

Rechnerarchitektur (AIN 2)

SoSe 2021

Kapitel 5

Die Speicherhierarchie

Prof. Dr.-Ing. Michael Blaich
mblaich@htwg-konstanz.de



Kapitel 5: Die Speicherhierarchie

5.1 Einführung

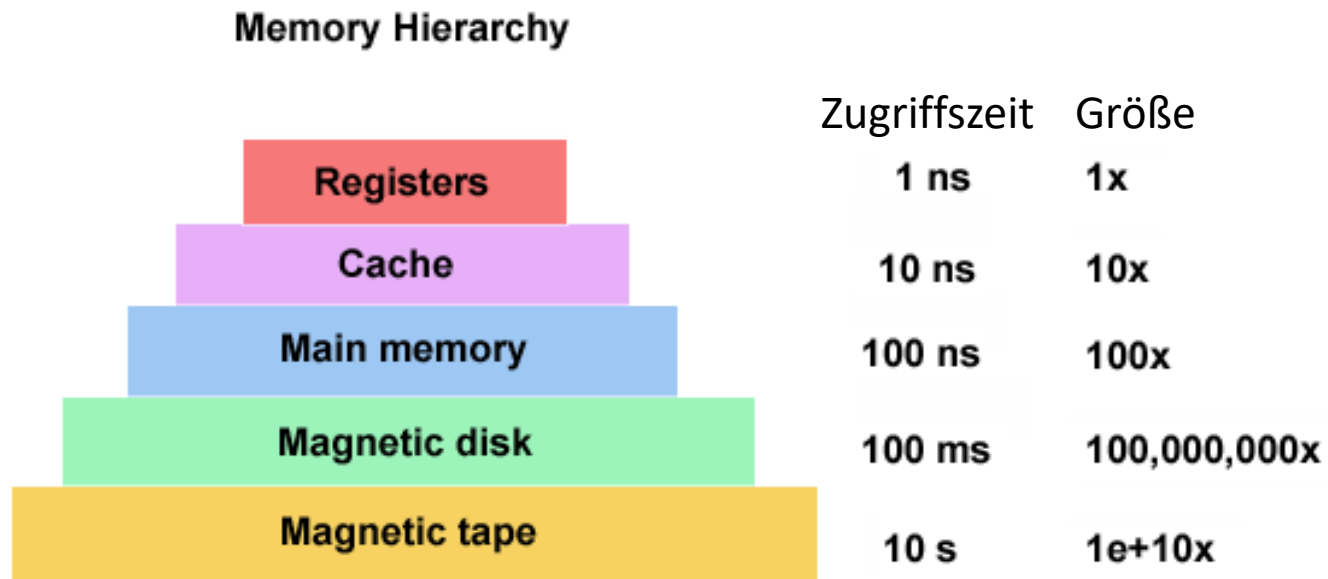
5.2 Caching

5.3 Virtueller Speicher

5.4 DRAM – Dynamic Random Access Memory

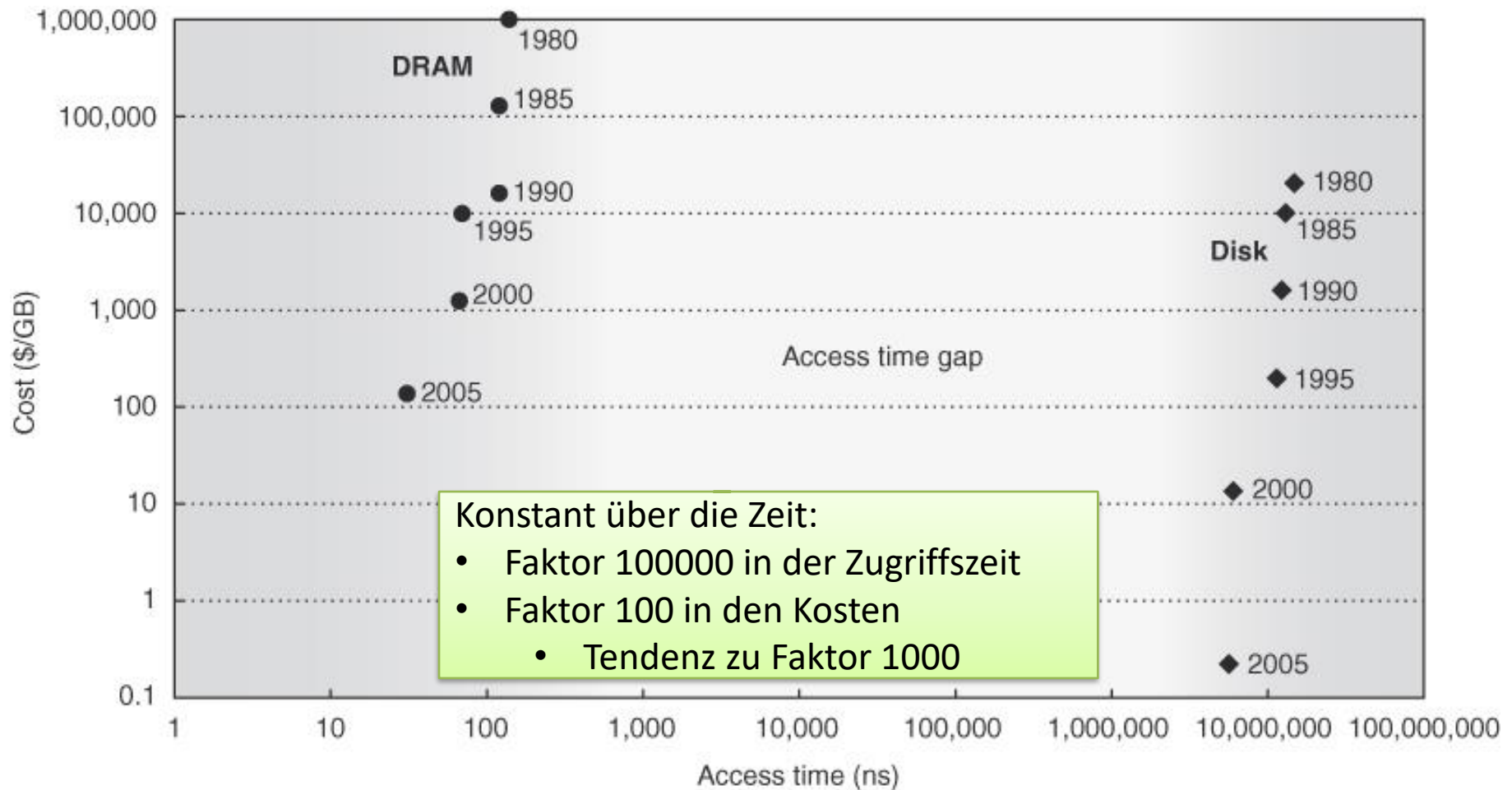
5.5 Zusammenfassung

- Möglichst großer Speicher
- Möglichst große Geschwindigkeit
 - geringe Zeit bis zur Verfügbarkeit eines Speicherinhalts (kleine latency)
 - hoher Durchsatz (großer throughput)
- Möglichst geringe Kosten
- Möglichst geringer Platzbedarf
- Programmierer wollen unendlichen Speicher mit minimaler Zugriffszeit (ein Taktzyklus)
 - schneller Speicher ist teuer
 - viel Speicher braucht Platz
 - nicht mit einem Typ von Speicher realisierbar, daher verschiedene Arten von Speicher organisiert in einer **Speicherhierarchie**

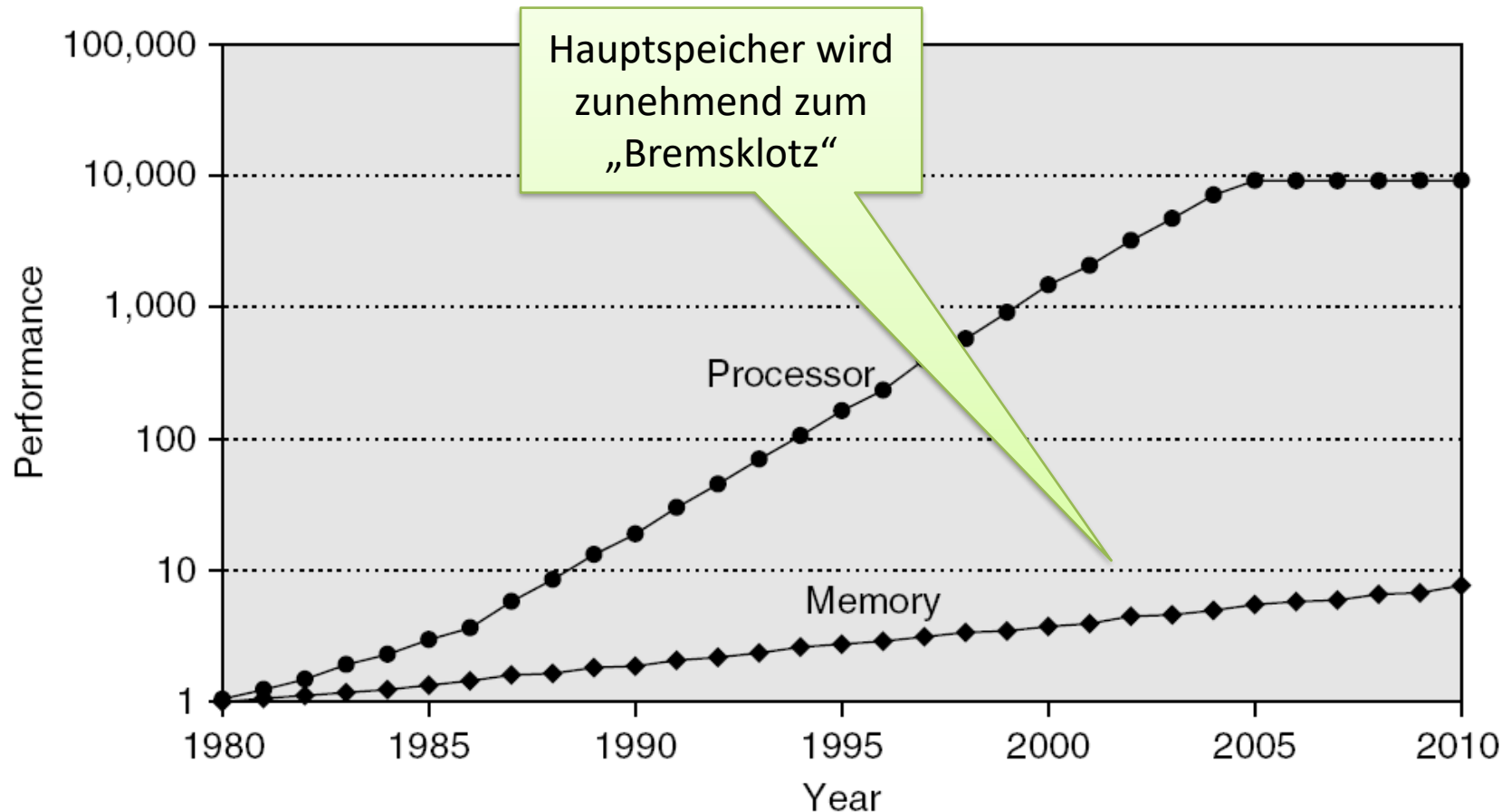


- Trade-Off zwischen Größe und Zugriffszeit
 - technisches Design erlaubt viele flexible Zwischenstufen

