (Cache Line; sprich: Käsch Lein) wird hier aus logischer Sicht als *Cache- Eintrag* und aus hardware-naher Sicht als *Behälter* (für abgebildete Arbeitsspeicher- Inhalte) bezeichnet.

Cache-Einträge haben im Speicheradreßraum stets integrale Adressen. Der Cache selbst stellt eine gewisse Anzahl von Behältern bereit, um Einträge aufzunehmen. Es gibt verschiedene Prinzipien, um Speicherinhalte diesen Behältern zuzuordnen (Abbildungsverfahren). Sie werden nachfolgend erläutert.

Vorbemerkung zum Fach-Englisch
Abbildung = Mapping (sprich: Mäpping). Assoziative Abbildung = Associative Mapping (sprich: Assoschiätiff...). Direktabbildung = Direct Mapping (sprich: Deireckt...). Blockassoziative Abbildung = Set Associative Mapping; genauer: n Way Set associative Mapping. Hierbei steht “1 Way” (sprich: wonn Wee) für einen Adreßumsetzungsweg aus TAG-RAM und Adreßvergleicher (Abbildungen 3.2, 3.4). Demgemäß spricht man z. B. vom 2 or 4 Way Set Associative Mapping. Abbildung 3.4 zeigt demnach ein Beispiel des 4 Way Set Associative Mapping (sprich: for Wee Set...). Die Direktabbildung (Abbildung 3.2) entspricht

einem 1 Way Set Associative Mapping. (“Set” wird gelegentlich weggelassen - es kann also auch beispielsweise 2 Way Associative Mapping heißen.)

Voll assoziative Abbildung (Fully Associative Mapping)

Im allgemeinen Fall kann jeder Behälter im Cache jeden entsprechenden Bereich aus dem Arbeitsspeicher aufnehmen. Das bedeutet: (1) in jedem Behälter muß die Adresse mitgespeichert werden, (2) bei Zugriffen müssen die Adreßangaben aller Behälter mit der Zugriffsadresse verglichen werden (Abbildung 3.1). Das muß aus Geschwindigkeitsgründen parallel erfolgen, so daß eine solche Anordnung ziemlich aufwendig wird. Wegen des gleichzeitigen Vergleichens wird dieses Prinzip als voll assoziativ (fully associative) bezeichnet.

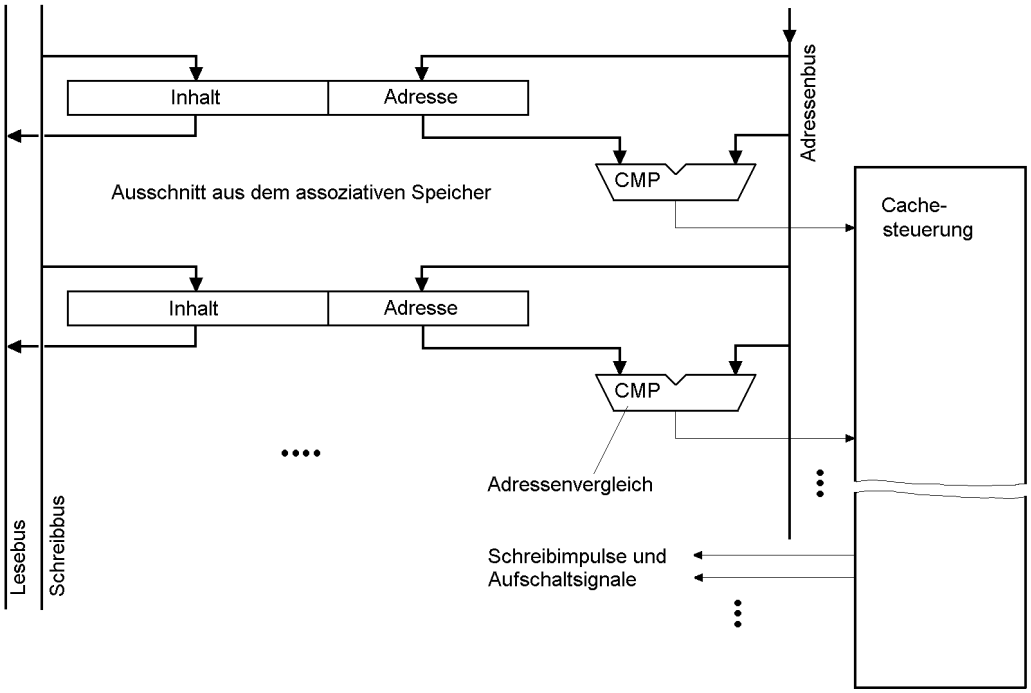

Abbildung 3.1 Cache-Organisation (1): voll assoziativ

Zur Funktionsweise:
Der Cache ist zunächst leer. Dann sind alle Adreßangaben im assoziativen Speicher als ungültig gekennzeichnet. Der erste Zugriff mit einer bestimmten Adresse (nennen wir sie A1) wird somit zum Cache Miss. Dementsprechend werden der gelesene Speicherinhalt sowie die Zugriffsadresse A1 in die erste Position des Assoziativspeichers eingetragen. Diese Position wird damit als gültig deklariert.

Weitere Zugriffe auf andere Adressen führen sinngemäß zu Cache Misses und somit zum weiteren Füllen des Caches.

Nun erfolgt ein weiterer Zugriff mit unserer Adresse A1. Dabei wird in der ersten Position des Assoziativspeichers der Adressenvergleich ein positives Ergebnis liefern (Cache Hit). Somit kann der Speicherinhalt aus dem Cache entnommen werden.

Praktische Anwendung:
Dieses Abbildungsverfahren wird kaum in Caches, aber gelegentlich in TLBs angewendet. Auf diese Weise erreichen auch vergleichsweise kleine TLBs befriedigende Trefferraten.

Direktabbildung (Direct oder 1 Way Set Associative Mapping)

Verglichen mit Assoziativspeichern sind RAM-Strukturen wesentlich kostengünstiger. Gemäß Abbildung 3.2 läßt sich ein transparenter Cache als RAM aufbauen. Jeder Behälter nimmt neben dem eigentlichen (Daten-) Inhalt einen höherwertigen Teil der Adresse auf. Dieser Adreßteil dient zur Kennzeichnung des Inhalts und wird als TAG*) bezeichnet. Mit einem niederwertigen Teil der Adresse wird der RAM direkt adressiert. Bei jedem Cache-Zugriff wird die gelesene TAG-Belegung mit den höherwertigen Bits der Adresse verglichen. Gleichheit zeigt einen Cache Hit an.

*): sprich: Tägg - im Fach-Englisch der Allerwelts-Ausruck für Kennzeichnung, Markierung usw.

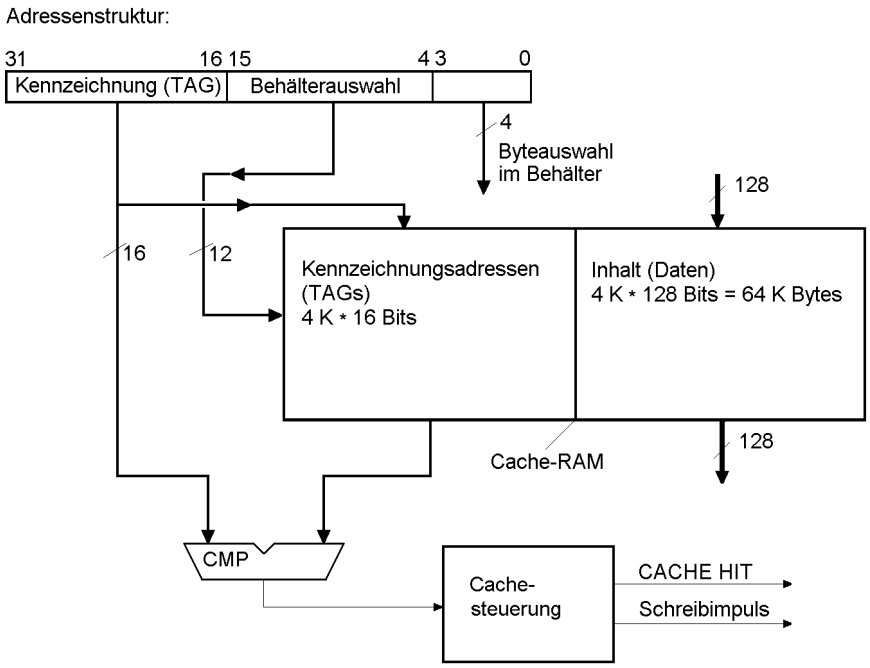

Abbildung 3.2 Cache-Organisation (2): Direktabbildung (Direct Mapping)

Im Beispiel handelt es sich um einen Cache von 64 kBytes, der 4k Behälter zu 128 Bits (= 16 Bytes) umfaßt. Jeder Behälter ist um eine Kennzeichnungsangabe (TAG) von 16 Bits erweitert.

Zur Funktionsweise (Abbildung 3.3):
Aus der Adressenstruktur ergibt sich eine Unterteilung des gesamten Adreßraums in viele gleich lange Bereiche. Bleiben wir beim Beispiel von Abbildung 3.2. Eine Kennzeichnungsadresse von 16 Bits unterteilt den Adreßraum von 2^{32} Bytes = 4 GBytes in 64k Bereiche zu je 64 kBytes (=

4k Cache-Einträge zu je 16 Bytes).

Der Cache ist zunächst leer. Dann sind auch alle Kennzeichnungsadressen im Cache als ungültig gekennzeichnet.

Der erste Zugriff mit einer bestimmten Adresse (nennen wir sie A1 - vgl. Abbildung 3.3d) wird somit zum Cache Miss. Dementsprechend wird der betreffende Behälter im Cache gefüllt: die Adreßbits 15...4 wählen den zu füllenden Behälter aus, und die Adreßbits 31...16 werden dort als Kennzeichnungsadresse (TAG) gespeichert (Abbildung 3.3b). Zugleich wird der Eintrag als gültig gekennzeichnet.

Nun werde beispielsweise mit einer Adresse A2 zugegriffen (Abbildung 3.3d). Die Adreßbits 15...4 sind genauso belegt wie bei der Adresse A1, die Adreßbits 31...16 hingegen abweichend. Die Adresse wählt somit denselben Behälter im Cache aus wie die Adresse A1. Die Cache-Steuerung erkennt aber die Kennzeichnungsadresse als gültig und wertet deshalb das Vergleichsergebnis aus. Da sich die Adressen in den Bitpositionen 31...16 voneinander unterscheiden, ergibt sich ein Cache Miss. Dementsprechend wird der Behälter erneut gefüllt, und zwar mit den Adreßbits 31..16 von A2 (Kennzeichnungsadresse) und mit den zugehörigen Daten aus dem Arbeitsspeicher (Abbildung 3.3c).

Ein weiterer Zugriff mit Adresse A2 wird dann zum Cache Hit - es wird wieder derselbe Behälter adressiert, diesmal aber ist die Belegung der Adreßbits 31...16 gleich der gespeicherten Kennzeichnungsadresse.

Zugriffe mit Adressen, deren Bitpositionen 15...4 anders belegt sind, führen dazu, daß im Cache jeweils andere Behälter adressiert werden.

Im besonderen ist erkennbar, daß, wenn in einem Speicherbereich immer wieder auf neue Adressen zugegriffen wird (fortlaufende Adressierung), sich der Cache allmählich mit den entsprechenden Einträgen füllt. Als Beispiel vgl. Adresse A3 in Abbildung 3.3d (Cache-Belegung: Abbildung 3.3c).

Dieses einfache Prinzip kann, vorausgesetzt die Speicherkapazität des Caches ist groß genug, durchaus befriedigende Trefferraten gewährleisten.

Andererseits ergeben sich geradezu lausige Trefferraten, wenn immer wieder mit gleichen Blockadressen auf andere Speicherbereiche zugegriffen wird. Um im Beispiel von Abbildung 3.3 zu bleiben: wir greifen mit gleichbleibender Blockadresse (911) immer wieder auf andere Bereiche zu - dann wird jeder dieser Zugriffe zum Cache Miss werden (und es gibt durchaus Anwendungen und Systemumgebungen, die ein ähnliches Zugriffsverhalten aufweisen (Stichworte: Multimedia und Multitasking)).

Praktische Anwendung:
U. a. L2-Caches auf 586- und 686-Motherboards. (Mit herkömmlichen Systemplattformen (Windows 3.x, 95/98) und Wald-und-Wiesen-Anwendungen - kein Multimedia, kein intensives Multitasking - ergeben sich annehmbare Trefferraten.)

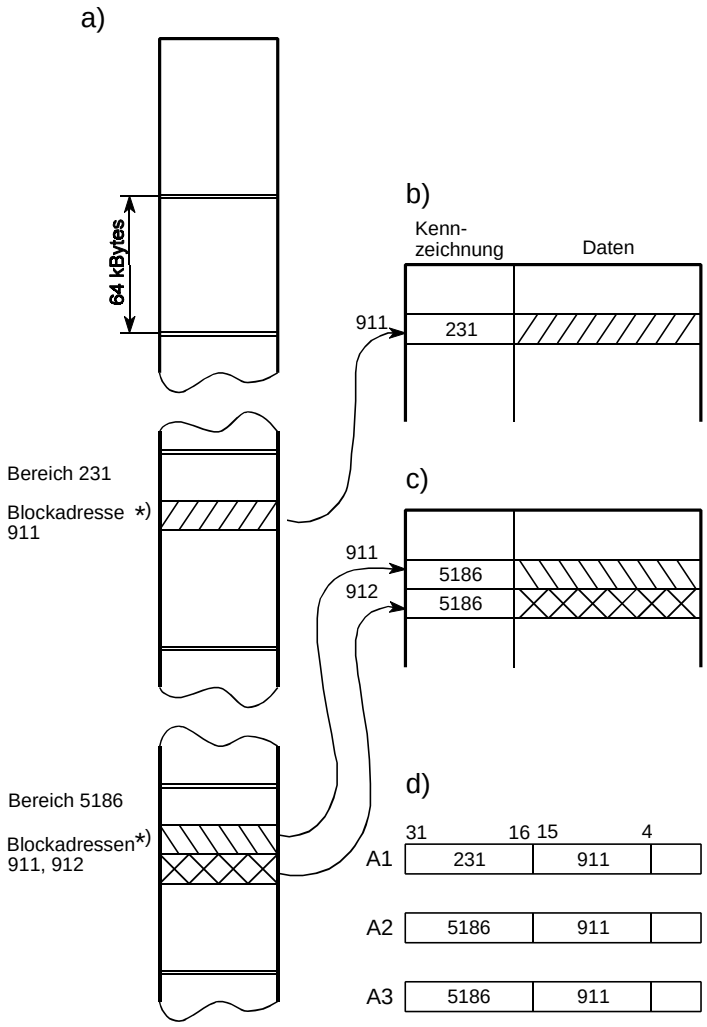

Abbildung 3.3 Zur Funktionsweise des direktabbildenden Caches. a) - der Speicheradreßraum; b) - Cache-Belegung nach dem Zugriff mit Adresse A1; c) - Cache-Belegung nach Zugriffen mit den Adressen A2, A3; d) - beispielhafte Zugriffsadressen; *): integrale Adressen (betrifft hier Cache-Einträge zu 16 Bytes)

Blockassoziative Abbildung (n Way Set Associative Mapping)

Dies ist eine Verbundlösung. Es werden mehrere direktabbildende Cache-Anordnungen vorgesehen, die gemeinsam adressiert und von einer gemeinsamen Cache-Verwaltung gesteuert werden. Bei Zugriffen auf gleiche Blockadressen werden die Cache-Anordnungen nacheinander belegt. Mit anderen Worten: es gibt mehrere Adreßumsetzungswege (“Ways”) ähnlich Abbildung 3.2 mit zugeordneten Datenspeichern und gemeinsamer Steuerung.

Beispiel: der interne Cache des i486. Er ist 4-fach blockassoziativ organisiert (Abbildung 3.4).

