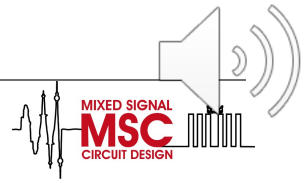
MIXED SIGNAL
MSC
CIRCUIT DESIGN

Technische Universität Berlin


Rechnerorganisation

Einstein-Prof. Dr.-Ing. Friedel Gerfers

Kapitel 9:
Caches und Hauptspeicher



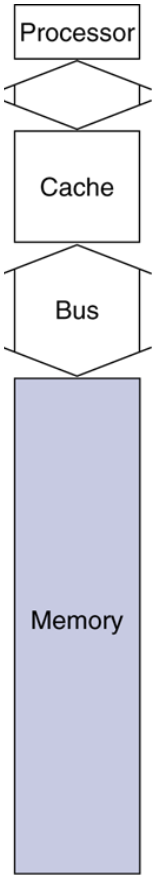
Ziele

Nach dieser Vorlesung sind Sie in der Lage:

- Folgende Begriffe zu erklären:
  - Satz-assoziativer Cache, voll-assoziativer Cache, Cache- Hierarchie bzw. Multi-level Cache, Bandbreite von Speichern
- Beschreibung eines Caches geben:
  - Einfluss des Caches auf die CPU-Zeit bestimmen
- Fehlzugriffsrate und –aufwand eines Caches zu reduzieren

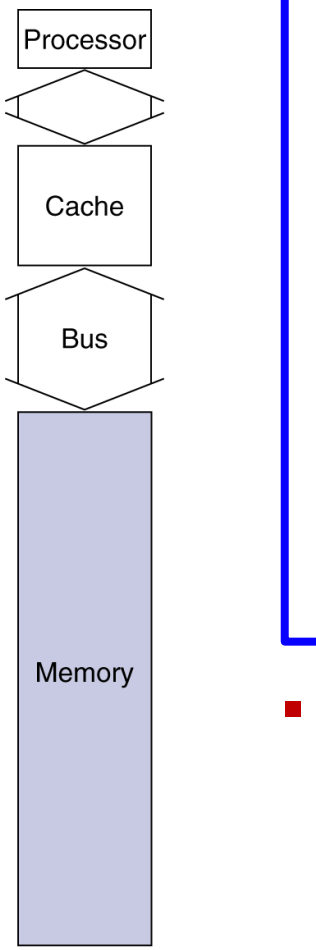
Hauptspeicher

- Hauptspeicher aus DRAM-Bausteinen aufgebaut
- DRAMs auf Dichte und nicht auf Zugriffszeit ausgelegt
- Latenz reduzieren schwierig; Bandbreite zwischen Cache und Speicher erhöhen möglich
- Cache und Speicher sind über einen Bus verbunden
  - Bus-Takt langsamer als CPU-Takt (bis zu 10x)
- Beispiel: Anzahl Bus-Takte
  - 1 Bus-Takt für Senden der Adresse
  - 15 Bus-Takte für jeden DRAM-Zugriff
  - 1 Bus-Takt für Senden eines Datenworts
- Blöcke von 4 Wörtern und 1-Wort breiten DRAM:
  - **Fehlzugriffswand**= 1 + 4×15 + 4×1 = 65 Bus-Takte
  - **Bandbreite** = 16 Byte / 65 Bus-Zyklen = 0.25 Byte/Takt