Entwicklung von Kosten und Zugriffszeiten

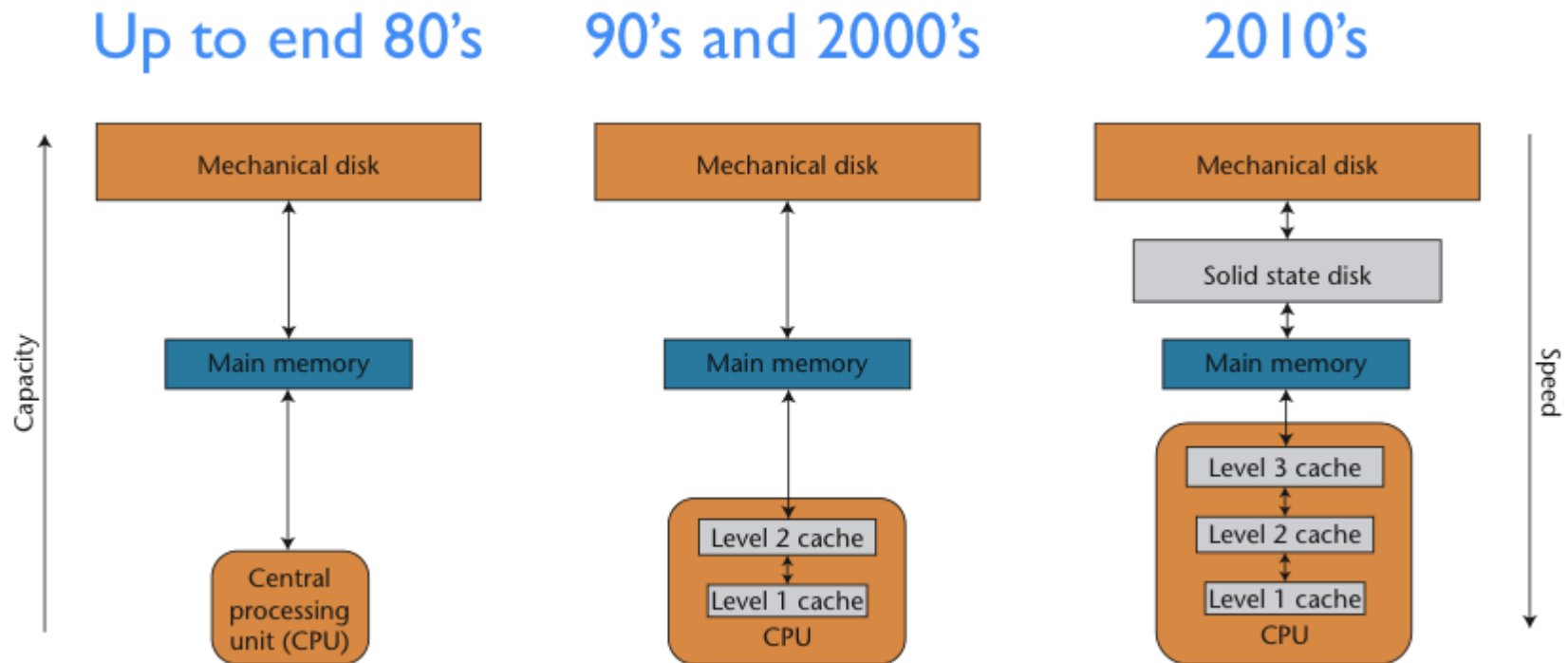


Lücke zwischen Speicher- und CPU-Geschwindigkeit



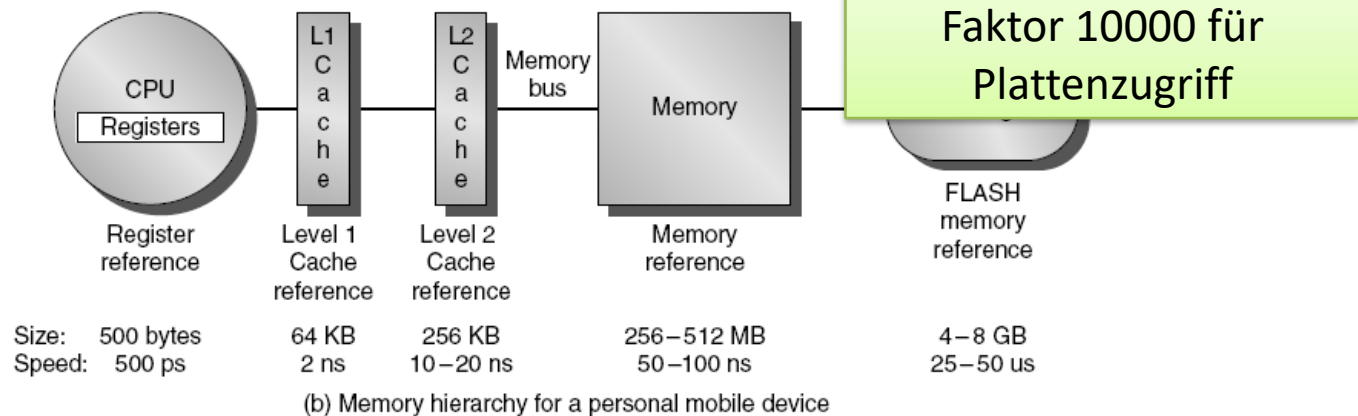
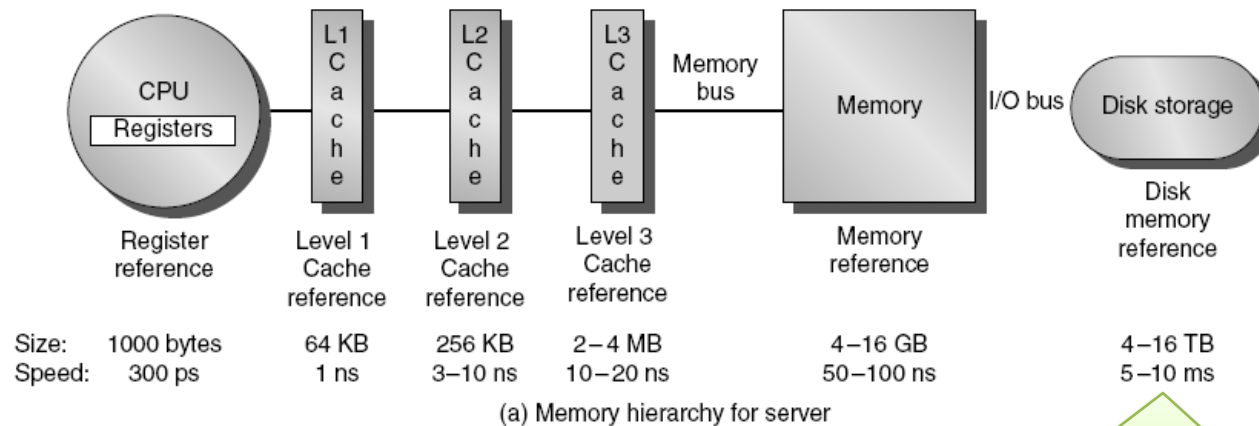
Entwicklung der Speicherhierarchie

- Steigende Anzahl von Cache-Hierarchien in PCs
 - weniger bis gar nicht in eingebetteten Systems
- SSD (Solid State Drive, Flash-Speicher) als weiterer Zwischenschritt zwischen Hauptspeicher und Festplatte



Speicherhierarchie in Zahlen

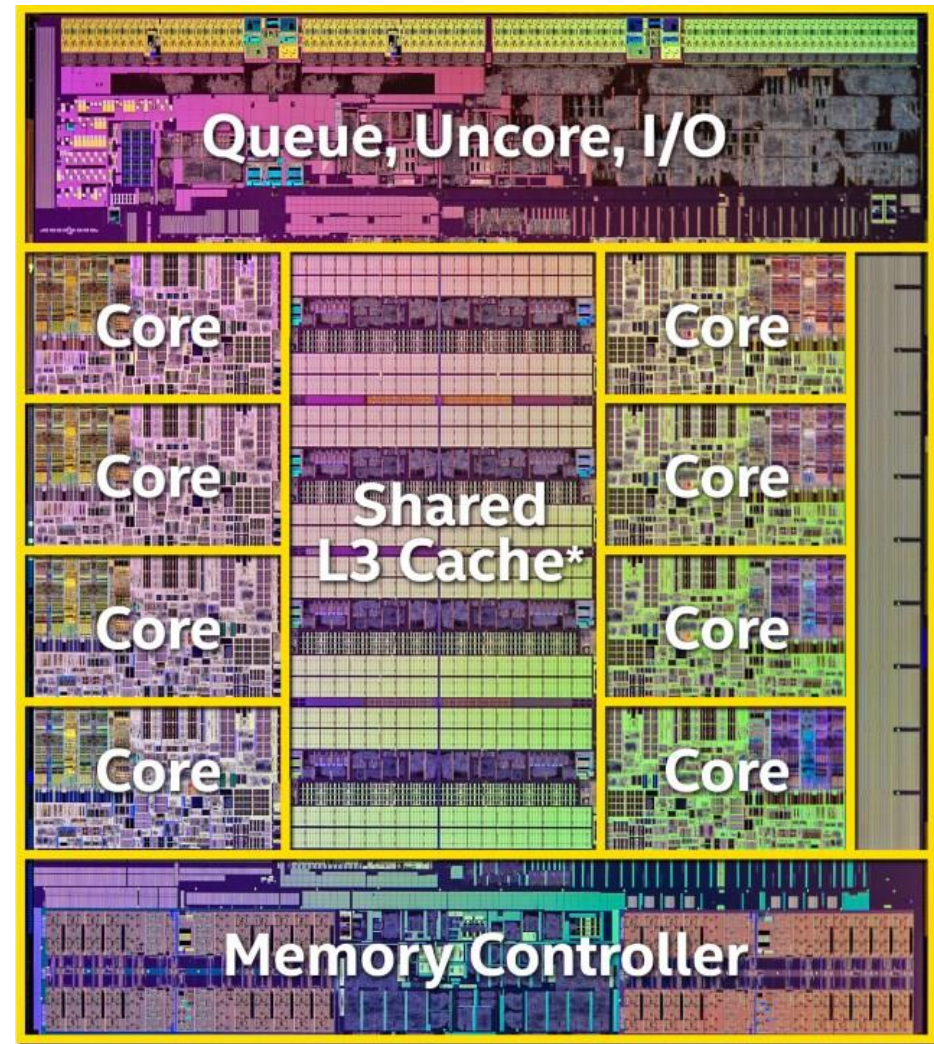
	Register	L1 Cache	L2 Cache	L3 Cache	Memory	Disk
Größe	1	x64 (64)	x4 (256)	x8 (2e3)	x2e3 (4e6)	x1e3 (4e9)
Taktzyklen	1	x3 (3)	x3 (10)	x3 (30)	x5 (150)	x1e4 (15e6)



Speicherhierarchie Beispiel

Core i7-5960X (2014)

Socket	LGA2011-3
Cores	8
Frequency	3GHz
Process	22nm
Level 1 cache	8 x 32Kb (512Kb)
Level 2 cache	8 x 256Kb (2MB)
Level 3 cache	20MB
Supported memory type	DDR4 (max. 64GB)
Power rating (TDP)	140W



Quelle: www.intel.com, product specification Core i7-5960X

Speicherhierarchie Beispiel

Core i7-5960X (2014)

Socket	LGA2011-3
Cores	8
Frequency	3GHz
Process	22nm
Level 1 cache	8 x 32Kb (512Kb)
Level 2 cache	8 x 256Kb (2MB)
Level 3 cache	20MB
Supported memory type	DDR4 (max. 64GB)
Power rating (TDP)	140W