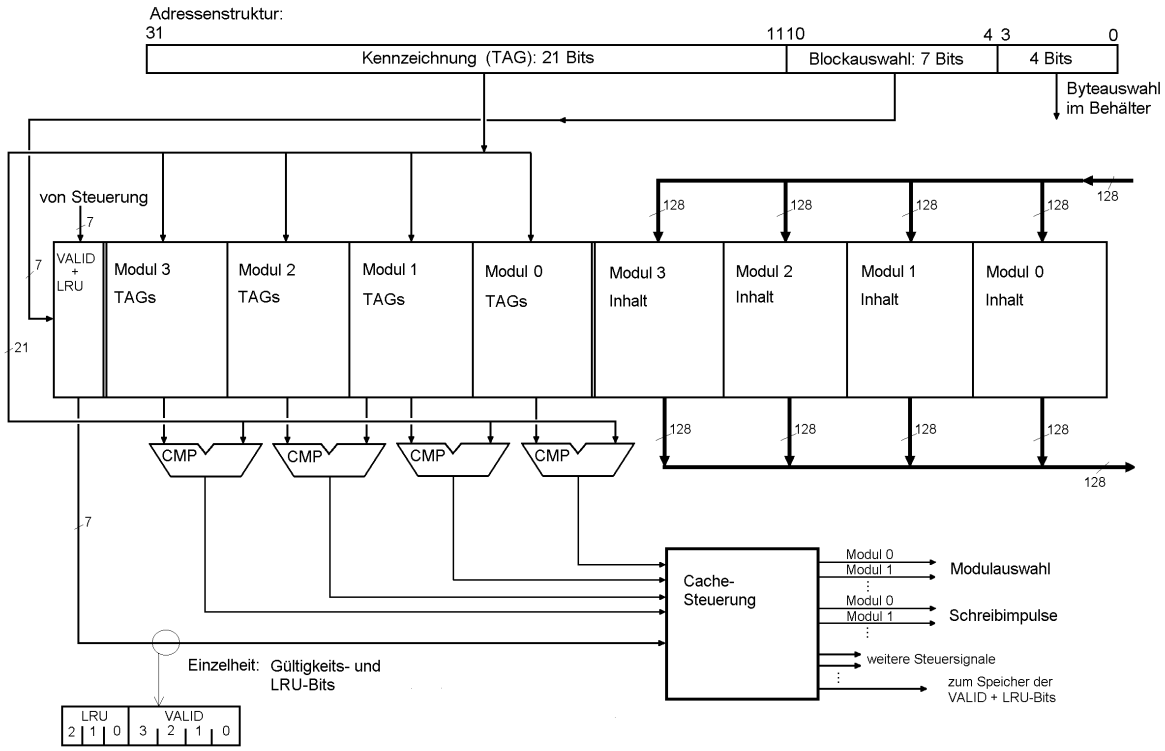

Abbildung 3.4 Cache-Organisation (3): blockassoziativ (Beispiel: der interne Cache des i486)

Die physische 32-Bit-Speicheradresse wird hier folgendermaßen interpretiert:

- die vier niedrigstwertigen Bits adressieren das Byte im jeweiligen Behälter,
- die folgenden 7 Bits wählen einen Behälter aus, und zwar aus einem von vier Speichermoduln*) mit jeweils 128 Behältern. Die vier Speichermoduln werden parallel adressiert.
- welcher der vier adressierten Behälter in Frage kommt, wird durch Vergleich mit den verbleibenden höchstwertigen 21 Adreßbits festgestellt (Kennzeichnungsadresse). Dazu hat jeder Speichermodul für jeden Behälter 21 weitere Bits (TAG-Bits), die die höherwertigen Bits der jeweiligen Adresse enthalten. Ist eine TAG-Belegung gleich den 21 Bits der anliegenden Adresse, so enthält der betreffende Behälter die zugehörigen Daten (Cache Hit). Fallen alle vier Vergleiche negativ aus, so befindet sich keine Abbildung des 16-Byte-Eintrags im Cache (Cache Miss).

*): “Speichermodul” (Memory oder Cache Module) bezeichnet nicht nur SIMMs, DIMMs usw., sondern wird gelegentlich als Allerwelts-Fachbegriff verwendet (allgemein für Speicheranordnung, Speicherblock usw.).

Es ist ersichtlich, daß sich der technische Aufwand im Vergleich zur voll assoziativen Organisation deutlich vermindert: für insgesamt 512 (4 · 128) Cache-Einträge braucht man nur vier Adreßvergleicher (bei voll assoziativer Organisation wären es hingegen 512), und die Adreßbits (TAG-Bits) können in üblichen RAM-Strukturen gespeichert werden.

Zur Funktionsweise:
Die Kennzeichnungsadresse von 16 Bits unterteilt den Adreßraum in 2M Bereiche zu je 2 kBytes (= 128 Cache-Einträge zu je 16 Bytes).

Der Cache ist zunächst leer. Dann sind auch alle Moduln als leer gekennzeichnet. Nun führen wir mehrere beispielhafte Zugriffe aus. (Abbildung 3.5 veranschaulicht die einzelnen Zugriffsadressen.)

1. Zugriff: mit Adresse A1. Hierbei werden gleichzeitig alle Behälter Nr. 99 in allen 4 Moduln ausgewählt - und als leer vorgefunden. Zunächst wird Behälter Nr. 99 in Modul 0 gefüllt. Die Bereichsadresse (231) kommt in den TAG-Teil von Behälter 99, Modul 0.

2. Zugriff: mit Adresse A2. Auch dabei werden die Behälter Nr. 99 in allen 4 Moduln ausgewählt. In Modul 0 ergibt sich ein Cache Miss, aber die anderen drei Moduln sind noch frei. Somit wird Behälter Nr. 99 in Modul 1 belegt. Die Bereichsadresse (5137) kommt in den TAG-Teil von Modul 1.

3. und 4. Zugriff: mit den Adressen A3 und A4. Sinngemäß werden die Behälter Nr. 99 in den beiden noch freien Moduln 2, 3 belegt.

5. Zugriff: mit Adresse A5. Jetzt werden die Behälter Nr. 100 in allen 4 Moduln ausgewählt - und als leer vorgefunden. Dementsprechend wird Modul 0 belegt.

6. Zugriff: mit Adresse A6. Es werden wiederum die Behälter Nr. 99 in allen 4 Moduln ausgewählt. Es ergibt sich ein Cache Hit für Modul 3 (Zugriffsadresse (16 428) = TAG-Belegung), so daß der Zugriff aus diesem Modul des Caches bedient werden kann.

7. Zugriff: mit Adresse A7. Nochmals werden die Behälter Nr. 99 in allen 4 Moduln ausgewählt. Alle diese Behälter sind belegt - und für jede Belegung ergibt sich ein Cache Miss. Nun muß eine der vorhandene Belegungen aufgegeben und durch die neue ersetzt werden.

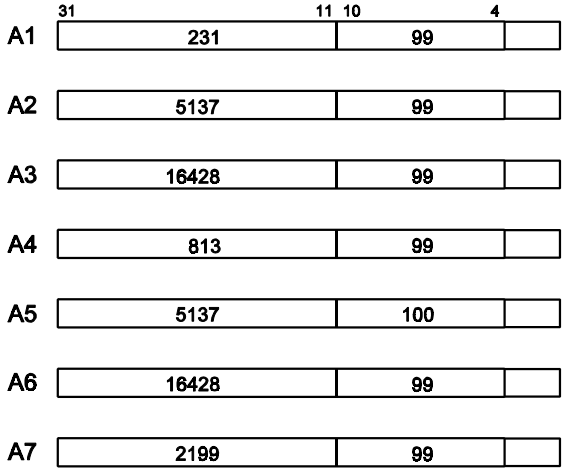

Abbildung 3.5 Zur Funktionsweise des blockassoziativen Caches: Beispiele von Zugriffsadressen

Hinweise:

1. Die Verwaltung der Moduln gewährleistet, daß derselbe TAG-Wert nicht in mehr als eines der 4 parallel adressierten Moduln eingetragen werden kann.
2. Beim Zugriff auf eine bestimmte Blockadresse (d. h. auf eine Reihe von 4 gemeinsam adressierten Behältern) können wir folgende Betriebszustände voneinander unterscheiden:

    1) ein Modul ist frei. Es kann demzufolge belegt werden (aktuelle Kennzeichnungsadresse wird zur TAG-Belegung).
    2) ein Modul enthält als TAG-Belegung die Kennzeichnungsadresse, mit der gerade zugegriffen wird: keine Änderung (Cache Hit).
    3) alle 4 Moduln sind belegt, und es wird mit einer neuen Kennzeichnungsadresse zugegriffen. Demzufolge wird keiner der Vergleicher aktiv (Cache Miss). Dann muß eines der Moduln gewissermaßen beräumt und neu belegt werden. Hierzu wird typischerweise jenes Modul ausgewählt, worauf in der letzten Zeit am wenigsten zugegriffen wurde (LRU-Verfahren (Einzelheiten dazu in Abschnitt 3.3.4.)).

Praktische Anwendung:
Caches aller Art, TLBs.

Abbildungsverfahren der Caches in Personalcomputern

Es sind folgende Verfahren üblich:

- Direktabbildung,
- 2- oder 4-fach blockassoziative Abbildung.

Wichtige Zusammenhänge im Überblick:

Anzahl der Adreßumsetzungswege ("Assoziativität")
Je mehr, desto besser die Trefferrate, desto höher aber auch der Aufwand.

Einfachste Lösungen
Direktabbildung im Verbund mit dem Schreibverfahren Write Through. Anwendung: externe Caches.

Kleine Caches (mit beispielsweise 32...128 kBytes)
Diese erreichen bei Direktabbildung nur unbefriedigende Trefferraten. Deshalb hatte man seinerzeit (Ende der 80er/Anfang der 90er Jahre) für externe 386- und 486-Caches eine wenigstens 2-fach blockassoziative Auslegung bevorzugt.

Abbildungsvermögen (Cacheability)
Bei gegebener Speicherkapazität ergibt Direktabbildung das größtmögliche Abbildungsvermögen (im Sinne von "x MBytes Cacheable Address Space"). Es sinkt bei 2-fach

blockassoziativer Organisation auf die Hälfte, bei 4-fach blockassoziativer Organisation auf ¼. (Anders herum: um ein bestimmtes Abbildungsvermögen beizubehalten, müßte die Speicherkapazität des Caches entsprechend verdoppelt oder vervierfacht werden.)

Die typische Auslegung vieler PCs mit Sockel-7-Prozessorinterface (586, 686)
Der L2-Cache ist ein größerer direktabbildender Cache (256 oder 512 kBytes). Synchrone SRAMs (Pipelined Burst) im Datenteil, asynchrone SRAMs als TAG-RAMs. Schreibverfahren: Write Back.

Externe Caches in Hochleistungssystemen
Der Leistungsgewinn infolge des Cache-Subsystems ist stets eine statistische Angelegenheit (wer es darauf anlegt, kann für jedes Cache-Subsystem Programme schreiben, die nachweisen, daß die Prozessorleistung schlechter ist als ohne jeden Cache*). Leistungsmessungen haben gezeigt, daß bei Cache-Größen zwischen 256 kBytes und ca. 1 MByte eine 2-fach blockassoziative Auslegung dieselbe Verbesserung der Trefferrate bewirkt wie die Verdoppelung eines direktabbildenden Cache. Deshalb bevorzugt man für Hochleistungssysteme auch im L2-Cache die blockassoziative Organisation.

*): Prinzip: die Behälter- bzw. Blockadresse beibehalten, die Kennzeichnungsadresse aber fortlaufend so ändern, daß sich bei jedem derartige Zugriff ein Cache Miss ergibt.

Schreibverfahren (Write Policy)
Es gibt drei Prinzipien, nach denen ein Cache Schreibzugriffe, die Cache Hits sind, verarbeiten kann:

1. Weiterreichen (Write Through)*

Die Daten werden gleichzeitig sowohl in den Cache als auch in den Arbeitsspeicher geschrieben. Folglich wird die Dauer des Zugriffs durch die Zugriffszeit des Arbeitsspeichers bestimmt.

*): sprich: wreit sruh. Vereinfachte Schreibweise: Write Thru.

2. Gepuffertes Weiterreichen (Buffered Write Through)

Die Daten werden in den Cache und in den Arbeitsspeicher geschrieben, wobei der Arbeitsspeicherzugriff jeweils zwischengepuffert und von autonomen Schaltmitteln ausgeführt wird. Der Prozessor muß deshalb nicht auf das Ende des Arbeitsspeicherzugriffs warten, sondern kann schon im folgenden Zyklus wieder Daten aus dem Cache lesen.

3. Bedarfsweises Zurückschreiben (Write Back)*

Die Daten werden zunächst in den Cache geschrieben. Sie werden nur dann in den Arbeitsspeicher übertragen, wenn der betreffende Eintrag aus dem Cache entfernt werden muß oder wenn dies vom System gefordert wird (Beenden der Arbeit; Bereitstellen der Daten zu Ausgabezwecken usw.).

*): sprich: wreit bäck.

Zur Funktionsweise:

- jedem Cache-Eintrag sind besondere Kennzeichnungsbits zugeordnet (üblicherweise als DIRTY oder MODIFIED bezeichnet)*),
- ist ein Schreibzugriff ein Cache Hit, so wird das betreffende Kennzeichnungsbit gesetzt,
- muß der jeweilige Eintrag aus dem Cache entfernt werden (um ihn durch einen anderen zu ersetzen - Cache Line Replacement), so wird das betreffende Kennzeichnungsbit abgefragt. Ist es gesetzt, wird der Eintrag in den Arbeitsspeicher zurückgeschrieben. Dann wird das Kennzeichnungsbit gelöscht.
- ist der Cache insgesamt zu leeren (um dies zu veranlassen, gibt es eigens Maschinenbefehle), werden alle Kennzeichnungsbits abgefragt. Alle als verändert gekennzeichneten Einträge werden in den Arbeitsspeicher zurückgeschrieben.

*): vgl. aber auch Hinweis 3 in Anschluß an die folgende Tabelle 3.1.

Schreibverfahren in modernen PCs

Der offensichtliche Vorteil des Write-Through-Prinzips ist die Einfachheit, der des Write-Back-Prinzips die Geschwindigkeit. Buffered Write Through ist ein Kompromiß zwischen beiden. In modernen Systemen sind die Schreibverfahren Write Through und Write Back üblich (Tabelle 3.1).

Schreibverfahren	Kennzeichen	Vorteile	Nachteile
Write Through	jeder Schreibzugriff ist stets auch ein Arbeitsspeicherzugriff (auch bei Cache Hit)	einfachere Hardware; Schreibzugriffe wirken sich unmittelbar im Arbeitsspeicher aus	geringere Systemleistung
Write Back	Schreibzugriffe, die Cache Hits sind, betreffen zunächst nur den Cache	geringere Bus-belastung, höhere Systemleistung	besondere Bitpositionen (DIRTY bzw. MODIFIED) erforderlich, um geänderte Cache-Einträge zu kennzeichnen*); kompliziertere Steuerung

*): siehe auch den Hinweis 3 im folgenden Text

Tabelle 3.1 Schreibverfahren

Hinweise:

1. Aus Leistungsgründen bevorzugt man in modernen Cache-Subsystemen das Write-Back-Verfahren.
2. Write-Back-Caches müssen im laufenden Betrieb von Zugriff zu Zugriff auf Write Through umschaltbar sein, um den Architektur-Vorgaben der Prozessoren zu entsprechen (vgl. die PWT-Bits in den Seitentabelleneinträgen).
3. Das Write-Back-Prinzip kann man auch *ohne* DIRTY- bzw- MODIFIED-Bits verwirklichen. Der Grundgedanke: (1) vor jedem Überschreiben eines gültigen Eintrags wird der Eintrag grundsätzlich (gleichgültig, ob verändert oder nicht) in den Arbeitsspeicher zurückgeschrieben, (2) beim Leeren des Caches wird stets der gesamte Cache-Inhalt zurückgeschrieben. Infolge der unnötigen Schreibvorgänge (Zurückschreiben ungeänderter Einträge) ergibt sich ein gewisser Leistungsverlust.

Trotzdem ist so ein Cache einem Write-Through-Cache überlegen.

Umgehen des Cache (Cache Policy)
In den meisten Systemen dürfen Zugriffe auf bestimmte Teile des Speicheradreßraums nicht über den Cache geführt werden, mit anderen Worten: sie sind vom "Caching" auszuschließen (Non-Cacheable Regions*)). Das betrifft z. B. Bildspeicher von Videoadaptern und andere E-A-Einrichtungen, die über den Speicheradreßraum angesprochen werden (Memory Mapped I/O; vgl. Abschnitt 5.4.3.).

*): sprich: Nonn Käschäbbl Riehtschns.

Bei jedem Zugriff auf den Speicheradreßraum muß bekannt sein, ob der Cache umgangen werden soll oder nicht. Des weiteren ist (wenn der Cache hierfür die Wahl läßt) bei Schreibzugriffen das Schreibverfahren zu bestimmen. Hierfür gibt es verschiedene Möglichkeiten:

Softwareseitige Steuerung über entsprechende Architektur-Vorkehrungen:

- die IA-32-Prozessoren haben Bitpositionen in Steuerregistern, über die man die Caches "global" steuern kann,
- im Protected-Modus der IA-32-Architektur kann man die Seitenverwaltung (Paging) ausnutzen. Die Seitenverzeichnis- und Seitentabelleneinträge enthalten die Bits PCD und PWT. Hierüber kann man den Speicheradreßraum seitenweise vom Caching allgemein ausschließen bzw. bestimmen, daß Schreibzugriffe auf bestimmte Seiten stets den Arbeitsspeicher mitbetreffen. Erweiterung: die Seitenattributtabelle PAT.
- Bereichsregister. Beispiel: die P6-Prozessoren von Intel haben spezielle Register, worüber die Speicherkonfiguration gesteuert werden kann (Memory Type Range Registers; MTRRs). Diese Register enthalten Typfelder, in denen zu jedem Speicherbereich die zugehörige Betriebsweise des Cache-Subsystems angegeben wird .

Steuerung über die Hardware
Manche Prozessoren und Steuerschaltkreise haben Steuereingänge, über die der Cache außer Betrieb gesetzt werden kann (übliche Signalbezeichungen: KEN ≙ Cache Enable, NCA ≙ Non Cacheable Access). Gelegentlich kann über weitere Eingänge das Schreibverfahren beeinflußt und ein Leeren des Caches veranlaßt werden (typische Bezeichnungen: WB/WT ≙ Write Back/Write Through, FLUSH). Bei jedem Speicherzugriff wird der KEN-Eingang abgefragt. Ist er inaktiv, so wird der Cache umgangen. Indem man den KEN-Eingang über entsprechende Adreßvergleicher oder -decoder ansteuert, kann man beliebige Adreßbereiche vom Caching ausschließen. (Extremfall: Cache gar nicht verwenden = KEN ständig inaktiv halten.) Sinngemäß läßt sich über den Eingang WB/WT bei Bedarf (z. B. bei Bildspeicherzugriffen) das Schreibverfahren Write Through erzwingen.

