Rechnerorganisation
Sommersemester 2023 – 11. Vorlesung

Prof. Stefan Roth, Ph.D.

Technische Universität Darmstadt

10. Juli 2023

Inhalt

Organisatorisches

Organisation und Inhalt

- Erinnerung: Evaluation der Veranstaltung Rechnerorganisation
  - Evaluation der Vorlesung & Übung
  - Evaluation Vorlesung und Übung müssen getrennt abgegeben werden
  - TANs (je eine TAN für Vorlesung sowie Übung) gibt es in Moodle
  - Evaluation möglich bis 13. Juli 2023
  - Gerne auch direktes Feedback an mich!
- Deadline für Theorietestat 04 ist heute nacht.
- Diese Woche ist die letzte Vorlesung & Übung.
- Die Tutor:innen bieten wöchentliche Sprechstunden an, in denen Fragen geklärt werden können.

Speicher

Speicherhierarchie – Übersicht

- Bisheriges Modell des Rechnersystems
  - Drei Phasen der Befehlsausführung
  - Prozessor (CPU) mit Registern (r0, ...)
  - Speicher (var1: .word 5)
- In diesem Modell besteht der Speicher[1] als ein lineares Feld aus Bytes
- Die CPU kann auf jede Speicherzelle in konstanter Zeit zugreifen
- Einfaches Modell, welches die Realität moderner Rechnersysteme nicht abbildet.
- In der Praxis wird ein Speichersystem verwendet, welches eine Hierarchie unterschiedlicher Speichertechnologien und -techniken darstellt.

[1] das Speichersystem

Speicherhierarchie – Übersicht

- Das Lesen und das Schreiben in einen Speicher dauert eine gewisse Zeit.
- Im Vergleich zur Taktfrequenz eines Mikroprozessors ist diese Zeit relativ lang.
- Wenn ein Speicher (von Neumann-Architektur) vorhanden ist, müssen sowohl die Befehle, als auch die Daten aus einem Speicher (nacheinander) geholt werden.
- Bei einer Harvard-Architektur mit getrennten Speichern kann sowas ggf. auch parallel ausgeführt werden.
- Außerdem können sogenannte Pipelinestufen eingeführt werden.
- Darstellung der Hierarchie als Pyramide

Speicherhierarchie – Übersicht

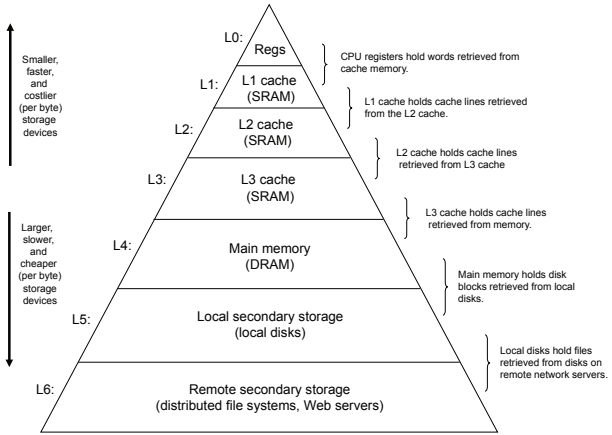


Abbildung: Quelle: [BO10, S. 48]

Speicherhierarchie – Eigenschaften von Speichern

- Kosten und Zugriffszeit
- Geschwindigkeit und Kapazität
- Zugriffsverfahren
- Änderbarkeit der Daten
- Permanenz der Daten

Speicherhierarchie – Kosten und Zugriffszeit

- Die Kosten werden in *$/Bit* oder *$/MByte* gemessen
- Kenngröße von Speichern
  - Zugriffszeit: durchschnittliche Zeit, um ein Wort aus dem Speicher zu lesen
  - Zykluszeit: minimale Zeit zwischen zwei Speicherzugriffen
  - hier spielt neben der Zugriffszeit auch das Busprotokoll eine Rolle
  - Bandbreite (Datenübertragungsrate): maximale Datenmenge, die pro Sekunde übertragen werden kann, gemessen in *Byte/sec*
- Entwicklung der Kosten und Zugriffszeit:
  - generell gilt: geringere Zugriffszeit, höhere Kosten