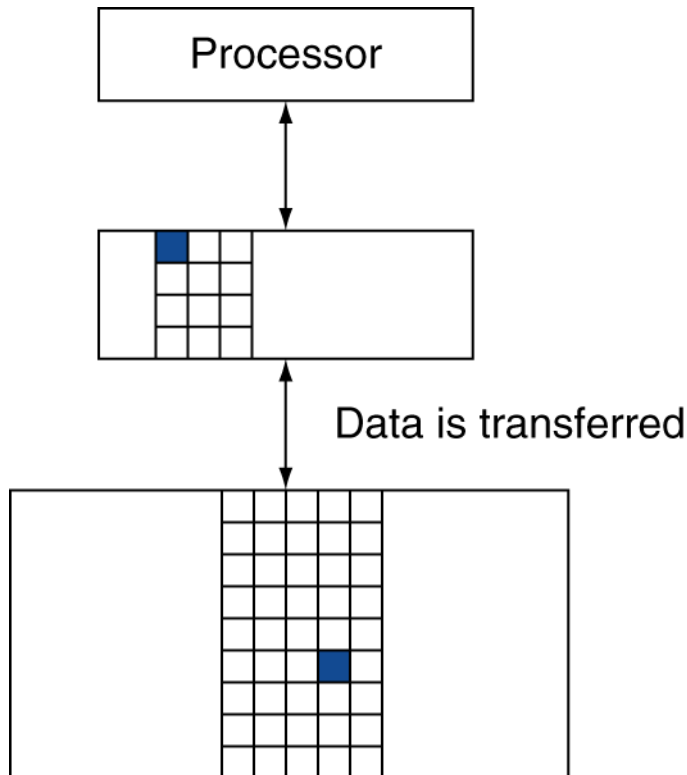
Core i9-10980XE (Q4 2019)

Socket	LGA2066
Cores	18
Frequency	3GHz
Process	14nm
Level 1 cache	18 x 32 kB Instruction (576kB) 18 x 32 kB Data (576kB)
Level 2 cache	18 x 1 MB (18MB)
Level 3 cache	24,75 MB
Supported memory type	DDR4 (max. 256GB)
Power rating (TDP)	165W

Grundlagen der Speicherhierarchie

- Anwendungen verhalten sich üblicherweise lokal
 - Räumliche Lokalität:** aufeinanderfolgender Speicherzugriffe auf **nahe-gelegene Speicherstellen**
 - aufeinanderfolgende Instruktionen, kurze Sprünge, Arrays
 - Zeitliche Lokalität:** wiederholter Zugriff auf **gleiche Speicherstelle** in kurzer Zeitspanne
 - Schleifen: gleiche Variablen und Instruktionen, mehrfacher Zugriff auf eine Variable
- Prinzip: Ablegen von häufig benötigte Speicherinhalte in kleinen Speichern, seltener benötigte Inhalte (nur) in großen Speichern abzulegen
 - **alle Daten** werden auf Festplatte gespeichert
 - Daten, auf die **in letzter Zeit zugegriffen** wurde, und deren Umgebung werden in den Hauptspeicher (DRAM) kopiert
 - Daten, auf die **vor noch kürzerer Zeit** zugegriffen wurde, und deren sehr nahe Umgebung werden in den Cache (SRAM) kopiert
 - Direkte Zugriff auf Cache-Speicher von der CPU
- Illusion eines **großen Speichers** mit (im Mittel) **kleinen Zugriffszeiten**

Terminologie



Block (auch Line): kleinste Speichereinheit, die zwischen Hierarchie-Ebenen kopiert wird

Hit: von oberer Ebene angefragter Speicherinhalt ist vorhanden

- **Hit Ratio:** Anteil erfolgreicher Speicherzugriffe

$$\text{Hit Ratio} = \frac{\#Hits}{\#Zugriffe}$$

Miss: Block ist nicht vorhanden und muss von nächster Ebene geladen werden

- **Miss Penalty:** Zeit, um Block "nachzuladen"
- **Miss Ratio:** Anteil nachzuladender Blöcke

$$\text{Miss Ratio} = \frac{\#Misses}{\#Zugriffe} = 1 - \text{Hit Ratio}$$

Kapitel 5: Die Speicherhierarchie

5.1 Einführung

5.2 Caching

5.2.1 Direct Mapped Cache

5.2.2 Cache Performance

5.2.3 Assoziative Caches

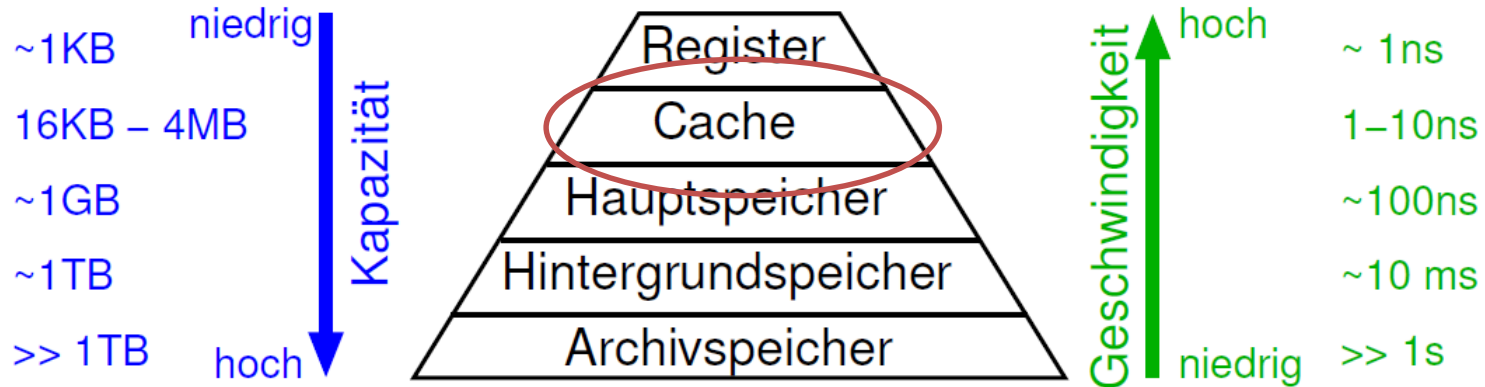
5.3 Virtueller Speicher

5.4 DRAM – Dynamic Random Access Memory

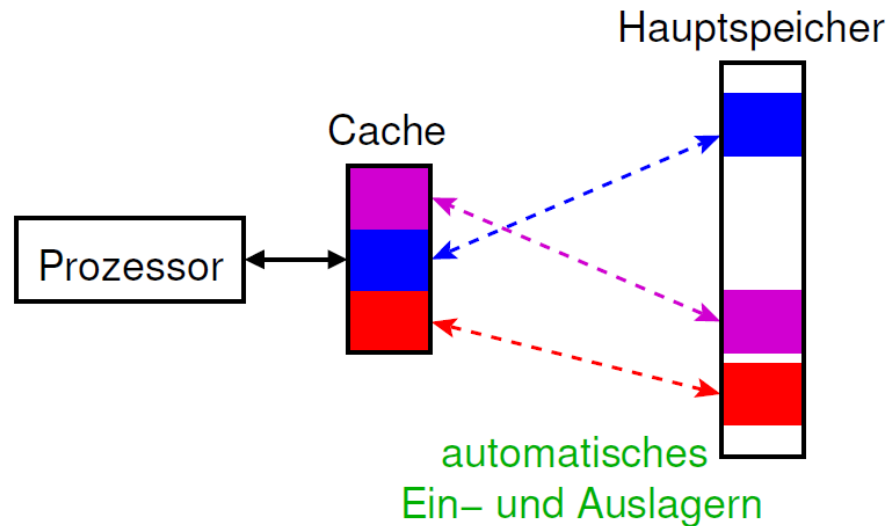
5.5 Zusammenfassung

Speicherhierarchie

- Speicherhierarchie



- Grundidee Caching



Grundlagen Cache-Speicher

- Cache-Speicher
 - Ebene der Speicherhierarchie, die der CPU am nächsten ist
- Herausforderung:
 - Zugriffe X_1, \dots, X_{n-1}, X_n
 - Welche Daten stehen im Cache?
 - Wo schauen wir nach?
 - Wie wird die Konsistenz der Daten gewährleistet?
 - Wie erzielen Caches eine hohe Effizienz bei kleinen Zugriffszeiten?

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

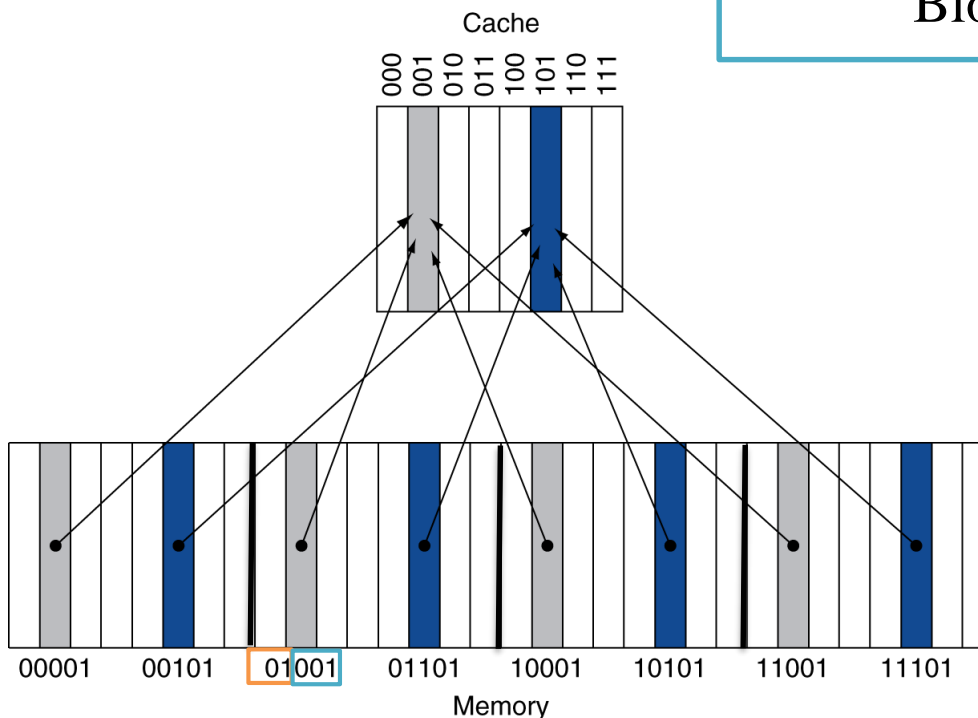
X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

b. After the reference to X_n

Einfachste Variante: Direct Mapped Cache

- Prinzip:
 - Aufteilung des Speichers in 2^k Blöcke
 - Transfer zwischen Speicher und Cache erfolgt in diesen Daten-Blöcken
 - Direkte Berechnung der Blocknummer im Cache (Index) aus der Blocknummer im Speicher
 - niedrigwertige Bits

Blocknummer im Cache (Index) =
Blocknummer modulo Cache - Größe



- Cache passt 4-mal in den Speicher
- jeder Cache-Block kann 4 möglichen Speicher-Blöcken entsprechen
- **Tag** höherwertige Bits der Blocknummer

Einfaches Beispiel für Datenhaltung in einem Cache

- Cache-Größe: 8 Blöcke mit einem Wort pro Block
- Datenhaltung:
 - V: Flag für Gültigkeit des Eintrags
 - Tag: Nummer des in Data gespeicherten Blocks
 - höherwertige Bits

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Änderung des Cache-Inhalts bei Speicherzugriffen

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

$$22 \% 8 = 6 \rightarrow 110$$

Blocknummer im Cache (Index) =
Blocknummer modulo Cache-Größe

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	MEM[22]
111	N		

Änderung des Cache-Inhalts bei Speicherzugriffen

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	MEM[26]
011	N		
100	N		
101	N		
110	Y	10	MEM[22]
111	N		


