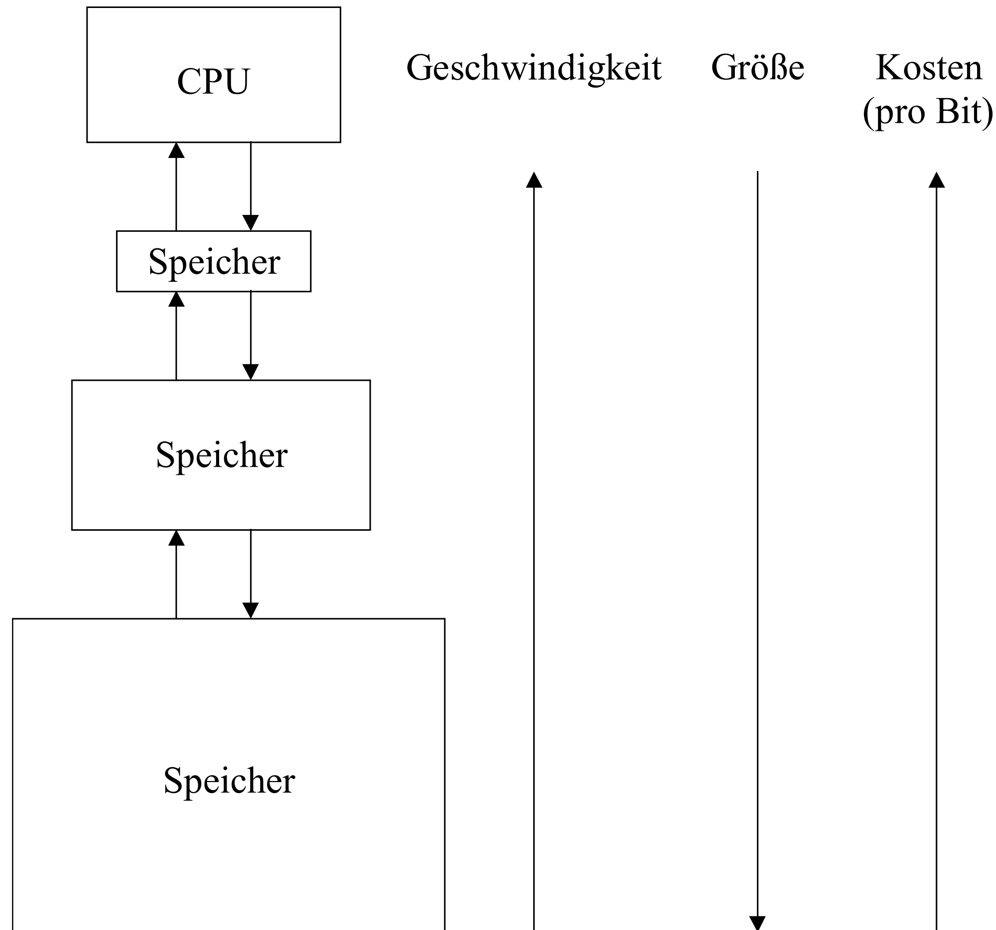
7. Speicherhierarchien



Im Folgenden:

Wie kann damit die "Illusion" eines sehr großen, sehr schnellen Speichers erzeugt werden?

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Prinzip der Lokalität

- **zeitliche Lokalität**
  - wenn eine Speicherzelle benutzt wurde, ist die Wahrscheinlichkeit, dass sie bald noch einmal benötigt wird, relativ groß
    - Programmcode enthält einen großen Anteil von Schleifen oder häufig benutzten Unterprogrammen
    - Daten werden gelesen, manipuliert und wieder geschrieben, dieselben Variablen werden für viele Zwischenergebnisse benutzt
- **räumliche Lokalität**
  - wenn eine Speicherzelle benutzt wurde, ist die Wahrscheinlichkeit, dass eine Speicherzelle mit einer in der Nähe liegenden Adresse bald benötigt wird, relativ groß
    - Programmcode wird sequentiell abgearbeitet
    - Daten liegen logisch gruppiert im Speicher vor (array's, struct's, Objekte)