Speicherhierarchie – Zugriffsverfahren

- Speicher können nach den beiden folgenden Zugriffsverfahren klassifiziert werden:
  - wahlweiser Zugriff (Random Access)
  - serieller Zugriff
- Speicher mit Random Access Zugriff
  - Register
  - Cache-Speicher
  - Hauptspeicher
- Speicher mit seriellem Zugriff
  - Festplatte
  - optische Platten
  - Magnetband

Speichertechnologien – Random-Access Memory (RAM)

- Random-Access Memory (RAM) gibt es in zwei Varianten
  - Statisches RAM (SRAM)
  - Dynamisches RAM (DRAM)
- SRAM ist schneller (und deutlich teurer) als DRAM
- SRAM $\Rightarrow$ Cache
- DRAM $\Rightarrow$ Hauptspeicher
- Typische Verteilung auf einem Rechner
  - 12 MB SRAM
  - 16 GB DRAM

Speichertechnologien – Random-Access Memory (RAM)

- Organisation des Speichers

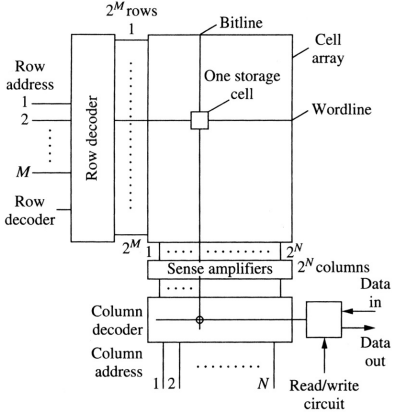


Abbildung: Quelle: [Jae10, S. 418]

Speichertechnologien – Statisches RAM I

- Statisches RAM speichert Informationen so lange, wie die Spannungsversorgung angeschaltet ist. (vgl. [Jae10, S. 417]
- Die gekoppelten Inverter haben zwei stabile Zustände ⇒ Bistabile Schaltung

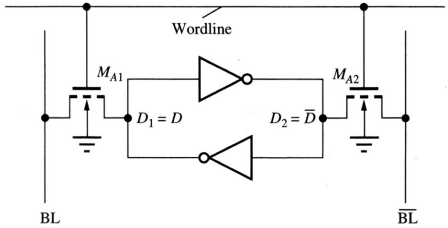


Abbildung: Quelle: [Jae10, S. 422]

- Realisierung der Inverter durch Transistoren

Speichertechnologien – Statisches RAM II

- 6T-Zelle, weil sechs Transistoren

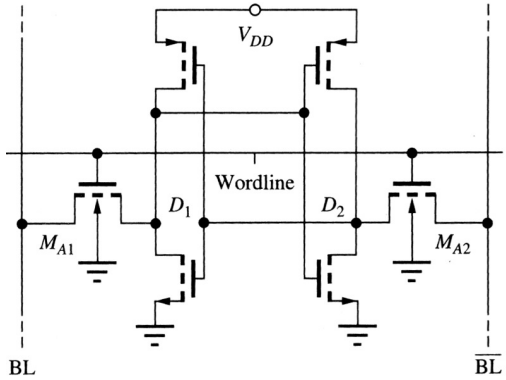


Abbildung: Quelle: [Jae10, S. 422]

Speichertechnologien – Dynamisches RAM I

- Dynamisches RAM speichert die Informationen in einem **Kondensator**

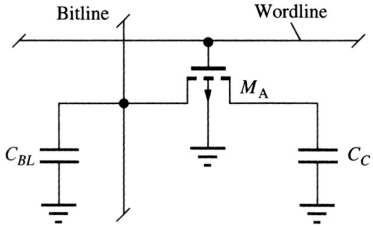
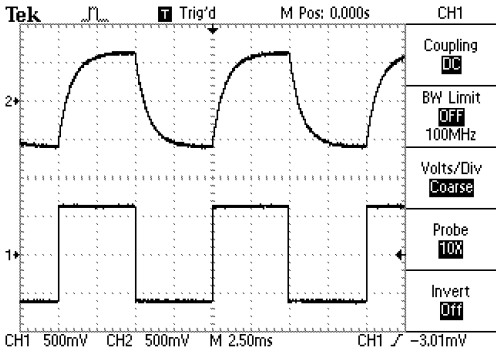