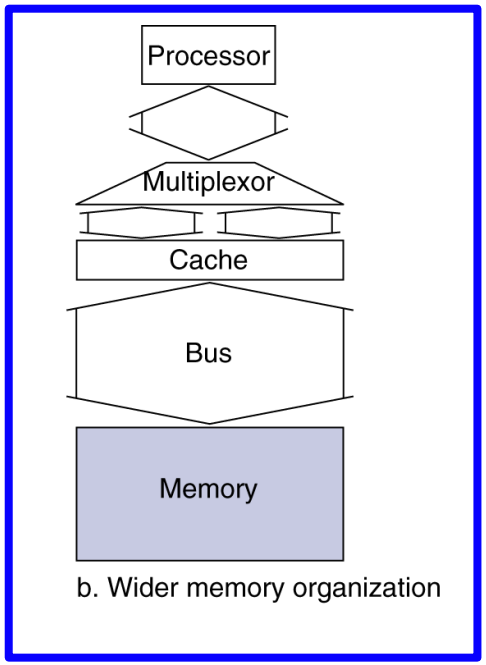
a. One-word-wide memory organization

Speicherbandbreite erhöhen

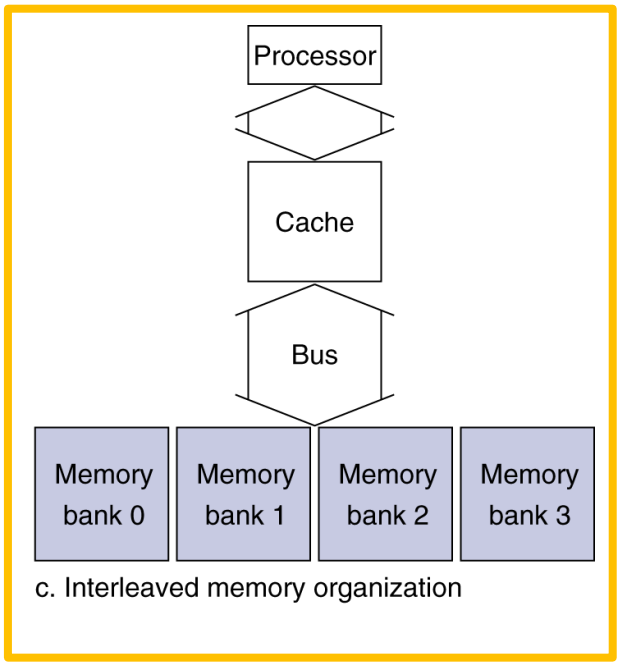


a. One-word-wide memory organization

a) 65 Bus-Takte



b. Wider memory organization



c. Interleaved memory organization

- Fehlzugriffsaufwand von
  - b1) 2-Wort-breiter Speicher: 1 + 2x15 + 2x1 = 33 Bus-Takte
  - b2) 4-Wort-breiter Speicher: 1 + 15 + 1 = 17 Bus-Takte
  - c) Verschachtelter Speicher mit 4 Banken: 1 + 1x15 + 4x1 = 20 Bus-Takte

Einfluss Cache-Leistung auf CPU-Zeit

- Kann man eigentlich nur durch Ausführen oder Simulation bestimmen
- Einfaches Modell:
  - CPU-Zeit = (CPU-Ausführungszyklen + Speicherstillstands-Zyklen) x Taktzeit
  - Speicherstillstands-Zyklen = Lesestillstands-Zyklen + Schreibestillstands-Zyklen
  - Lesestillstands-Zyklen = #Leseoperationen x Lese-Fehlrate x Lese-Fehlaufwand
  - Schreibestillstands-Zyklen = #Schreiboperationen x Schreib-Fehlrate x Schreib-Fehlaufwand + Schreibpuffer-Stillstand-Zyklen
- Annahmen:
  - Lese-Fehlaufwand = Schreib-Fehlaufwand
  - Schreib-Fehlzugriff erfordert, dass der Block geladen wird
  - Schreibpuffer-Stillstand-Zyklen nur durch detaillierte Simulation zu bestimmen und meist sehr gering. Wir nehmen an 0
- Speicherstillstands-Zyklen =
  = #Speicherzugriffe x Fehlzugriffsrate x Fehlaufwand
  = #Befehle x Fehlzugriffe/Befehl x Fehlaufwand

Beispiel: Einfluss der Cache-Leistung auf CPU-Zeit

Technische Universität Berlin

- Gegeben:

  CPI = Clock cycles per instruction

  - Fehlzugriffsrate Befehlscache: 2%
  - Fehlzugriffsrate Datencache: 4%
  - CPI ohne Speicherstillstände (perfekter Cache): 2
  - Fehleraufwand: 100 Zyklen
  - 36% Lade/Speicher-Befehle (SPECint2000)
- Um wie viel ist der Prozessor langsamer im Vergleich zu einem Prozessor mit perfektem Cache?
- Lösung (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = N_{instr} x 0.02 x 100 = 2 x N_{instr}
  - Datenfehlzugriffs-Zyklen = N_{instr} x 0.36 x 0.04 x 100 = 1.44 x N_{instr}
  - CPI mit Speicherstillstand = 2 + 3.44 = 5.44
- Prozessor mit perfektem Cache ist Faktor 5.44/2 = 2.72 schneller

Einfluss Hauptspeichergeschwindigkeit

- Angenommen wir verdoppeln die Taktgeschwindigkeit, aber die Geschwindigkeit des Hauptspeichers bleibt unverändert.
  - Fehleraufwand vergrößert sich auf 200 Zyklen

- Um wie viel schneller ist der Prozessor mit dem schnelleren Takt?
- Antwort (N_{instr} = Gesamtzahl ausgeführter Befehle):
  - Befehlsfehlzugriffs-Zyklen = N_{instr} x 0.02 x 200 = 4 x N_{instr}
  - Datenfehlzugriffs-Zyklen = N_{instr} x 0.36 x 0.04 x 200 = 2.88 x N_{instr}
  - CPI (schneller Takt) = 2 + 6.88 = 8.88
  - CPI (langsamer Takt) = 5.44

- Leistungssteigerung = 5.44 / (8.88/2) = 1.23

Cache-Leistung verbessern

- AMAT = (hit time) + (miss rate) x (miss penalty)
  - Trefferzeit verbessern
  - Fehlzugriffsrate reduzieren
  - Fehlzugriffsaufwand reduzieren

- Was passiert, wenn wir die Blöcke größer machen?