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

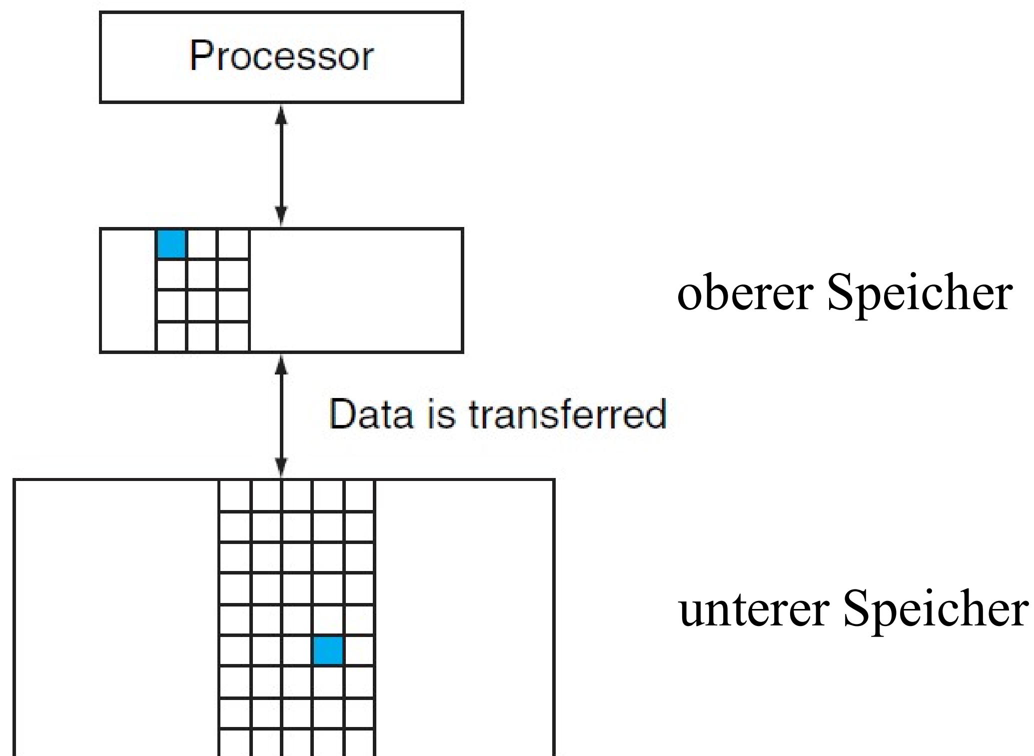
Ausnutzung der Lokalität

- **zeitliche Lokalität**
  - einmal benutzte Daten werden im schnellen Speicher nahe der CPU gehalten
    - schneller Zugriff, wenn sie noch einmal benötigt werden
- **räumliche Lokalität**
  - Daten werden zwischen den Hierarchie-Stufen in großen Blöcken transferiert
    - höherer Durchsatz als bei Transfer einzelner Worte
    - anschließend schneller Zugriff auch auf benachbarte Daten

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Obere und untere Ebene

- wir betrachten zunächst nur zwei benachbarte Ebenen (obere und untere)
- Daten werden zwischen oberer und unterer Ebene in Blöcken ausgetauscht



Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Begriffe

- **Block**
  - die kleinste Datenmenge, die in einer Ebene vorhanden oder nicht vorhanden sein kann
- ***hit* (Treffer)**
  - das angeforderte Datum befindet sich in einem Block des oberen Speichers
- ***miss* (Fehlwurf)**
  - das angeforderte Datum, befindet sich in keinem Block des oberen Speichers
  - der untere Speicher wird dann nach der Information befragt

Begriffe (2)

- ***hit rate*, *hit ratio* (Trefferquote)**
  - Bruchteil der Speicherzugriffe, die auf der oberen Ebene Erfolg haben
- ***miss rate*, *miss ratio* (1-Trefferquote)**
  - Bruchteil der auf der oberen Ebene erfolglosen Speicherzugriffe
- ***hit time***
  - Gesamtzeit für einen erfolgreichen Speicherzugriff
  - einschließlich der Zeit zum Feststellen, dass es sich um einen *hit* handelt
- ***miss penalty* (Strafe, Kosten für Fehlwurf)**
  - Gesamtzeit für das Ersetzen eines Blocks im oberen Speicher durch einen Block aus dem unteren Speicher plus der Übertragungszeit in die obere Ebene