Hinweis:
Die Prozessoren der 486- und der P5-Klasse sowie 686-Typen mit Sockel-7-Interface haben entsprechende Steuereingänge (KEN, FLUSH, ggf. WB/WT), die vom Motherboard-

Steuerschaltkreis aus erregt werden. Die P6-Prozessoren von Intel haben Bereichsregister (MTRRs), manche auch eine Seitenattributtabelle (PAT). Ein FLUSH-Eingang ist vorgesehen, es gibt aber keine Eingänge KEN, WB/WT o. dergl.

Softwareseitige Steuerung über Steuerschaltkreise
Hat der Prozessor keine Seitenattributtabelle und keine Bereichsregister, so muß der Steuerschaltkreis die Zugriffsadressen auswerten und den KEN-Eingang entspechend ansteuern. Manche Steuerschaltkreise enthalten universell nutzbare Bereichsregister (somit können z. B. durch Laden von Adreß- und Längenangaben die auszuschließenden Adreßbereiche gekennzeichnet werden). Oftmals hat man aber die Beeinflussungsmöglickeiten an die PC-typische Aufteilung des Speicheradreßraums angepaßt. Es sind beispielsweise feste Bereiche vorgesehen, denen man bestimmte Attribute (u. a. auch “Cache ein/aus”) zuordnen kann. Der Intel-Fachbegriff für entsprechende Steuerregister: PAM-Register (PAM = Programmable Attribute Map).

Einlagerungsverfahren (Allocation Policy)
Grundsätzlich sind nur Zugriffe auf Adressen zu berücksichtigen, die nicht vom Caching ausgeschlossen sind. Das Problem der Einlagerung (Allocation) ergibt sich stets, wenn ein Zugriff auf eine dem Caching unterworfene (cacheable) Adresse ein Cache Miss ist. Folgende Einlagerungsverfahren als Reaktion auf einen Cache Miss sind üblich:

a) beim Lesen
Es wird *stets* ein Cache-Eintrag aufgebaut (Allocation on Reads).

b) beim Schreiben
Hier gibt es 2 Alternativen:

- es wird *kein* Cache Eintrag aufgebaut (No Allocation on Writes),
- es wird *stets* ein Cache Eintrag aufgebaut (Allocation on Writes).

Hinweise:

1. Allocation on Writes hat folgende Vorteile: (1) bessere Trefferrate (es ist recht wahrscheinlich, daß auf benachbarte Adressen (die den gleichen Cache-Eintrag betreffen) zugegriffen wird, (2) Verringerung der Busbelastung.
2. Allocation on Writes verspricht nur im Verbund mit dem Schreibverfahren Write Back nennenswert mehr Leistung.
3. Intel bevorzugt für i486 und Pentium *No Allocation on Writes*. Bei den P6-Prozessoren ist hingegen ausschließlich *Allocation on Writes* vorgesehen.
4. Es gibt Steuerschaltkreise, die es ermöglichen, das Einlagerungsverfahren des externen Caches programmseitig (über Steuerregister) einzustellen.

Auslagerungsverfahren (Replacement Policy)
Das Auslagern “alter” Cache-Einträge (um in den Behältern Platz für neue zu schaffen) ist an sich nur ein Problem bei blockassoziativen Abbildungsverfahren (bei Direktabbildung gibt es jeweils nur einen Behälter im Cache - und der wird einfach mit dem neuen Eintrag

überschrieben). Um den freizumachenden Behälter zu bestimmen, wird meistens ein näherungsweises LRU-Verfahren verwendet.

Hinweis: Beim Write-Back-Cache müssen veränderte (modifizierte) Cache-Einträge in den Arbeitsspeicher zurückgeschrieben werden.

Gültigkeit von Cache-Einträgen (Validity)
Zugriffe auf den Speicheradreßraum dürfen nur Cache-Einträge ansprechen, die "gültig" (valid; sprich: fällid) sind (d. h. ein Cache-Eintrag ist dann gültig, wenn sein Inhalt dem Inhalt des Arbeitsspeichers unter der jeweiligen Adresse entspricht). Es gibt zwei Möglichkeiten, die Gültigkeit zu kennzeichnen:

- jedem Eintrag ist ein Gültigkeitsbit (VALID-Bit) zugewiesen (vgl. auch Abbildung 3.4),
- es gibt keine VALID-Bits; statt dessen wird der Cache so initialisiert, daß anfänglich alle Einträge gültig sind. (Bevor der Cache zur Nutzung freigegeben wird, werden in einem Sonderablauf (unter Steuerung des BIOS) alle Behälter mit gültigen Werten gefüllt.)

Cache-Kohärenz (Cache Coherency)
Der Begriff (sprich: Käsch Kohärensie) bezeichnet folgendes Problem:

- es gibt im System mehrere Einrichtungen, die den Inhalt des Arbeitsspeichers verändern können (mehrere Prozessoren, aber auch DMA-Hardware und Busmaster),
- der Prozessor sieht den Arbeitsspeicher aber nur über sein Cache-Subsystem,
- es ist notwendig, zu gewährleisten daß alle Einrichtungen bei Zugriff auf eine bestimmte Adresse dort gleichermaßen die aktuellen Daten vorfinden.

(Anders herum gesehen: gibt es in einem System nur einen einzigen Prozessor und ist dies die einzige Einrichtung, die den Speicherinhalt verändern kann, so gibt es kein Problem der Cache-Kohärenz.)

Was ist zu leisten? - Offensichtlich gibt es zwei Fälle:

1. die wirklich aktuellen Daten wurden vom Prozessor erzeugt bzw. verändert und befinden sich im Cache. Nun will eine andere Einrichtung darauf zugreifen. Deshalb müssen die Daten vor diesem Zugriff zunächst in den Arbeitsspeicher geschafft werden.
2. eine andere Einrichtung hat die Daten im Arbeitsspeicher verändert (bzw. neu eingeschrieben - z. B. im Rahmen eines Eingabevorgangs). Der Prozessor muß also veranlaßt werden, diese Daten unbedingt aus dem Arbeitsspeicher zu holen. Das einfachste Verfahren: eventuell vorhandene Cache-Einträge werden ungültig gemacht - somit wird der nächste entsprechende Zugriff des Prozessors zum Cache Miss werden.

Die bevorzugte Lösung: Fremdzugriffsprüfung (Snooping[*)]). Jeder Cache beobachtet alle Zugriffe auf den Speicheradreßraum. Handelt es sich (1) um den Zugriff einer anderen Einrichtung, betrifft dieser aber (2) einen gültigen Cache-Eintrag, so muß etwas getan werden. Hierzu stehen folgende Maßnahmen zur Wahl:

- der betroffene Eintrag wird als “ungültig” (invalid) gekennzeichnet (Cache Line Invalidation; sprich: Käsch Lein Infällideeschn). Greift der Prozessor erneut auf den entsprechenden Adreßbereich zu, so ergibt sich ein Cache Miss. Demzufolge wird der Eintrag neu aufgebaut (= aus dem Arbeitsspeicher geholt) und hat somit den aktuellen Inhalt. Invalidation-Abläufe bestehen im wesentlichen im Zurücksetzen des betreffenden Gültigkeitsbits (im TAG-RAM).
- der betroffene Eintrag wird automatisch (vom Cache Controller) neu aufgebaut (d. h. aktualisiert; dieses ziemlich komplizierte Verfahren ist notwendig, wenn man auf Gültigkeitsbits im TAG-RAM verzichtet hat).

*): sprich: Snuhping - der Allerwelts-Ausdruck für das Beobachten bzw. “Mithören” von Signalspielen.

Hinweise:

1. Kann man über mehrere Bussysteme auf den Speicheradreßraum zugreifen (z. B. über den Prozessorbus, DRAM-Bus, PCI-Bus, AGP, ISA-Bus usw.), so sind alle Bussysteme gleichzeitig zu überwachen (was auf üblichen Motherboards dadurch erleichtert wird, daß alle Bussysteme in einem Steuerschaltkreis zusammengeführt sind). Siehe hierzu auch (ergänzend) das Technische Handbuch 01.
2. Der Cache Controller muß alle Caches (externe und interne) gleichermaßen berücksichtigen.
3. Bei Write-Thru-Caches können nur fremde Schreibzugriffe die Kohärenz beeinträchtigen. Bei Write-Back-Caches ist aber damit zu rechnen, daß sich die aktuellen (= vom Prozessor veränderten) Daten im Cache befinden, nicht aber im Arbeitsspeicher. Somit müssen hier auch fremde *Lesezugriffe* berücksichtigt werden (vor dem Lesen sind die veränderten Daten erst einmal in den Arbeitsspeicher zu schaffen).
4. Besonders schwierig wird es, wenn es mehrere Prozessoren mit Caches gibt.

Abbildungsvermögen (Cacheability)
Der Begriff ist mit einer Zahlenangabe verbunden, die besagt, wie groß der Ausschnitt aus dem Speicheradreßraum ist, den der Cache abbilden kann. Der Wert hängt von folgenden Gegebenheiten ab (Abbildung 3.6):

- der Länge des Cache-Eintrags (Cache Line),
- der Speicherkapazität des Cache (Cache Size),
- der Adreßbreite im TAG-RAM.

Das Abbildungsvermögen in der Servicepraxis
Je moderner der PC, desto wahrscheinlicher erledigt sich das Problem von selbst - mit dem Prozessor zusammengebaute Caches haben typischerweise ein Abbildungsvermögen, das den gesamten Adreßraum umfaßt. Die nachfolgend erläuterten Spitzfindigkeiten brauchen Sie aber dann, wenn PCs zu erweitern sind, die einen externen Cache auf dem Motherboard haben (vgl. die Abbildungen 3.7, 3.8).

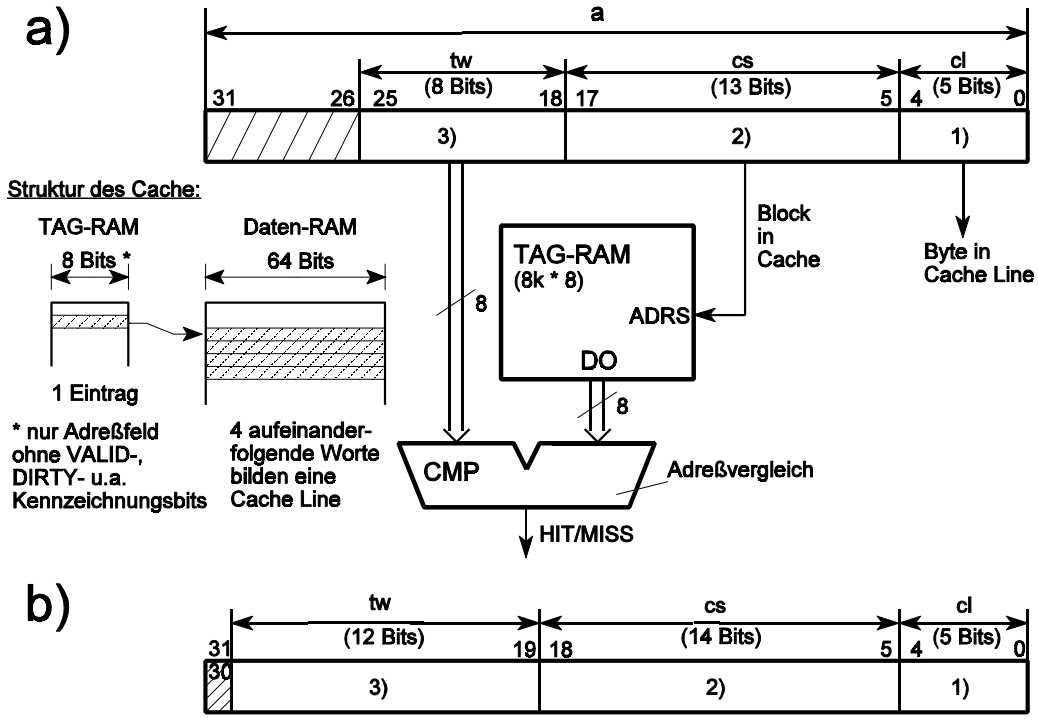

Abbildung 3.6 Zum Abbildungsvermögen. a) Beispiel einer herkömmlichen Adreßaufteilung (64 MBytes Cacheability); b) Beispiel einer Adreßaufteilung für 2 GBytes Cacheability

Erklärung:
Eine (Byte-) Adresse der Länge a (in Bits) beschreibt einen Adreßraum von 2^a Bytes. Anhand eines direktabbildenden Caches ist dargestellt, wie die Adreßbits ausgenutzt werden:

1) die niedrigstwertigen Adreßbits adressieren das einzelne Byte im Cache-Eintrag. Einem Adreßabschnitt von *cl* Bits entspricht ein Cache-Eintrag von 2^{cl} Bytes.
2) die folgenden Adreßbits adressieren den einzelnen Behälter bzw. Block im Cache. Einem Adreßabschnitt von *cs* Bits entspricht ein Cache mit 2^{cs} Behältern bzw. Blöcken.
3) diese Adreßbits *tw* dienen zum Vergleichen mit dem Adreßfeld im TAG-RAM (Kennzeichnungsadresse).

Wieviele Bytes der Cache abbilden kann, wird offensichtlich dadurch bestimmt, wieviele Adreßbits in der Cache-Hardware ausgewertet werden. Das Abbildungsvermögen umfaßt den gesamten Adreßraum (2^a Bytes), wenn gilt:

$$tw + cs + cl = a$$

Das Abbildungsvermögen eines gegebenen Caches ergibt sich als Zweierpotenz der Adreßbits, die tatsächlich ausgewertet werden:

$$\text{Abbildungsvermögen (Cacheability; in Bytes)} = 2^{tw+cs+cl}$$

a) Beispiel einer Adreßaufteilung (64 MBytes Cacheability):

- a = 32 Bits (die gängige 32-Bit-Byteadresse),
- cl = 5 Bits (1 Cache Line = 32 Bytes = 256 Bits = 4 Worte zu 64 Bits ≙ 1 Pentium-Burstzyklus),
- cs = 13 Bits (≙8k Einträgen bzw. Blöcken = 32k Worten zu 64 Bits ≙ Aufbau des Cache mit zwei 32k · 32 PBSRAMs),
- tw = 8 Bits (≙ TAG-RAM 8k · 8 (ohne Gültigkeits- und andere Kennzeichnungsbits)).

Die Cache-Hardware nutzt also 8 + 13 + 5 = 26 Bits aus. Also ergibt sich eine Cacheability von 2^{26} Bytes = 64 MBytes.

b) Maßnahmen zur Verbesserung des Abbildungsvermögens (auf 2 GBytes Cacheability):

- Vergrößerung des Caches. Beispiel: Datenspeicher aus vier 32k · 32 oder zwei 32k · 64 PBSRAMs (= Verdoppelung der Speicherkapazität = 16k Einträge); demgemäß muß der TAG-RAM 16k Adreßvergleichsangaben aufnehmen können. Hierdurch wird ein Adreßbit mehr ausgewertet und somit das Abbildungsvermögen auf 128 MBytes verdoppelt.
- breiteres Adreßfeld im TAG-RAM. Beispiel: 12 Bits. Hierdurch werden 4 weitere Adreßbits ausgewertet, so daß sich das Abbildungsvermögen auf das 16-fache erhöht.

Beide Maßnahmen zusammen beziehen also 5 weitere Adreßbits in das “Caching” ein. Das ergibt eine Cacheability von insgesamt 2^{31} Bytes = 2 GBytes.

Hinweise:

1. Der Übergang auf blockassoziative Abbildung bringt in Hinsicht auf das Abbildungsvermögen gar nichts (im Gegenteil: bei gleicher Gesamt-Kapazität des Datenspeichers hat z. B. ein 2-fach blockassoziativer Cache nur die halbe Cacheability eines direktabbildenden).
2. Wenn (aus Kostengründen) eine bestimmte Breite des TAG-RAM vorgegeben ist (z. B. 8 Bits - um mit einem SRAM in · 8-Organisation auszukommen), gleichzeitig aber ein größtmögliches Abbildungsvermögen gefordert wird, so muß man auf VALID- und DIRTY-Bits verzichten und dafür einen höheren Steuerungsaufwand in Kauf nehmen (eine typische Auslegung preisgünstiger PCs aus der 2. Hälfte der 90er Jahre). Es kommt nur Direktabbildung in Frage.
3. Zur Systemoptimierung: im praktischen Betrieb muß das Abbildungsvermögen nicht größer sein als nötig (d. h. als die installierte Arbeitsspeicherkapazität). Unter Beachtung dieser Bedingung kann man ohne weiteres - falls die Hardware die Möglichkeit bietet - einen Cache auf blockassoziativen Betrieb umstellen.
4. Die Cache-Steuerung legt den vom Cache erfaßten Adreßbereich typischerweise an den Anfang des Adreßraums. (Bei einer Cacheability von 64 MBytes erstreckt sich der erfaßte (“cacheable”) Adreßraum also von Adresse 0 bis zur Adresse 2^{26}-1.)
5. Die internen Caches der Intel-Prozessoren gewährleisten die Cacheability des gesamten linearen Adreßraums (vgl. die Adreßstruktur in Abbildung 3.4 - es sind tatsächlich alle 32 Adreßbits in das “Caching” einbezogen).