Abbildung: Quelle: [Jae10, S. 430]

- Wird als 1T-Zelle bezeichnet.
- Die Kondensator entlädt sich und verliert damit die Information.
- **Refresh-Zyklus** notwendig, um Informationen aufzufrischen

Speichertechnologien – Dynamisches RAM II

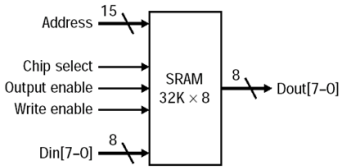
- Lade- und Entladekurve eines **Kondensators**



Speicherorganisation

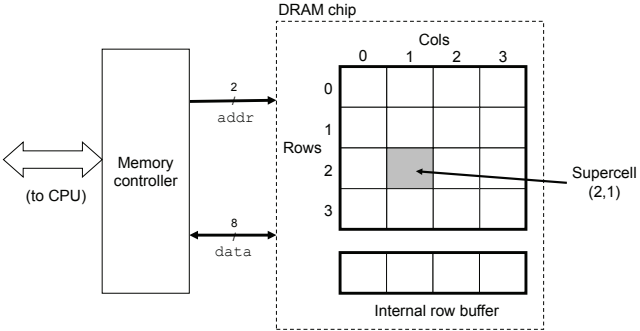
Speicherorganisation

- Eigenschaften eines RAM-Chips sind durch zwei Größen gegeben
  - Anzahl adressierbarer Plätze
  - Breite (in Bit) jedes adressierbaren Platzes
- Beispiel: 256K x 1 SRAM
  - 256K Einträge mit 1 Bit Breite
  - $256K = 2^{18}$, also 18 Adresseingänge sowie ein 1 Bit Dateneingang/-ausgang
- 32K x 8 SRAM
  - 32K Einträge mit 8 Bit Breite
  - $32K = 2^{15}$, also 15 Adresseingänge sowie 8 Bit Dateneingang/-ausgang



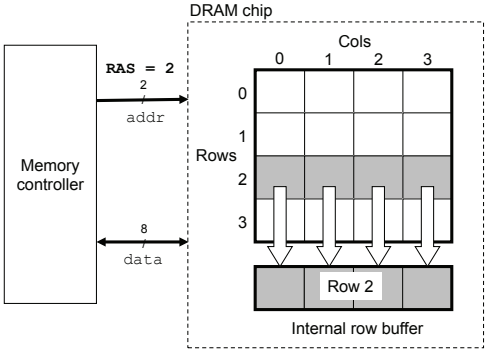
Speicherorganisation

- Abstrakte Darstellung eines 128 Bit 16 x 8 DRAM Chips



Speicherorganisation

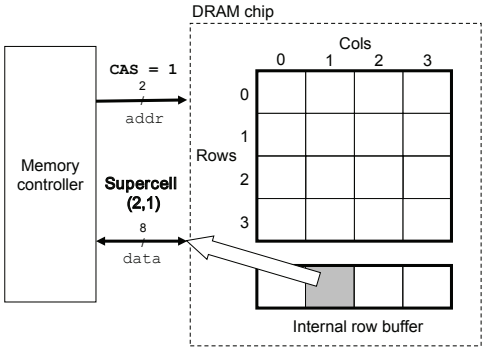
- Lesezugriff RAS[2]



