Superskalare Architekturen (4)

Dynamische Sprungvorhersage mit „Vorgeschichte“

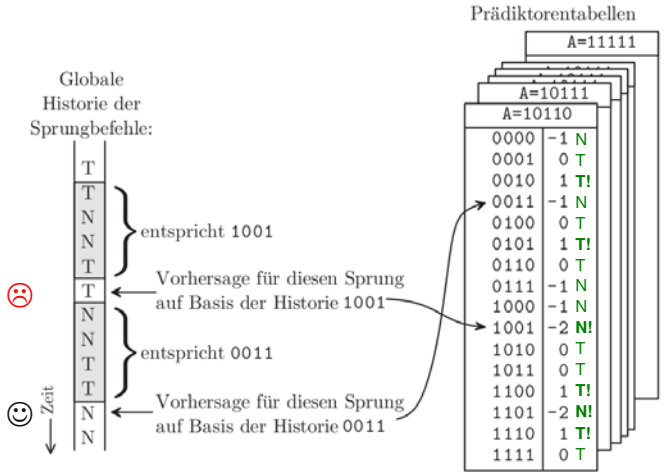
- (hoffentlich wieder kehrende) Muster in Programmausführung erkennen
- exakte Historie der Verzweigungen (Taken / Not Taken) berücksichtigen, speichern in Schieberegister
- Globale / Lokale Historie:
  - **lokal:** für jeden Verzweigungsbefehl eine separate Historie speichern
  - **global:** nur eine einzige Historie; speichert Ergebnisse der letzten ausgeführten Verzweigungen

Globale Historie (1)

- nur **ein** Schieberegister für das ganze Programm (das ist das „globale“)
- aber Prädiktortabellen für verschiedene Adressen (Adress-Hashing)
- So geht's:
  - Blick in Schieberegister
  - Suche der richtigen Prädiktortabelle für akt. Adresse
  - Sch.-Reg.-Inhalt in Prädiktortabelle suchen

```
0000 0000 → 00   0
0000 0100 → 01   1
0000 1000 → 10   2
0000 1100 → 11   3
0001 0000 → 00   0
0001 0100 → 01   1
...
```


Globale Historie (2)



Lokale Historie

- für jede Adresse (genauer: Adress-Hashing):
  - **separates** Schieberegister (das ist das „lokale“)
  - Prädiktortabelle für diese Adresse
- So geht's:
  - abhängig von Adresse: Auswahl des richtigen Schieberegisters
  - Suche der richtigen Prädiktortabelle für akt. Adresse
  - Sch.-Reg.-Inhalt in Prädiktortabelle suchen

Beispiel: 4-stufige Historie, 2-bittige Prädiktoren

Beobachtete Folge	Historie	Prädiktoren NNNN	NNNT	NNTT	NTTN	TTNN	TNNT
1. T	NNNN	00 ☺	00	00	00	00	00
2. T	NNNT	**01** ↗	00 ☺	00	00	00	00
3. N	NNTT	01	**01** ↗	00 ☹	00	00	00
4. N	NTTN	01	01	**10** ↘	00 ☹	00	00
5. T	TTNN	01	01	10	**10** ↘	00 ☺	00
6. T	TNNT	01	01	10	10	**01** ↗	00 ☺
7. N	NNTT	01	01	10 ☺	10	01	**01** ↗
8. N	NTTN	01	01	**11** ↘	10 ☺	01	01
9. T	TTNN	01	01	11	**11** ↘	01 ☺	01
10. T	TNNT	01	01	11	11	01	01 ☺

01: **T!**
00: T
10: N
11: **N!**

Folien 7-15: nur Entwicklung dieser Darstellung

Beispiel: 4-stufige Historie, 2-bittige Prädiktoren

- *simple.mms* mit YesNo = 11001100...
- nur ein Sprungbefehl (eine Tabelle)
- Alle Prädiktoren auf 00 (T) initialisiert
  - „leicht optimistisch“, dass gesprungen wird;
  - Skala: 01 = **T!**, 00 = T, 10 = N, 11 = **N!**
- Historie vor dem Start: NNNN
- Nach jedem Sprung: Prädiktor aktualisieren

Platzbedarf

- Adress-Hashing: a relevante Bits, 2^a unterscheidbare Adressen
- Länge des/der Schieberegister: b
- n-bittige Prädiktoren (2 Bit: 01=**T!**, 00=T, 10=N, 11=**N!**)

Lokale Historie

$$2^a \cdot b + 2^a \cdot 2^b \cdot n$$
$$= 2^a \cdot (b + 2^b \cdot n)$$

Anzahl der Schieberegister

Globale Historie

$$b + 2^a \cdot 2^b \cdot n$$

Anzahl Bits pro Prädiktor
Anzahl möglicher Inhalte eines Schieberegisters
Anzahl Prädiktor-Tabellen
Länge des Schieberegisters

Speicher (1)