6. Tabelle 3.2 gibt einen Überblick über das Abbildungsvermögen typischer L2-Cache-Konfigurationen (optimiert für Pentium-Prozessoren: Cache-Eintrag = 32 Bytes, Direktabbildung)*).
7. In Abschnitt 4.3.5. ist dargestellt, wie sich ein unzureichendes Abbildungsvermögen auf die Systemleistung auswirken kann.

*): die typische Auslegung externer Caches auf Motherboards der Baujahre 1995...98.

Cache-Größe (Bytes)	Realisierungs-beispiel (PBSRAMs)	Cache-Einträge = Speicherplätze des TAG-RAM	Breite der Adreßfelder im TAG-RAM bei gefordertem Abbildungsvermögen (Cacheable Memory Size) von 64 M...4 GBytes (in Bits)					
			64 M	256 M	512 M	1 G	2 G	4 G
128k	2 · 16k · 32	4 k	9	11	12	13	14	15
256k	2 · 32k · 32	8 k	8	10	11	12	13	14
512k	4 · 32k · 32	16 k	7	9	10	11	12	13
1M	4 · 32k · 64	32 k	6	8	9	10	11	12

Tabelle 3.2 Das Abbildungsvermögen (Cacheability) direktabbildender Caches in Abhängigkeit von Cache-Größe und Organisation des TAG-RAMs

Erklärung:
Die Tabelle enthält den Zusammenhang zwischen Cache-Größe, Abbildungsvermögen und der Breite der Adreßfelder im TAG-RAM.

1. Beispiel: Welches Abbildungsvermögen (Cacheability) hat ein Cache von 256 kBytes mit einem TAG-RAM, der 10 Bits breite Adreßfelder enthält? - In der Zeile “Cache-Größe 256k” suchen wir die Adreßfeld-Breite = 10 Bits auf und finden eine “Cacheable Memory Size” von 256 MBytes.

2. Beispiel: Welche Cache-Speicherkapazität müßten wir installieren, wenn der TAG-RAM 11 Bits breite Adreßfelder enthält und eine Cacheability von 512 MBytes gefordert wird? - In der Spalte “512 M” suchen wir die Zeile mit dem Wert 11 und finden links außen die Cache-Kapazität zu 256 kBytes.

3. Beispiel: Wie breit müssen die Adreßfelder im TAG-RAM sein, wenn bei einem Cache von 512 kBytes ein Abbildungsvermögen von 2 GBytes gefordert wird? - In der Zeile “Cache-Größe 512k” suchen wir die Spalte “Cacheable Memory Size” = 2 GBytes und finden im Schnittpunkt von Zeile und Spalte 12 Bits.

Hinweise:

1. Für n-fach blockassoziative Caches sind die Werte in der Spalte “Cache-Größe” mit n zu multiplizieren.
2. Geht es um Systeme mit 16 Bytes breiten Cache-Einträgen, so ist Tabelle 4.2 weiterhin nutzbar, wenn man die Angaben der “Cacheable Memory Size” jeweils halbiert (die Tabelle reicht dann von 32 MBytes bis 2 GBytes).

Einzelheiten und Ausführungsbeispiele

Externe Caches (L2-Caches)

Ein moderner L2-Cache (Abbildung 3.7) besteht aus folgenden Funktionseinheiten:

- dem Datenspeicher (Daten-RAM),
- dem Kennzeichnungsspeicher (TAG-RAM),
- der Cache-Steuerung (Cache Controller).

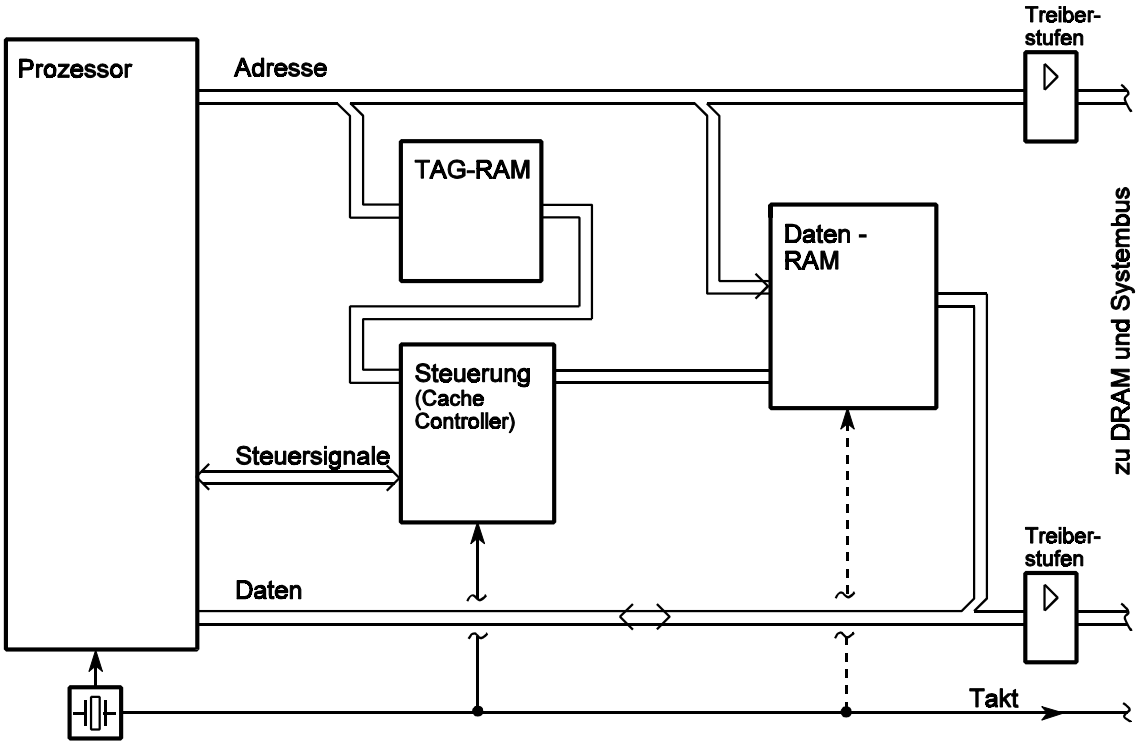

Abbildung 3.7 Struktur eines L2-Caches

Es gibt vielfältige Möglichkeiten, diese Funktionseinheiten auf Schaltkreise aufzuteilen (Tabelle 3.3), mit bestimmten Schaltkreistypen zu realisieren und die Schaltkreise z. B. auf Motherboards oder steckbaren Moduln anzuordnen (Tabelle 3.4).

Cache-Steuerung	TAG-RAM	Daten-RAM
▪ spezielle Cache-Controller-Schaltkreise, ▪ Funktionseinheiten in Motherboard-Steuerschaltkreisen (Chipsets), ▪ mit programmierbarer Logik*)	▪ übliche SRAMs, ▪ spezielle TAG-RAMs, ▪ im Controllerschaltkreis	▪ übliche SRAMs, ▪ synchrone SRAMs (Burst- oder Pipelined-Burst-SRAMs (PBSRAMs)), ▪ im Controllerschaltkreis

*): in PCs kaum üblich (ggf. nur in ziemlich alter Hardware (386/486))

Tabelle 3.3 Realisierungsmöglichkeiten der Funktionseinheiten externer Caches

Bei der Gestaltung des Cache-Subsystems können verschiedene Entwicklungsziele Vorrang haben:

- höchste Schaltungsintegration/höchste Geschwindigkeit,
- minimale Kosten,
- Erweiterbarkeit (Modularität).

Demzufolge finden wir eine Vielzahl von Ausführungen vor. Tabelle 3.4 gibt einen Überblick über Auslegungen externer Caches üblicher Personalcomputer.

Prozessor-Generation	Realisierung des externen Caches	Organisationsprinzipien	Länge eines Cache-Eintrags	Speicherkapazität*)
386	Controller mit eingebautem TAG-RAM, Datenspeicher mit üblichen (asynchronen) SRAMs	direktabbildend oder 2-fach block-assoziativ, Write Thru	1 Cache Line = 4 Bytes (32 Bits), 1 Block = 8 oder 16 aufeinanderfolgende Cache Lines (32 oder 64 Bytes)	32 oder 64 kBytes
486	TAG-RAM im Controller bzw. mit asynchronen SRAMs, Datenspeicher mit üblichen SRAMs	direktabbildend oder 2-fach block-assoziativ, Write Thru oder Write Back	1 Cache Line = 16 Bytes (128 Bits)	64 oder 128 kBytes
Pentium (586; 686 mit Sockel-7-Interface)	TAG-Speicher mit üblichen SRAMs oder speziellen TAG-RAMs, Datenspeicher mit üblichen (asynchronen) oder mit synchronen (Pipelined Burst) SRAMs	direktabbildend, Write Back	1 Cache Line = 32 Bytes (256 Bits)	128, 256 oder 512 kBytes
P6: Pentium Pro	gesamter L2-Cache auf einem Schaltkreis in gemeinsamem Gehäuse mit dem Prozessor	4-fach block-assoziativ, mehrere wählbare Schreibverfahren (u. a. Write Back)	1 Cache Line = 32 Bytes (256 Bits)	256 oder 512 kBytes oder 1 MBytes
P6/P7: Pentium II/III/4, Celeron usw.	L2 Cache auf Prozessorschaltkreis oder auf Prozessormodul (unabhängiger Cache-Bus**))	wie Pentium Pro	wie Pentium Pro	128 kBytes... 2 MBytes

*): typische Werte; **): Intel-Fachbegriff: DIB = Dual Independent Bus Architecture

Tabelle 3.4 Typische Auslegungsformen externer Caches

Technische Gestaltung kostengünstiger und modularer Cache-Subsysteme

Der Kostendruck des Marktes zwingt gleichsam den Entwickler dazu, bestimmte Auslegungen zu bevorzugen. Typische Anforderungen sind u. a.:

- Kostenminimierung: es sind vorwiegend Speicherschaltkreise aus der Massenfertigung einzusetzen,
- Produktdifferenzierung: Motherboards sind so zu gestalten, daß man auf Grundlage eines einzigen Typs mehrere nach Leistungsvermögen und Kosten abgestufte Modelle fertigen kann,
- Modularität (Baukastenprinzip): der Anwender soll seine Konfiguration nach Bedarf erweitern können.

Die gewählten Lösungen hängen vom jeweiligen Stand der Technik ab und sind dementsprechend dem Wandel (bisweilen auch der Mode) unterworfen (Tabelle 3.5).

ältere PCs (386, 486, ältere Pentium-Modelle)	herkömmliche PCs des Massen-Marktes (586, 686)	moderne PCs, Hochleistungs-systeme
▪ Controller mit eingebautem TAG-RAM, Fassungen für Daten-RAMs, ▪ TAG- und Daten-RAMs sind asynchrone SRAMs in Fassungen	▪ steckbare Speichermoduln mit TAG- und (vorzugsweise synchronen) Daten-RAMs (z. B. COASt-Moduln), ▪ TAG- und Daten-RAMs fest auf Motherboard (bei preisgünstigeren Modellen selektiv bestückt), ▪ L2-Cache auf Prozessorschaltkreis*)	▪ komplettes Cache-Subsystem auf einem Schaltkreis (Pentium Pro), ▪ L2-Cache auf Prozessorschaltkreis (z. B. Pentium III**)), ▪ Cache-Subsystem zusammen mit Prozessor in Steckkassette (Pentium II/III, AMD Athlon usw.), ▪ Verbindung von Prozessor und L2-Cache über ein besonderes Bussystem (z. B. Dual Independent Bus Architecture (Intel))

*): dies ist z. B. eine Möglichkeit, hochleistungsfähige Prozessoren mit dem herkömmlichen Anschlußbild “Sockel 7" anzubieten (Beispiel: AMD K6-III); **): Intel-Fachbegriff: Advanced Transfer Cache (ATC)

Tabelle 3.5 Auslegungs-Varianten externer Caches (Auswahl)

Pentium-Cache-Subsystem mit Intel 82439HX

Der Schaltkreissatz 82430HX ist zum Aufbau von PCs mit Pentium-Prozessor und PCI-Bus vorgesehen (Abbildung 3.8). Es handelt sich um eine typische Bestückung von Sockel-7-Motherboards der Baujahre 1997...98. Die Cache-Steuerung ist Teil des Schaltkreises 82439HX (System Controller TXC).

Merkmale:

- Abbildungsverfahren: direktabbildend,
- Schreibverfahren: Write-Back,
- Datenspeicherkapazität: 256 oder 512 kBytes,
- Struktur der TAG-Einträge: 8...11 Bits Vergleichsadresse, 1 VALID-Bit, 1 DIRTY-Bit,
- Aufbau des TAG-Speichers: extern mit asynchronem SRAM (die Kennzeichnungsbits (VALID, DIRTY) befinden sich bei 256 kBytes vollständig, bei 512 kBytes teilweise im Steuerschaltkreis),

- Aufbau des Datenspeichers: mit externen Pipelined-Burst-SRAMs (PBSRAMs),
- Länge eines Cache-Eintrags (Cache Line Size): 32 Bytes (= 4 64-Bit-Worte ≙ 1 Pentium-Burstzyklus),
- Abbildungsvermögen: üblicherweise 64 MBytes (zusätzlich einstellbar: 128, 256 oder 512 MBytes),
- Umgehen des Cache: über Konfigurationsregister steuerbar,
- Gewährleistung der Cache-Kohärenz: siehe Abschnitt 3.3.7. (Seite 76),
- Leistungsvermögen: siehe Tabelle 3.6.

******* entfällt *******

Abbildung 3.8 Blockschaltbild eines Motherboards mit 82430HX-Schaltkreisen (Intel)

Art des Zugriffs	Dauer des Zugriffs (Prozessor-Bustakte)
wahlfreies Lesen (Single Read)	3
wahlfreies Schreiben (Single Write)	3
Lesen im Burstbetrieb	3-1-1-1
Zurückschreiben (Write Back L1 → L2) im Burstbetrieb	3-1-1-1
aufeinanderfolgende (Pipelined Back-to-Back) Lesezugriffe im Burstbetrieb	3-1-1-1-1-1-1-1

Tabelle 3.6 Zum Leistungsvermögen eines modernen externen Caches

Caches und TLBs von IA-32-Prozessoren

Vom i486 an werden IA-32-Prozessoren mit eingebauten Caches (L1-Caches) ausgerüstet. Das Cache-Subsystem typischer IA-32-Prozessoren umfaßt darüber hinaus noch TLBs und externe (L2-) Caches (Abbildung 3.9).

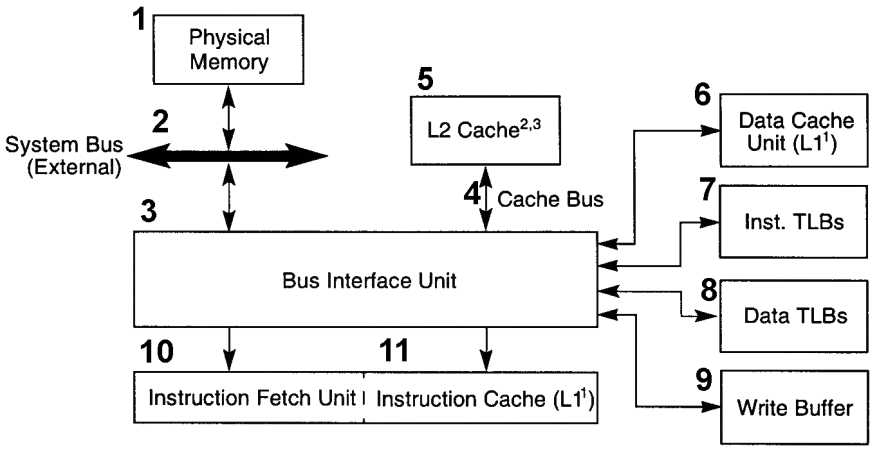

Abbildung 3.9 Caches und TLBs in IA-32-Prozessoren (Intel). Erklärung der Anmerkungen im Text

Erklärung:
1 - installierte Speichereinrichtungen als Gesamtheit (Arbeitsspeicher, Bildspeicher usw.);
2 - Prozessorbus; 3 - Bussteuerung im Prozessor; 4 - besonderes Interface zum L2-Cache;

5 - L2-Cache; 6 - interner Datencache (L1); 7 - TLBs für Befehlszugriffe; 8 - TLBs für Datenzugriffe; 9 - Schreibpuffer (für Write Posting); 10 - Befehlsleseeinheit; 11 - Befehls-Cache (L1).

Zu den Anmerkungen in Abbildung 3.9:

1) i486: nur ein universeller L1-Cache (für Daten und Befehle),
2) i486 und Pentium: kein besonderes Cache-Interface. L2-Caches sind bedarfsweise an den Prozessorbus anzuschließen.
3) P6-Klasse (PentiumPro, II, III usw.): L2-Cache ist über besonderes Interface angeschlossen und mit Prozessor zusammengebaut (in Schaltkreisgehäuse oder Steckkassette).