Änderung des Cache-Inhalts bei Speicherzugriffen

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	MEM[26]
011	N		
100	N		
101	N		
110	Y	10	MEM[22]
111	N		

Änderung des Cache-Inhalts bei Speicherzugriffen

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000



Index	V	Tag	Data
000	Y	10	MEM[16]
001	N		
010	Y	11	MEM[26]
011	Y	00	MEM[3]
100	N		
101	N		
110	Y	10	MEM[22]
111	N		

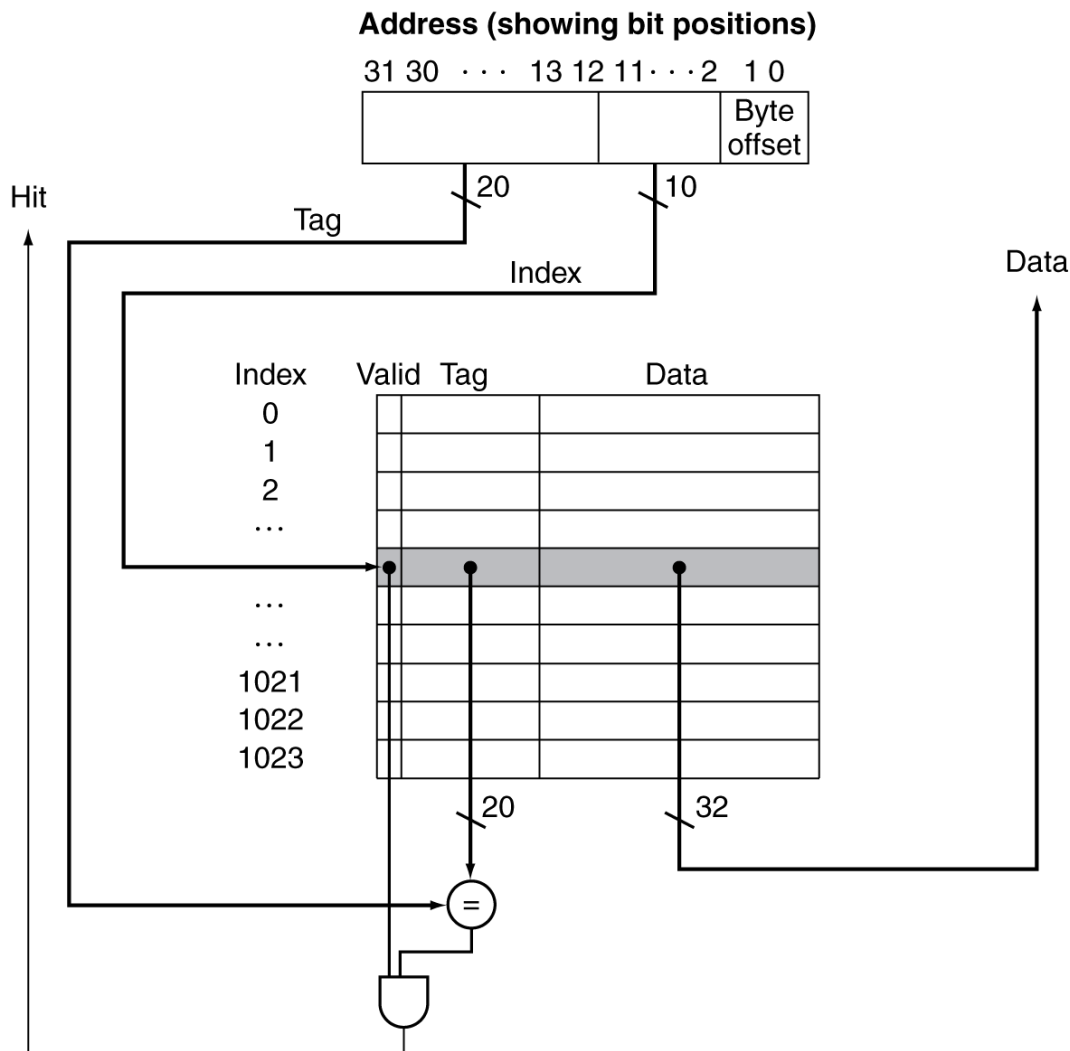
Änderung des Cache-Inhalts bei Speicherzugriffen

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Block muss
nachgeladen
werden

Index	V	Tag	Data
000	Y	10	MEM[16]
001	N		
010	Y	10	MEM[18]
011	Y	00	MEM[3]
100	N		
101	N		
110	Y	10	MEM[22]
111	N		

Zugriff auf Cache in Hardware mit Speicheradresse



Eingang: 32-Bit Speicheradresse mit

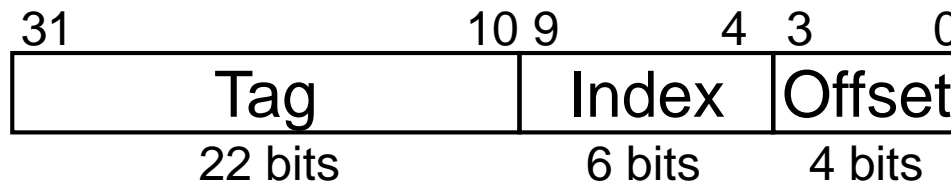
- Byte Offset (2 Bits)
 - Cache enthält Words
- Index (10 Bits)
 - Cache enthält 1024 Blöcke oder 4096 Bytes
- Tag (20 Bits)
 - Speicherbereich ist 2^{20} -mal größer als der Cache

Ausgang:

- Hit, wenn
 - $V(\text{Index})$ gültig ist und
 - $\text{Tag}(\text{Index})$ gleich Tag ist
- $\text{Data}(\text{Index})$

Blockgröße

- Cache mit 64 Blöcken à 16 Bytes
- Welchen Index hat Speicheradresse 1200?
 - $\text{Blocknummer} = \text{floor}(\text{Adresse} / \text{Blockgröße}) = 1200 / 16 = 75$
 - $\text{Blockindex} = \text{Blocknummer} \bmod \text{Blockanzahl} = 75 \bmod 64 = 11$
- Welche Bits der Adresse entsprechen Byte-Offset, Index und Tag?
 - $\text{Tag} = \text{floor}(\text{Blocknummer} / \text{Blockanzahl}) = 1$
 - $\text{Byte-Offset} = \text{Adresse} \bmod \text{Blockgröße} = 0$



1200: 0000000...000001 001011 0000

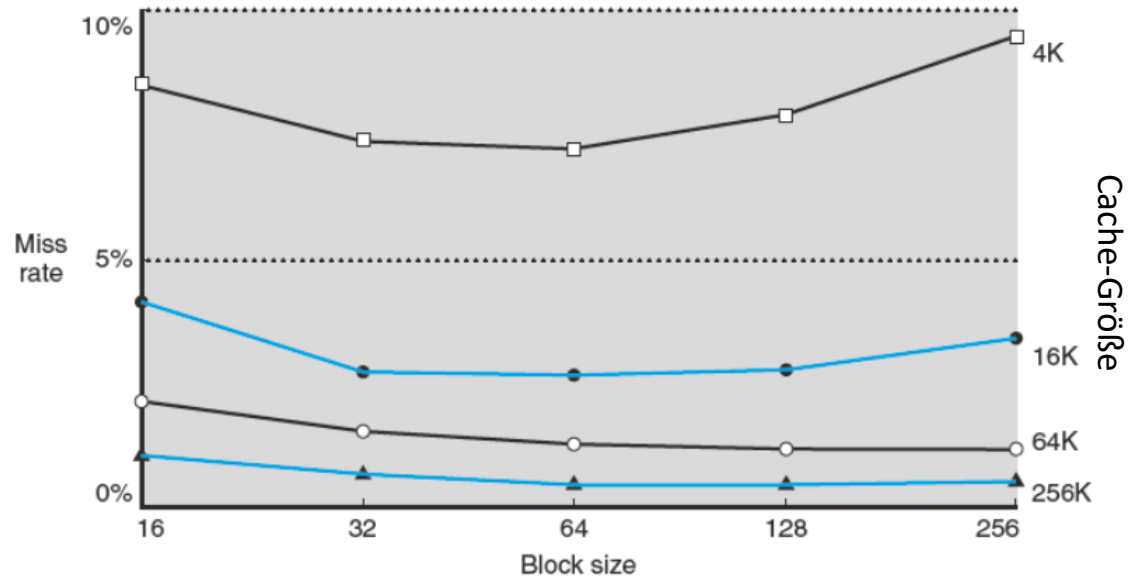
Noch ein Beispiel?

- Cache mit 8192 Bytes Speicher und 64 Byte-Blöcken
- Speicheradresse 100000?
- Anzahl Blöcke: 128 ($8192/64 = 2^{13}/2^6 = 2^{(13-6)} = 2^7$)
- Blocknummer: $\text{floor}(100000 / 64) = 1562$
- Blockindex: $1562 \bmod 128 = 26 = \text{floor}(100000 \bmod 8192 / 64)$
- Tag: $\text{floor}(1562 / 128) = 12 = \text{floor}(100000 / 8192)$
- Offset: $100000 \bmod 64 = 32$

Erwägungen zur Blockgröße

- Miss Rate in Abhängigkeit der Blockgröße
 - große Blöcke verringern Miss-Rate aufgrund von Lokalität
 - je größer die Blöcke, desto weniger Blöcke gibt es
 - mehr Konkurrenz um die Blöcke führt zu erhöhter Miss Rate
 - Trade-Off zwischen räumlicher und zeitlicher Lokalität

Größere Speicherblöcke führen nicht immer zu einer Verbesserung der Miss-Rate: es gibt ein Optimum in Abhängigkeit der Speichergröße



- Kosten für einen Miss (Miss Penalty) steigen bei größeren Blöcken
 - Zeit für Datentransfer steigt mit Blockgröße
 - Abhilfe Early Restart: Prozessor arbeitet direkt mit benötigtem Wort und wartet nicht auf vollständigen Transfer des Datenblocks
 - vor allem bei Instruktionen

Kapitel 5: Die Speicherhierarchie

5.1 Einführung

5.2 Caching

5.2.1 Direct Mapped Cache

5.2.2 Cache Performance

5.2.3 Assoziative Caches

5.3 Virtueller Speicher

5.4 DRAM – Dynamic Random Access Memory

5.5 Zusammenfassung

Was passiert bei einem Miss?