Speicher-Organisation

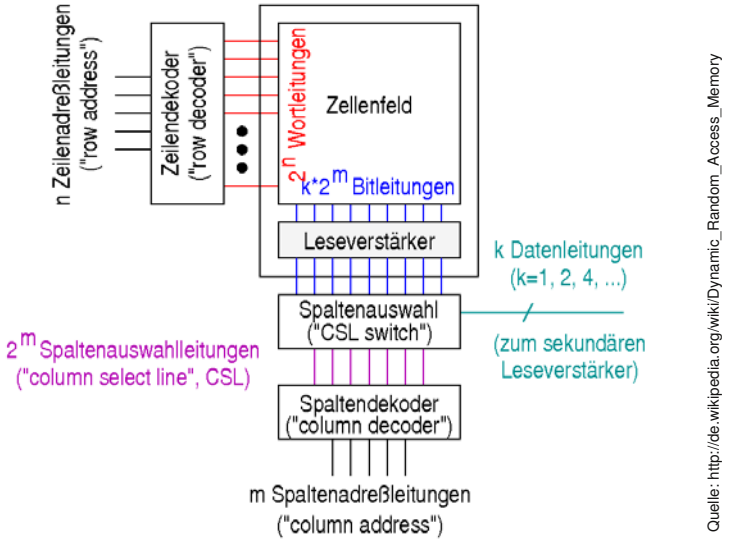
- zweidimensional: „Zeilen“ und „Spalten“
- Einsparen von Anschlussleitungen: Zeilen- und Spaltenadresse nacheinander über die gleichen Leitungen übermitteln
- Aufeinanderfolgende Zugriffe auf die gleiche Zeile (**page**) erfordern kein erneutes Dekodieren / Auslesen dieser Zeile (**page hit**: Seite ist bereits „offen“)
- Bänke: erlauben parallele Zugriffe auf Chip

Speicher-Typen

- SRAM (Static RAM)
- DRAM (Dynamic RAM)
  - SDRAM (Synchronous DRAM)
    - PC66/100/133: 168 Pins, 64 Datenleitungen
    - DDR (Double Data Rate) PC1600/2100/2700: 184 Pins
    - DDR2, DDR3: 240 Pins
    - **S**DRAM **s**ynchron (mit Takt) les-/beschreibbar

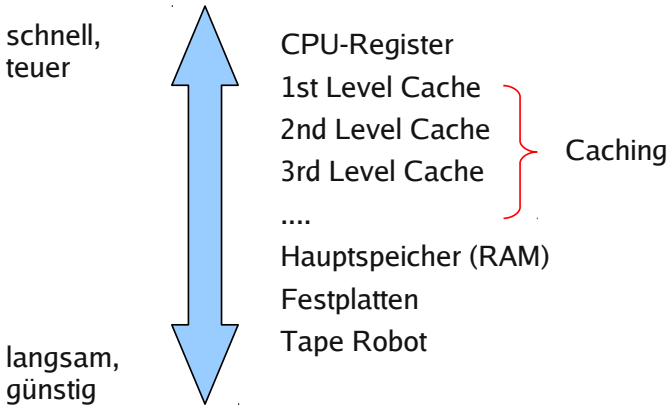
DDR3-RAM-Modul
(Quelle: Wikipedia)

Speicherzugriff



Quelle: http://de.wikipedia.org/wiki/Dynamic_Random_Access_Memory

Speicherhierarchie

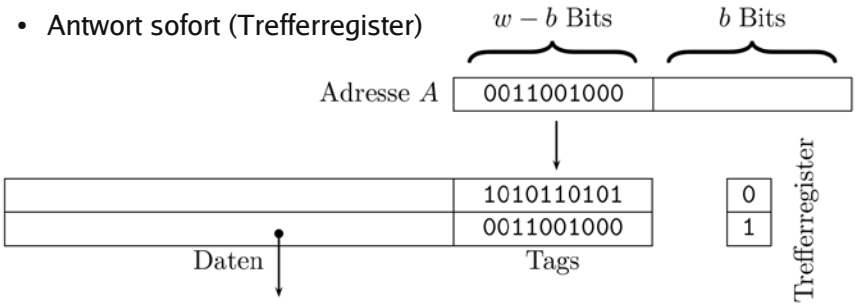


Caching

- Cache: sehr schneller Speicher
- speichert kürzlich verwendete RAM-Inhalte zwischen
- beschleunigt erneuten Zugriff auf dieselbe Speicherzelle
- muss Paare (Adresse, Inhalt) speichern
- Zugriff auf Cache
  - gewünschte Adresse anlegen
  - Antwort sofort (assoziativer Speicher): Inhalt (**cache hit**) oder Fehler (**cache miss**)

Caching

- Cache enthält immer ganze **Cache-Line** (Block von Speicheradressen, die sich nur in den unteren *b* Bits unterscheiden)
- Simultaner Vergleich aller „Tags“ mit oberen $w - b$ Bits der angeforderten Adresse
- Antwort sofort (Trefferregister)