Interne Caches herkömmlicher IA-32-Prozessoren

Hinweis:
Die älteren Prozessortypen sind in ihrem Aufbau noch vergleichsweise überschaubar - und man hat sich seinerzeit noch Mühe gegeben, die Einzelheiten zu dokumentieren. Diese “veralteten” Prozessoren eignen sich deshalb gut als Lehrbeispiele. (Auch die ganz neumodischen Typen beruhen auf den gleichen Prinzipien*) - nur sind die Speicherkapazitäten und Taktfrequenzen höher, es ist in den Einzelheiten komplizierter, und es wird zumeist nicht so ausführlich dokumentiert.)

*): gerade Caches und TLBs sind Erfahrungssache - die Entwickler sind froh, wenn sie eine Grundsatzlösung endlich zum Funktionieren gebracht haben. In der Folge werden sie immer wieder darauf zurückgreifen und sich davor hüten, Bewährtes ohne Not über den Haufen zu werfen (Marketingleute, Journalisten usw. dürften dies allerdings kaum so darstellen...).

Der interne Cache des i486

Ergänzend zu Abbildung 3.4 veranschaulicht Abbildung 3.10 den physischen Aufbau des internen Cache. Es sind vier Moduln mit je 128 Behältern zu 16 Bytes vorgesehen. Jedes Modul enthält für jeden Behälter zusätzlich 21 TAG-Bits, die die höchstwertigen 21 Adreßbits des jeweiligen Eintrags aufnehmen (Kennzeichnungsadresse). Alle Moduln werden über die Adreßbits 10...4 gemeinsam adressiert. Die unter einer Adresse zugänglichen Behälter aller vier Moduln werden als Block bezeichnet. Jeder Block ist um 7 zusätzliche Bits erweitert, nämlich um 4 Gültigkeitsbits (VALID3...0) und um 3 LRU-Bits L2, L1, L0 (der Cache enthält also 128 Blöcke zu 512 Datenbits (= 4 Einträge zu 128 Bits), 84 TAG-Bits (= 4 Kennzeichnungsadressen zu 21 Bits), 4 Gültigkeitsbits und 3 LRU-Bits).

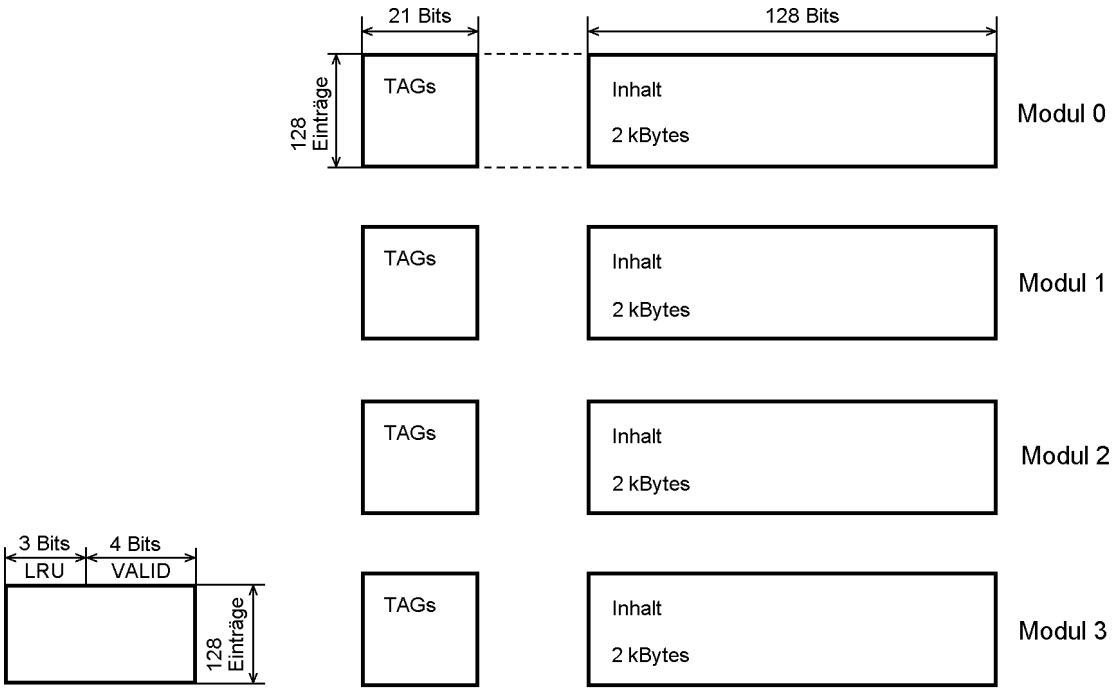

Abbildung 3.10 Die physische Organisation des Cache im i486

Cache-Verwaltung
Für jeden Behälter im Cache ist eine Belegungsanzeige erforderlich. Nicht belegte (freie) Behälter müssen vom Adreßvergleich ausgeschlossen werden; sie stehen aber zur Verfügung, um neue Einträge aufzunehmen. Für die einschlägigen Verwaltungsaufgaben sind die Gültigkeits- und LRU-Bits vorgesehen.

Gültigkeitsbits
Der Cache führt für jeden der 512 Behälter ein Gültigkeitsbit. Die Gültigkeitsbits VALID3...0 eines Blocks aus vier Behältern werden zusammen gespeichert. Bei gesetztem Gültigkeitsbit VALIDn ist der Behälter in Modul n mit einem Eintrag belegt; bei gelöschtem VALIDn ist er frei. Der Cache ist leer (flushed), wenn alle 512 Gültigkeitsbits gelöscht sind.

LRU-Bits
Soll ein neuer Eintrag in den Cache aufgenommen werden, so wird zunächst ein freier Behälter gesucht. Bei der 4-fach blockassoziativen Organisation hat man höchstens die Wahl zwischen vier Behältern im Block. Das Entscheidungsprinzip: es wird der erste freie Behälter belegt.

Sind alle vier Behälter im adressierten Block belegt, so muß einer davon freigemacht und neu belegt werden. Um diesen Behälter zu bestimmen, verwendet man einen Algorithmus, der den Behälter im Block identifiziert, zu dem in der letzten Zeit am wenigsten zugegriffen wurde (Prinzip "Least Recently Used"; LRU). Das wird näherungsweise ermittelt, wozu für jeden Block von 4 Behältern drei LRU-Bits verwaltet werden (Abbildung 3.11).

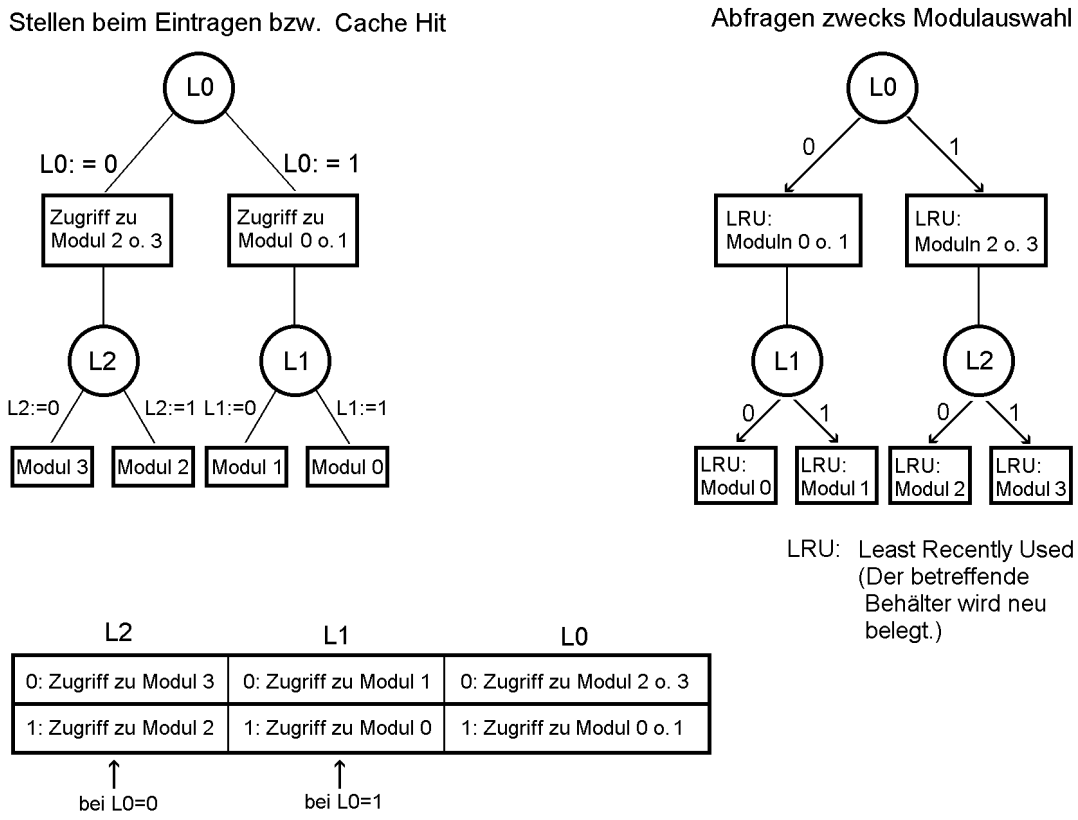

Abbildung 3.11 Nutzung der LRU-Bits

Im Fall eines Cache Hit bzw. nach dem Füllen eines Behälters wird zunächst Bit L0 gestellt (es wird gelöscht, wenn zu einem der Moduln 2 oder 3 zugegriffen wurde, ansonsten wird es gesetzt). Bit L1 unterscheidet zwischen Modul 0 (L1 = 1) und Modul 1 (L1 = 0). Sinngemäß zeigt Bit L2 an, ob Modul 2 (L2 = 1) oder Modul 3 (L2 = 0) belegt wurde.

Geht es darum, den Behälter zu finden, der zuvor am wenigsten benutzt wurde, so werden die Bits genau anders herum ausgewertet: ist L0 = 0 (Kennzeichnung für Zugriff zu Modul 2 oder 3), so gab es logischerweise unmittelbar vorher keinen Zugriff zu den Moduln 0 und 1. Das heißt, es ist L1 auszuwerten. Ist L1 = 1, so bedeutet dies, daß zuvor zu Modul 0 zugegriffen wurde, also nicht zu Modul 1, so daß Modul 1 als "least recently used" freigemacht und neu belegt wird.

Leeren des Caches
Beim Hardware-Rücksetzen bzw. beim Leeren des Cache werden alle LRU-Bits und alle Gültigkeitsbits gelöscht.

Füllen des Caches
Resultiert aus einem Lesezugriff mit einer bestimmten Adressse ein Cache Miss, so müssen die betreffenden Daten in den Cache geschafft werden. Es werden grundsätzlich nur vollständige Cache Lines (zu 16 Bytes) nachgefüllt, auch wenn beispielsweise nur ein Byte gelesen werden soll. Dieses Nachfüllen wird üblicherweise im Rahmen eines Burst-Zyklus aus vier aufeinanderfolgenden Datenzyklen durchgeführt. Ein solcher Burst-Zyklus erfordert im Idealfall (keine Wartezustände) nur insgesamt 5 Bustakte (Schema 2-1-1-1). Eine weitere

(leistungssteigernde) Besonderheit: Der Cache-Eintrag wird nicht etwa vom niedrigstwertigen 32-Bit-Wort an beginnend geholt. Vielmehr wird die Zugriffs-Adresse (die zum Cache Miss geführt hat) ausgewertet, und es wird im ersten Datenzyklus jenes 32-Bit-Wort herangeschafft, welches das adressierte Byte enthält*). Zudem werden die nachfolgenden Adressen gemäß einem besonderen Zählschema bestimmt**).

*): Der Vorteil: die Prozessor-Hardware kann sofort weiterarbeiten, nachdem die geforderte Informationsstruktur (Byte, Wort usw.) im Schaltkreis eingetroffen ist, muß also nicht warten, bis alle 16 Bytes im Cache bereitstehen.

**): dies ist eine Intel-Eigenheit, die sinngemäß auch bei den P5-Prozessoren zu finden ist (nur geht es hier um 64-Bit-Worte). Manche Prozessoren anderer Hersteller haben ein anderes Zugriffsschema. Wichtig ist: *externe Caches* müssen das jeweilige Verfahren der Burst-Adreßzählung unterstützen.

Die Caches der herkömmlichen Pentium-Prozessoren (P5-Klasse)

Interne Caches

Das interne Cache-Subsystem der Pentium-Prozessoren ist aus dem des i486 durch Weiterentwicklung hervorgegangen (Tabelle 3.7). Die wesentlichen Unterschiede: (1) es sind getrennte Befehls- und Datencaches zu je 8 kBytes vorgesehen, (2) der einzelne Cache-Eintrag ist doppelt so lang (32 Bytes), (3) die Organisation ist 2-fach-blockassoziativ, (4) im Datencache ist das Write-Back-Prinzip implementiert. Abbildung 3.12 veranschaulicht den Aufbau der internen Caches.

Prozessor	i486	Pentium
Anzahl der Caches	1; universell für Daten und Befehle	2; ein Daten- und ein Befehlscache
Größe	8 kBytes	2 · 8 kBytes
Organisation	4-fach blockassoziativ	2-fach blockassoziativ (jeder der beiden Caches)
Länge eines Eintrags (Cache Line)	16 Bytes	32 Bytes
Anzahl der Einträge	512 (4 · 128)	256 (2 · 128) je Cache
Ersetzungsalgorithmus	LRU näherungsweise (3 LRU-Bits)	LRU näherungsweise (1 LRU-Bit)
Hardware-Kontrollmaßnahmen	keine	Paritätskontrolle
Maßnahmen zur Gewährleistung der Cache-Kohärenz (in Multiprozessorsystemen)	Fremdadressenprüfung; Cache Invalidation	MESI-Protokoll

Tabelle 3.7 Die internen Caches von i486 und Pentium

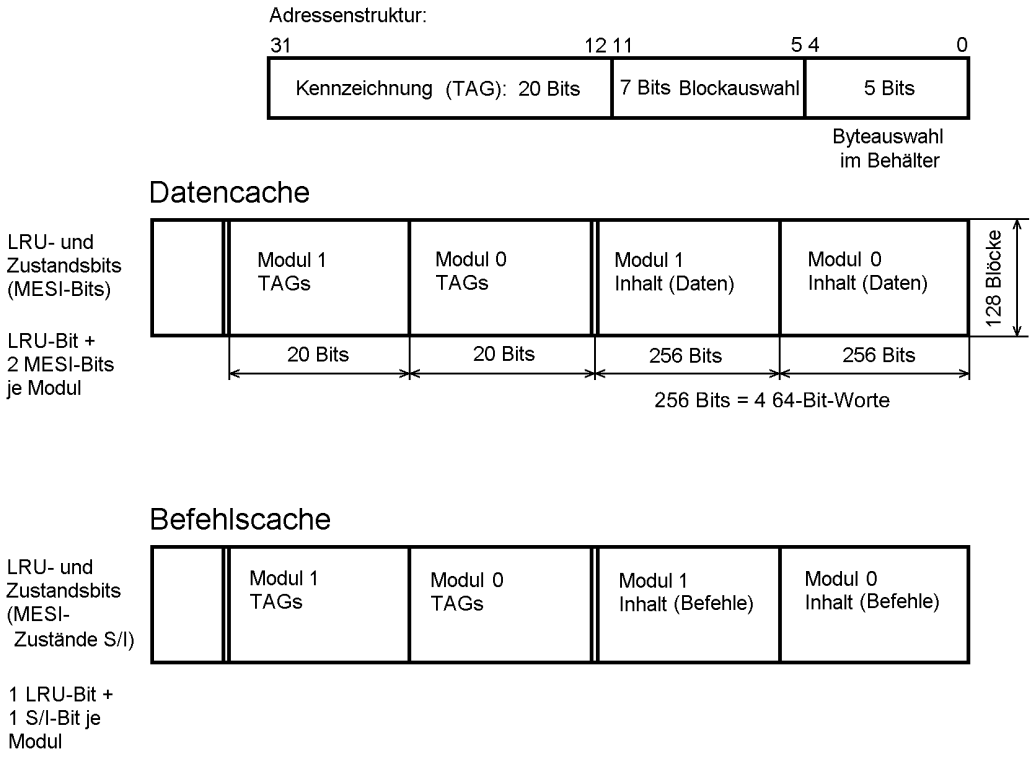

Abbildung 3.12 Die internen Caches des Pentium

Organisation des Datencaches
Der Pentium ist eine Superskalarmaschine mit zwei parallelen Verarbeitungspipelines (U- und V-Pipelines).

Beide müssen gleichzeitig aus dem Datencache bedient werden können. Dazu sind die einzelnen Behälter (zu jeweils 32 Bytes) in 8 Blöcke zu 4 Bytes unterteilt (Interleaving- Organisation). So wird es möglich, daß beide Pipelines U, V gleichzeitig auf verschiedene 4-Bytes-Doppelworte im gleichen Behälter zugreifen können, ohne sich gegenseitig zu behindern.

Write-Back-Betriebsweise
Der Datencache kann wahlweise im Write-Through- oder im Write-Back-Modus betrieben werden. Steuerung: (1) bei eingeschalteter Seitenverwaltung mittels des PWT-Bits im jeweiligen Seitentabelleneintrag, (2) hardwareseitig durch Erregen des Eingangs WB/WT.

Befehlscache
Der Befehlscache ist nur für Befehlslesezugriffe vorgesehen. Ergibt ein Schreibzugriff einen Cache Hit im Befehlscache, wird der betreffende Eintrag ungültig gemacht.

