Rechnerorganisation – Kapitel 8

Prof. Sarah Harris
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

TECHNISCHE UNIVERSITÄT DARMSTADT

SoSe 2016

Kapitel 8: Themen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einleitung
- Leistungsvergleich von Speichersystemen
- Caches
- Virtueller Speicher
- Speichereinblendung von Ein-/Ausgabegeräten
- Zusammenfassung

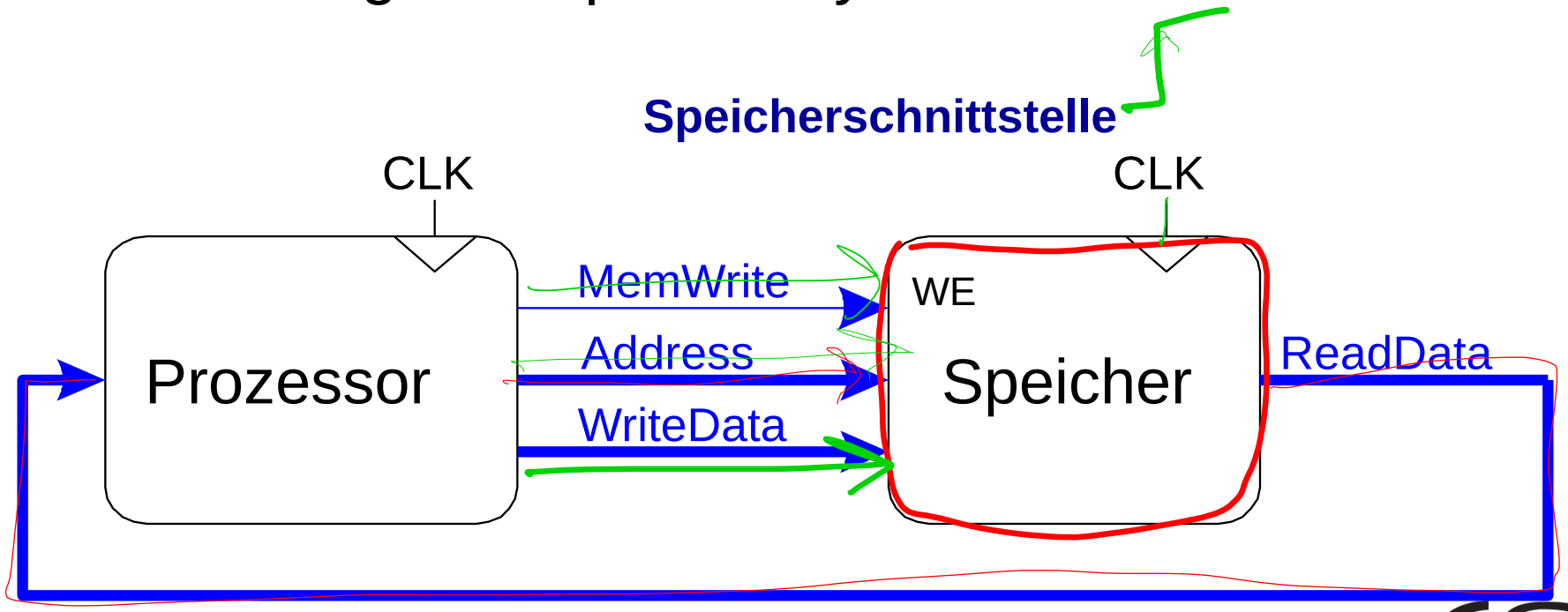
Embedded Systems & Applications

Einleitung

TECHNISCHE UNIVERSITÄT DARMSTADT

Rechenleistung hängt ab von:

- Prozessorleistung
- Leistung des Speichersystems

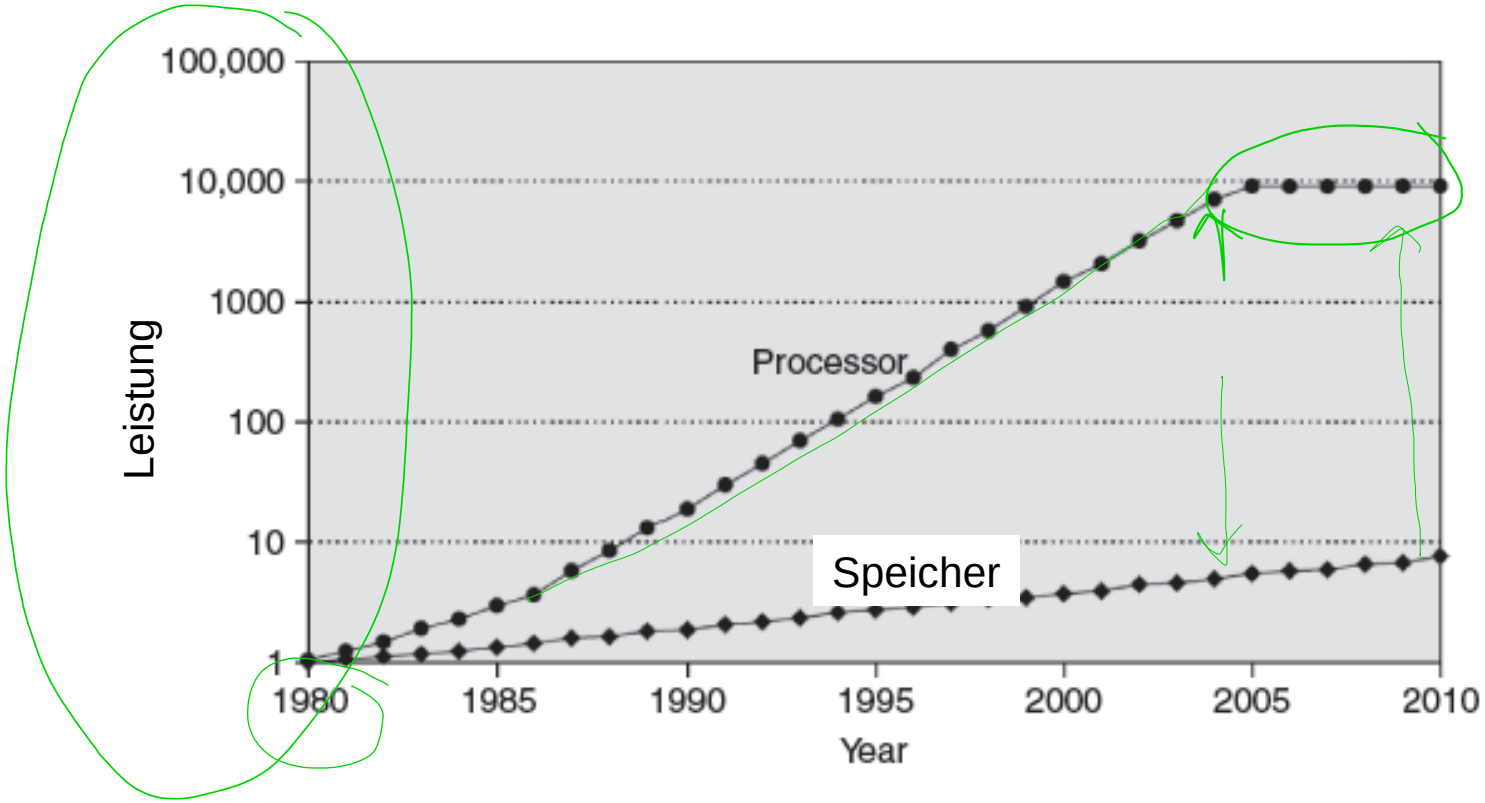


Embedded Systems & Applications

Prozessor/Speicher Leistung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Annahme bisher in der Vorlesung: Speicherzugriffe dauern 1 Takt
- Ist aber seit den 1980‘er Jahren nicht mehr wahr



Embedded Systems & Applications

Herausforderungen beim Entwurf von Speichersystemen

TECHNISCHE UNIVERSITÄT DARMSTADT

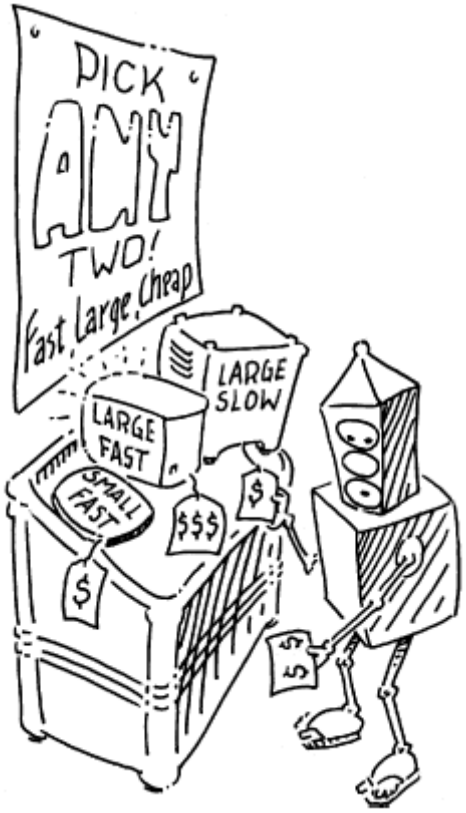
Speichersystem soll so schnell sein wie Prozessor

