2. Ansatzpunkt: Reduktion der Penalty

2.1. Early Restart und critical word first

Beide Techniken basieren darauf, die Wartezeit der CPU auf das Mindestmaß zu beschränken. Early restart lädt den Block wie bisher, und erkennt anhand der Offset-Bits der Adresse, wann das Wort im Block vorhanden ist, das die CPU jetzt braucht. Dieses wird dann sofort (gleichzeitig mit dem Laden des Restes des Blocks) an die CPU übermittelt, die weitermachen kann.
Komplizierter ist critical word first. Hier wird der Block nicht in seiner ursprünglichen Reihenfolge im Speicher gelesen, sondern zuerst wird nur der Teil gelesen, der von der CPU gebraucht wird. Dazu werden ebenfalls die Offset-Bits interpretiert. Critical word first = wrapped fetch = requested word first.

Beide Techniken lohnen sich nur bei großen Blöcken (z.b. > 64Byte).

2.2. Level-2-Caches

Der Architekt steht bei der gestiegenen Prozessorleistung vor dem Konflikt, dass er einerseits einen schnelleren und andererseits einen größeren Cache braucht, aber small is fast. Wie kann er nun einen größeren Cache mit der Geschwindigkeit eines kleinen Caches bekommen?
Durch hinzuschalten einer weiteren Ebene der Speicherhierarchie zwischen Cache und Hauptspeicher: den Level-2-Cache.
1. Level 1: klein und schnell genug um mit der Geschwindigkeit des Prozessors Schritt halten zu können,
2. Level 2: groß genug, um die Gesamt-Miss-Rate unter einen erträglichen Wert zu senken.

$\text{Durchschnittliche Zugriffszeit}_{L1} = \text{Hit Zeit}_{L1} + \text{Miss Rate}_{L1} * \text{Miss Penalty}_{L1}$
$\text{Miss Penalty}_{L1} = \text{Hit Zeit}_{L2} + \text{Miss Rate}_{L2} * \text{Miss Penalty}_{L2}$

Also
$\text{Durchschnittliche Zugriffszeit}_{L1} = \text{Hit Zeit}_{L1} + \text{Miss Rate}_{L1} * (\text{Hit Zeit}_{L2} + \text{Miss Rate}_{L2} * \text{Miss Penalty}_{L2})$

In dieser Formel benutzen wir die „lokale Miss-Rate“ für L2

Lokale Miss-Rate := Anzahl Misses / Anzahl Zugriffe auf diesen Cache

Globale Miss-Rate := Anzahl Misses / Zugriffe auf den Speicher insgesamt

Beispiel:
Level 1: 40 Misses auf 1000 Zugriffe
Level 2: 20 Misses auf 1000 Zugriffe

wie sind die globalen und lokalen Miss-Raten?

Level 1: globale Miss-Rate = lokale Miss-Rate = 40 / 1000 = 4%

Level 2: globale Miss-Rate = 20 / 1000 = 2%
lokale Miss-Rate = 20 / 40 = 50%

Beispiel:

Wir betrachten ein Cache-System mit einem Level-1-Cache, der 95% Trefferrate vorweist und einem Level-2-Cache, der eine globale Trefferrate von 99,9% hat. Der Zugriff auf den L-2-Cache dauert 3 Zyklen, der auf den L-1-Cache nur einen. Ein Fehlzugriff im L-2-Cache bedeutet einen Penalty von 70 Zyklen.
Wie ist die Durchschnittliche Zugriffszeit? Wie wäre sie ohne L2-Cache?

Beispiel:

Wir betrachten ein Cache-System mit einem Level-1-Cache, der 95% Trefferrate vorweist und einem Level-2-Cache, der eine globale Trefferrate von 99,9% hat. Der Zugriff auf den L-2-Cache dauert 3 Zyklen, der auf den L-1-Cache nur einen. Ein Fehlzugriff im L-2-Cache bedeutet einen Penalty von 70 Zyklen.
Wie ist die Durchschnittliche Zugriffszeit? Wie wäre sie ohne L2-Cache?

$$DZZ_{L1} = \text{Hit Zeit}_{L1} + \text{Miss Rate}_{L1} * (\text{Hit Zeit}_{L2} + \text{Miss Rate}_{L2} * \text{Miss Penalty}_{L2})$$

Wie ist die Miss Rate_{L2}? Es ist die lokale. Die globale Miss Rate ist 0,1%. Aber nur 5% der Zugriffe erfolgt auf den L2-Cache, da die Trefferrate L1 bereits 95% ist. Also ist die lokale Miss Rate von L2 = 0,1% / 5% = 2%.

$$DZZ_{L1} = 1 + 5\% * (3 + 2\% * 70) = 1 + 5\% * 4{,}4 = 1{,}22$$

$$DZZ_{oL2} = 1 + 5\% * 70 = 4{,}5$$

Das folgende Diagramm zeigt das Verhalten eines L-2-Cache im Vergleich zu dem eines einstufigen Caches derselben Größe in Bezug auf die Miss-Raten. Wir sehen zwei Effekte:
1. Die globale Miss-Rate unterscheidet sich kaum, jedenfalls nicht für größere Caches und nur für diese ist die L-2-Variante interessant.
Wir gewinnen also nur durch die verminderte Penalty.
2. Die lokale Miss-Rate gibt wenig Information über die Qualität des Systems aus L-1 und L-2-Cache. Sie ist ja eine Funktion der Miss-Rate des L-1-Caches und daher lediglich für die Berechnung der durchschnittlichen Zugriffszeit ein notwendiges Maß.