[2]Row Address Strobe

Speicherorganisation

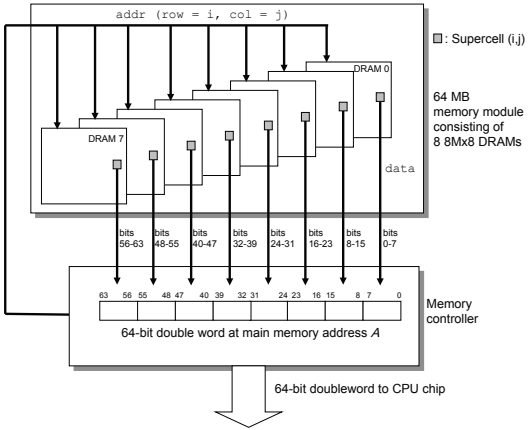
- Lesezugriff CAS[3]



[3]Column Address Strobe

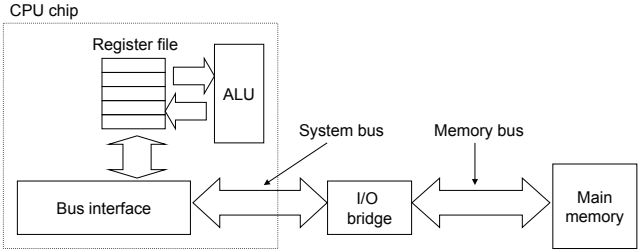
Speicherorganisation

- Verschaltung mehrerer Speichermodule



Speicherorganisation

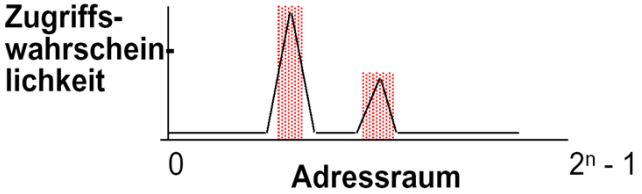
- Verbindung von CPU und Speicher



Lokalität

Lokalität I

- Für eine funktionierende Speicherhierarchie ist **Lokalität** wichtig [BO10, S. 621]
- Das Lokalitätsprinzip stellt fest:
  - eine Eigenschaft bei der Ausführung von Programmen ist, dass meistens nur auf einen relativ geringen Teil des Adressraumes zugriffen wird.



Lokalität II – Formen der Lokalität

- zeitliche (temporale) Lokalität
  - Nach Zugriff auf einen bestimmten Datensatz wird mit großer Wahrscheinlichkeit bald erneut darauf zugegriffen
  - Beispiel: Schleifen
- räumliche Lokalität
  - Nach dem Zugriff auf einen bestimmten Datensatz wird mit großer Wahrscheinlichkeit auch auf einen Datensatz zugegriffen, der in unmittelbarer Nähe im Speicher steht.
  - Beispiel: sequentielle Instruktionsfolgen (ohne Sprünge)
  - Beispiel: Reihungen, Matrizen

Lokalität III – Vorteile

- Gut geschriebene Programme haben eine gute Lokalität
- Die Anwendung des Lokalitätsprinzips hat eine enorme Auswirkung auf die Performanz eines Rechnersystems (Hardware/Software).
- Programme mit guter Lokalität laufen schneller, als Programme mit schlechter Lokalität
- Zweifache Unterscheidung ist möglich:
  - Lokalität der Daten
  - Lokalität der Befehle
- Im Folgenden einige Beispiele für Programme mit guter und schlechter Lokalität

Lokalität IV – Erstes Beispielprogramm

- Berechnen der Summe eines Vektors

```
1  ...
2  int sumvec (int v[N])
3  {
4   int i,sum=0;
5
6   for (i=0 ; i < N ; i++)
7    sum += v[i];
8   return sum;
9  }
10 ...
```

Lokalität V – Erstes Beispielprogramm

- Hat diese Funktion eine gute Lokalität?
- Elemente des Vektors werden sequentiell gelesen
- Beispielhafte Anordnung im Speicher für Vektor v (N=8)

Adresse	0	4	8	12	16	20	24	28
Inhalt	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Zugriffsreihenfolge	1	2	3	4	5	6	7	8