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Cache

- engl.: Versteck, geheimes Lager
- zwei Bedeutungen
  - Speicher zwischen CPU und Hauptspeicher
  - jeder Speicher, der die Lokalität von Zugriffen ausnutzt, z.B.
    - Festplatten-Cache
    - Software-Technik, um Werte nicht mehrfach aufwändig zu berechnen

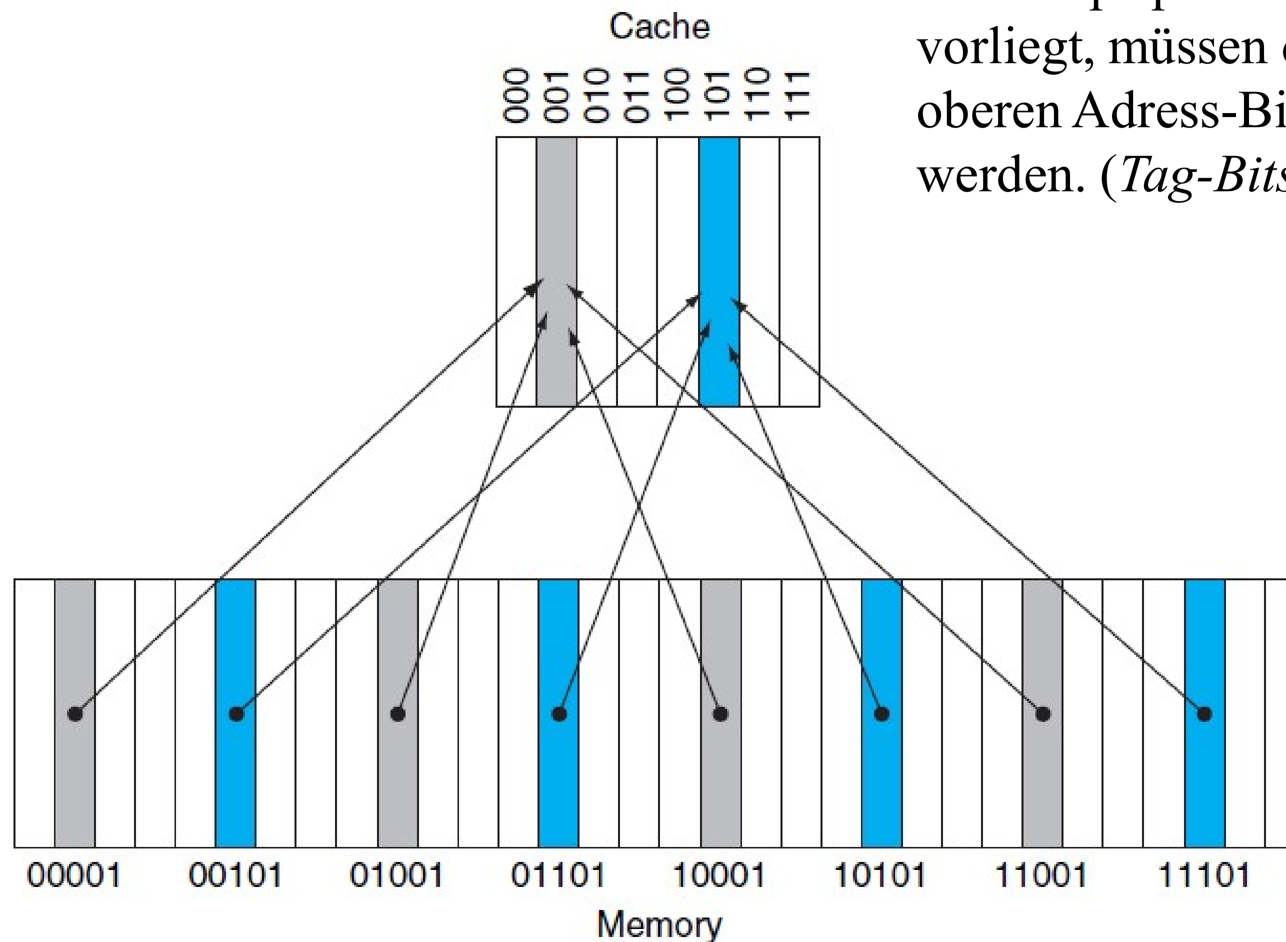
- **Probleme, die gelöst werden müssen**
  - Wie weiß man, ob sich ein Datenelement bereits im Cache befindet?
  - Falls es sich dort befindet: wie findet man es?