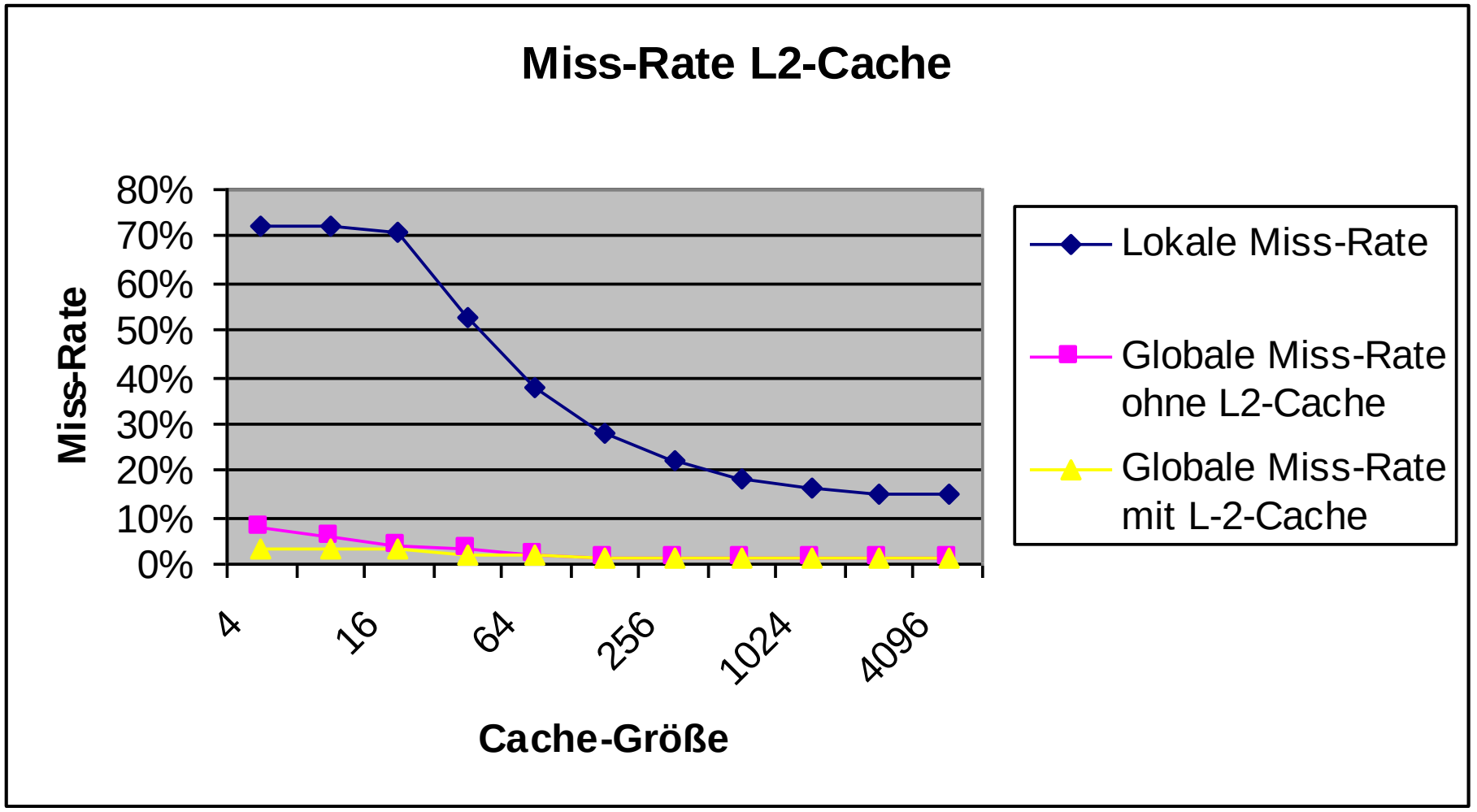
Was ist bei L-2-Caches anders als bei L1?
1. Sollte viel größer sein, Faktor 256 bis 1024

2. Höhere Assoziativität lohnt sich, um Konflikt-Misses zu minimieren

3. Größere Blockgrößen sind sinnvoll, obwohl es auch sehr effizient sein kann, die Blockgröße an die des L1-Caches anzugleichen. Dies zahlt sich nämlich aus, wenn zurückgeschrieben werden muß. Dann müssen nur die dirty-Daten ersetzt werden.