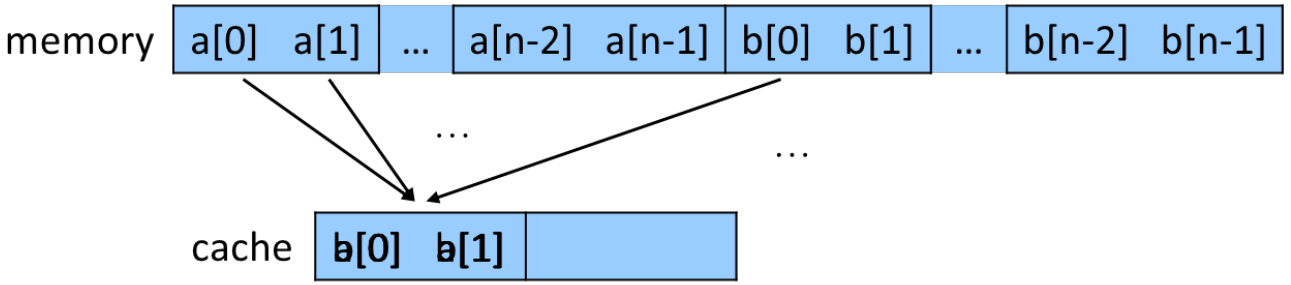
- Wir besprechen 2 Methoden um Cache-Leistung zu verbessern
  - assoziative Caches: reduzieren Fehlzugriffsrate durch flexiblere Platzierung von Blöcken
  - Cache-Speicherhierarchie (multilevel cache): reduziert Fehlzugriffsaufwand

Motivation Assoziative Caches

- Beispiel: for (i=0; i<n; i++)

  dotprod += a[i]*b[i];

- Angenommen a[i] und b[i] werden auf gleichen Cache-Block abgebildet
  - Nicht unüblich, wenn a und b hintereinander im Speicher allokiert wurden und ihre Größe ein Vielfaches der Cachegröße ist



  - Jeder Zugriff auf a und b generiert Fehlzugriff!
- Lösung: Flexiblere Platzierung von Blöcken → assoziativer Cache

Assoziative Caches

- vollassoziativer Cache (*fully associative cache*)
  - Auch Inhaltsadressierbarer Speicher (*Content Addressable Memory*) genannt
  - Block kann an jeder Stelle im Cache platziert werden
  - Alle Einträge müssen parallel durchsucht werden
  - 1 Vergleicher pro Cache-Eintrag → teuer
- n-fach satzassoziativer Cache (*n-way set associative cache*)
  - Besteht aus einer Menge von Sätzen (*sets*)
  - die aus jeweils *n* Ways bestehen
  - Blockadresse bestimmt auf welchen Satz Block abgebildet wird
    - (Cache-Index) = (Blockadresse) modulo (#Sätze)
  - Gleichzeitiges Durchsuchen aller Einträge eines Satzes
  - *n* Vergleicher benötigt

Mögliche Organisationen für Cache mit 8 Blöcken

1-fach satzassoziativ (direkt abgebildet)		
Satz	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Vergleich Caches mit verschiedener Assoziativität / 1

- Vergleich dreier 4-Block-Caches
  - Direct mapped, 2-way set associative, fully associative
- Sequenz von Blockadressen: 0, 8, 0, 6, 8
- Wie viele Treffer sind möglich?

- Direkt abgebildeter Cache:

Block - adresse	Cache-Index	Hit/miss	Cache-Inhalt nach Zugriff			
			0	1	2	3
0	0 mod 4 = 0	miss	**Mem[0]**			
8	8 mod 4 = 0	miss	**Mem[8]**			
0	0 mod 4 = 0	miss	**Mem[0]**			
6	6 mod 4 = 2	miss	Mem[0]		**Mem[6]**	
8	8 mod 4 = 0	miss	**Mem[8]**		Mem[6]	

Cache-Index = BlockAdresse modulo (Anzahl **Blocks** in Cache) = BlockAdresse modulo **4**

Vergleich Caches / 2

2-fach satz-assoziativer Cache:

Satz-Index = BlockAdresse modulo (Anzahl **Sets** in Cache) = BlockAdresse modulo **2**

Block-adresse	Cache-Index	Hit/miss	Cache-Inhalt nach Zugriff			
			Set 0		Set 1	
0	0 mod 2 = 0	miss	**Mem[0]**			
8	8 mod 2 = 0	miss	Mem[0]	**Mem[8]**		
0	0 mod 2 = 0	hit	**Mem[0]**	Mem[8]		
6	6 mod 2 = 0	miss	Mem[0]	**Mem[6]**		
8	8 mod 2 = 0	miss	**Mem[8]**	Mem[6]		