- Cache Hit:
 - CPU macht normal weiter
 - nächste Instruktion kann ausgeführt werden
 - Load/Store-Instruktion benötigt eingeplante Takte
- Cache Miss:
 - CPU Pipeline wird gestoppt (stall)
 - Block wird von der nächsten Ebene der Speicherhierarchie geladen
 - im Falle einer geladenen Instruktion:
 - Wiederholung der Fetch-Stage
 - im Fehler eines Datenzugriffs
 - Datenzugriff fertig ausführen



Write-Through und Write-Back

- Was passiert bei einem Data-Write Hit, d.h. wenn ein Wort in einen Speicherblock geschrieben wird, der im Cache vorhanden ist?
 - Möglichkeit 1: **Write-Through**
 - Wort im Cache und direkt auch im Hauptspeicher ändern
 - Nachteil: Schreiben dauert länger
 - Beispiel: CPI = 1 (ohne Write Through), 10% Store-Instruktionen, Schreiben eines Worts in den Speicher dauert 100 Takte
 - Was ist der effektive CPI?
 - » $\text{Effective CPI} = 1 + 0.1 \times 100 = 11$
 - Möglichkeit 2: **Write-Back**
 - Wort zunächst nur im Cache ändern und Block als geändert (dirty) markieren
 - beim Ersetzen des Blocks Daten zurückschreiben
 - Nachteil: inkonsistente Daten in Cache und Hauptspeicher, Ersetzen dauert länger
 - Vorteil: weniger Schreib-Operationen

Write-Through und Write-Back

- Schreib-Puffer (Write Buffer)
 - Speichert Daten zwischen, die darauf warten, in den Hauptspeicher geschrieben zu werden
 - CPU kann direkt weiterarbeiten
 - muss aber stoppen, falls der Schreib-Puffer voll ist
 - Einsatz bei Write-Back und Write-Through
 - Write-Back: CPU muss bei einem Miss (Ersetzen eines Blocks) nicht zusätzlich auf das Rückschreiben in den Hauptspeicher warten
 - Write-Through: CPU muss nicht auf Schreiben in Hauptspeicher warten
- Was passiert bei einem Data-Write-Miss?
 - Alternativen bei Verwendung von Write-Through:
 - Allocate on miss: in Hauptspeicher schreiben und Block in Cache laden
 - Write around: nur in Hauptspeicher schreiben und den Block nicht in den Cache laden
 - oftmals schreiben Programme einen ganzen Block ohne zu lesen
 - Write-back:
 - Daten in Hauptspeicher schreiben und Block in Cache laden

Bewertung der Cache Performance

- CPU Zeit wird verbraucht durch
 - Taktzyklen zur Ausführung des Programms
 - inklusive Cache-Hits zum Laden/Speichern von Daten und Laden von Instruktionen
 - Taktzyklen durch Speicher-Stalls (Memory Stall Cycles), d.h. wenn die Pipeline aufgrund von Speicherzugriffen warten muss
 - verursacht durch Cache-Misses
 - vereinfachte Berechnung:

Memory stall cycles

$$= \frac{\# \text{ Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\# \text{ Instructions}}{\text{Program}} \times \frac{\# \text{ Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

- Gegeben:
 - System mit getrenntem Instruktions- und Daten-Cache
 - Instruktions-Cache: 2% Miss-Rate
 - Daten-Cache: 4% Miss Rate
 - Miss Penalty: 100 Takte
 - Basis-CPI (bei 100% Hit-Rate): 2
 - Anteil Lade- und Speicherbefehle: 36%
- Mittlere Anzahl Memory-Miss-Cycles pro Instruktion?
 - Instruktions-Cache: $0.02 \times 100 = 2$
 - Daten-Cache: $0.36 \times 0.04 \times 100 = 1.44$
- Was ist der effektive CPI?
 - Effektiver CPI: $2 + 2 + 1.44 = 5.44$
 - Ideale CPU bei 100% Hit Rate ist über zweieinhalb-mal schneller
 - niedrige Hit-Rate vor allem im Instruktions-Cache ist entscheidend für die Performance des Programms

- Hit-Time:
 - Speicherzugriffszeit im Falle eines Hits
 - Hit-Time ist eine wichtige Kenngröße für die Gesamt-Performance
- Mittlere Speicherzugriffszeit
 - Average Memory Access Time (AMAT)
 - $AMAT = \text{Hit time} + \text{Miss-Rate} \times \text{Miss-Penalty}$
- Berechnungsbeispiel:
 - CPU mit 1ns Taktung, Hit-Time = 1 Takt, Miss-Penalty = 20 Takte, Cache-Miss-Rate = 5%
 - $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$
 - im Mittel 2 Zyklen pro Load-/Store-Instruktion

- DRAM als Hauptspeicher
 - feste Breite (z.B. 1 Word)
 - angebunden über getakteten Bus mit fester Breite
 - Bus normalerweise langsamer getaktet als CPU
- Beispiel: Lesen eines Cache Blocks von 4 Words
 - 1 Bus-Takt zur Übertragung der Adresse
 - 15 Bus-Takte für den DRAM Speicherzugriff
 - 1 Bus-Takt für die Datenübertragung
 - DRAM-Breite: 1 Word
 - Miss-Penalty: $1 + 4 \times 15 + 4 \times 1 = 65$ Bus-Takte
 - Bandbreite = $16 \text{ Bytes} / 65 \text{ Takte} = 0.25 \text{ B/Takt}$

- Bei steigender CPU Performance wird Einfluss von Miss-Penalty signifikant
 - je niedriger der Basis-CPI desto größer wird der Anteil der Zeit, die in Memory-Stalls verbraucht wird
 - höhere CPU-Taktfrequenz führt zu mehr Taktzyklen in Memory Stalls
- Bemerkung:
 - Spekulation und Multi-Threading (siehe Kapitel 6) mindern die Auswirkungen von Misses, da während der Miss-Penalty-Dauer andere Instruktionen ausgeführt werden können
- Cache-Effizienz hat einen entscheidenden Einfluss auf die Gesamtperformanz des Systems

Kapitel 5: Die Speicherhierarchie

5.1 Einführung

5.2 Caching

5.3.1 Direct Mapped Cache

5.3.2 Cache Performance

5.3.3 Assoziative Caches

5.3 Virtueller Speicher

5.4 DRAM – Dynamic Random Access Memory

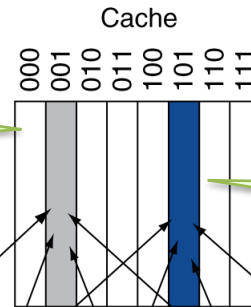
5.5 Zusammenfassung

Direct Mapped Caches

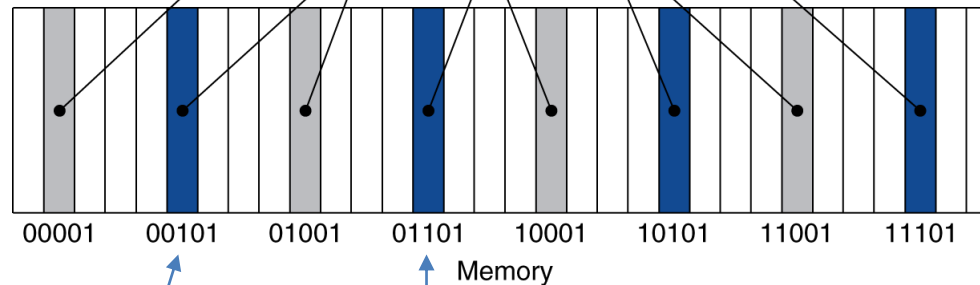


Ist das wirklich effizient?

... während andere Cache-Blöcke evtl. ungenutzt sind



Cache Block wird andauernd ersetzt ...



Array A

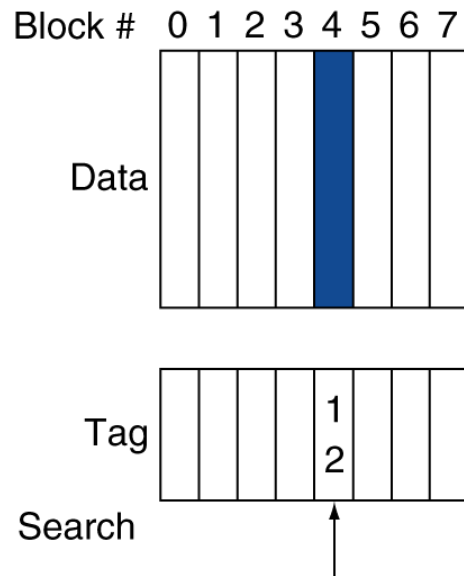
Array B

Prozedur arbeitet mit A und B. Was passiert?

- Assoziatives Feld:
 - Zugriff auf Feld über Key und nicht über Index
 - Abbildung von Key auf Index wird separat gespeichert
- Assoziativer Cache:
 - Cache-Block wird nicht direkt über Speicheradresse bestimmt
 - **Fully-associative** (voll-assoziativer) Cache:
 - beliebige Zuordnung von Speicherblock auf Cache-Block
 - Vorteil: die vielversprechendsten Blöcke können im Cache gehalten werden
 - Nachteil: aufwändiges Überprüfen von allen Cache-Blöcken auf Treffer
 - **N-way-set-associative** (N-fach mengen-assoziativer) Cache:
 - **Kompromiss** zwischen Direct-Mapped und Fully-Associative Cache
 - Einteilung des Caches in Speicherbereiche (Sets) mit jeweils N Blöcken
 - Direct-Mapped-Zugriff auf Sets
 - » $\text{Set-Nummer} = (\text{Block-Nummer}) \bmod (\text{\#Sets})$
 - Associative-Zugriff innerhalb der Sets
 - » n Vergleiche für Treffer

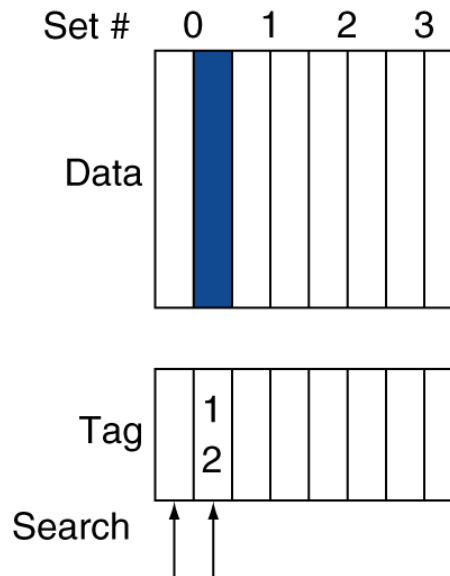
Übersicht der Cache-Organisation

Direct mapped



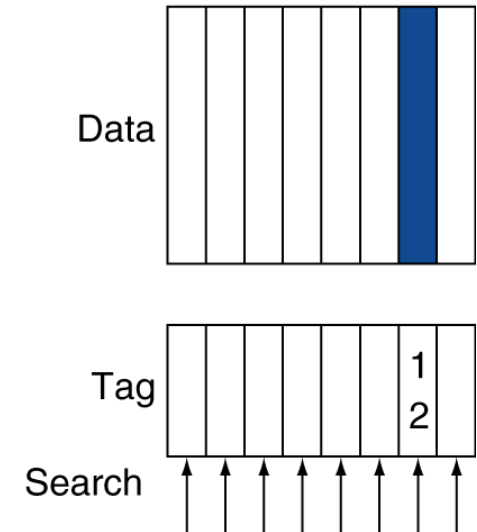
Vorteil: Block kann berechnet werden
Nachteil: keine Auswahlmöglichkeit der Blöcke im Cache

Set associative



Kompromiss: Grad der Assoziativität steigt mit der Größe der Sets

Fully associative



Vorteil: volle Auswahlmöglichkeit der Blöcke im Cache
Nachteil: aufwändige Suche nach Treffern

Grad der Assoziativität

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

- Hauptspeicher mit N Blöcken
- Cache mit M Blöcken
- Abbildung/Speichern von n Hauptspeicherblöcken auf m Cacheblöcke

	n	m
Direct Mapped	N/M	1
Two-Way Set Assoc.	$2 \cdot N/M$	2
k-Way Set Assoc.	$k \cdot N/M$	k
Fully Associative	N	M

- $\text{Cache-Größe} = (\text{Anzahl Sets}) \times (\text{Blöcke pro Set}) \times (\text{Bytes pro Block})$

Beispiel: 4-Way Set-Associative Cache

- 4-Way Set-Associative Cache mit 8192 Bytes und 16-Byte Blöcken
- Byte-Adresse 4000?