4. Übliche und sinnvolle Kombination: Write-through mit schreib-Puffer bei L1, write Back bei L2.

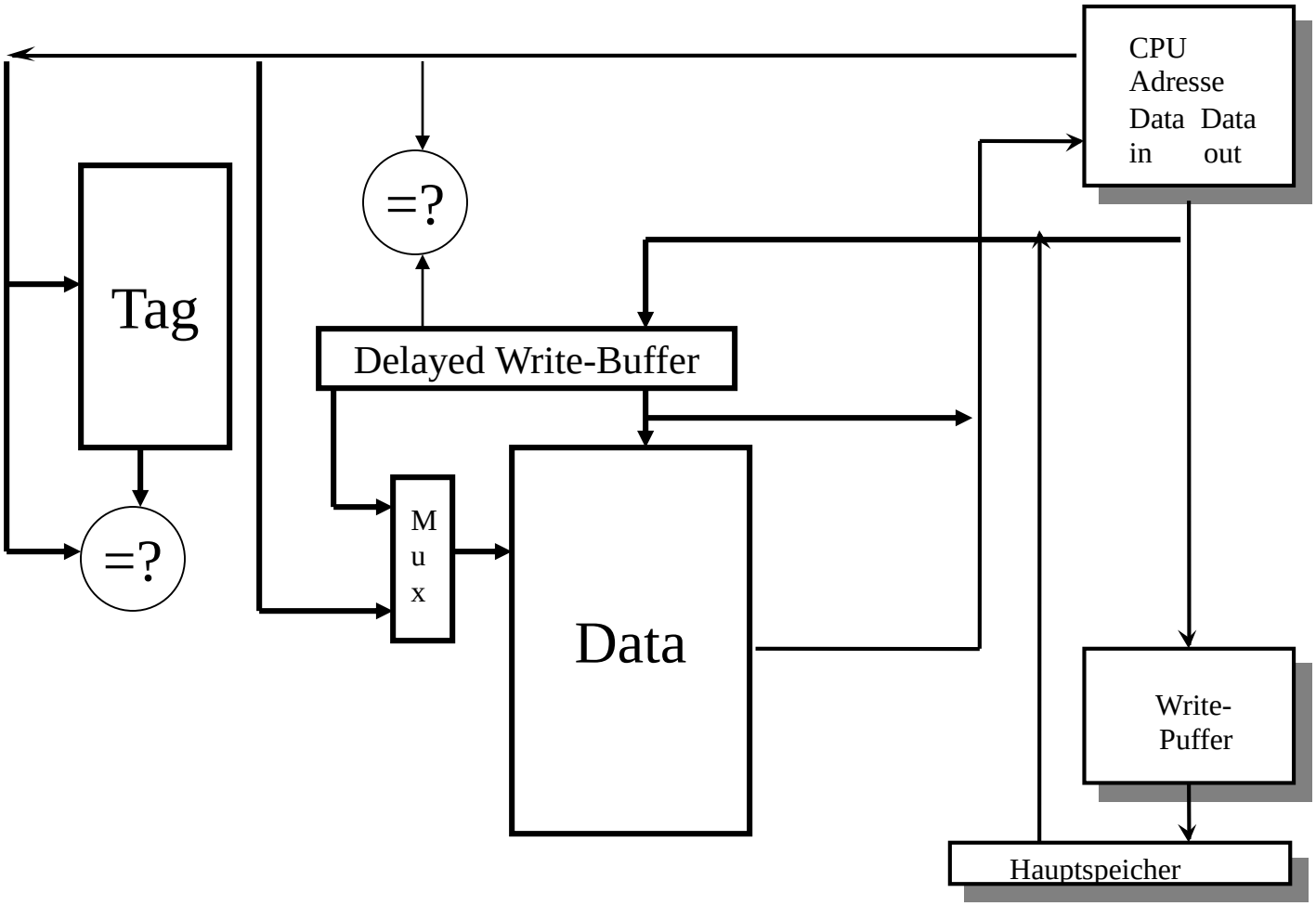
Miss-Raten mit L-2-Cache



3. Technik: Verringerung der Hit-Zeit
3.1. Kleine und einfache (direkt gemappte) Caches
small is fast

3.2. Gepipelinete Schreibzugriffe
Schreiben dauert beim Hit länger, weil zuerst der Tag-Check erforderlich ist, und erst dann geschrieben werden darf. Daher fügt man ein Register zwischen CPU und Cache, in das zuerst geschrieben werden kann. Dann macht die CPU weiter, als ob es ein Hit war. Währenddessen wird der Tag geprüft. Wenn Hit: kopieren des Registers in den Cache, wenn Miss, ganz normales wirte around oder fetch on write Verhalten.

Bild siehe nächste Folie.