Ersetzungsschema: *least recently used*

Ersetzungsschema

- Keine Wahl bei direkt abgebildeter Cache

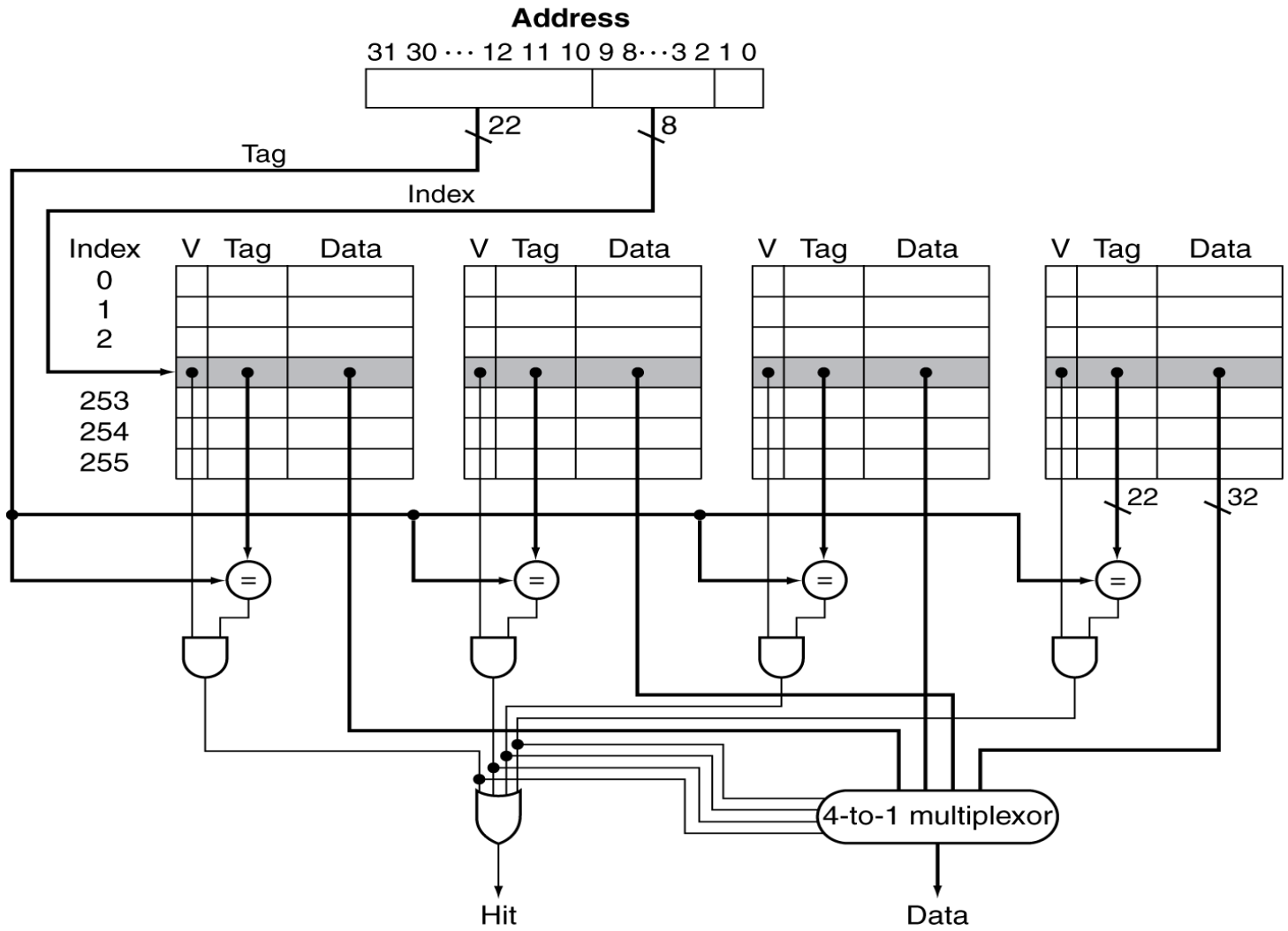
- Random
  - Ersetze einen zufälligen Block im Satz (einfache Hardware)
- First-In-First-Out (FIFO)
  - Ersetze Block, welcher als erster im Cache geladen ist
- Least Recently Used (LRU)
  - Ersetze Block, auf den am längsten nicht mehr zugegriffen wurde
  - Komplizierte Hardware (> 4-way: Approximation)
- Optimaler Algorithmus (OPT oder MIN)
  - Ersetze Block, auf den am längsten nicht mehr zugegriffen wird

Wie assoziativ?