Beispiel: 4-Way Set-Associative Cache

- 4-Way Set-Associative Cache mit 8192 Bytes und 16-Byte Blöcken

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

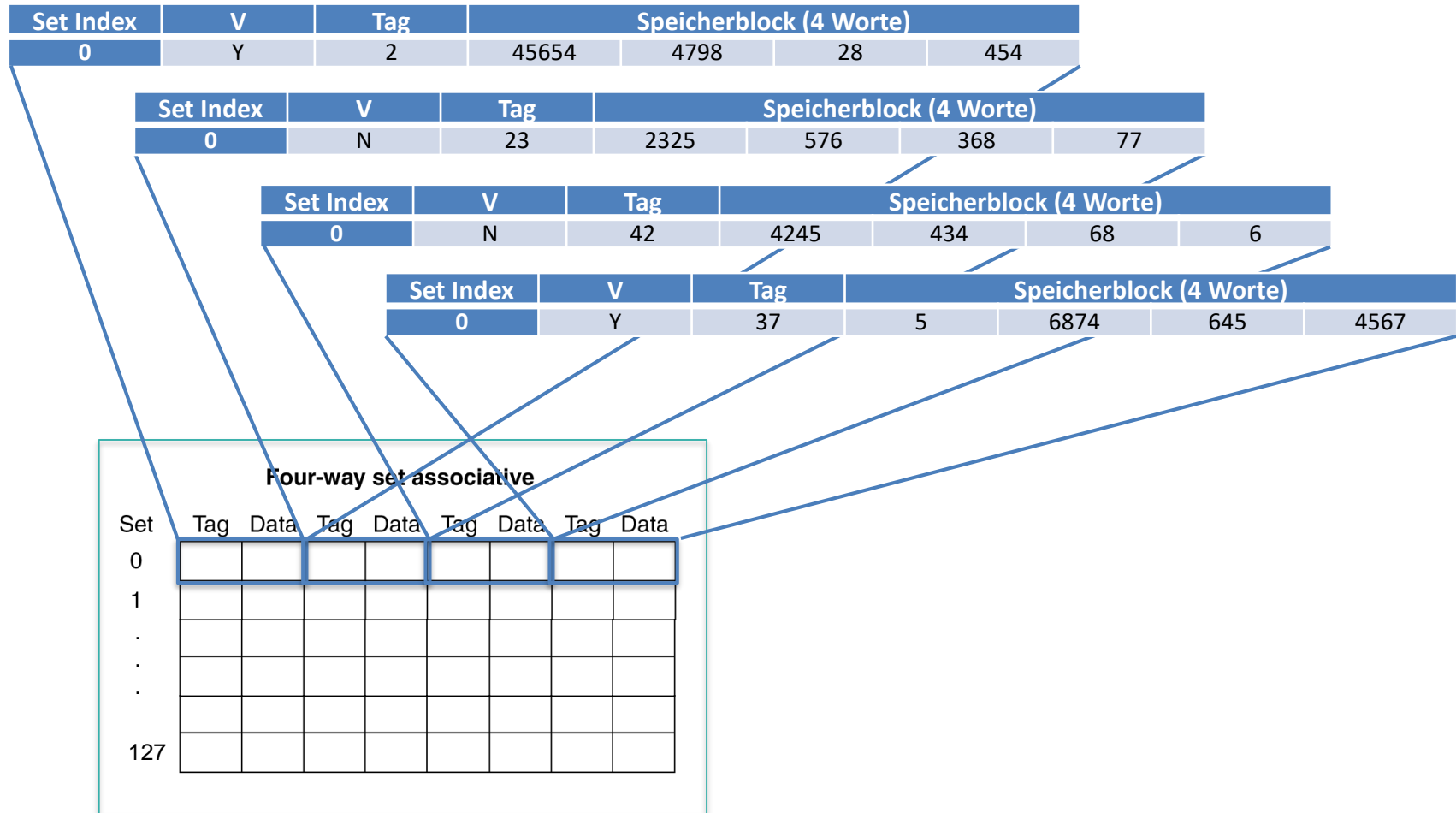
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Beispiel: 4-Way Set-Associative Cache

- 4-Way Set-Associative Cache mit 8192 Bytes und 16-Byte Blöcken
- Byte-Adresse 4000?
- Anzahl Blöcke: $8192/16 = 512$
- Anzahl Sets: $\text{Anzahl Blöcke} / 4 = 128$

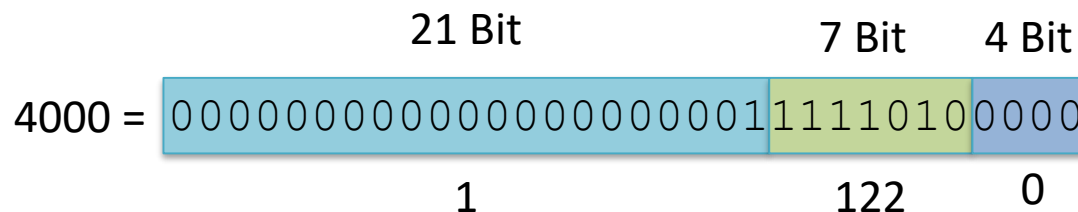
Four-way set associative								
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Beispiel: 4-Way Set-Associative Cache



Beispiel: 4-Way Set-Associative Cache

- 4-Way Set-Associative Cache mit 8192 Bytes und 16-Byte Blöcken
- Byte-Adresse 4000?
- Anzahl Blöcke: $8192/16 = 512$
- Anzahl Sets: Anzahl Blöcke / 4 = 128
- Blocknummer: $\text{floor}(4000 / 16) = 250$
- Set Index: $250 \bmod 128 = 122$
- Tag: $\text{floor}(250 / 128) = 1$
- Byte-Offset: $4000 \bmod 16 = 0$



Beispiel: 4-Way Set-Associative Cache

- Für das Set mit Index 122 sind vier Blöcke mit den Tags 0, 17, 1 und 2345 gespeichert.
- Um festzustellen, ob Block (250) im Cache vorhanden ist, muss der Tag (1) des Blocks mit den im Cache gespeicherten Tags verglichen werden.

Set Index	V	Tag	Speicherblock (4 Worte)			
120	Y	2	45646	5413	7198	1476
121	Y	6	198	213	617	948
122	Y	0	720	260	350	2240
123	N	2	898	9456	4560	8181
124	N	14	0	0	0	0

1. Block pro Set

Set Index	V	Tag	Speicherblock (4 Worte)			
120	Y	80	1232	1432	198	1776
121	Y	7	982	253	67	4498
122	Y	17	702	670	50	440
123	N	20	82	779	10	114
124	N	12	2	7	4560	9870

2. Block pro Set

Set Index	V	Tag	Speicherblock (4 Worte)			
120	Y	23	3123	13222	4198	1376
121	Y	89	398	232	6734	938
122	Y	1	730	640	504	430
123	N	2	8877	956	1043	131
124	N	5	54	890	4320	303

3. Block pro Set

Set Index	V	Tag	Speicherblock (4 Worte)			
120	Y	234	123	132	198	176
121	Y	5674	98	23	67	98
122	Y	2345	70	60	50	40
123	N	7346	8	9	10	11
124	N	1448	0	0	0	0

4. Block pro Set

Beispiel: Zugriffssequenz

Cache mit 4 Blöcken und Zugriffssequenz: 0, 8, 0, 6, 8

	Block address	Cache index	Hit/miss	Cache content after access			
				0	1	2	3
Direct-Mapped	0						
	8						
	0						
	6						
	8						
2-Way-Set Associative				Set 0		Set 1	
	0						
	8						
	0						
	6						
	8						
Fully Associative	0						
	8						
	0						
	6						
	8						

Beispiel: Zugriffssequenz

Cache mit 4 Blöcken und Zugriffssequenz: 0, 8, 0, 6, 8

	Block address	Cache index	Hit/miss	Cache content after access			
				0	1	2	3
Direct-Mapped	0	0	miss	Mem[0]			
	8	0	miss	Mem[8]			
	0	0	miss	Mem[0]			
	6	2	miss	Mem[0]		Mem[6]	
	8	0	miss	Mem[8]		Mem[6]	
				Set 0		Set 1	
2-Way-Set Associative							
Fully Associative							

Mem[0]: erster Zugriff, unvermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermeidbarer Miss

Beispiel: Zugriffssequenz

Cache mit 4 Blöcken und Zugriffssequenz: 0, 8, 0, 6, 8

	Block address	Cache index	Hit/miss	Cache content after access			
				0	1	2	3
Direct-Mapped							
				Set 0		Set 1	
2-Way-Set Associative	0	0	miss	Mem[0]			
	8	0	miss	Mem[0]	Mem[8]		
	0	0	hit	Mem[0]	Mem[8]		
	6	0	miss	Mem[0]	Mem[6]		
	8	0	miss	Mem[8]	Mem[6]		
Fully Associative							

Mem[0]: erster Zugriff, unvermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermiedener Miss

Beispiel: Zugriffssequenz

Cache mit 4 Blöcken und Zugriffssequenz: 0, 8, 0, 6, 8

	Block address	Cache index	Hit/miss	Cache content after access			
				0	1	2	3
Direct-Mapped							
2-Way-Set Associative							
Fully Associative	0		miss	Mem[0]			
	8		miss	Mem[0]	Mem[8]		
	0		hit	Mem[0]	Mem[8]		
	6		miss	Mem[0]	Mem[8]	Mem[6]	
	8		hit	Mem[0]	Mem[8]	Mem[6]	

Mem[0]: erster Zugriff, unvermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermiedener Miss

Beispiel: Zugriffssequenz

Cache mit 4 Blöcken und Zugriffssequenz: 0, 8, 0, 6, 8

	Block address	Cache index	Hit/miss	Cache content after access				
				0	1	2	3	
Direct-Mapped	0	0	miss	Mem[0]				5 Misses
	8	0	miss	Mem[8]				
	0	0	miss	Mem[0]				
	6	2	miss	Mem[0]		Mem[6]		
	8	0	miss	Mem[8]		Mem[6]		
				Set 0		Set 1		
2-Way-Set Associative	0	0	miss	Mem[0]				4 Misses
	8	0	miss	Mem[0]	Mem[8]			
	0	0	hit	Mem[0]	Mem[8]			
	6	0	miss	Mem[0]	Mem[6]			
	8	0	miss	Mem[8]	Mem[6]			
Fully Associative	0		miss	Mem[0]				3 Misses
	8		miss	Mem[0]	Mem[8]			
	0		hit	Mem[0]	Mem[8]			
	6		miss	Mem[0]	Mem[8]	Mem[6]		
	8		hit	Mem[0]	Mem[8]	Mem[6]		

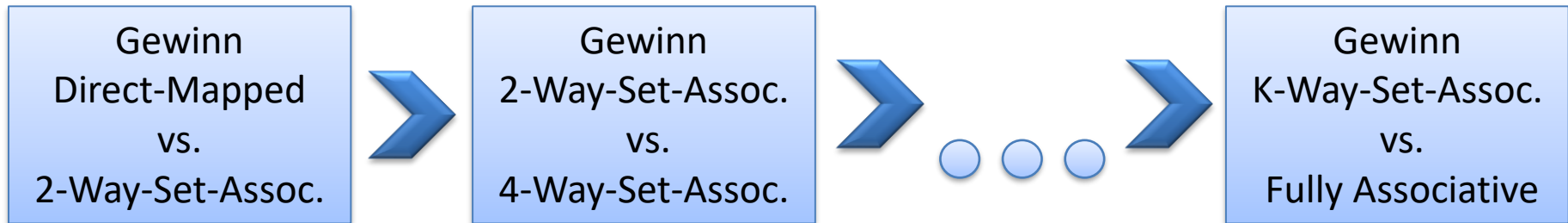
Mem[0]: erster Zugriff, unvermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermeidbarer Miss