Direct Mapped Cache

- **Erstes Beispiel**
  - Jeder Block hat eine eindeutig definierte Adresse im Cache.
  - Blockgröße entspricht im Beispiel nur einem Wort.
  - Für das *Mapping* gilt praktisch immer
    - Adresse im Cache ist Adresse im Hauptspeicher modulo Anzahl der Blöcke im Cache
    - Falls Anzahl der Blöcke eine Zweierpotenz (auch das praktisch immer)
      - niedere Adressbits als Cacheadresse verwenden.
    - Viele Blöcke im Hauptspeicher teilen sich einen Block im Cache.

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Direct Mapped Cache (2)

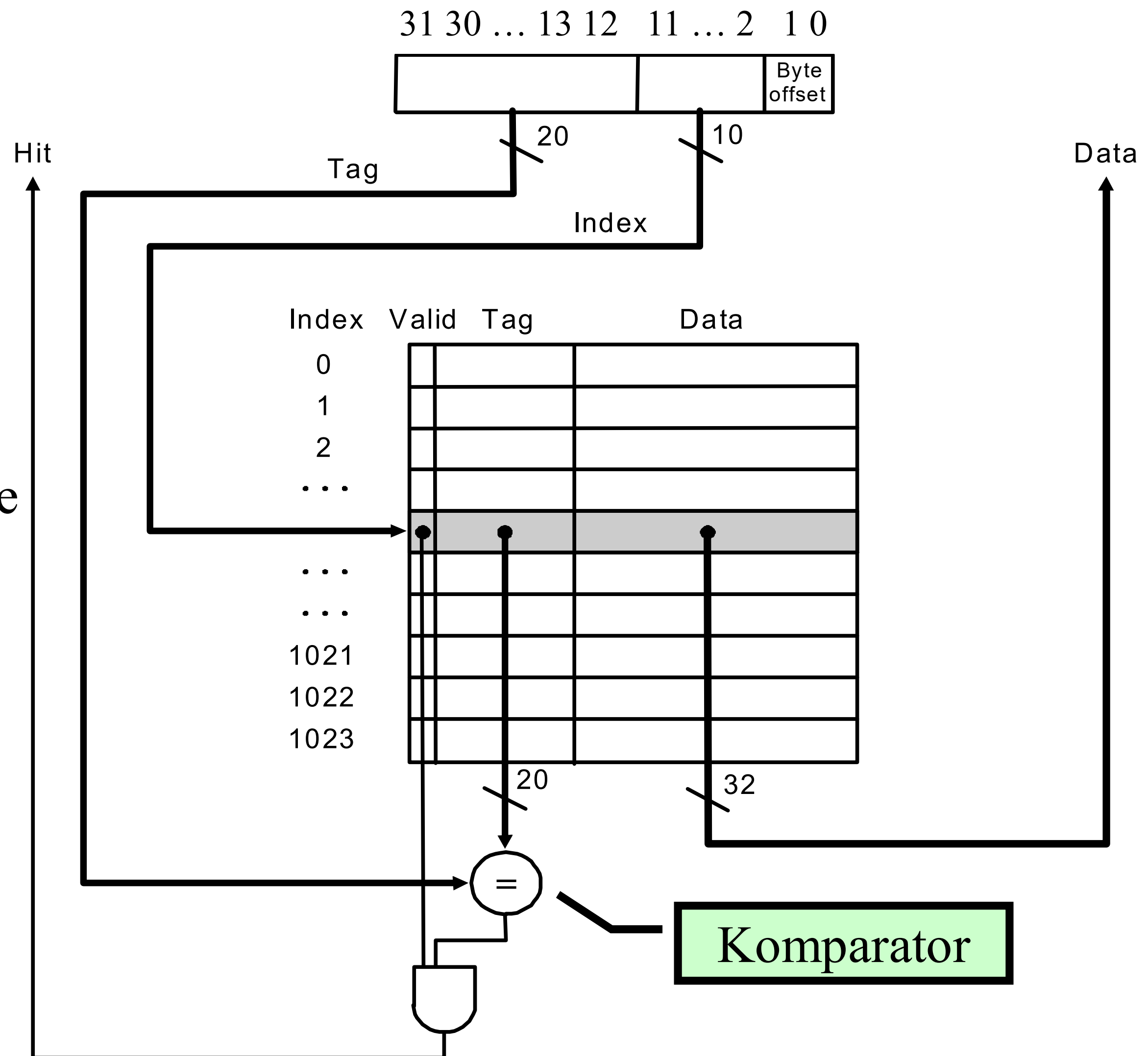


Um zu erkennen, welcher Block des Hauptspeichers im Cache vorliegt, müssen die restlichen, oberen Adress-Bits gespeichert werden. (*Tag-Bits*: engl. *tag*: Etikett)