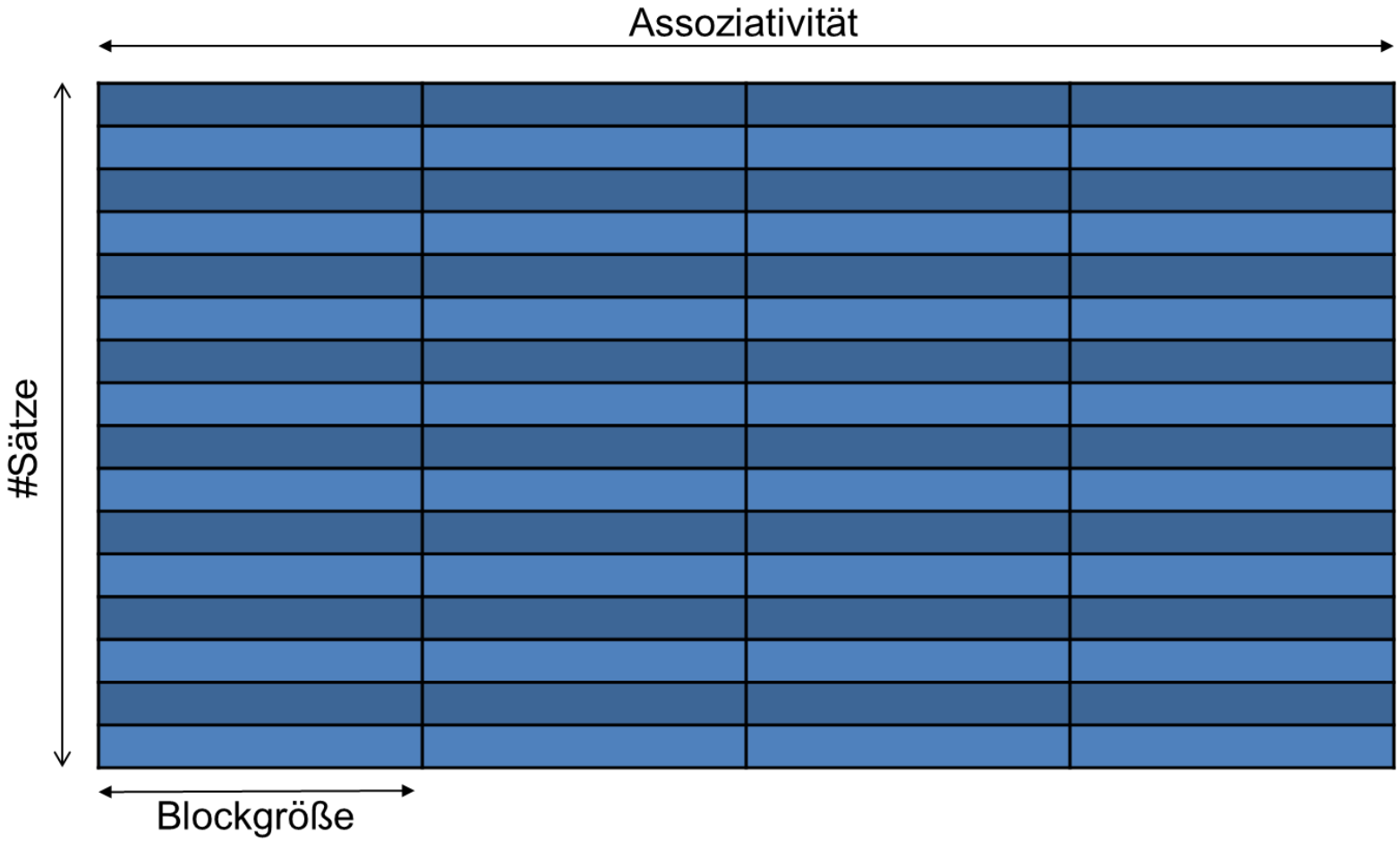
- Erhöhung der Assoziativität reduziert Fehlerrate
  - Aber mit abnehmende Rendite (*diminishing returns*)

- Simulation eines 64KB Datencaches, 64-Byte Blöcke, SPEC2000:
  - 1-fach: 10.3%
  - 2-fach: 8.6%
  - 4-fach: 8.3%
  - 8-fach: 8.1%

Organisation eines 4-fach satzassoziativen Caches



Kapazität eines satzassoziativen Caches



Cache-Größe = #Sätze x Assoziativität x Blockgröße

Cache Übungen

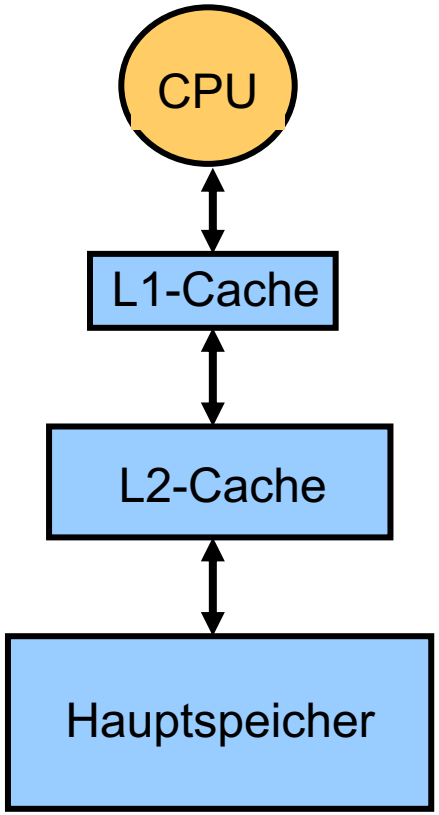
- Gegeben:
  - Cache-Größe = 4 KB
  - 4-fach satzassoziativ
  - Blockgröße = 4 Wörter, Wort = 4 Bytes
  - Adresslänge = 32 Bit