- Zumindest dem Anschein nach

Idealer Speicher:

- Schnell
- Billig
- Hohes Fassungsvermögen

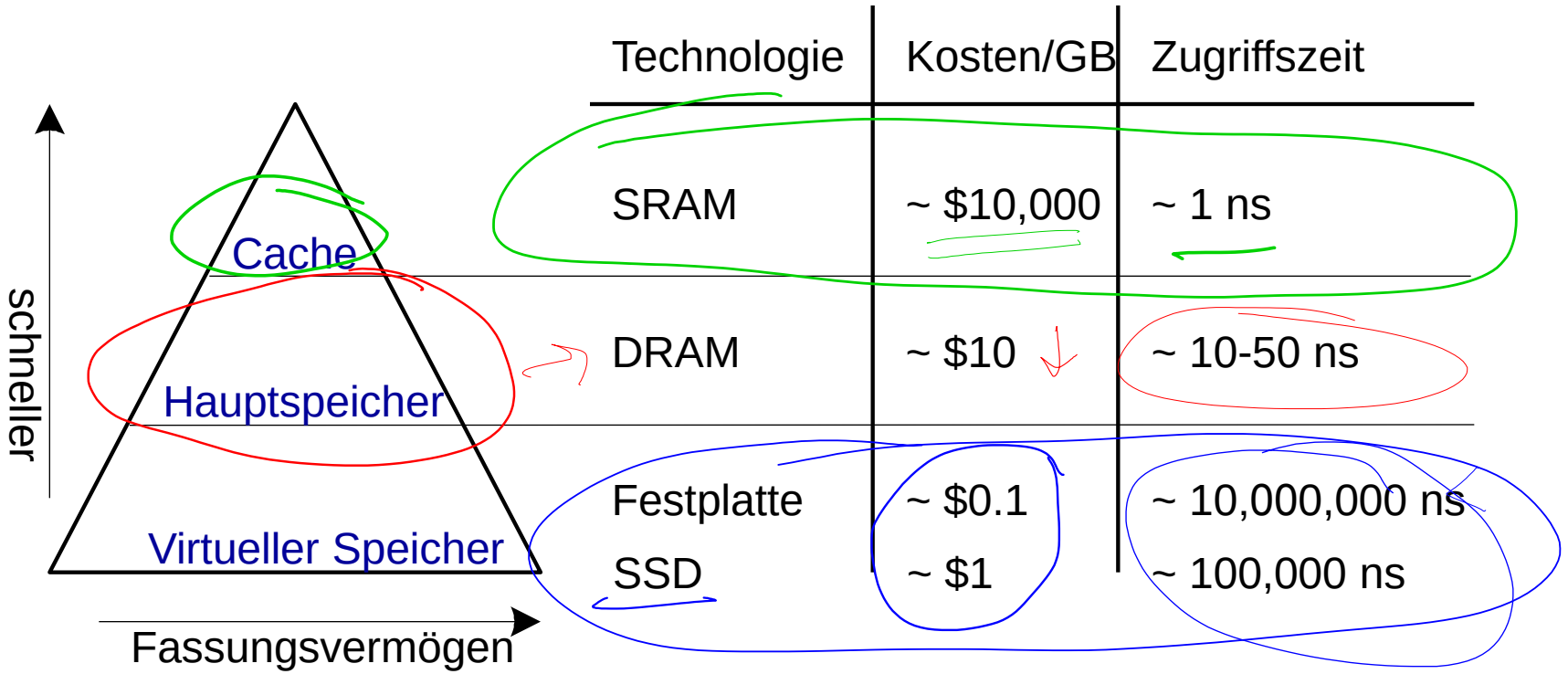
Praktisch: Nur zwei von drei Eigenschaften realisierbar!



Verwende Hierarchie von Speichern

Embedded Systems & Applications

Speicherhierarchie: Beispiele für Ebenen



Lokalität

TECHNISCHE UNIVERSITÄT DARMSTADT

Nutze Lokalität zur Beschleunigung von Speicherzugriffen aus

Zeitliche Lokalität:

- Ein gerade benutztes Datum wird wahrscheinlich bald wieder gebraucht
- **Ausnutzung:** gerade benutzte Daten in höheren Ebenen der Speicherhierarchie halten

Räumliche Lokalität:

- Um ein benutztes Datum herum liegende Daten werden wahrscheinlich auch bald gebraucht
- **Ausnutzung:** Beim Zugriff auf ein Datum auch benachbarte Daten in höhere Ebenen der Speicherhierarchie bringen

Leistung eines Speichersystems

TECHNISCHE UNIVERSITÄT DARMSTADT

Treffer (*hit*): Datum wird auf dieser Ebene der Speicherhierachie gefunden

Verfehlt (*miss*): nicht gefunden (suche auf tieferer Ebene)