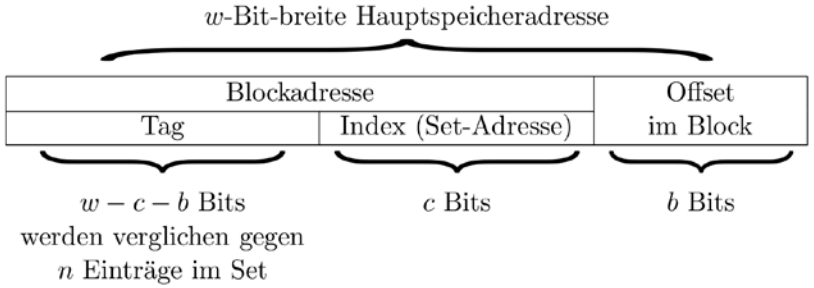


Caching

- Alternativ zu assoziativem Speicher: direkt-abbildender Cache
  - Cache besteht aus *n* Blöcken C[i]
  - RAM „partitioniert“ in *n* gleich große Teile R[i]
  - RAM-Inhalte aus R[i] in C[i] ablegen
- einfacher als assoziativer Speicher, aber schlechtere Ausnutzung des Cache-Speichers

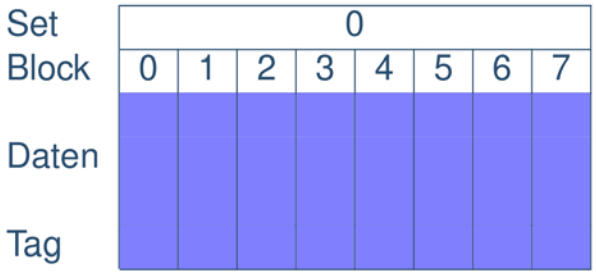
Caching

- Kombination der beiden Caching-Methoden
  - Cache in 2^c sog. **Sets** unterteilt
  - jedes Set besteht aus $n = 2^a$ Blöcken der Größe 2^b
  - Set-Nummer (a-bittig): **Index, Set-Adresse**



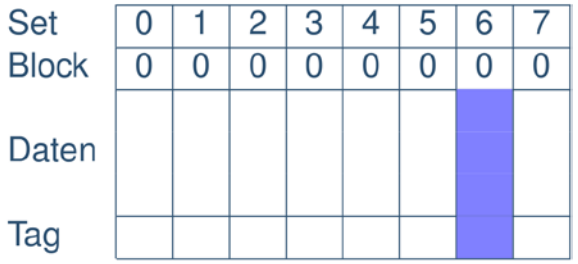
Varianten direkt-abb./assoz. (1)

- voll-assoziativer Cache
  - nur 1 Set ($c = 0$)
  - Block an beliebiger Position speichern



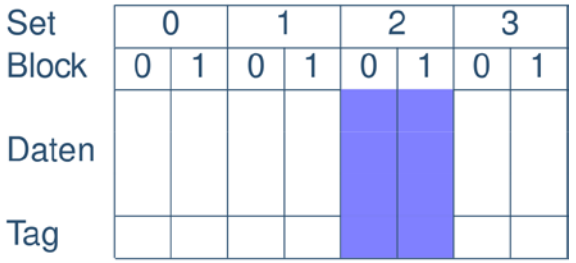
Varianten direkt-abb./assoz. (2)

- Direkt abbildender Cache
  - enthält in jedem Set nur einen Block: $a = 0$
  - Block immer nur an eine feste Position schreibbar



Varianten direkt-abb./assoz. (3)

- n-way associative cache
  - Sets mit je n Blöcken
  - Beispiel $n = 2$: 2-Wege-assoziativer Cache
  - Set durch Adresse festgelegt; in Set freie Wahl



Allgemeines zum Caching

- Bei Speicherzugriffen sucht die CPU zuerst im Cache
- Wird sie dort nicht fündig (cache miss), muss sie auf den Hauptspeicher zugreifen (was deutlich länger dauert)
- **Lokalitätsprinzip:** Zugriffe bleiben häufig „in der Nähe“ von bereits verwendeten Adressen

  → darum auch kleine Caches hilfreich
- kein Platz im Cache? → **Verdrängung**

Vorschau 16.12.2010

- mehr Caching: Verdrängung

Cache: Schreibzugriffe

- **Write through:** Daten werden gleichzeitig in den Cache und die nächst niedrigere Speicherhierarchie geschrieben.
- **Write back:** Daten werden nur in den Cache geschrieben. Entsprechender Block wird dort als „**dirty**“ gekennzeichnet. Rückschreiben erst erforderlich, wenn der Block ersetzt werden soll (hier kann also ein Lesevorgang einen Schreibvorgang auslösen). Spart Speicherbandbreite, ist aber aufwendiger zu realisieren