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Direct Mapped Cache (3)

- **bei MIPS**



Valid-Bit: zeigt an, ob *Tag* gültig ist (d.h. ob Block gerade benutzt wird)

– *Welche Art der Lokalität wird ausgenutzt?*

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Einbau des Caches in Pipeline

- Caches ersetzen Programm- und Datenspeicher in der MIPS Pipeline

- **Lesen aus dem Cache**
  - *read hit*
    - Daten werden, wie bereits angenommen, innerhalb eines Taktes gelesen
  - *read miss*
    - Lese-Adresse zum Hauptspeicher schicken
    - Pipeline anhalten (*stall*), bis Daten aus dem Speicher eintreffen und in den Cache geschrieben worden sind
    - anschließend erneut auf Cache zugreifen (diesmal ein *Hit*)
- **Schreiben in den Cache**
  - schreibt man Daten nur in den Cache, stimmen Cache und Speicherinhalt nicht überein
    - Cache und Hauptspeicher sind inkonsistent
    - Daten müssen irgendwann in den Hauptspeicher

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Schreibstrategien

- **Zwei Schreibstrategien**
  - *write-through*
    - immer gleichzeitig in Cache und Hauptspeicher schreiben
    - falls Speicher zu langsam, muss Prozessor angehalten werden
    - Abhilfe: *write-buffer*
      - die Daten, die in den Hauptspeicher geschrieben werden sollen, werden zusammen mit den Schreibadressen in einen Zwischenspeicher abgelegt, der selbständig die Daten schreibt, sobald der Bus zum Speicher frei ist
      - meist hat der *write-buffer* Platz für mehrere (4-16) Worte, um *write-bursts* (schnelle Folge von Schreibbefehlen) abzufangen
      - Prozessor arbeitet sofort weiter, sofern der *write-buffer* nicht voll ist
      - Achtung: Lesen
        - » Bei einem *cache miss* muss auch der *write-buffer* abgefragt werden, ob er noch Schreibaufträge für den betroffenen Block bearbeitet.

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Schreibstrategien (2)

- *write-back*
  - meist bessere Alternative zu *write-through*
  - *dirty bit*
    - zeigt an, dass Block im Cache inkonsistent zum Speicher ist
    - wird beim Laden aus dem Hauptspeicher auf 0 gesetzt
    - wird bei jedem Schreiben in den Block auf 1 gesetzt
  - Die Daten werden erst dann zurück geschrieben, wenn der Platz im Cache für andere Daten benötigt wird (Verdrängung).
    - Also nur bei einem *read miss*, der einen *dirty* Block verdrängt.
    - Dann wird erst zurückgeschrieben, und dann gelesen.

Beispiel (Zustand immer nach einem *miss*)