CPU
Adresse
Data in
Data out
=?
Tag
Delayed Write-Buffer
=?
M
u
x
Data
Write-
Puffer
Hauptspeicher

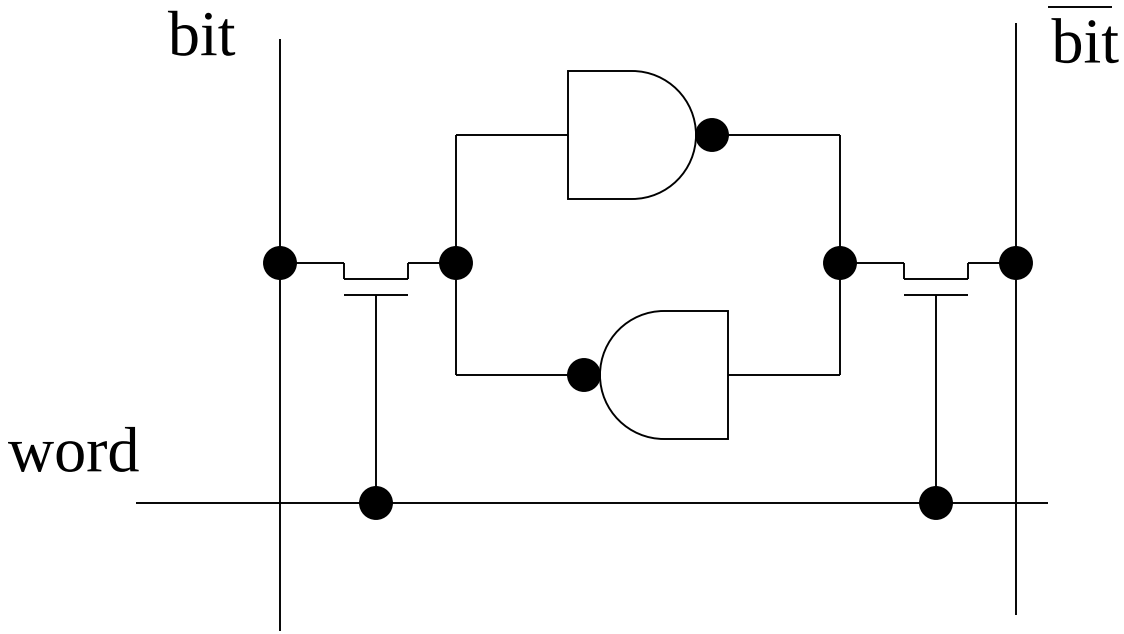
6.4. Hauptspeicher

Der Hauptspeicher ist die nächst niedrige Stufe nach dem Cache (ggf. L-2-Cache) in der Speicherhierarchie. Die Performance des Hauptspeichers hängt von zwei Größen ab: Zugriffszeit und Bandbreite.

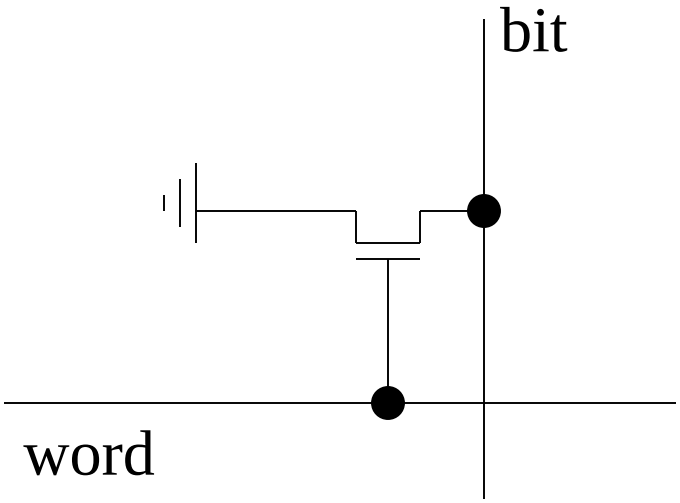
Außer der Zugriffszeit ist auch die Zykluszeit ein wichtiger Parameter für Speicher. Gemeint ist nicht Taktzyklusszeit sondern Speicherzykluszeit. Ein Speicherzyklus besteht aus dem Anlegen der Adresse, dem Lesen der Daten und dem Schreiben der Daten. Es verwundert zunächst, daß auch bei einem normalen Lesen aus dem Speicher der letzte Teil dieses Zyklus, das Schreiben, erforderlich ist. Dies hängt mit der gegenwärtigen (und sicher in den nächsten 10 Jahren noch verwendeten) DRAM-Technologie zusammen, wie wir gleich sehen werden.

Die Hauptunterscheidung bei Speicherchips ist zwischen SRAM und DRAM zu treffen. SRAM (static random access memory) ist schnell und teuer, DRAM (dynamic random access memory) ist langsam und billig. SRAM ist etwa 8 - 16 mal so schnell und 8-16 mal so teuer wie DRAM.