Cache - Größe = #Sätze x Assoziativität x Blockgröße

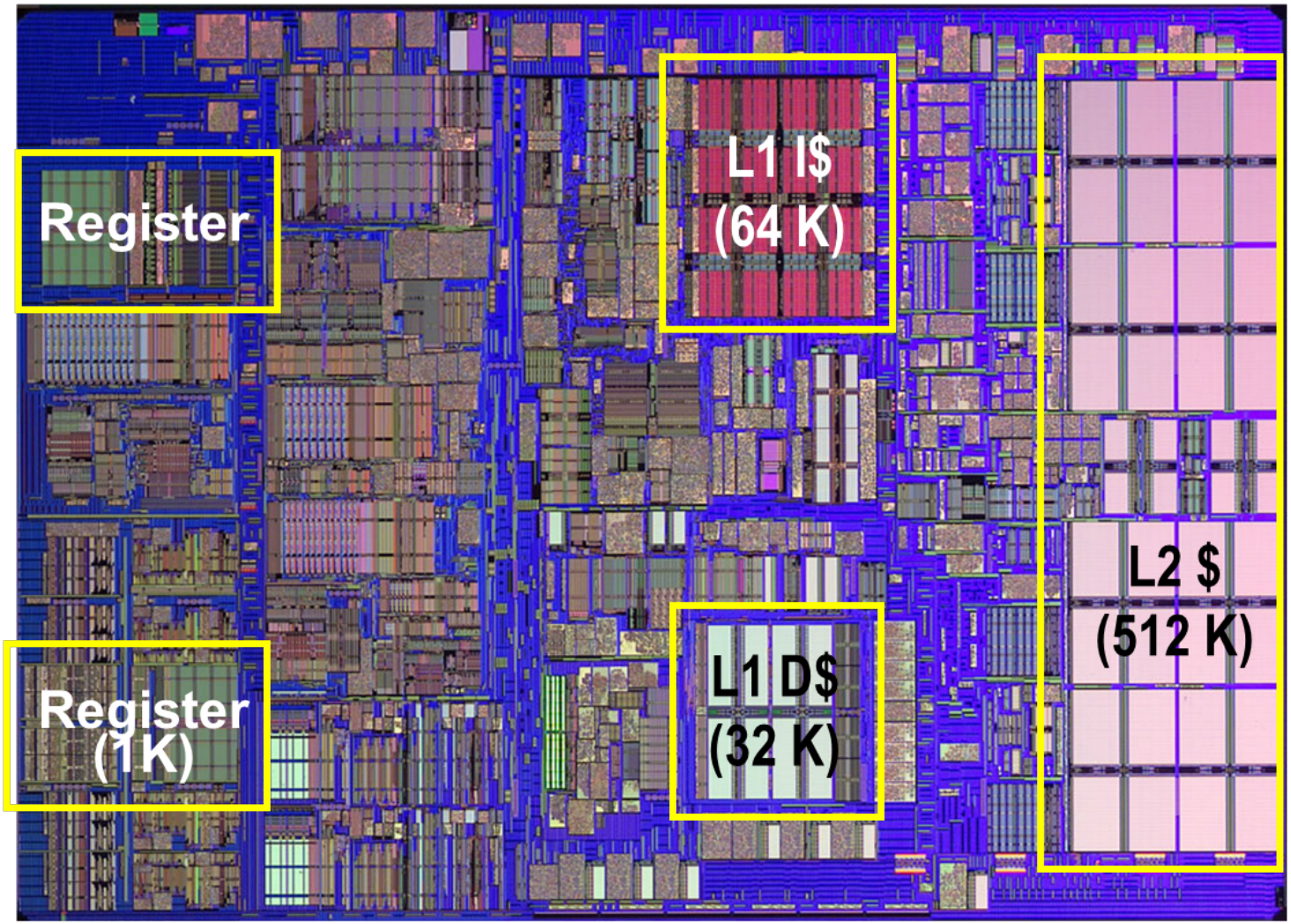
Frage	Lösung
Wie viel Sätze?	4096 / 4 / (4*4) = 64 Einträge
Wie viele Bits für Index?	64 = 2^6 → 6 Bit + 4 Bit Byte Select
Wie viele Bits für Tag?	32 - 6 - 4 = 22 Bit
Auf welchen Satz wird Byteadresse 4196 abgebildet?	4196 = 1 0|00 0110|0100$_2$ → 000110$_2$ = 6