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

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

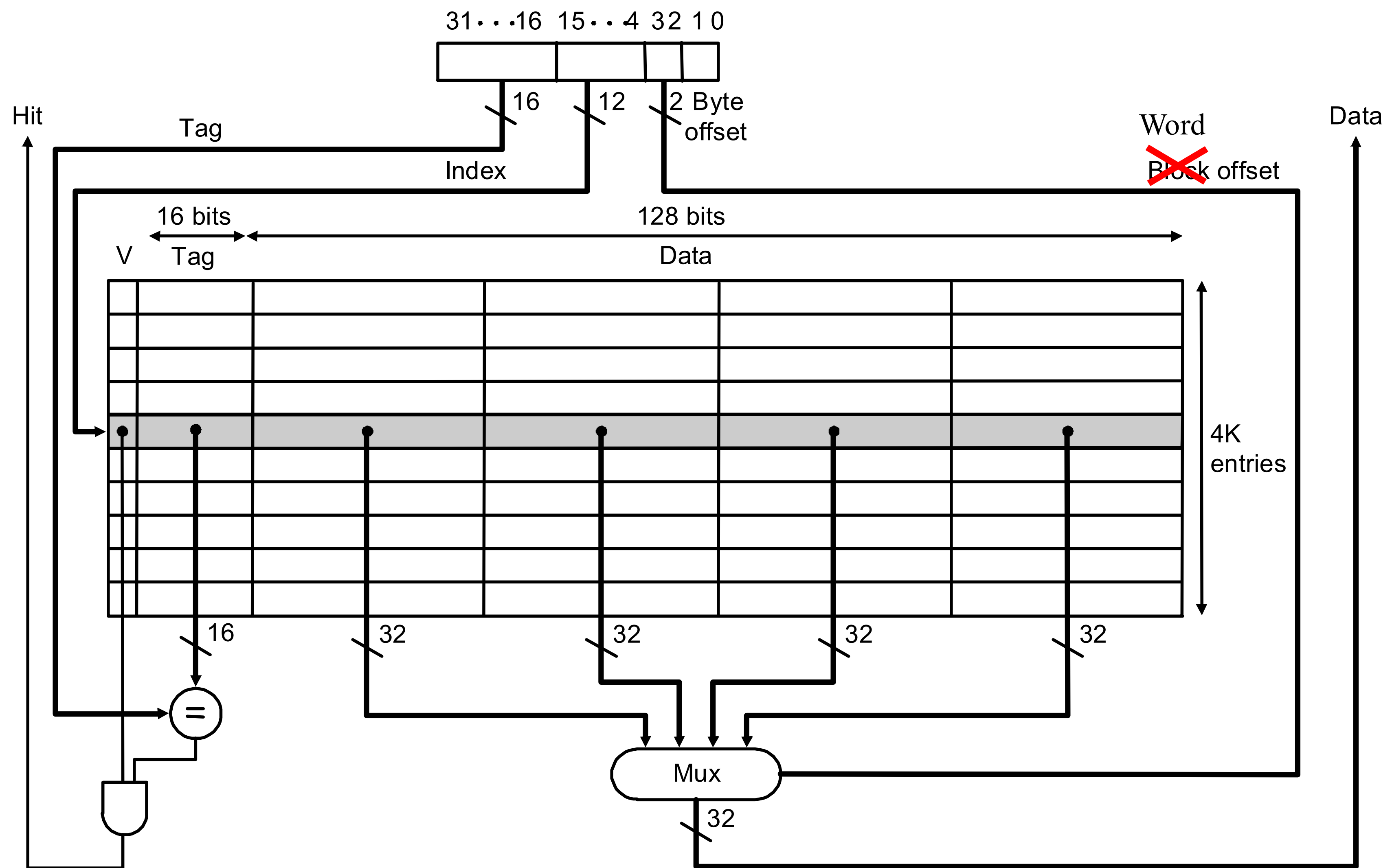
Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Direct Mapped Cache (4)

- **Ausnutzung der räumlichen Lokalität**
  - Block umfasst mehrere Worte
    - z.B. 1 Block = 4 Worte (praktisch immer Zweierpotenz)
  - Daten werden immer als ganzer Block aus dem Hauptspeicher gelesen oder dorthin zurück geschrieben.
  - Zugriff auf Bytes über
    - Byte Offset (Byteadresse innerhalb eines Wortes, 2 Bits)
      - Bei lw, sw oder Instruction Fetch immer 00, da Wortadresse
    - Word Offset (Wortadresse innerhalb Block)
      - niederwertige Bits
    - Index (Blockadresse im Cache)
      - mittlere Bits
    - Tag (Nummer des Blocks im Hauptspeicher)
      - höherwertige Bits
      - Vergleich mit gespeichertem Tag, um *hit/miss* festzustellen

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Direct Mapped Cache (5)



Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Hits vs. Misses

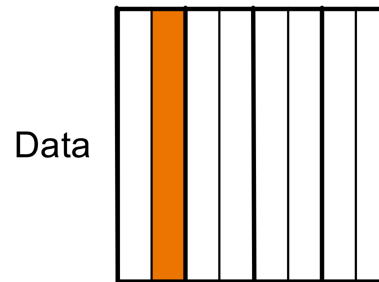
- ***read***
  - *read hit*: Wort aus Cache lesen
  - *read miss*
    - bei *write back* und gesetztem *dirty bit*: alten Block in Hauptspeicher schreiben
    - neuen Block aus Hauptspeicher lesen
- ***write***
  - es reicht nicht, *tag* und Wort zu schreiben, da die anderen drei Worte zu einer anderen Adresse gehören können
    - beim Schreiben *tag* mit Adresse vergleichen (wie beim Lesen)
  - *write hit*: in Cache schreiben (später: *write back*)
  - *write miss*: vor Schreiben Block aus Speicher laden
    - bei *write-back* und gesetztem *dirty bit* Block vorher sogar noch in den Hauptspeicher zurückschreiben (also alten Block schreiben, neuen Block lesen, dann erst Wort in den Cache schreiben)

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Verringerung des *miss ratio*

- **Flexiblere Platzierung der Blöcke**
  - bisher: *direct mapped*
    - jeder Block kann nur an einer Stelle im Cache abgelegt werden