Nachfüllen bzw. Zurückschreiben
Die Burst-Zyklen sind an sich genauso organisiert wie beim i486; sie erfordern - ohne Wartezustände - ebenfalls 5 Prozessortakte. Die wesentlichen Unterschiede:

- die Datenwegbreite beträgt 64 Bits = 8 Bytes, somit werden in 4 Datenzyklen $4 \cdot 8 = 32$ Bytes übertragen,
- es gibt auch Burst-Zyklen für Schreibzugriffe (Zurückschreiben von Cache-Einträgen bei Write-Back-Betrieb).

LRU-Algorithmus
Bei einem zweifach-blockassoziativem Cache wird der näherungsweise LRU-Algorithmus - verglichen mit Abbildung 3.11 - recht einfach. Man braucht für jeden Block nur ein LRU-Bit, das bei jedem Zugriff zu einem der beiden Moduln entsprechend gestellt wird. Beispiel: bei Zugriff auf Modul 1 wird das Bit gesetzt, bei Zugriff auf Modul 0 gelöscht. Wird das Bit als gesetzt vorgefunden, so wurde zuvor auf Modul 1 zugegriffen, demzufolge wird der Eintrag im Modul 0 überschrieben.

Adreßumsetzungspuffer (TLBs)

Aufbau und Wirkungsweise des i486-TLB

Der TLB ist ein vierfach-blockassoziativer Schnellspeicher. Er kann 32 Seitentabelleneinträge aufnehmen. Abbildung 3.13 zeigt seine Struktur.

Der Inhaltsteil besteht aus vier Moduln mit je acht Behältern zu 22 Bits. Jeder Behälter nimmt eine physische Seitenrahmenadresse (20 Bits) sowie die seitenbezogenen Steuerbits PWT, PCD auf.

Jeder Behälter ist um 21 Bits erweitert, die in vier Moduln des Kennzeichnungsteils (TAG-Teils) gespeichert werden. Diese 21 Bits enthalten die höherwertigen 17 Bits der linearen Adresse (TAG-Bits), die Seitenattributbits U/S, R/W und D sowie ein Gültigkeitsbit V.

Die Bits 12-14 der linearen Adresse adressieren einen Block von vier Behältern. In jedem Modul werden die TAG-Belegungen mit den höherwertigen 17 Bits der linearen Adresse verglichen, wobei ein Vergleich nur dann wirksam wird, wenn das jeweilige Gültigkeitsbit (V) gesetzt ist. Bei Adressengleichheit werden die höherwertigen 20 Bits der physischen Adresse und die zugehörigen Seitenattribute aus dem betreffenden Behälter entnommen.

Wird in keinem der vier Moduln Adressengleichheit festgestellt, so läuft die Adreßumsetzung über explizite Zugriffe zu Seitentabellenverzeichnis und Seitentabelle ab. Die so ermittelten Umsetzungsangaben werden in den TLB eingetragen. Dabei wird zunächst der erste freie Behälter im betreffenden Modul gesucht. Ein freier Behälter ist durch ein gelöschtes Gültigkeitsbit (V) gekennzeichnet. Wurde ein solcher Behälter gefunden, so werden die Umsetzungsangaben eingetragen, und das V-Bit wird gesetzt.

Wurde kein freier Behälter gefunden (alle vier V-Bits sind bereits gesetzt), so wird einer der vier Behälter mit der aktuellen Umsetzungsangabe belegt. Dazu wird das gleiche Verfahren wie beim internen Cache verwendet (näherungsweises Last Recently Used mit 3 LRU-Bits je Block; vgl. Abbildung 3.11).

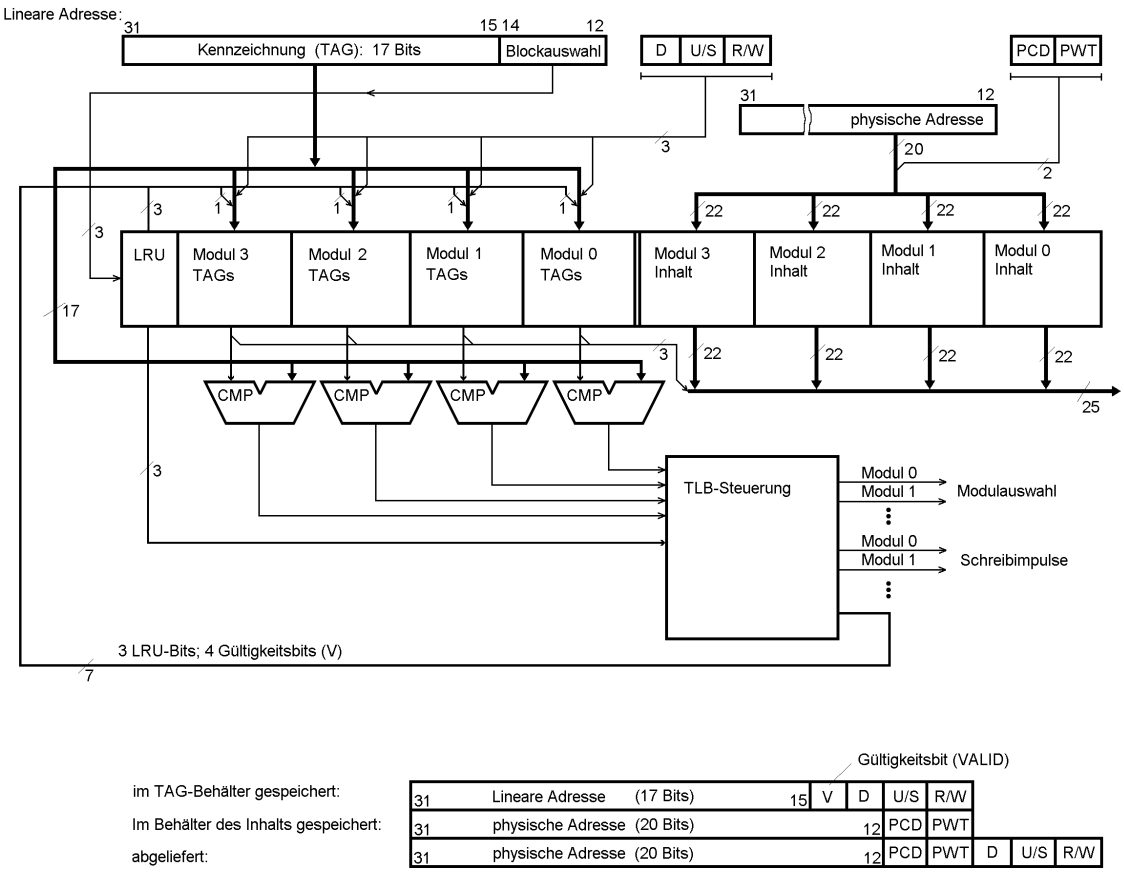

Abbildung 3.13 Der TLB des i486

Nutzung

Ein TLB-Eintrag kommt in folgenden Schritten zustande:

- der Eintrag aus dem Seitentabellenverzeichnis wird geholt,
- erforderlichenfalls wird das Zugriffsanzeigebit (A) im Verzeichniseintrag gesetzt,
- der Seitentabelleneintrag wird geholt,
- erforderlichenfalls werden die Bits der Zugriffs- und Änderungsanzeige (D, A) gesetzt,
- aus diesen Angaben wird der TLB-Eintrag aufgebaut und in den TLB geschrieben.

Trefferrate

Für den TLB wird eine durchschnittliche Trefferrate von 98% angegeben, d. h. nur 2% aller Adressenumsetzungen erfordern Speicherzugriffe zu den Tabellen.

Programmseitige Beeinflussung des TLB

Der TLB ist nicht abschaltbar und wirkt für Programme völlig transparent (alle Zugriffe werden von der Hardware gesteuert).

Der TLB muß geleert werden, wenn das Seitentabellenverzeichnis bzw. Seitentabellen geändert wurden (ansonsten könnten "alte" TLB-Inhalte für die folgenden Adreßumsetzungen verwendet werden). Hierzu gibt es entsprechende Maschinenbefehle (vgl. Tabelle 3.9).

Die TLBs der Pentium-Prozessoren

Die Pentium-Prozessoren enthalten insgesamt 3 TLBs, die den beiden internen Caches direkt zugeordnet sind (Tabelle 3.8, Abbildung 3.14).

Prozessor	i486	Pentium
Anzahl der TLBs	1; universelle Nutzung für Daten und Befehle	3; einer für Befehlszugriffe, einer für Datenzugriffe mit 4 kBytes Seitenlänge, einer für Datenzugriffe mit 4 MBytes Seitenlänge
Größe	32 Einträge	32 Einträge für Befehlszugriffe, 64 Einträge für Datenzugriffe mit 4 kBytes Seitenlänge, 8 Einträge für Datenzugriffe mit 4 MBytes Seitenlänge
Organisation	4-fach blockassoziativ	4-fach blockassoziativ (jeder der drei TLBs)
Ersetzungsalgorithmus	LRU näherungsweise (3 LRU-Bits)	LRU näherungsweise (3 LRU-Bits)
Hardware-Kontrollmaßnahmen	keine	Paritätskontrolle

Tabelle 3.8 Die TLBs von i486 und Pentium

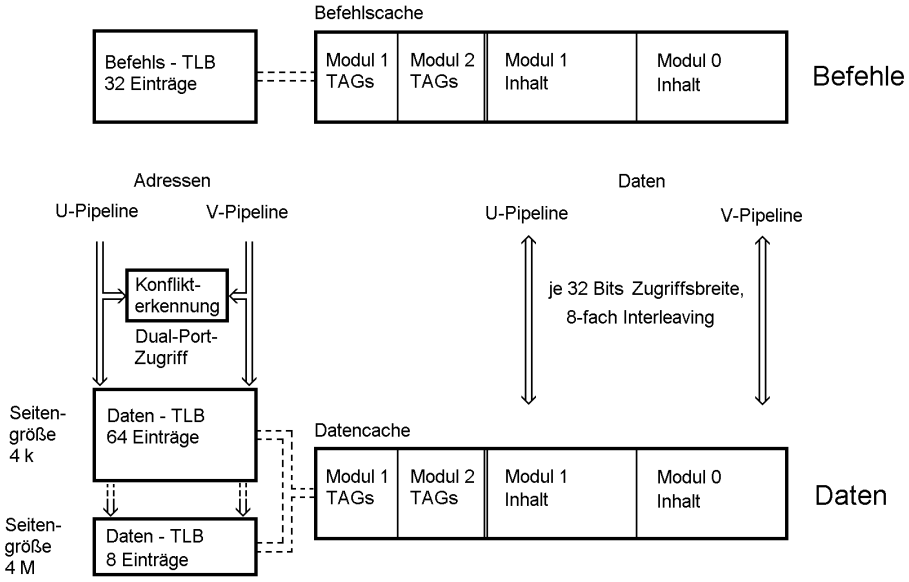

Abbildung 3.14 Die TLBs der Pentium-Prozessoren

Hinweis:
Moderne Hochleistungsprozessoren enthalten typischerweise mehrere TLBs (vgl. Anhang 1).

Aufteilung:

- nach Art des Zugriffs (TLBs für Daten, TLBs für Befehle),
- nach der Seitenlänge (TLBs für "gewöhnliche" Seiten (zu 4 kBytes), TLBs für lange Seiten (zu 2 oder 4 MBytes)).

Programmseitige Steuerung von Caches und TLBs
Wir beziehen uns weiterhin auf IA-32. Caches und TLBs können durch Steuerbits in Steuerregistern und in den Datenstrukturen der Seitenverwaltung gesteuert sowie mittels verschiedener Maschinenbefehle beeinflußt werden (Abbildung 3.15, Tabelle 3.9).

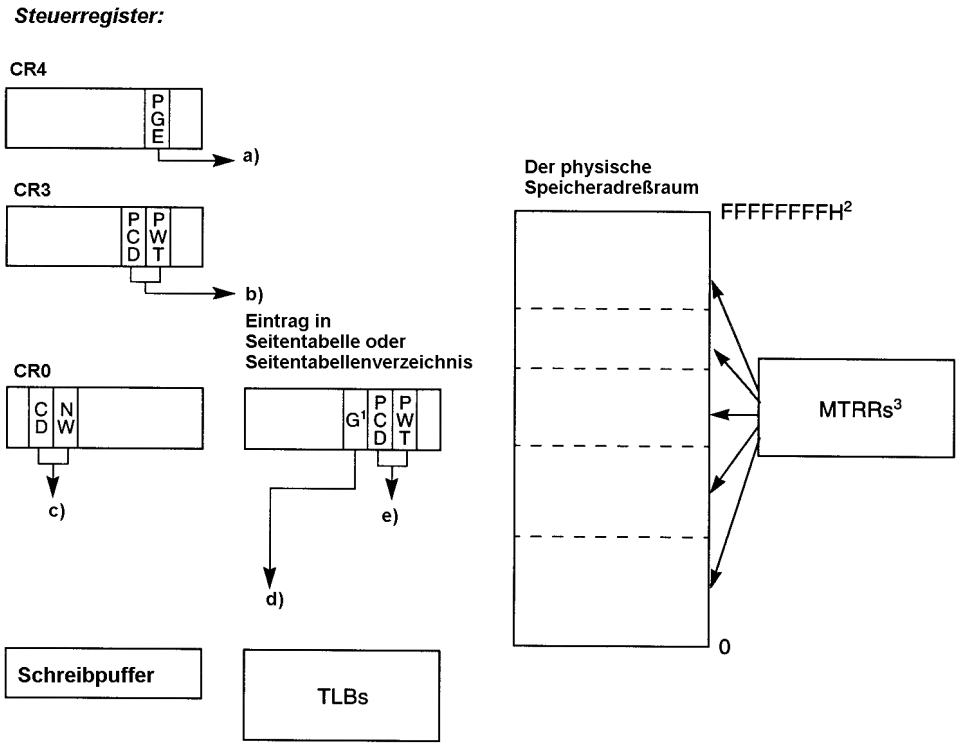

Abbildung 3.15 Zur programmseitigen Beeinflußbarkeit der Caches und TLBs (Intel)

Erklärung:

a) Steuerregister CR4 (nur P6). PGE = Page Global Enable. Schaltet die Wirksamkeit des G-Bits in den Seitentabelleneinträgen ein.

b) Steuerregister CR3: die Bits PCD, PWT betreffen das Seitentabellenverzeichnis.

c) Steuerregister CR0: allgemeine Betriebsartensteuerung. CD = Cache Disable (generelles Ein- und Ausschalten), NW = Not Write Through (Ein- und Ausschalten des Write-Through-Betriebs).

d) G-Bit (Global). Ist PGE in Steuerregister CR4 gesetzt, so bewirkt das G-Bit in einem Seitentabelleneintrag, daß die betreffenden Umsetzungsangaben auch dann im TLB verbleiben, wenn der TLB-Inhalt als ungültig gekennzeichnet wird (TLB Invalidation).

e) Bits in den Tabelleneinträgen der Seitenverwaltung. Werden wirksam, wenn auf die betreffende Seite zugegriffen wird. PCD = Page-Level Cache Disable. Bei Zugriff wird

Cache umgangen. PWT = Page-Level Write Through. Bei Zugriff wird Write-Through-Betrieb erzwungen.

Zu den Anmerkungen in Abbildung 3.15:

1) nur in P6-Prozessoren,
2) Endadresse = FFFF FFFFH bei 32-Bit-Adressierung, F FFFF FFFFH bei 36-Bit-Adressierung,
3) Speicherbereichsregister gibt es nur in P6-Prozessoren. Bei Pentium: hardwareseitige Steuerung über Eingänge KEN (≙ Caching ein/aus) und WB/WT (≙ erzwinge Write Through). Bei i486 nur Eingang KEN. Eingänge werden vom Steuerschaltkreis (North Bridge) erregt.

Maschinenbefehl	Wirkung
INVD: Invalidate Internal Caches	Leeren der internen Caches (alle Einträge werden ungültig). Externe Caches werden zum Leeren aufgefordert. Kein Zurückschreiben veränderter Cache-Einträge
WBINVD: Write Back and Invalidate Caches	Leeren der internen Caches (alle Einträge werden ungültig). Zuvor werden alle veränderten Cache-Einträge zurückgeschrieben. Externe Caches werden zum Zurückschreiben und Leeren aufgefordert
jedes Schreiben in's Steuerregister 3, Verändern einzelner Bits in anderen Steuerregistern	Leeren der TLBs (alle Einträge werden ungültig)
INVLPG: Invalidate TLB Entry	ein einzelner Eintrag - der der im Befehl angegebenen Seite entspricht - wird in den TLBs ungültig gemacht
PREFETCH: vorbeugendes Holen von Cache-Einträgen (Intel SSE (ab Pentium III), AMD 3DNow)	von der im Befehl angegebenen Adresse wird ein Cache-Eintrag geholt und in bestimmte Cache-Ebenen eingetragen (so daß er bei ggf. nachfolgenden wirklichen Zugriffen schon bereitsteht)

Tabelle 3.9 Maschinenbefehle zum Beeinflussen von Caches und TLBs

Hinweis:
Die Einzelheiten sind ziemlich spitzfindig und im Service kaum von Nutzen. Grundsätzlich finden wir aber derartige Funktionen in praktisch allen modernen Hochleistungsprozessoren. Wir sollten deshalb wissen, worum es hierbei geht:

- das "Caching" ist ein- und auszuschalten,
- bestimmte Speicherbereiche sind vom "Caching" auszuschließen,
- bei bestimmten Zugriffen sind bestimmte Funktionen der Cache-Verwaltung zu erzwingen (betrifft Write Through, Cache-Kohärenz usw.),
- Caches und TLBs sind gelegentlich zu leeren (dabei sind veränderte Einträge von Write-Back-Caches in den Arbeitsspeicher zurückzuschreiben).