Speicherchips sind quadratisch organisiert, in Zeilen und in Spalten. Ein Bit wird im SRAM durch zwei rückgekoppelte Inverter gespeichert, die Ihren Zustand gegenseitig stabilisieren, solange die Versorgungsspannung anliegt. Zum Lesen und zum Schreiben wird durch das Anlagen einer Adresse über einen Adressdekodierer eine (von sovielen, wie der Bausteine Worte speichern kann) word-Leitung aktiviert, die zwei n-Transistoren ansteuert. Über diese Transistoren kann dann der Inhalt der RAM-Zelle gelesen oder überschrieben werden.



Während SRAM auf Performance (kurze Zugriffszeit) hin optimiert werden, ist bei DRAMS die Kapazität (Anzahl der zu speichernden Bits) zu maximieren und die Pin-Anzahl zu minimieren (um die Bausteine hinreichend klein zu halten). Letzteres wird erreicht durch zeitmultiplexen der Adressleitungen in Zeilenadresse (RAS, row-adress-strobe) und Spaltenadresse (CAS, column adress strobe). Ersteres erreicht man durch Speicherung eines Bits mit einem einzigen Transistor und einer speziell auf reguläre Strukturen optimierten CMOS-Fertigungsprozess. Dies bringt zwei Nachteile mit sich: Da der gespeicherte Wert nur auf der GATE-SORUCE Kapazität des Transistors gespeichert wird, verliert er mit der Zeit sein Potential. Das bedeutet,



daß nach einer gewissen Zeit die RAM-Zelle „refreshed“ werden muß. Dies geschieht durch Auslesen und wieder Schreiben desselben Wertes. Der zweite Nachteil ist die Tatsache, das beim Lesen die Information in der RAM-Zelle zerstört wird. Dies wird ebenfalls dadurch gelöst, daß der gelesene Wert sofort nach dem Lesen wieder zurückgeschrieben wird. Aus der Notwendigkeit dieses zusätzlichen Zurückschreibens ergibt sich die oben erwähnte Differenz zwischen Zugriffs- und Zykluszeit.

Typischerweise wird DRAM für Hauptspeicher verwendet und SRAM für Caches.

Man kann beobachten, daß die Größe der Hauptspeicher etwa mit der Leistungssteigerung der Prozessoren mithält, also sich etwa alle drei Jahre vervierfacht. Leider nimmt aber die Zykluszeit nicht in entsprechendem Maße ab. Stattdessen haben sich die Verhältnisse im Speicherchipbereich etwa entsprechend folgender Tabelle entwickelt:

Jahr	Chip-Kapazität	RAS	CAS	Zykluszeit
1980	64KBit	180ns	75ns	250ns
1983	256KBit	150ns	50ns	220ns
1986	1MBit	120ns	25ns	190ns
1989	4MBit	100ns	20ns	165ns
1992	16MBit	80ns	15ns	120ns
1995	64MBit	65ns	10ns	90ns
1998	256MBit	50ns	10ns	75ns
2005	1GBit	30ns	7ns	37ns
2012	4GBit	15ns	5ns	21ns
2016	16GBit	???	???	< 10ns

Durch die langsame Entwicklung der DRAMs im Gegensatz zu den Prozessoren entstand mittlerweile eine deutliche Lücke. Wie können wir die Memory-Performance erhöhen?

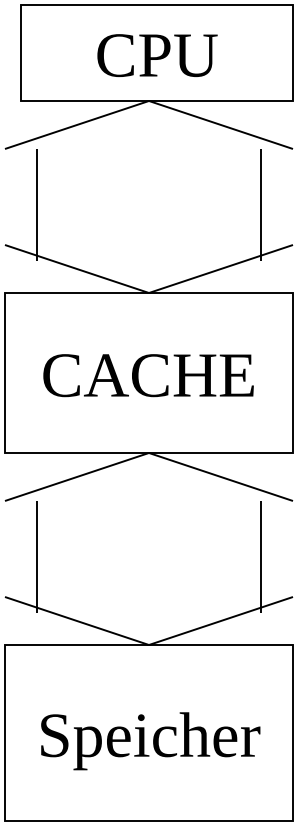
1. Technik: Höhere Bandbreite: Breiterer Hauptspeicher

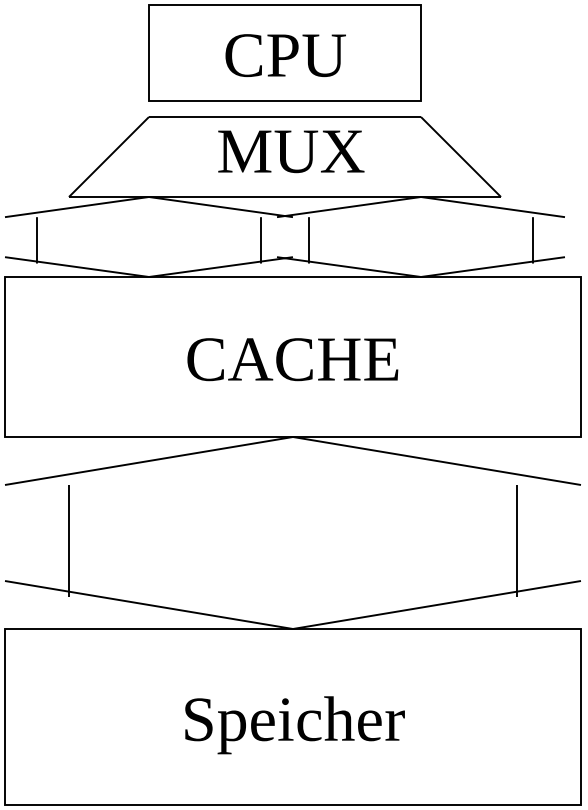
Def: Die Breite eines Speichers ist die Anzahl der Bits, die bei einem Lesezugriff der Außenwelt bereitgestellt werden können.