direct mapped



set associative

Data

Tag

1

2

Search

fully associative

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

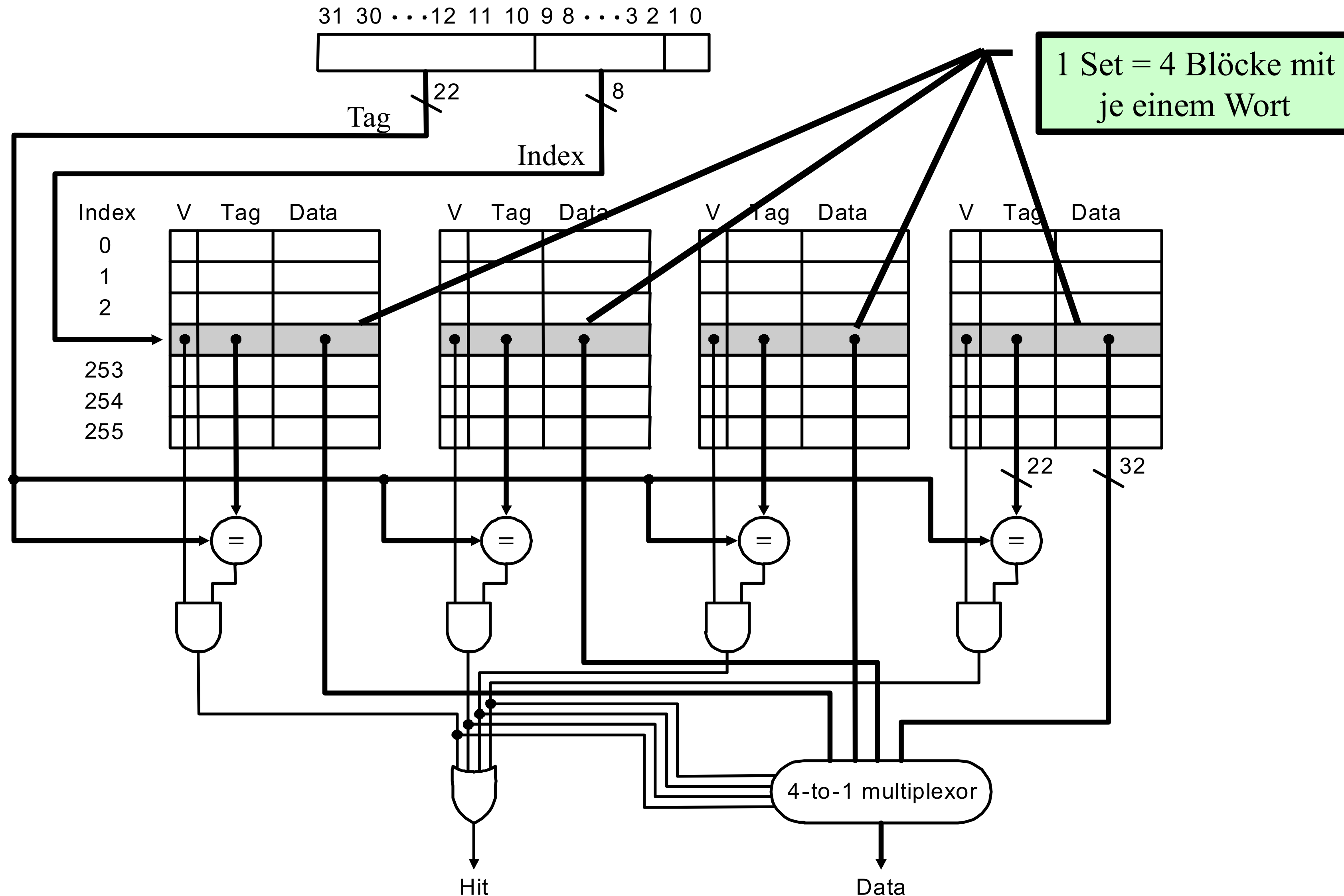
Cache-Organisation

- *fully associative*
  - Jeder Block kann an jeder Stelle des Caches abgelegt werden.
  - Cache-Blöcke müssen nach der richtigen Adresse abgesucht werden.
  - Suche muss parallel durchgeführt werden.
  - Ein Komparator für ***jeden*** Cache-Eintrag
  - Hardwarekosten extrem hoch
  - Nur sinnvoll bei sehr kleinen Caches
- Kompromiss: *set associative*
  - Feste Anzahl von Speicherplätzen (mindestens 2), in denen ein Block abgelegt werden kann (*set*)
  - Bei *n* Plätzen pro *set* spricht man von
    - *n*-way set-associative Cache (n-Wege Cache)
  - *Direct mapping* in den *set*, *fully associative* innerhalb des *sets*

- **Erhöhung des Grades an Assoziativität verringert die *miss rate***

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Implementierung (4-way set associative)



Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Verdrängung von Blöcken

- **Auswahl des zu verdrängenden Blockes**
  - Bei *direct mapped*: keine Wahl
  - Bei *set associative* oder *fully associative* muss man entscheiden, welcher Block verdrängt werden soll.
    - Idealerweise der Block, der noch am längsten in der Zukunft nicht mehr gebraucht werden wird.
    - Das kann der Prozessor nicht wirklich wissen.

Verdrängungsstrategien

- RANDOM
  - entferne einen zufällig ausgewählten Block
- LIFO
  - entferne den zuletzt geladenen Block
- FIFO
  - entferne den zuerst geladenen Block
- LFU (*least frequently used*)
  - entferne den Block, auf den am wenigsten häufig zugegriffen wurde
- LRU (*least recently used*)
  - entferne den Block, auf den am längsten nicht mehr zugegriffen worden ist
  - gilt als beste Strategie

Verdrängung von Blöcken (2)

- **LRU (*least recently used*) erweist sich meist als das beste Verfahren.**
  - LRU bei 2-way Cache: ein Extra-Bit
  - beim Mehr-Wege Cache ist das erheblich aufwändiger
    - Häufig verzichtet man deshalb auf exakte Implementierung von LRU.
    - Näherungen, wie FIFO sind erheblich einfacher zu implementieren.

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Performancevergleich

- **Beispiel**
  - 10 Spec2000 Benchmarks
  - Blockgröße 64 Bytes
  - Anzahl der *cache misses* für 1000 Instruktionen

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

  - geringer Unterschied zwischen LRU und anderen Algorithmen bei großen Cachegrößen

Weitere wichtige Themen

- **Virtueller Speicher**
  - Unterstützung in Prozessoren
    - Memory Management Unit

Nicht mehr prüfungsrelevant!

Bei Interesse:
Siehe Folien aus dem SS 2008
(Rechnerorganisation 4 SWS)

- **Thread-Level Parallelism**
  - Unterstützung zur parallelen Verarbeitung mehrerer Threads
    - Hyperthreading
      - ein Prozessor, der mit mehreren Registersätzen zwei Threads parallel ausführen kann
    - Mehrprozessorsysteme
      - mehrere Prozessoren
      - Cache-Kohärenz
    - Multi Core Prozessoren
      - mehrere komplette Prozessoren in einem Gehäuse

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Abschließende Bemerkung

David. A. Patterson, John L. Hennessy:

"Acceptance of hardware ideas requires acceptance by software people; therefore hardware people should learn about software. And if software people want good machines, they must learn more about hardware to be able to communicate with and thereby influence hardware engineers."

Copyright © 2019, Prof. Dr. Joachim K. Anlauf, Institut für Informatik VI, Universität Bonn

Weitere wichtige Themen (2)

Nicht vergessen:
Klausur Donnerstag, 18.7.19, 12:30 Uhr, HS 2

Und dann: Schöne Semesterferien!