- Lokalität
  - schlechte zeitliche Lokalität
  - gute räumliche Lokalität

Lokalität VI – Zweites Beispielprogramm

- Berechnen der Summe eines Arrays

```
1  ...
2  int sumarrayrows(int a[M][N])
3  {
4      int i,j,sum = 0;
5
6      for (i=0 ;i < M ; i++)
7       for (j=0 ; j < N ; j++)
8        sum += a[i][j];
9      return sum;
10 }
11 ...
```

Lokalität VII – Zweites Beispielprogramm

- Hat diese Funktion eine gute Lokalität?
- Elemente des Vektors werden sequentiell gelesen
- Beispielhafte Anordnung im Speicher für Array a

Adresse	0	4	8	12	16	20
Inhalt	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Zugriffsreihenfolge	1	2	3	4	5	6

- Lokalität
  - schlechte zeitliche Lokalität
  - gute räumliche Lokalität

Lokalität VIII – Drittes Beispielprogramm

- Berechnen der Summe eines Arrays

```
1  ...
2  int sumarrayrows(int a[M][N])
3  {
4      int i,j,sum = 0;
5
6      for (j=0 ; j < N ; j++)
7        for (i=0 ; i < M ; i++)
8          sum += a[i][j];
9      return sum;
10 }
11 ...
```

Lokalität IX – Drittes Beispielprogramm

- Hat diese Funktion eine gute Lokalität?
- Elemente des Vektors werden sequentiell gelesen
- Beispielhafte Anordnung im Speicher für Array a

Adresse	0	4	8	12	16	20
Inhalt	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Zugriffsreihenfolge	1	3	5	2	4	6

- Lokalität
  - schlechte zeitliche Lokalität
  - schlechte räumliche Lokalität

Lokalität X – Lokalität der Befehle

- Wie die Daten sind auch die Befehle im Speicher abgelegt.
- Auch für die Befehle ist also das Lokalitätsprinzip anwendbar
- Für die `for` Schleifen der Beispiele gilt eine gute räumliche Lokalität
- Da der Schleifenkörper wiederholt ausgeführt wird, gibt es auch eine gute zeitliche Lokalität
- Bei der Programmierung an das Lokalitätsprinzip denken.
- Lokalitätsprinzip läßt die Speicherhierarchie funktionieren

Lokalität XI – Viertes Beispielprogramm

- Schreiben in ein Array, zeilenweise

```
1  ...
2  #define DIM 22000
3
4  int main()
5  {
6          long i, j;
7          static long matrix[DIM][DIM];
8
9          for(i= 0; i < DIM; i++)
10                 for(j= 0; j < DIM; j++)
11                         matrix[i][j]= 1;
12         return 0;
13 }
14 ...
```

Lokalität XII – Fünftes Beispielprogramm

- Schreiben in ein Array, spaltenweise

```
1  ...
2  #define DIM 22000
3
4  int main()
5  {
6          long i, j;
7          static long matrix[DIM][DIM];
8
9          for(j= 0; j < DIM; j++)
10                 for(i=0 ; i < DIM; i++)
11                         matrix[i][j]= 1;
12         return 0;
13 }
14 ...
```

Lokalität XIII – Diskussion

- Vergleich zeigt
  - clientssh-arm: DIM 80000
    - Viertes Beispielprogramm: 38.5 s
    - Fünftes Beispielprogramm: 1m48.3 s
  - Raspberry Pi: DIM 10000
    - Viertes Beispielprogramm: 1.5 s
    - Fünftes Beispielprogramm: 13.5 s
- Auch beim Schreiben hat eine zeilenweise Traversierung des Arrays Vorteile.
- Achtung: Verhältnis der Beschleunigung hängt auch von der Problemgröße ab.

Prinzip des Caches

Prinzip des Caches

- Ein Cache ist ein kleiner und schneller Speicher (SRAM)
- Der Prozess, einen Cache zu benutzen wird auch als *caching* bezeichnet
- Zentrale Idee der Speicherhierarchie
- Ein schnellerer und kleinerer Speicher auf dem Level k fungiert als Cache für einen langsameren und größeren Speicher auf dem Level $k + 1$.
- Vergleich mit Speicherhierarchie zeigt, dass das mehrfach angewendet wird
- Deshalb ist das Lokalitätsprinzip für eine sinnvolle Nutzung der Speicherhierarchie notwendig.