L-1-Caches werden in der Regel mit der physikalischen Breite eines Wortes der CPU ausgelegt, weil die meisten Zugriffe diese Wortbreite erfordern. Systeme ohne L-2-Cache passen häufig die Breite des Hauptspeichers dieser Wortbreite an. Verdoppeln oder vervierfachen von Cache und Hauptspeicher verdoppeln und vervierfachen dementsprechend die Bandbreite zwischen den beiden Stufen der Hierarchie.

Beispiel:
Sei der Hauptspeicher folgendermaßen organisiert:
4 Zyklen zum Senden der Adresse
24 Zyklen Zugriffszeit pro Wort
4 Zyklen zum Senden eines Wortes zum Cache
Breite eines Cache-Blockes: 4-Worte

Die Miss-Penalty wäre im einfachsten Falle 4*(4+24+4) = 128 Zyklen.
Bei einer doppelwortbreiten Speicherorganisation würde sich die Penalty halbieren, bei einer vier-Wort-breiten Organisation vierteln.
Dies würde allerdings einen doppelt- bzw. vierfach-breiten Bus zwischen Cache und Hauptspeicher implizieren.

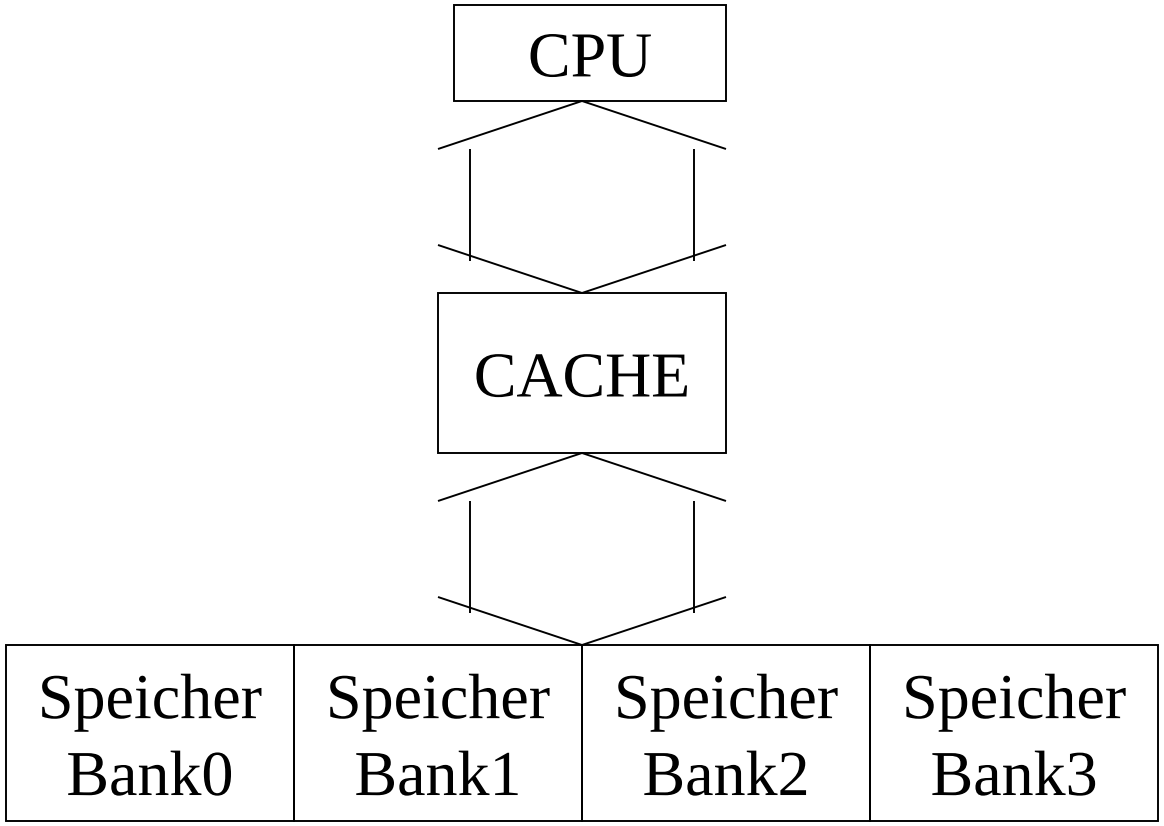




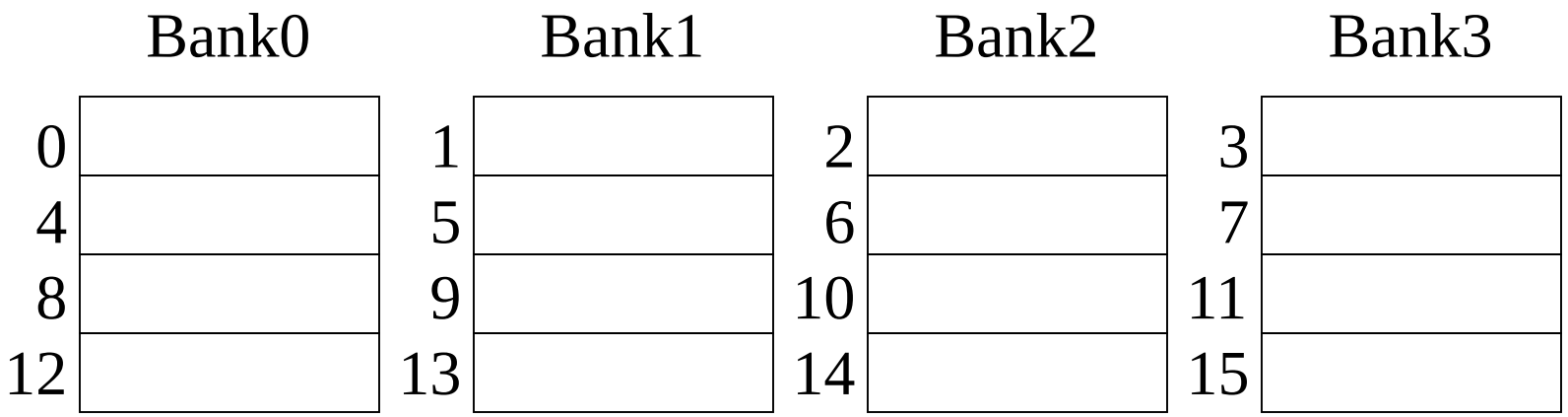
Der Multiplexer zwischen Cache und CPU ist natürlich unerwünscht, da er den kritischen Pfad bezüglich Laufzeit darstellen kann. Ein L-2-Cache kann hier hilfreich sein, wobei der Multiplexer dann zwischen L-2-Cache und L-1-Cache angeordnet wird.
Ein zweiter Nachteil der breiteten Speicherbusorganisation ist folgender: Normalerweise ist Speicher durch den Benutzer erweiterbar. Durch den breiteren Bus wird die kleinste Erweiterungseinheit doppelt bzw. viermal so groß.