Cache-Speicherhierarchie

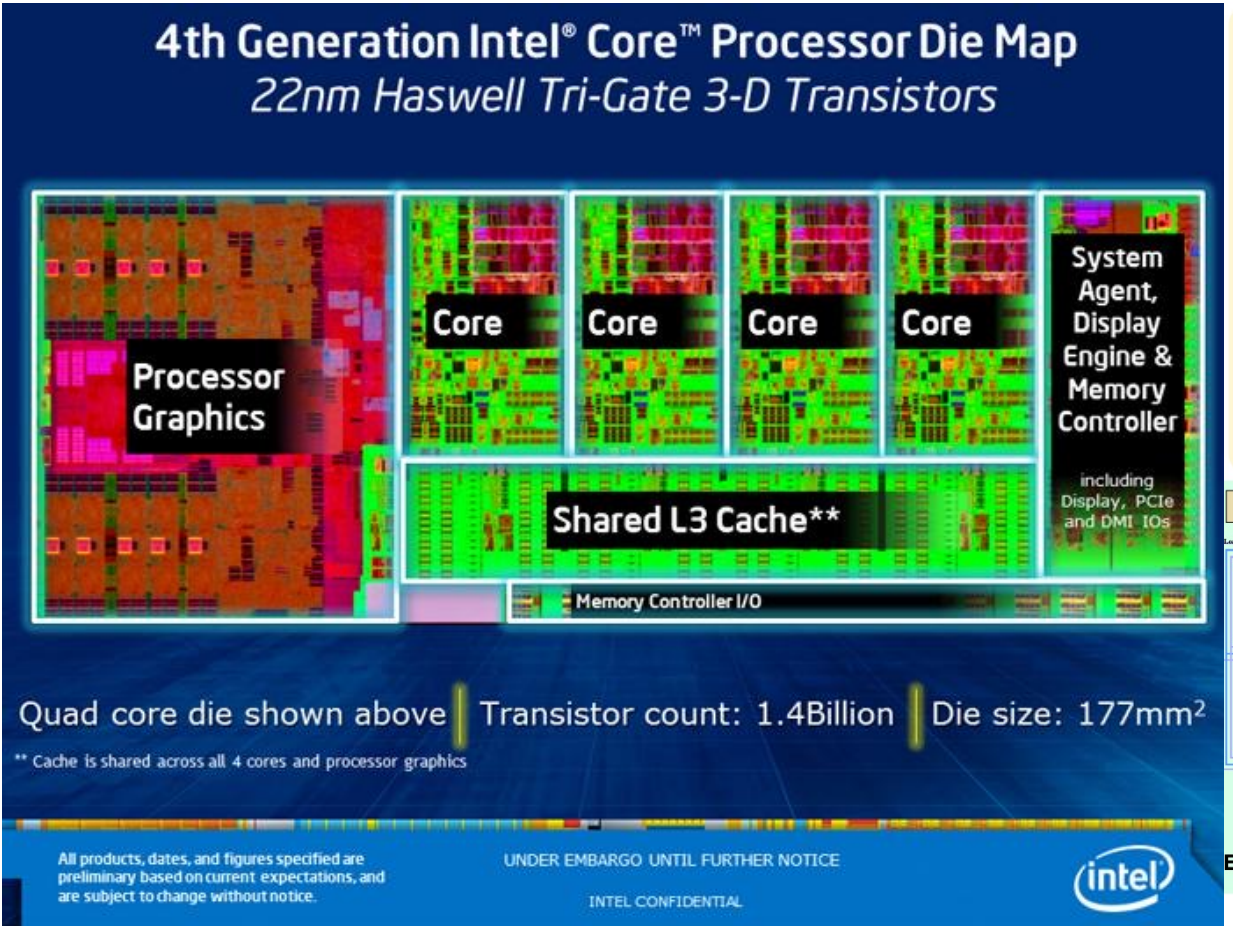
- Fehleraufwand reduzieren durch mehrere Cache-Ebenen (*cache levels*)
  - Cache-Speicherhierarchie
- Primärer Cache (L1-Cache) klein und schnell
  - Trefferzeit 1-3 Taktzyklen
- Sekundärer Cache (L2-Cache) größer und langsamer als L1, aber immer noch viel schneller als Hauptspeicher
  - Trefferzeit L2: 10-50 Taktzyklen
  - Hauptspeicher: 100-500 Taktzyklen
- Einige Hochleistungsprozessoren haben L3-Cache