Prinzip des Caches

- Grundlegendes Prinzip des cachings der Speicherhierarchie

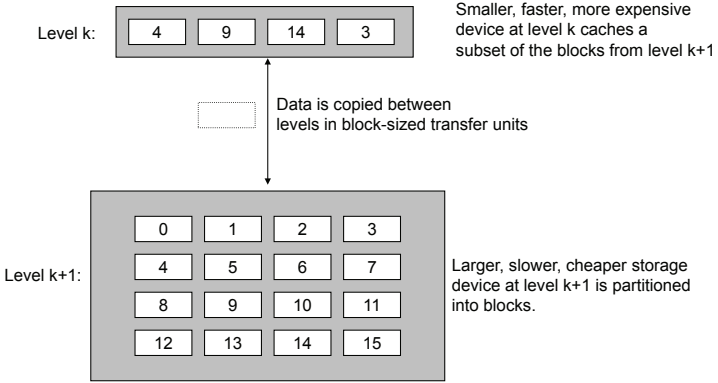


Abbildung: Quelle: [BO10, S. 627]

Prinzip des Caches – Cache Hits

- Wenn ein Programm Daten von einem *Objekt d* braucht, wird geschaut, ob d in einem der Blöcke auf dem Level k gespeichert ist.
- Wenn d in dem Cache auf Level k gefunden wird, nennt man das einen **Cache Hit**
- Das Programm liest dann d direkt aus dem Level k
- Da Level k schneller ist, als Level $k+1$ ergibt sich dadurch ein Geschwindigkeitsvorteil
- Bemerkung: Auch wenn L1, L2 und L3 Cache alle als SRAM ausgeführt sind, gibt es trotzdem Geschwindigkeitsunterschiede

Prinzip des Caches – Cache Miss

- Wenn ein Programm Daten von einem *Objekt d* braucht, wird geschaut, ob d in einem der Blöcke auf dem Level k gespeichert ist.
- Wenn d in dem Cache auf Level k **nicht** gefunden wird, nennt man das einen **Cache Miss**
- Bei einem Cache Miss holt der Cache auf dem Level k den Block mit d von dem Cache auf Level $k + 1$.
- Möglicherweise muss dann ein existierender Block (wenn Cache auf Level k voll ist) überschrieben werden.
- Der Prozess des Überschreibens wird auch als *Ersetzung* bezeichnet
- Die Entscheidung, welcher Block ersetzt wird, kann mit unterschiedlichen Strategien erfolgen
  - Zufallsersetzung
  - Least-recently used (LRU) Ersetzung
- Bedeutung der Lokalität beachten

Prinzip des Caches – Speicherhierarchie

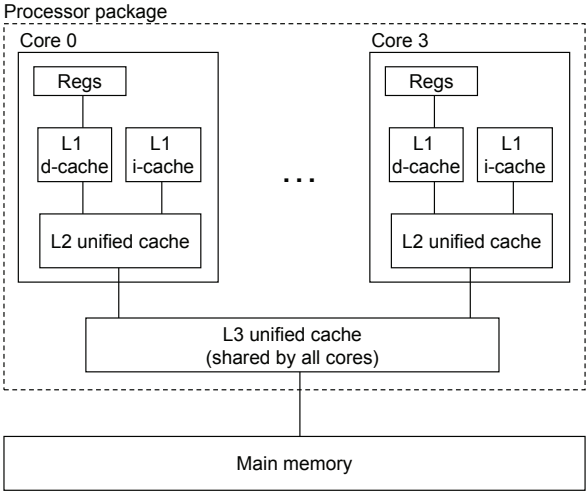
- Caching in modernen Rechnersystemen

Typ	What cached	Latenz (cycles)
CPU Register	4 Byte oder 8 Byte	0
L1 Cache	64 Byte Block	1
L2 Cache	64 Byte Block	10
L3 Cache	64 Byte Block	30
...	...	...
Platten Cache	Disk Sektors	100000