Mem[0]: wiederholter Zugriff, vermiedener Miss

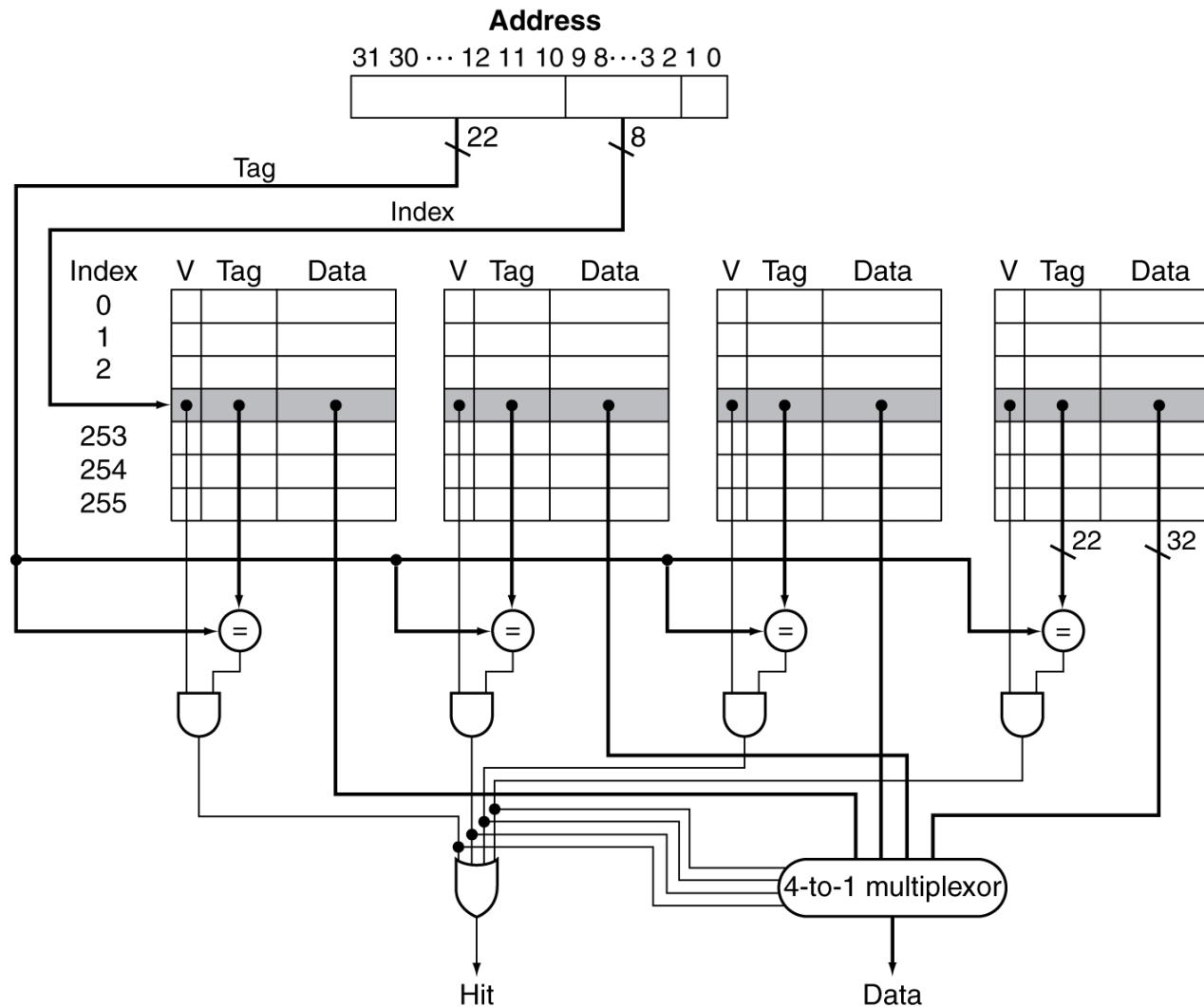
Wieviel Assoziativität braucht der Cache?

- Mehr Assoziativität bedeutet niedrigere Miss-Rate
 - ABER: je größer die Assoziativität desto geringer der Gewinn



- SPEC2000: Simulation eines Systems mit 64KB Data-Cache, 16-Word Blöcken ergibt folgende Miss-Rates:
 - 1-way: 10.3%
 - 2-way: 8.6%
 - 4-way: 8.3%
 - 8-way: 8.1%

4-Way-Set-Assoziative Cache in Hardware



1024 Blöcke
aufgeteilt in 256
Sets mit je 4
Blöcken

8 Bit für Set Index
22 Bits für Tag
2 Bits für Offset

4 Vergleiche der
Tags, einer pro
Block im Set

Hit bei einem
Treffer, Auswahl
der richtigen
Daten über MUX

- Ersetzen von Cache-Blöcken
 - bei einem Miss wird der benötigte Block in den Cache geladen und ein anderer Block im Cache muss gelöscht werden
 - der neue Block ersetzt den existierenden Block
 - während bei Direct-Mapped Caches keine Auswahl besteht, kann bei einem assoziativen Cache entschieden werden, welcher Block im Set ersetzt wird
- Strategien zum Ersetzen von Blöcken (Replacement Policy)
 - 1. Kriterium: ungültige Blöcke
 - Strategien für gültige Böcke
 - Least-Recently-Used (LRU)
 - Ersetzen des Blocks, für den der letzte Zugriff am längsten zurückliegt
 - einfach für 2-Way, machbar für 4-way, sonst zu schwierig in Hardware umzusetzen
 - Random
 - zufällige Wahl des zu ersetzenden Blocks
 - bei hoher Assoziativität (großen Sets) näherungsweise gleiche Performance wie LRU

Illustration: Vorteil Associative Cache

- Vergleich L1 Instruktionscache mit 8192B, Blockgröße 8B
 - Variante 1: Direct-Mapped Cache
 - Variante 2: 2-Way Set Associative Cache
 - Variante 3: 4-Way Set Associative Cache
- Betrachten folgende Schleife:

```
16000 LOOP:  ...
...
16008      jal  F
...
16016      jal  G
...
16028      beq  $s0,$s1,LOOP
```

```
7816      F:      ...
...
7836      jr $ra
```

```
32384     G:      ...
...
32396     jr $ra
```

- die Speicheradressen der Funktionen sind so gewählt, dass alle drei Code-Stücke auf überlappende Cache-Blöcke abgebildet werden

Abbildung auf Cache-Blöcke

	Adresse	Blocknummer	Blockindex (Direct-Mapped)	Set Index (2-Way)	Set-Index (4-Way)
Schleife	16000	2000			
	16004	2000			
	16008	2001			
	16012	2001			
	16016	2002			
	16020	2002			
	16024	2003			
	16028	2003			
F	7816	977			
	7820	977			
	7824	978			
	7828	978			
	7832	979			
	7836	979			
G	36480	4560			
	36484	4560			
	36488	4561			
	36492	4561			

Abbildung auf Cache-Blöcke

	Adresse	Blocknummer	Blockindex (Direct-Mapped)	Set Index (2-Way)	Set-Index (4-Way)	Anzahl Sets
			1024	512	256	
Schleife	16000	2000				←
	16004	2000				
	16008	2001				
	16012	2001				
	16016	2002				
	16020	2002				
	16024	2003				
	16028	2003				
F	7816	977				
	7820	977				
	7824	978				
	7828	978				
	7832	979				
	7836	979				
G	36480	4560				
	36484	4560				
	36488	4561				
	36492	4561				

Abbildung auf Cache-Blöcke

	Adresse	Blocknummer	Blockindex (Direct-Mapped)	Set Index (2-Way)	Set-Index (4-Way)	Anzahl Sets
		Anzahl Sets	1024	512	256	
Schleife	16000	2000	976	464	208	
	16004	2000	976	464	208	
	16008	2001	977	465	209	
	16012	2001	977	465	209	
	16016	2002	978	466	210	
	16020	2002	978	466	210	
	16024	2003	979	467	211	
	16028	2003	979	467	211	
F	7816	977	977	465	209	
	7820	977	977	465	209	
	7824	978	978	466	210	
	7828	978	978	466	210	
	7832	979	979	467	211	
	7836	979	979	467	211	
G	36480	4560	464	464	208	
	36484	4560	464	464	208	
	36488	4561	465	465	209	
	36492	4561	465	465	209	

Abbildung auf Cache-Blöcke

	Adresse	Blocknummer	Blockindex (Direct-Mapped)	Set Index (2-Way)	Set-Index (4-Way)	Anzahl Sets
			1024	512	256	←
Schleife	16000	2000	976	464	208	
	16004	2000	976	464	208	
	16008	2001	977	465	209	
	16012	2001	977	465	209	
	16016	2002	978	466	210	
	16020	2002	978	466	210	
	16024	2003	979	467	211	
	16028	2003	979	467	211	
F	7816	977	977	465	209	
	7820	977	977	465	209	
	7824	978	978	466	210	
	7828	978	978	466	210	
	7832	979	979	467	211	
	7836	979	979	467	211	
G	36480	4560	464	464	208	
	36484	4560	464	464	208	
	36488	4561	465	465	209	
	36492	4561	465	465	209	

Illustration: Vorteil Associative Cache

- Vergleich L1 Instruktionscache mit 8192B, Blockgröße 8B
 - Variante 1: Direct-Mapped Cache
 - Variante 2: 2-Way Set Associative Cache
 - Variante 3: 4-Way Set Associative Cache
- Betrachten folgende Schleife:

```
16000 LOOP:  ...
...
16008      jal F
...
16016      jal G
...
16028      beq $s0,$s1,LOOP
```

```
7816      F:      ...
...
7836      jr $ra
```

```
32384     G:      ...
...
32396     jr $ra
```

- die Speicheradressen der Funktionen sind so gewählt, dass alle drei Code-Stücke auf überlappende Cache-Blöcke abgebildet werden

"Ausgerollte Schleife" (Unrolled Loop)

Adresse	Blocknummer	Tag	Blockindex	Set Index (2-Way)	Set Index (4-Way)
16000	2000	1	976	464	208
16004	2000	1	976	464	208
16008	2001	1	977	465	209
7816	977	0	977	465	209
7820	977	0	977	465	209
7824	978	0	978	466	210
7828	978	0	978	466	210
7832	979	0	979	467	211
7836	979	0	979	467	211
16012	2001	1	977	465	209
16016	2002	1	978	466	210
16020	2002	1	978	466	210
36480	4560	4	464	464	208
36484	4560	4	464	464	208
36488	4561	4	465	465	209
36492	4561	4	465	465	209
16024	2003	1	979	467	211
16028	2003	1	979	467	211
16000	2000	1	976	464	208
16004	2000	1	976	464	208

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1							
16004	2000	1							
16008	2001	1							
7816	977	0							
7820	977	0							
7824	978	0							
7828	978	0							
7832	979	0							
7836	979	0							
16012	2001	1							
16016	2002	1							
16020	2002	1							
36480	4560	4							
36484	4560	4							
36488	4561	4							
36492	4561	4							
16024	2003	1							
16028	2003	1							

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976						
16004	2000	1	976						
16008	2001	1	977						
7816	977	0	977						
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	20	20	20	20
16004	2000	1	976						
16008	2001	1	977						
7816	977	0	977						
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	20	20	20	20
16004	2000	1	976						
16008	2001	1	977						
7816	977	0	977						
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	1	20	20	20
16004	2000	1	976						
16008	2001	1	977						
7816	977	0	977						
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	1	20	20	20
16004	2000	1	976	20	20	1	20	20	20
16008	2001	1	977						
7816	977	0	977						
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	1	20	20	20
16004	2000	1	976	20	20	1	20	20	20
16008	2001	1	977	20	20	1	1	20	20
7816	977	0	977						
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	1	20	20	20
16004	2000	1	976	20	20	1	20	20	20
16008	2001	1	977	20	20	1	1	20	20
7816	977	0	977	20	20	1	0	20	20
7820	977	0	977						
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