2. Technik: Einfaches interleaved Memory

Man kann auch die Bandbreite erhöhen, indem man die potentielle Parallelität in den Speicherbausteinen nutzt. Speicher kann in Banken organisiert werden, die in der Lage sind, mehrere Worte gleichzeitig anstelle nur eines Wortes zur Zeit zu lesen oder schreiben. Die Banken sind in der Regel ein Wort breit, so daß die Breite des Speicherbusses und des Caches nicht verändert werden müssen. Dadurch, daß dieselbe Adresse an alle Bänke angelegt wird, ist die Zeit für das Senden der Adresse und die Zugriffszeit nur einmal erforderlich. Lediglich die Übertragungszeit über den gemeinsamen Speicherbus hängt nun ab von der Anzahl der Speicherworte, die ich aus genausovielen Bänken geholt habe.


CPU
CACHE
Speicher Bank0
Speicher Bank1
Speicher Bank2
Speicher Bank3

In unserem Beispiel von eben bedeutet das, der Zugriff von vier Worten aus vier Bänken kostet jetzt 4+24+4*4Zyklen = 44 Zyklen. Zwar mehr als 32 aber nur mit dem einfach breiten Speicherbus.
Auf dem unteren Bild ist gezeigt, wie man die Adressen jetzt auf die einzelnen Speicherbanken verteilen muß:



Interleaved Speicher bringen dann Vorteile, wenn sequentiell auf mehrere Worte im Speicher zugegriffen wird. Das ist zum Beispiel bei Caches grundsätzlich der Fall, denn wir laden ja immer ganze Blöcke aus dem Speicher. Write-Back Caches können in besonderem Maße von diesem Vorteil Gebrauch machen, da sie ja beim Schreiben wie beim Lesen auf ganze Blöcke sequentiell zugreifen.