Zudem werden gelegentlich weitere Funktionen vorgesehen, die es der Systemsoftware ermöglichen, den Caches und TLBs gleichsam Hilfestellung zu geben*) (vgl. das "dauernde Einlagern" von TLB-Einträgen (G-Bits) und den Prefetch-Befehl).

*): Grundgedanke: die Software weiß am besten, welche Seiten ständig benötigt werden, auf welche Adreßbereiche demnächst zugegriffen werden wird usw. Anwendung: in wirklich

zeitkritschen Abläufen, vor allem in solchen, die umfangreiche Datenströme betreffen (Audio- und Videoverarbeitung, Darstellung bewegter Bilder usw.).

Zur Cache-Kohärenz
Das Problem ist immer dann zu lösen, wenn mehrere Einrichtungen (z. B. Prozessoren in Multiprozessorsystemen) Speicherinhalte verändern können, aber damit zu rechnen ist, daß sich Kopien dieser Speicherinhalte in verschiedenen Caches befinden (vgl. Abschnitt 3.2.7.). In IA-32-Systemen wird dieses Problem nach dem Prinzip der Adreßprüfung (Address Snooping) gelöst. Hierzu hat man verschiedene Signalprotokolle implementiert. Um das Grundsätzliche zu zeigen, wollen wir drei davon kurz vorstellen.

1. Das 486-Protokoll

Der Prozessor kann seine Cache-Einträge prüfen, es ist aber Sache des Systems (mit anderen Worten: zusätzlicher Hardware), diese Prüfung anzufordern und dem Prozessor die zu prüfenden Adressen zuzuführen.

Der Ablauf:

1. der Adreßbus wird von außen durch Erregen des Eingangssignals AHOLD freigeschaltet,
2. die zu prüfende Adresse wird auf den Bus gelegt,
3. die Cache-Steuerung prüft, ob für diese Adresse ein gültiger Eintrag im Cache vorhanden ist,
4. ist dies der Fall (Cache Hit), wird der Eintrag ungültig gemacht (Cache Line Invalidation),
5. die aufgeschaltete Adresse wird wieder vom Bus genommen,
6. der nächste 486-Lesezugriff mit einer entsprechenden Adresse führt somit zu einem Cache Miss - und der sorgt dafür, daß der entsprechende 16-Byte-Block wiederum in den Cache eingetragen wird.

2. Das MESI-Protokoll (Pentium u. a.)

MESI (in anderer Schreibweise: M.E.S.I.) ist ein prozessor-unabhängiges Protokoll zur Gewährleistung der Cache-Kohärenz (also praktisch ein Industriestandard). Das Protokoll weist jedem Cache-Eintrag einen von vier möglichen Zuständen zu und legt die Regeln für die Zustandsübergänge fest. Die Bezeichnung (MESI) ergibt sich aus der Aufzählung dieser vier Zustände (Modified, Exclusive, Shared, Invalid).

Abbildung 3.16 veranschaulicht das Szenarium, in dem das MESI-Protokoll von Bedeutung ist: mehrere Prozessoren mit lokalen Caches sind über einen Systembus (Speicherbus) an einen gemeinsamen Arbeitsspeicher angeschlossen. Der lokale Cache eines Prozessors ist dabei typischerweise ein Verbund von internem und externem Cache. Hierbei gilt das Prinzip "Consistency by Inclusion", das heißt, der externe Cache enthält grundsätzlich alle Einträge des internen. Gemäß dem MESI-Protokoll überwachen alle Cache-Controller (interne und externe) die Zugriffe, die über den Systembus laufen (Bus Snooping) und reagieren gegebenenfalls entsprechend.

Im folgenden wollen wir die einzelnen Zustände lediglich kurz beschreiben. Zu Einzelheiten

müssen wir auf die einschlägigen Schaltkreis- und Architekturhandbücher verweisen.

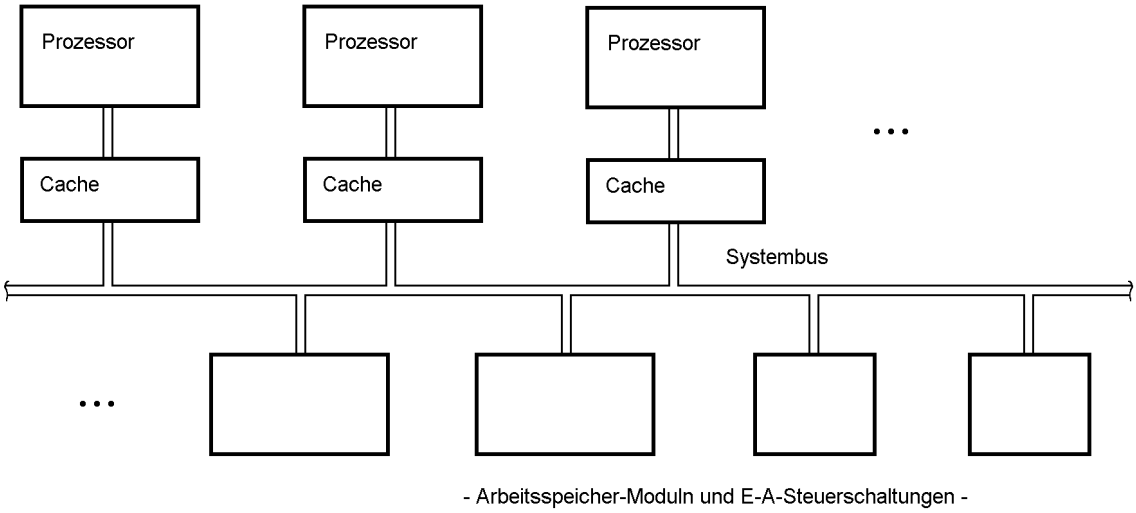

Abbildung 3.16 Multiprozessorsystem mit lokalen Caches

Modified
Der Eintrag befindet sich nur in einem Cache und wurde vom Prozessor verändert (modifiziert). Der betreffende Inhalt des Arbeitsspeichers ist somit nicht auf aktuellem Stand. Der "eigene" Prozessor kann uneingeschränkt mit einem solchen Eintrag arbeiten (Lesen und Schreiben). Erst wenn ein anderer Prozessor auf eine entsprechende Adresse zugreifen will (der Cache Controller muß dies durch Überwachen des Systembus erkennen), ist ein Zurückschreiben in den Arbeitsspeicher erforderlich.

Exclusive
Der Eintrag befindet sich nur in einem Cache, und zwar unverändert. Somit enthält auch der Arbeitsspeicher den aktuellen Stand. Ein Schreibzugriff des Prozessors führt zum Übergang in den Zustand "Modified".

Shared
Der Eintrag kann sich auch in einem anderen Cache befinden. Der Prozessor kann den Eintrag jederzeit lesen. Schreibzugriffe werden hingegen automatisch gemäß dem Write-Thru-Prinzip ausgeführt, der Arbeitsspeicher wird also aktualisiert. Die anderen Cache-Controller müssen dies erkennen und gegebenenfalls auch die Aktualisierung veranlassen (bzw. den Eintrag ungültig machen, so daß beim nächsten Zugriff zwangsläufig der aktuelle Wert aus dem Arbeitsspeicher geholt wird).

Invalid
Für die betreffende Adresse gibt es keinen gültigen Cache-Eintrag, folglich ist ein Zugriff zum Arbeitsspeicher notwendig. Auch in diesem Falle müssen die anderen Cache-Controller gegebenenfalls den Zustand in ihren Caches ändern. (Beispiel: in Prozessor A gibt es keinen Cache-Eintrag, wohl aber in Prozessor B. Dessen Eintrag hat demzufolge den Zustand "Exclusive". Wenn nun Prozessor A die betreffenden Daten aus dem Arbeitsspeicher holt, so müssen beide Einträge (in den Caches von A und von B) als "Shared" gekennzeichnet werden.)

Hinweis:
Typischerweise werden die vier Zustände in zwei Bits je Eintrag codiert. Bei internen Befehlscaches sind nur die Zustände "Shared" und "Invalid" von Bedeutung, so daß dort jeweils ein Bit ausreicht.

3. Kohärenzprüfung im Verbund von L1- und L2-Cache

Dies ist Aufgabe des Steuerschaltkreises. Unser Beispiel: das Motherboard gemäß Abbildung 3.8. Der Steuerschaltkreis (TXC) überwacht die Aktivitäten auf dem PCI-Bus und kann somit zweckmäßig reagieren, wenn PCI-Busmaster auf den Arbeitsspeicher zugreifen wollen (über den PIIX-Schaltkreis werden auch DMA- und ISA-Busmaster-Zugriffe erfaßt). Infolge der Write-Back-Organisation sind sowohl Lese- aus auch Schreibzugriffe zu berücksichtigen. Der Schaltkreis steuert die entsprechenden Prüfabläufe (Snooping) in beiden Caches (L1 und L2) und verhält sich so, wie in Tabelle 3.10 angegeben.

Fremdzugriff (von Busmaster)	Treffer (Snoop-Hit)		Reaktion
	im L1-Cache	im L2-Cache	
Lesen	nein	nein	-
	nein	ja[1)]	zurückschreiben und ungültig machen
	ja[1)]	nein	zurückschreiben
	ja[1)]	ja[1)]	aus dem L1-Cache zurückschreiben, im L2-Cache ungültig machen
Schreiben	nein	nein	-
	nein	ja	aus dem L2-Cache zurückschreiben[2)] und ungültig machen
	ja	nein	aus dem L1-Cache zurückschreiben[2)] und ungültig machen
	ja	ja	aus dem L1-Cache zurückschreiben[2)], in beiden Caches ungültig machen

1): beim Lesen sind nur Treffer auf modifizierte Einträge (enthalten geänderte Daten, DIRTY-Bit ist gesetzt) von Bedeutung; 2): ein Treffer auf einen modifizierten Eintrag führt zum Zurückschreiben, ein Treffer auf einen unmodifizierten Eintrag lediglich zum Ungültigmachen (VALID-Bit → 0)

Tabelle 3.10 Zur Wirkungsweise der Kohärenzprüfung (Snooping)

Der Fremdzugriff betrifft stets den Arbeitsspeicher. Der eigentliche Arbeitsspeicherzugriff findet erst *nach* einem eventuellen Zurückschreiben statt. Es wird stets versucht, vor Ausführung des Fremdzugriffs im Arbeitsspeicher aktuelle Daten bereitzustellen. Hierzu werden modifizierte Cache-Einträge zurückgeschrieben. Das Zurückschreiben eines Cache-Eintrags bedeutet nichts anderes, als daß der entsprechende Speicherbereich (hier: von 32 Bytes Länge) im Arbeitsspeicher aktualisiert wird. Ein Lesezugriff findet somit stets aktuelle Daten vor, und beim Schreiben werden die akutellen Daten des Prozessors und des Busmasters gleichsam zusammengemischt. (Beispiel: der Prozessor hat das 3. und 4. Byte in dem betreffenden Speicherbereich verändert, und der Busmaster schreibt Daten in die 19. und 20. Byteposition desselben Bereichs.) Durch das Ungültigmachen des Cache-Eintrags wird gewährleistet, daß

beim nächsten Zugriff des Prozessors wieder eine aktuelle Kopie des Speicherbereichs als neuer Cache-Eintrag herangeschafft wird.

Optimierung

Was ist zu optimieren?

- die Systemleistung (= Durchsatz (Throughput)),
- die Kosten.

Die Optimierung betrifft:

- die Systemstrukturen im allgemeinen,
- die Wirkprinzipien und Technologien,
- die zu installierenden Speicherkapazitäten,
- vielfältige Feinabstimmungen (Speichertechnologie - Speicherkapazität - Bustakt (Timing) - Größe und Assoziativität von Caches usw.).

Im folgenden wollen wir ausgewählte Optimierungsfragen im PC-Bereich diskutieren. Hierbei stützen wir uns sowohl auf anschauliche Überlegungen als auch auf Beispiele und Meßergebnisse.

Sind die Beispiele und Ergebnisse veraltet?
Sie sind teils älteren Datums, aber deswegen noch längst nicht veraltet oder unbrauchbar. Machen wir uns folgendes klar:

- moderne Hochleistungssysteme sind so kompliziert, daß die vielfältigen Zusammenhänge gar nicht mehr überblickt werden können (auch die heutigen Wissenschaftler greifen sich stets nur Teilprobleme heraus),
- wir beginnen naheliegenderweise mit dem Einfacheren, gehen Schritt für Schritt vor und betrachten jedes Teilproblem der Optimierung zunächst für sich allein,
- Ergebnisse, die bereits vor Jahren gefunden wurden, wird man heutzutage als gegeben voraussetzen und nicht wieder von neuem erforschen - es bleibt also gar nichts anderes übrig, als sich mit älterem Datenmaterial zufrieden zu geben und daraus seine Schlüsse zu ziehen.

Caches
Die Cache-Optimierung ist im wesentlichen eine Erfahrungswissenschaft. Es bleibt kaum anderes übrig, als sich verschiedene Konzepte einfallen zu lassen und deren Nutzen durch Leistungsmessung (bzw. Simulation) anhand typischer, häufig vorkommender Einsatzfälle zu bestimmen.

Trefferraten
Tabelle 4.1 zeigt Trefferraten für direktabbildende und blockassoziative Caches verschiedener Auslegung.

Hinweis:
Die Werte resultieren aus statistischen Analysen von Programmabläufen und geben eher vorsichtige Erwartungen wieder. In der Praxis können Trefferraten in weiten Bereichen schwanken.

Faustregel:
Kein Cache ist besser als ein zu kleiner. Erklärlich: denn beim Cache Hit läuft alles bestens, jeder Cache Miss hingegen kostet sogar mehr Zeit als ein gewöhnlicher Arbeitsspeicherzugriff - weil der gesamte Behälter nachgefüllt werden muß. Wir wollen das einmal grob durchrechnen. Es bezeichnet

h	die Trefferrate in Prozent,
b	die Länge eines Eintrags in Bytes,
t_{ah}	die Zugriffszeit bei einem Cache Hit,
t_{aw}	die Zugriffszeit für einen wahlfreien Zugriff,
t_{ae}	die (gesamte) Zugriffszeit für das Holen eines Cache-Eintrags.

Ohne Cache müssen wir in allen Zugriffsfällen (≙ 100%) die Zugriffszeit t_{aw} ansetzen. Bei einem Cache mit der Trefferrate h haben wir in h% aller Zugriffsfälle mit t_{ah} zu rechnen und in (100-h)% mit t_{ae}. Wenn der Cache also etwas nützen soll, muß gelten:

$$t_{ah} \cdot h + t_{ae} \cdot (100 - h) < t_{aw} \cdot 100$$

Um die Formel etwas zu vereinfachen, setzen wir t_{ah} gleich Null (ist bei einem internen Cache durchaus gerechtfertigt, da keine besonderen Taktzyklen für den Zugriff erforderlich sind). Des weiteren vernachlässigen wir Beschleunigungsmöglichkeiten für die aufeinanderfolgenden Zugriffe beim Holen eines Cache-Eintrages und setzen $t_{ae} = b \cdot t_{aw}$. Damit ergibt sich:

$$b \cdot t_{aw} \cdot (100 - h) < t_{aw} \cdot 100$$

Nach entsprechender Umformung der Ungleichung erhalten wir:

$$h > \frac{100\ (b\ -\ 1)}{b} \quad (\text{in } \%)$$

Setzen wir für b einige technisch sinnvolle Werte ein:

- b = 8 erfordert eine Trefferrate h von mindestens 87,5%,
- b = 16 erfordert eine Trefferrate h von mindestens 93,75%,
- b = 32 erfordert eine Trefferrate h von mindestens 96,9%,

um gegenüber einer Ausführung ohne Cache überlegen zu sein. Aus Tabelle 4.1 ist ersichtlich, wie groß der Cache wenigstens sein sollte, um die jeweilige Trefferrate zu gewährleisten.

(Noch ein Hinweis, falls Sie genauer mitgerechnet haben: Je kürzer der Cache-Eintrag, um so besser - so scheint es jedenfalls. Bei 1-Byte-Einträgen wäre auch der kleinste Cache überlegen. Diese Auslegung ist aber zu aufwendig und hat in der Praxis erhebliche leistungsseitige Nachteile. Erklärlich, denn zum Holen von 1-Byte-Einträgen kann man keine Burst-Zyklen vorsehen.)