10 Misses

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	1	20	20	20
16004	2000	1	976	20	20	1	20	20	20
16008	2001	1	977	20	20	1	1	20	20
7816	977	0	977	20	20	1	0	20	20
7820	977	0	977	20	20	1	0	20	20
7824	978	0	978						
7828	978	0	978						
7832	979	0	979						
7836	979	0	979						
16012	2001	1	977						
16016	2002	1	978						
16020	2002	1	978						
36480	4560	4	464						
36484	4560	4	464						
36488	4561	4	465						
36492	4561	4	465						
16024	2003	1	979						
16028	2003	1	979						

10 Misses

Direct Mapped Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	20	20	20	20	20	20
16000	2000	1	976	20	20	1	20	20	20
16004	2000	1	976	20	20	1	20	20	20
16008	2001	1	977	20	20	1	1	20	20
7816	977	0	977	20	20	1	0	20	20
7820	977	0	977	20	20	1	0	20	20
7824	978	0	978	20	20	1	0	0	20
7828	978	0	978	20	20	1	0	0	20
7832	979	0	979	20	20	1	0	0	0
7836	979	0	979	20	20	1	0	0	0
16012	2001	1	977	20	20	1	1	0	0
16016	2002	1	978	20	20	1	1	1	0
16020	2002	1	978	20	20	1	1	1	0
36480	4560	4	464	4	20	1	1	1	0
36484	4560	4	464	4	20	1	1	1	0
36488	4561	4	465	4	4	1	1	1	0
36492	4561	4	465	4	4	1	1	1	0
16024	2003	1	979	4	4	1	1	1	1
16028	2003	1	979	4	4	1	1	1	1

10 Misses

Direct Mapped Cache – Schleifendurchgang N>2

				Cache-Inhalt (Tag)					
			Cache-Block	464	465	976	977	978	979
Adresse	Block	Tag	Blockindex	4	4	1	1	1	1
16000	2000	1	976	4	4	1	1	1	1
16004	2000	1	976	4	4	1	1	1	1
16008	2001	1	977	4	4	1	1	1	1
7816	977	0	977	4	4	1	0	1	1
7820	977	0	977	4	4	1	0	1	1
7824	978	0	978	4	4	1	0	0	1
7828	978	0	978	4	4	1	0	0	1
7832	979	0	979	4	4	1	0	0	0
7836	979	0	979	4	4	1	0	0	0
16012	2001	1	977	4	4	1	1	0	0
16016	2002	1	978	4	4	1	1	1	0
16020	2002	1	978	4	4	1	1	1	0
36480	4560	4	464	4	4	1	1	1	0
36484	4560	4	464	4	4	1	1	1	0
36488	4561	4	465	4	4	1	1	1	0
36492	4561	4	465	4	4	1	1	1	0
16024	2003	1	979	4	4	1	1	1	1
16028	2003	1	979	4	4	1	1	1	1

6 Misses

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464												
16004	2000	3	464												
16008	2001	3	465												
7816	977	1	465												
7820	977	1	465												
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	20	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464												
16008	2001	3	465												
7816	977	1	465												
7820	977	1	465												
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464												
16008	2001	3	465												
7816	977	1	465												
7820	977	1	465												
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16008	2001	3	465												
7816	977	1	465												
7820	977	1	465												
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16008	2001	3	465	3	21	3	3	23	3	24	25	25	26	27	27
7816	977	1	465												
7820	977	1	465												
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16008	2001	3	465	3	21	3	3	23	3	24	25	25	26	27	27
7816	977	1	465	3	21	3	3	1	1	24	25	25	26	27	27
7820	977	1	465												
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16008	2001	3	465	3	21	3	3	23	3	24	25	25	26	27	27
7816	977	1	465	3	21	3	3	1	1	24	25	25	26	27	27
7820	977	1	465	3	21	3	3	1	1	24	25	25	26	27	27
7824	978	1	466												
7828	978	1	466												
7832	979	1	467												
7836	979	1	467												
16012	2001	3	465												
16016	2002	3	466												
16020	2002	3	466												
36480	4560	8	464												
36484	4560	8	464												
36488	4561	8	465												
36492	4561	8	465												
16024	2003	3	467												
16028	2003	3	467												

2-Way-Set-Associative Cache – Schleifendurchgang 1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	20	21	21	22	23	23	24	25	25	26	27	27
16000	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16004	2000	3	464	3	21	3	22	23	23	24	25	25	26	27	27
16008	2001	3	465	3	21	3	3	23	3	24	25	25	26	27	27
7816	977	1	465	3	21	3	3	1	1	24	25	25	26	27	27
7820	977	1	465	3	21	3	3	1	1	24	25	25	26	27	27
7824	978	1	466	3	21	3	3	1	1	1	25	1	26	27	27
7828	978	1	466	3	21	3	3	1	1	1	25	1	26	27	27
7832	979	1	467	3	21	3	3	1	1	1	25	1	1	27	1
7836	979	1	467	3	21	3	3	1	1	1	25	1	1	27	1
16012	2001	3	465	3	21	3	3	1	3	1	25	1	1	27	1
16016	2002	3	466	3	21	3	3	1	3	1	3	3	1	27	1
16020	2002	3	466	3	21	3	3	1	3	1	3	3	1	27	1
36480	4560	8	464	3	8	8	3	1	3	1	3	3	1	27	1
36484	4560	8	464	3	8	8	3	1	3	1	3	3	1	27	1
36488	4561	8	465	3	8	8	3	8	8	1	3	3	1	27	1
36492	4561	8	465	3	8	8	3	8	8	1	3	3	1	27	1
16024	2003	3	467	3	8	8	3	8	8	1	3	3	1	3	3
16028	2003	3	467	3	8	8	3	8	8	1	3	3	1	3	3

9 Misses

2-Way-Set-Associative Cache – Schleifendurchgang >1

				Cache-Inhalt (Tag)											
			Cache-Block	464	464	LRU	465	465	LRU	466	466	LRU	467	467	LRU
Adresse	Blocknummer	Tag	Set Index (2-Way)	3	8	8	3	8	8	1	3	3	1	3	3
16000	2000	3	464	3	8	3	3	8	8	1	3	3	1	3	3
16004	2000	3	464	3	8	3	3	8	8	1	3	3	1	3	3
16008	2001	3	465	3	8	3	3	8	3	1	3	3	1	3	3
7816	977	1	465	3	8	3	3	1	1	1	3	3	1	3	3
7820	977	1	465	3	8	3	3	1	1	1	3	3	1	3	3
7824	978	1	466	3	8	3	3	1	1	1	3	1	1	3	3
7828	978	1	466	3	8	3	3	1	1	1	3	1	1	3	3
7832	979	1	467	3	8	3	3	1	1	1	3	1	1	3	1
7836	979	1	467	3	8	3	3	1	1	1	3	1	1	3	1
16012	2001	3	465	3	8	3	3	1	3	1	3	1	1	3	1
16016	2002	3	466	3	8	3	3	1	3	1	3	3	1	3	1
16020	2002	3	466	3	8	3	3	1	3	1	3	3	1	3	1
36480	4560	8	464	3	8	8	3	1	3	1	3	3	1	3	1
36484	4560	8	464	3	8	8	3	1	3	1	3	3	1	3	1
36488	4561	8	465	3	8	8	3	8	8	1	3	3	1	3	1
36492	4561	8	465	3	8	8	3	8	8	1	3	3	1	3	1
16024	2003	3	467	3	8	8	3	8	8	1	3	3	1	3	3
16028	2003	3	467	3	8	8	3	8	8	1	3	3	1	3	3

2 Misses

4-Way-Set-Associative Cache – Schleifendurchgang 1

				Notation: höhergestellte Zahlen stellen Zugriffsreihenfolge dar; LRU = "1"															
			Cache-Block	208				209				210				211			
Adresse	Block	Tag	Set Index (4-Way)	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹
16000	2000	7	208	7 ¹	21 ⁴	22 ³	23 ²	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹
16004	2000	7	208	7 ¹	21 ⁴	22 ³	23 ²	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹
16008	2001	7	209	7 ¹	21 ⁴	22 ³	23 ²	7 ¹	21 ⁴	22 ³	23 ²	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹
7816	977	3	209	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹
7820	977	3	209	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	20 ⁴	21 ³	22 ²	23 ¹	20 ⁴	21 ³	22 ²	23 ¹
7824	978	3	210	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²	20 ⁴	21 ³	22 ²	23 ¹
7828	978	3	210	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²	20 ⁴	21 ³	22 ²	23 ¹
7832	979	3	211	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²	3 ¹	21 ⁴	22 ³	23 ²
7836	979	3	211	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²	3 ¹	21 ⁴	22 ³	23 ²
16012	2001	7	209	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²	3 ¹	21 ⁴	22 ³	23 ²
16016	2002	7	210	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ²	7 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²
16020	2002	7	210	7 ¹	21 ⁴	22 ³	23 ²	7 ²	3 ¹	22 ⁴	23 ³	3 ²	7 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²
36480	4560	17	208	7 ²	17 ¹	22 ⁴	23 ³	7 ²	3 ¹	22 ⁴	23 ³	3 ²	7 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²
36484	4560	17	208	7 ²	17 ¹	22 ⁴	23 ³	7 ²	3 ¹	22 ⁴	23 ³	3 ²	7 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²
36488	4561	17	209	7 ²	17 ¹	22 ⁴	23 ³	7 ³	3 ²	17 ¹	23 ⁴	3 ²	7 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²
36492	4561	17	209	7 ²	17 ¹	22 ⁴	23 ³	7 ³	3 ²	17 ¹	23 ⁴	3 ²	7 ¹	22 ⁴	23 ³	3 ¹	21 ⁴	22 ³	23 ²
16024	2003	7	211	7 ²	17 ¹	22 ⁴	23 ³	7 ³	3 ²	17 ¹	23 ⁴	3 ²	7 ¹	22 ⁴	23 ³	3 ²	7 ¹	22 ⁴	23 ³
16028	2003	7	211	7 ²	17 ¹	22 ⁴	23 ³	7 ³	3 ²	17 ¹	23 ⁴	3 ²	7 ¹	22 ⁴	23 ³	3 ²	7 ¹	22 ⁴	23 ³

9 Misses

4-Way-Set-Associative Cache – Schleifendurchgang >1

				Notation: höhergestellte Zahlen stellen Zugriffsreihenfolge dar; LRU = "1"															
			Cache-Block	208				209				210				211			
Adresse	Block	Tag	Set Index (4-Way)	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16000	2000	7	208	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16004	2000	7	208	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16008	2001	7	209	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
7816	977	3	209	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
7820	977	3	209	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
7824	978	3	210	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
7828	978	3	210	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
7832	979	3	211	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
7836	979	3	211	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16012	2001	7	209	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16016	2002	7	210	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16020	2002	7	210	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
36480	4560	17	208	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
36484	4560	17	208	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
36488	4561	17	209	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
36492	4561	17	209	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16024	2003	7	211	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3
16028	2003	7	211	7 2	17 1	22 4	23 3	7 3	3 2	17 1	23 4	3 2	7 1	22 4	23 3	3 2	7 1	22 4	23 3

0 Misses

- Direct Mapped Caches können bei ungünstiger Konstellation zu vermeidbaren Misses führen
 - zyklischer Aufruf von mehreren Prozeduren, deren Instruktionsadressen auf gleiche Indizes abgebildet werden
 - paralleles Arbeiten auf Speicherstellen (z.B. Arrays), die auf gleiche Indizes abgebildet werden
- Assoziative Arrays erlauben paralleles Arbeiten auf mehreren Speicherstellen, die auf gleiche Indizes abgebildet werden
 - Nachteil: Caches mit größerer Assoziativität sind in Hardware komplizierter zu realisieren
 - Trade-Off zwischen Performance und Implementierungskomplexität
 - oft 4-Way- oder 8-Way-Associative Arrays