Beispiel:
Betrachten wir die folgende Maschine und ihre Cache Performance:
Block-Größe: 1 Wort
Speicherbusbreite: 1 Wort
Miss-Rate: 3%
Speicherzugriffe pro Instruktion 1,2
Cache-Miss-Penalty = 32 Zyklen
CPI ohne Cache-Misses 2.0

Durch Ändern der Blockbreite auf 2 Worte sinkt die Miss-Rate auf 2% und bei einer 4-Wort-breiten Organisation auf 1%. Wir setzen die Zugriffszeiten von oben voraus. Was bringt Verdopplung der Zugriffsbreite und was bringt 2-Weg- bzw. 4-Weg-interleaving?

Antwort:

Die Original CPI mit Cache-Misses ist
2 + (1,2*3%*32) = 3,15

Da IC und Zykluszeit sich nicht verändern, können wir alle hier relevanten Effekte durch die CPI quantifizieren:

Vergrößerung der Blockgröße auf zwei Worte liefert folgende Fälle:

32-Bit Bus, nicht interleaved:	2 + (1,2*2%*2*32) = 3,54
32-Bit Bus, interleaved	2 + (1,2*2%*36) = 2,86
64-Bit Bus, nicht interleaved	2 + (1,2*2%*32) = 2,77

Vergrößerung der Blockgröße auf vier Worte liefert folgende Fälle:

32-Bit Bus, nicht interleaved:	2 + (1,2*1%*4*32) = 3,54
32-Bit Bus, interleaved	2 + (1,2*1%*44) = 2,53
64-Bit Bus, nicht interleaved	2 + (1,2*1%*2*32) = 2,77

Letztlich gewinnt interleaved sogar gegen die doppelte Busbreite, weil mit dem Faktor vier interleaved wird.

Wieviele Speicherbanken sollte ich haben?
Soviele, wie der Zugriff in einer Bank an Zyklen dauert.

Beispiel: Ein Zugriff dauert 6 Zyklen, ich habe aber nur 4 Banken. Angenommen, ich will 24 Worte vom Speicher in den Cache übertragen. Ich beginne bei Bank0 zur Zeit 0. Zur Zeit 6 steht das Datum am Ausgang von Bank 0 zur Verfügung und wird über den Bus zum Cache gebracht. Jetzt kann an Bank 0 der nächste Zugriff beginnen. Das nächste Datum aus Bank 1 läuft zur Zeit 7 über den Bus usw. das vierte Datum zur Zeit 9. Danach passiert für zweit Takte nichts, denn an Bank 0 ist erst zum Zeitpunkt 11 wieder ein Datum zur Verfügung.

Häufig tauchen Bank-Konflikte auf, weil sequentielle Zugriffe alle auf dieselbe Bank zugreifen wollen.

Beispiel:

```
for i:=0 to 7 do
   for j:=0 to 7 do
      a[j,i]:=2*a[j,i]+5;
```

Wenn wir z.B. vier Banken haben, greifen aufeinanderfolgende Zugriffe immer auf dieselbe Bank zu, weil alle a[j,i] für festes i in derselben Bank stehen. Das liegt

daran, daß die Array-Breite und die Anzahl der Banken nicht teilerfremd sind. Häufig löst man dieses Problem durch eine krumme Zahl von Banken, im Idealfall eine Primzahl, denn die ist ja zu allem teilerfremd.

In welcher Bank steht mein Datum?
Bank Nummer = Adresse mod (Anzahl der Banken)
Adresse innerhalb der Bank = Adresse div (Anzahl der Banken)

Um nun aber die Berechnung, in welcher Bank ich ein Datum finde, nicht zu kompliziert zu machen, wird eine Primzahl der Form 2^k-1 gewählt. Dann gilt nämlich:

Adresse innerhalb der Bank = Adresse mod (Anzahl der Worte in der Bank).

Da dies in der Regel eine Zweierpotenz ist, hat man also die Division durch eine Bit-Selection ersetzt.

3. Technik: DRAM-spezifische Optimierungen

DRAM-Bausteine sind quadratisch in Zeilen und Spelaten organisiert. Bei jedem Zugriff wird zunächst die ganze Zeile in einen Puffer gelesen, und aus diesem mit der Spaltenadresse die relevanten Daten ausgesucht. Solche Zeilen sind groß, z.B. bei einem 4 MB DRAM-Baustein 2 Kbits. Wir können hier vom Prinzip der Lokalität profitieren, indem wir aus derselben Zeile weitere Bits benutzen, bevor (am Ende des Lesezyklus) die Zeile wieder zurückgeschrieben wird. Diese Technik gibt es in drei Standardvarianten, die heute von allen DRAM-Bausteinen unterstützt werden:

Nibble-mode: Mit der RAS-Flanke werden drei weitere Bits (sequentiell folgend) zur Verfügung gestellt.

Page-mode: Der Puffer wirkt wie SRAM. Durch Wechsel der Spaltenadresse kann auf beliebig viele weitere Bits wahlfrei zugegriffen werden. Erst beim nächsten Refresh oder bei der nächsten Zeilenadresse geht das nicht mehr.

Static column: Wie Page mode, wobei kein CAS erforderlich ist bei Wechsel der Spaltenadresse.