Speicherkapazität	Eintrag	Abbildungsprinzip	Trefferrate
1 kBytes	4 Bytes	direkt	41%
8 kBytes	4 Bytes	direkt	73%
16 kBytes	4 Bytes	direkt	81%
32 kBytes	4 Bytes	direkt	86%
32 kBytes	4 Bytes	2-fach	87%
32 kBytes	8 Bytes	direkt	91%
64 kBytes	4 Bytes	direkt	88%
64 kBytes	4 Bytes	2-fach	89%
64 kBytes	4 Bytes	4-fach	89%
64 kBytes	8 Bytes	direkt	92%
64 kBytes	8 Bytes	2-fach	93%
128 kBytes	4 Bytes	direkt	89%
128 kBytes	4 Bytes	2-fach	89%
128 kBytes	8 Bytes	direkt	93%

Tabelle 4.1 Trefferraten verschiedener Cache-Auslegungen (Intel)

Welche Maßnahme nützt am meisten?
Wir gehen von einem vorhandenen (typischerweise: kleineren und einfacheren) Cache aus. Ein genauerer Blick in Tabelle 4.1 zeigt, daß sehr kleine Caches recht geringe Trefferraten haben und deshalb zunächst einmal auf eine höhere Speicherkapazität gebracht werden müssen. Von da an (laut Tabelle ab 32 kBytes) ist eine bestimmte Rangfolge der Nützlichkeit zu erkennen:

1. Verlängerung des Cache-Eintrags von 4 auf 8 Bytes (32 kBytes: von 86% auf 91%, 64 und 128 kBytes: von ca. 89% auf ca. 93%),
2. Verbesserung des Abbildungsverfahrens (höhere Assoziativität). Folgende Auslegungen haben praktisch gleiche Trefferraten: (1) 4 Bytes lange Einträge: 64 kBytes 2- oder 4-fach blockassoziativ und 128 kBytes direktabbildend; (2) 8 Bytes lange Einträge: 64 kBytes 2-fach blockassoziativ und 128 kBytes direktabbildend. (Bestätigt die Faustregel: ein 2-fach blockassoziativer Cache hat die gleiche Trefferrate wie ein direktabbildender doppelter Kapazität.)
3. Vergrößerung der Speicherkapazität.

Schreibverfahren: Write Through oder Write Back?
Für das Write-Back-Prinzip wird ein Leistungsgewinn von rund 10% angegeben. Des weiteren gewährleistet das Write-Back-Prinzip eine deutlich geringere Busbelastung (Simulationswerte nach Intel: 15...25% bei Write Back; 50...75% bei Write Through).

Für dicht gekoppelte Hochleistungs-Multiprozessorsysteme ist Write Back eine praktische Notwendigkeit: denn nur so können mehrere Hochleistungsprozessoren am selben Bus arbeiten, ohne sich gegenseitig zu behindern.

Größe und Auslegung externer Caches
Die wesentlichen Parameter sind:

- die gesamte Größe bzw. (Daten-) Speicherkapazität des Cache,
- die Länge eines Cache-Eintrags (Cache Line),
- die Anzahl der parallel vorgesehenen Speicherblöcke bzw. Adreßumsetzungswege (Assoziativität),
- das Schreibverfahren (Write Thru oder Write Back).

Abbildung 4.1 zeigt die Auswirkungen verschiedener Cache-Größen und Auslegungen auf das Leistungsvermögen eines Pentium-Systems. Abbildung 4.2 veranschaulicht die Auswirkungen des gewählten Schreibverfahrens.

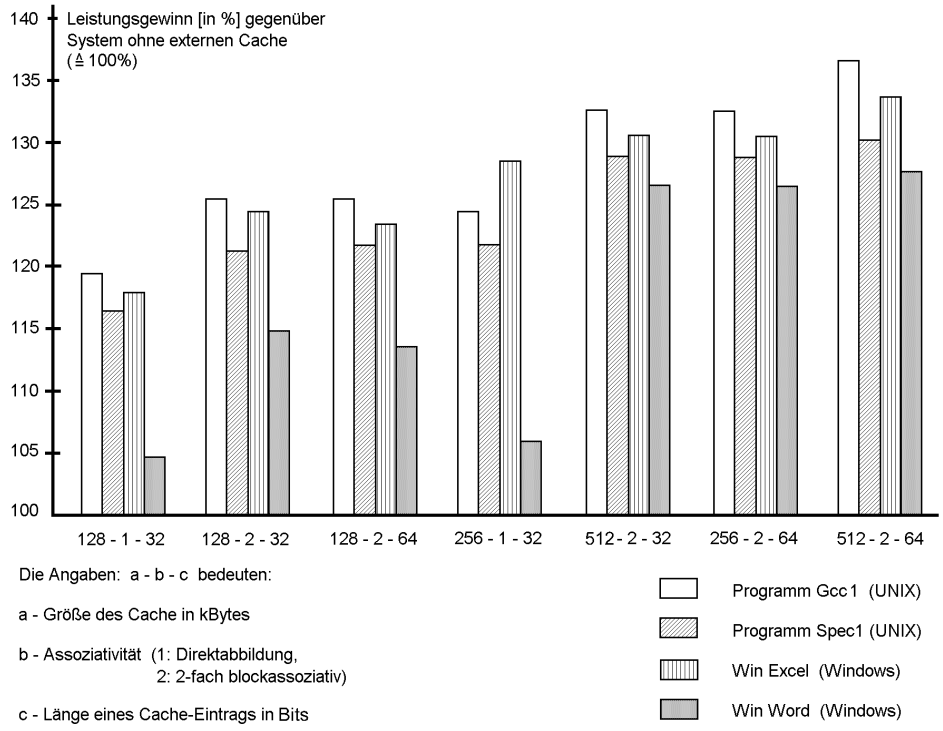

Abbildung 4.1 Auswirkungen verschiedener Cache-Größen und -Auslegungen (Intel)

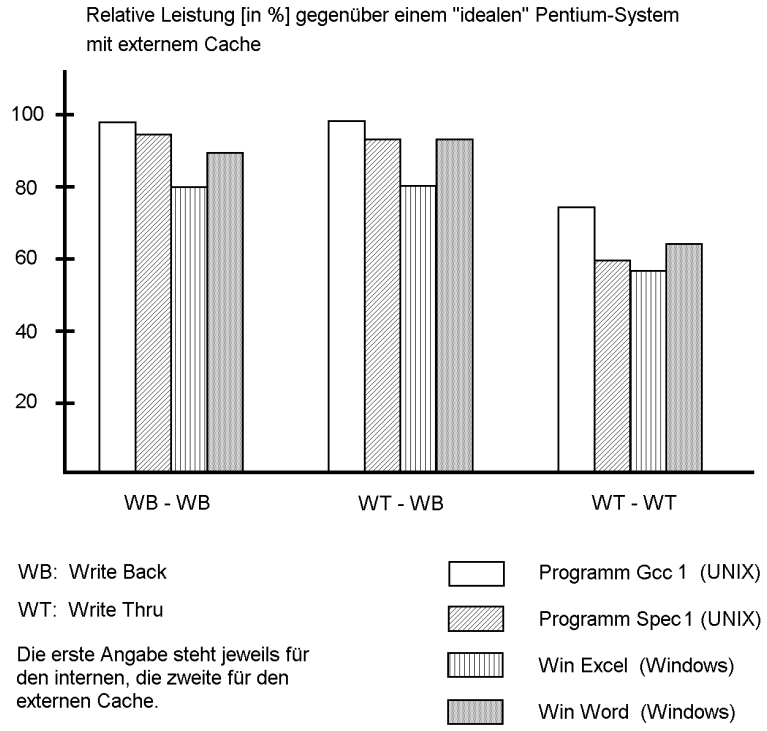

Abbildung 4.2 Auswirkungen des gewählten Schreibprinzips (Intel)

Die Ergebnisse:

1. bereits der kleinste der untersuchten Second-Level-Caches bringt in den meisten Fällen eine Leistungssteigerung von mehr als 15 % gegenüber einem System ohne externen Cache,
2. die Erhöhung der Assoziativität bringt den meisten Leistungsgewinn, dann folgt die Verlängerung der Cache Line, dann erst die Vergrößerung des Caches[*)],
3. eine Auslegung mit 256 kBytes, 2-fach blockassoziativer Organisation und Einträgen von 32 Bytes verspricht das beste Kosten-Nutzen-Verhältnis. Insbesondere die recht kostspielige Vergrößerung von 256 auf 512 kBytes bringt nur 2...3% mehr an Leistung.
4. das Leistungsvermögen hängt wesentlich vom Schreibverfahren des externen Caches ab - vgl. die schlechte Leistung der Konfiguration WT - WT. Weshalb? - weil das Zurückschreiben in den Arbeitsspeicher wirklich Zeit kostet. Der externe Cache sollte deshalb ein Write-Back-Cache sein

*): die Caches sind hier wesentlich größer als die von Tabelle 4.1. Deshalb die andere Reihenfolge (im Vergleich zu Seite 79).

Kein Cache - interner Cache

Bei Prozessoren ohne Cache (IA-32-Beispiel: 386) ist zu erwarten, daß die weitaus meisten Buszugriffe Lesezugriffe sein werden, wovon wiederum ein beträchtlicher Anteil auf das Holen der auszuführenden Befehle entfällt*).

*): betrachten wir einen typischen Verknüpfungsbefehl: ADD A, B (Wirkung: <A> := <A> + <B>). Es sind zu holen (1) der Befehl selbst, (2) der Operand A, (3) der Operand B. Schließlich ist (4) das Ergebnis (A) zurückzuschreiben (also: entfallen rund gerechnet 3 Lesezugriffe auf einen Schreibzugriff).

Die Nutzung eines internen Caches (IA-32-Beispiel: i486) hat demgegenüber folgende Auswirkungen:

- der Prozessor benötigt den Bus weniger häufig,
- der Anteil der Schreibzugriffe übertrifft deutlich jenen der Lesezugriffe.

Siehe dazu Abbildung 4.3.

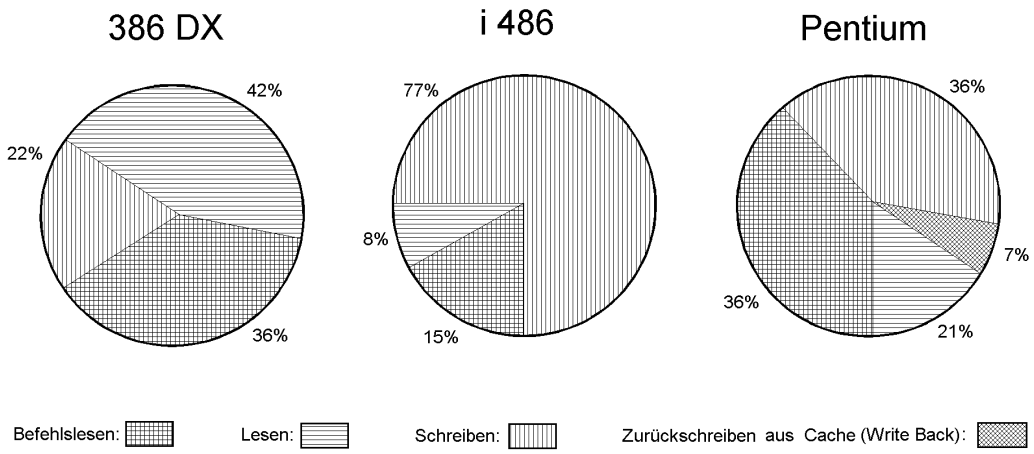

Abbildung 4.3 Ausnutzung des Prozessorbus im Vergleich (Intel)

Hinweise:

1. Die Busausnutzung des 386 (als Beispiel eines klassischen Einzelprozessors) entspricht etwa unseren Erwartungen (1 Schreibzugriff auf 3 Lesezugriffe ≙ Schreiben 25%, Lesen 75%).
2. Der hier zugrundeliegende i486 hat einen Write-Thru-Cache. Ersichtlich ist der hohe Anteil an Schreibzugriffen. Die Folgerung: weitere Leistungssteigerung erfordert Übergang auf Write Back (moderne 486-Typen, P5, P6 usw.).

Die Verarbeitungsleistung in verschiedenen Betriebsweisen

Der Bus moderner Hochleistungsprozessoren ist für extrem schnelle Zugriffs-Zyklen (Burst-Zyklen) ausgelegt, um die internen Speichermittel (Caches, Befehls-Warteschlangen usw.) in kürzester Zeit auffüllen zu können. Diese Buszugriffe haben den Charakter von Blocktransporten. Die typische Auslegung bei IA-32: der erste Zyklus erfordert zwei Takte, während die folgenden mit je einem Takt auskommen. Im System sollten die Verbindungen zwischen Prozessor und Speicher (als Verbund von externem Cache und Arbeitsspeicher) so gestaltet sein, daß solche Zugriffsweisen auch tatsächlich ausgenutzt werden können. Die Wichtigkeit solcher Vorkehrungen wird anhand von Leistungsmessungen deutlich.

Abbildung 4.4 zeigt den Anteil der verschiedenartigen Zugriffe beim Durchlauf eines Meßprogramms (Dhrystone) unter verschiedenen Betriebsbedingungen. Abbildung 4.5 veranschaulicht dazu den Leistungsgewinn bei Ausnutzung des internen Caches und der Burst-Zugriffe. Bezugsgröße ist die Laufzeit unter folgenden Bedingungen: (1) Cache nicht genutzt, (2) alle Zugriffe sind Einzelzugriffe (keine Bursts). Der Leistungsgewinn LG errechnet sich folgendermaßen:

$$LG = \left(\frac{\text{Laufzeit ohne Cache und Burst}}{\text{Laufzeit im jeweiligen Betriebsfall}}\right) \cdot 100\%$$

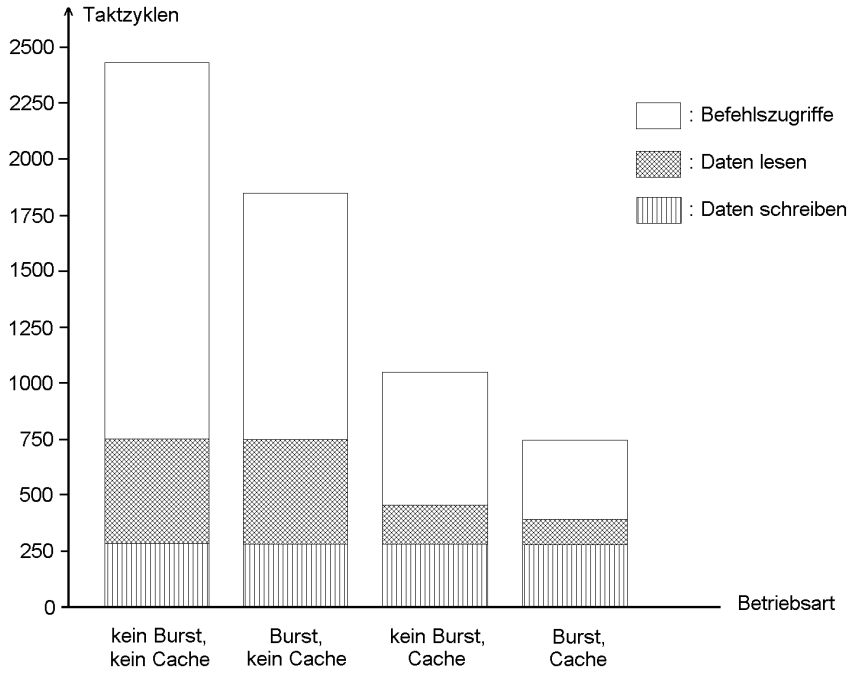

Abbildung 4.4 Art und Dauer der Buszyklen während eines Dhrystone-Durchlaufs

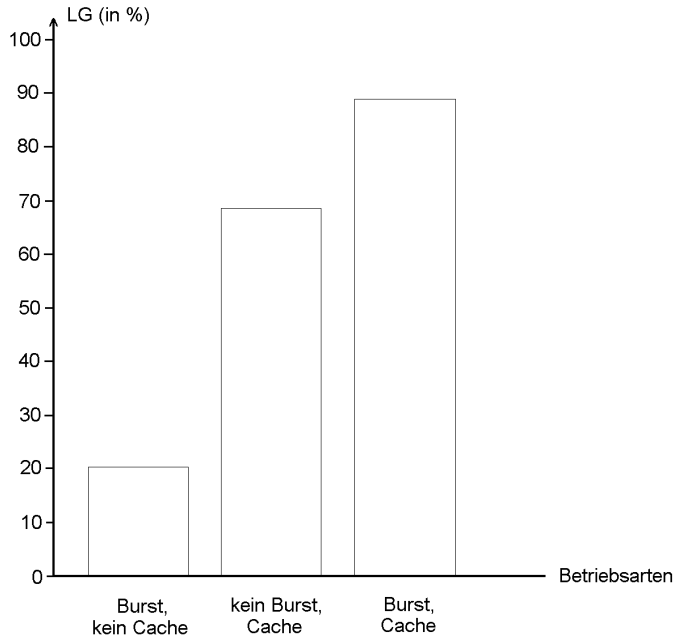

Abbildung 4.5 Leistungsgewinne in verschiedenen Betriebsarten (beide Abbildungen nach: Funk/Baumgartner: Leistungssteigerung beim i486. Design & Elektronik 14 vom 2. 7. 91)

Hinweise:

1. Abbildung 4.4 veranschaulicht auf andere Weise das gleiche wie Abbildung 4.3: ohne Cache sind die weitaus meisten Zugriffe Lesezugriffe.
2. Aus beiden Abbildungen ist ersichtlich, daß der Burst-Betrieb allein deutlich weniger Leistungsgewinn bringt als der Cache. Nutzanwendung: vor allem den Cache optimieren - dann sind auch einige “restliche” Wartezustände am Bus zu verschmerzen. Mit anderen Worten: ein Prozessor mit eingebautem Cache und einfachem Bus ist besser als einer ohne Cache, aber mit schnellem Bus.
3. Ein PC mit Hochleistungsprozessor sollte auch ein Hochleistungs-Speichersubsystem haben, das über ein entsprechend breites und schnelles Interface angeschlossen ist.