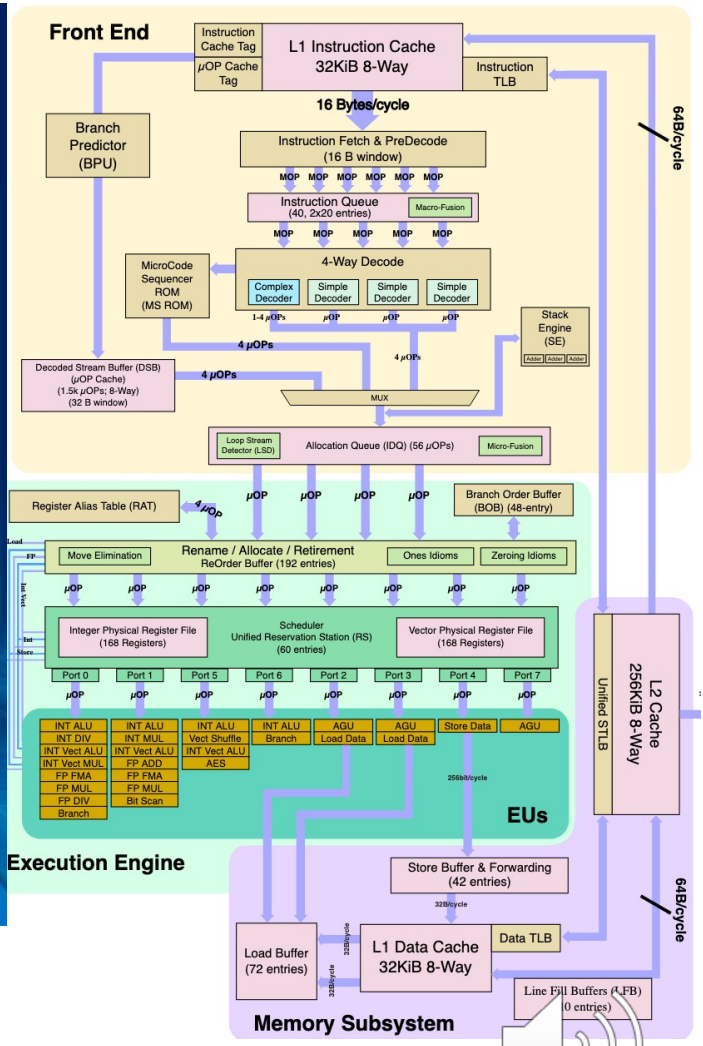


iMAC PowerPC: Alle Cache auf dem Chip



Intel Haswell Prozessor





Memory Hierarchy of Intel Haswell CPU

- Die Cache-Bandbreite beträgt 64B/Zyklus für das Laden und 32B/Zyklus für das Speichern
  - L1I Cache:
    - 32 KB 8-way set associative
      - 64 B line size, Write-back policy, shared by the two threads, per core
  - L1D Cache:
    - 32 KB 8-way set associative
      - 64 B line size, shared by the two threads, per core, 4 cycles for fastest load-to-use
      - 64 Bytes/cycle load bandwidth, 32 Bytes/cycle store bandwidth, Write-back policy
  - L2 Cache:
    - unified, 256 KB 8-way set associative
      - 11 cycles for fastest load-to-use, 64B/cycle bandwidth to L1$, Write-back policy
  - L3 Cache:
    - 1.5 - 3 MB
      - Write-back policy, per core

Leistung einer Cache-Speicherhierarchie

- Wir haben Prozessor mit
  - Grund-CPI von 1,0 (wenn alle Zugriffe im primären Cache treffen)
  - Taktgeschwindigkeit von 4 GHz (0,25ns Taktzykluszeit)
  - Hauptspeicher mit Zugriffszeit von 100ns (400 Taktzyklen)
  - Fehlrate_{L1} = 2% (Befehle und Daten)
- Nur L1-Cache:
  - CPI = Grund-CPI + Fehlrate_{L1} x Fehlaufwand = 1,0 + 0,02x400 = 9,0
- Nun mit L2-Cache
  - Zugriffszeit von 5ns (20 Taktzyklen)
  - Fehlrate_{L2} = L2 lokaler Fehlrate (#Fehler in L2 / #Zugriffe auf L2) = 25%
- CPI = Grund-CPI + Fehlrate_{L1} x Fehlaufwand_{L1}
- Fehlaufwand_{L1} = Trefferzeit_{L2} + Fehlrate_{L2} x Fehlaufwand_{L2}
- CPI = 1,0 + 0,02 x (20 + 0,25 x 400) = 3,4
- Prozessor mit L2-Cache ist 9,0/3,4 = 2,6 x schneller

Überlegungen zum Multilevel-Cache

- L1-Cache
  - zielt auf minimale Trefferzeit (*hit time*) ab
- L2-Cache:
  - zielt auf geringe Fehlzugriffsrate ab, um Hauptspeicher-Zugriffe zu vermeiden
  - Trefferzeit hat weniger Einfluss
- Fazit:
  - L1-Caches meist kleiner als bei Systemen, bei denen nur eine Cache-Ebene verwendet wird
  - L1-Blöcke kleiner als L2-Blöcke
  - L2 höhere Assoziativität als L1

Fazit

- Um Prozessor/DRAM-Leistungslücke zu schließen, wird Cache-Speicherhierarchie verwendet
  - Moderne Hochleistungssysteme haben mehrere Cache-Ebenen
- Instruktionen/Daten werden mit hoher Wahrscheinlichkeit im Cache gefunden wegen temporaler und räumliche Lokalität
- Cache kann direkt abgebildet, satz-assoziativ oder voll-assoziativ sein.
- Hilfreiche Cache-Formeln:
  - Cache-Größe = (# Sätze) x Assoziativität x (Blockgröße)
  - Blockadresse = (Byteadresse) / (Blockgröße in Bytes)
  - Cache-Index = (Blockadresse) modulo (# Sätze)
- Nicht merken, sondern verstehen!