- Die Blöcke werden größer
- Die Verzögerung (Latenz) wird größer

Cachehierarchie ARM®/Intel Core i7

Cachehierarchie ARM®/Intel Core i7 – Quelle: [BO10, S. 647]



Cachehierarchie ARM® A53

- Getrennte Caches für Daten (d-cache) und Instruktionen (i-cache) auf Level 1
- ARM® Cortex®-A53 MPCore Processor, Technical Reference Manual

Cachehierarchie Intel Core i7

- Getrennte Caches für Daten (d-cache) und Instruktionen (i-cache) auf Level 1
- Charakteristische Werte [BO10, S. 647]

Cache Typ	Zugriffszeit (cycles)	Cache Größe
L1 i-Cache	4	32 KB
L1 d-Cache	4	32 KB
L2 unified cache	11	256 KB
L3 unified cache	30–40	8 MB

- Alle SRAM Cache Speicher sind auf dem CPU Chip (on die).

Cache Speicher – Anordnung und Organisation

- Anordnung des Caches

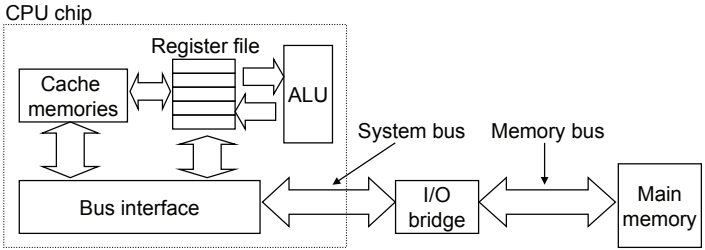


Abbildung: Quelle: [BO10, S. 630]

- Die Organisation eines Cache Speichers kann auf mehrere Arten erfolgen
- Direkt-abbildender Cache, einfachste Form (vgl. Betriebssystemvorlesung)

Klausurvorbereitung

Klausurvorbereitung I

- Häufige Frage: was ist für die Klausur relevant?
- Antwort 1
  - Vorlesungen 1 – 11
  - Übungen 1 – 11
  - Theorie- und Programmiertestate!
- Antwort 2
  - Das (Grund-)Verständnis des Faches Rechnerorganisation

Klausurvorbereitung II

- Das (Grund-)Verständnis des Faches Rechnerorganisation
  - Begrifflichkeiten – die Sprache der Informatik verinnerlichen und anwenden (nicht auswendig lernen)
  - Konzepte der Assemblerprogrammierung
  - Funktionsweise der Mikroarchitektur
- Und wie bereitet man sich darauf vor?
  - Studium der Vorlesung und der Übungen
  - Praktische Programmierung am System
- Ist das Anschauen von alten Klausuren wichtig?
  - Nein!
  - Man ist auf die Fachprüfung sehr gut vorbereitet, wenn man Vorlesung und Übung angeschaut und bearbeitet hat.
- Wie stellen in der kommenden Woche noch einige Beispielaufgaben in Moodle bereit, um ein Gefühl für die Klausur zu geben.

Zusammenfassung und Ausblick

Zusammenfassung und Ausblick

Zusammenfassung

- Speicher

Ausblick

- Schönen Sommer!
- Viel Erfolg bei der Fachprüfung Rechnerorganisation

Lernkontrolle

- Ich habe die Idee der Speicherhierachie nachvollziehen können ☑
- Lokalität und deren Auswirkung auf Programmlaufzeit habe ich nachvollziehen können ☑
- Bei der Nutzung von Matrizen werde ich stets die „richtige“ Traversierung vornehmen ☑
- ...

Literatur

Literatur

[BO10] Bryant, Randal E. und David R. O´Hallaron: *Computer Systems - A Programmer´s Perspective*.
Prentice Hall, 2010.

[HH16] Harris, David Money und Sarah L. Harris: *Digital Design and Computer Architecture, ARM® Edition*.
Morgan Kaufmann, 2016.

[Jae10] Jaeger, Richard C.: *Microelectronic Circuit Design*.
McGRAW-Hill, 2010.