Hit-Rate = # hits / # Speicherzugriffe
= 1 – Miss Rate

Miss-Rate (*MR*) = # misses / # Speicherzugriffe
= 1 – Hit Rate

Durchschnittliche Speicherzugriffszeit (*average memory access time, AMAT*): Durchschnittliche Zeit, die der Prozessor braucht, um auf ein Datum zuzugreifen

$$\textbf{AMAT} = t_{\text{cache}} + MR_{\text{cache}} (t_{MM} + MR_{MM}\, t_{VM})$$

MM = main memory, Hauptspeicher
VM = virtual memory, Virtueller Speicher

Embedded Systems & Applications

Beispiel 1: Leistung eines Speichersystems

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein Programm hat 2000 Load- und Store-Befehle
- 1250 der Daten werden im Cache vorgefunden
- Der Rest kommt aus anderen Ebenen der Speicherhierarchie

- **Was sind die Hit- und Miss-Rates des Caches?**

Hit Rate = $\frac{1250}{2000}$.

Miss Rate = $\frac{750}{2000}$ = 1 − Hit Rate

Embedded Systems & Applications

Beispiel 1: Leistung eines Speichersystems

- Ein Programm hat 2000 Load- und Store-Befehle
- 1250 der Daten werden im Cache vorgefunden
- Der Rest kommt aus anderen Ebenen der Speicherhierarchie

- **Was sind die Hit- und Miss-Rates des Caches?**

Hit Rate = 1250/2000 **= 0,625**

Miss Rate = 750/2000 = **0,375** = 1 – Hit Rate

Beispiel 2: Leistung eines Speichersystems

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: Prozessor hat zwei Hierarchieebenen:

- Cache
- Hauptspeicher (Main Memory)

t_{cache} = 1 Takt

t_{MM} = 100 Takte

Wie lang ist die AMAT des Programmes aus Beispiel 1?

AMAT = 1 + 0,375 (100)

Embedded Systems & Applications

Beispiel 2: Leistung eines Speichersystems

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: Prozessor hat zwei Hierarchieebenen:

- Cache
- Hauptspeicher (Main Memory)

t_{cache} = 1 Takt

t_{MM} = 100 Takte

Wie lang ist die AMAT des Programmes aus Beispiel 1?

$$\textbf{AMAT} = t_{cache} + MR_{cache}\, t_{MM}$$

$$= [1 + 0{,}375 * 100] \text{ Takte}$$

$$= \textbf{38,5 Takte}$$

Embedded Systems & Applications

Gene Amdahl, 1922 -

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Amdahls Gesetz:** Die Optimierung eines Subsystems bringt nur dann etwas, wenn das Subsystem tatsächlich großen Einfluss auf die Gesamtrechenleistung hat
- Gründete drei Firmen, darunter auch die Amdahl Corporation in 1970
  - IBM-kompatible Großrechner
    - I.d.R. schneller und/oder billiger

Embedded Systems & Applications

Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

„Verstecktes Lager“

- Höchste Ebene der Speicherhierarchie
- Schnell (oft Zugriffszeit ~ 1 Takt)
- Stellt dem Prozessor idealerweise die meisten benötigten Daten zur Verfügung
- Speichert (i.d.R.) die zuletzt benutzten Daten

Embedded Systems & Applications

Aufbau von Caches: Entwurfsentscheidungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Daten werden im Cache gehalten?
- Wie werden die Daten gefunden?
- Wie werden Daten ersetzt?

- Schwerpunkt hier auf Loads
- Stores werden aber ähnlich gehandhabt

Embedded Systems & Applications

Welche Daten werden im Cache gehalten?

Welche Daten werden im Cache gehalten?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Idealfall: Cache "ahnt" im Voraus, welche Daten der Prozessor benötigen wird und hält diese bereit
- Praxis: Prophezeiungen in der Regel ungenau
- Basiere Vorhersagen auf bisherigem Verhalten
  - **Zeitliche Lokalität**: kopiere gerade benutzte Daten in den Cache. Bei nächster Verwendung werden die Daten im Cache gefunden (*Cache Hit*).
  - **Räumliche Lokalität**: kopiere benachbarte Daten auch in den Cache
    - Blockgröße: Anzahl von Bytes, die immer zusammen in den Cache kopiert werden

Aufbau von Caches: Entwurfsentscheidungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Daten werden im Cache gehalten?
- Wie werden die Daten gefunden?
- Wie werden Daten ersetzt?

Embedded Systems & Applications

Welche Daten werden im Cache gehalten?

TECHNISCHE UNIVERSITÄT DARMSTADT

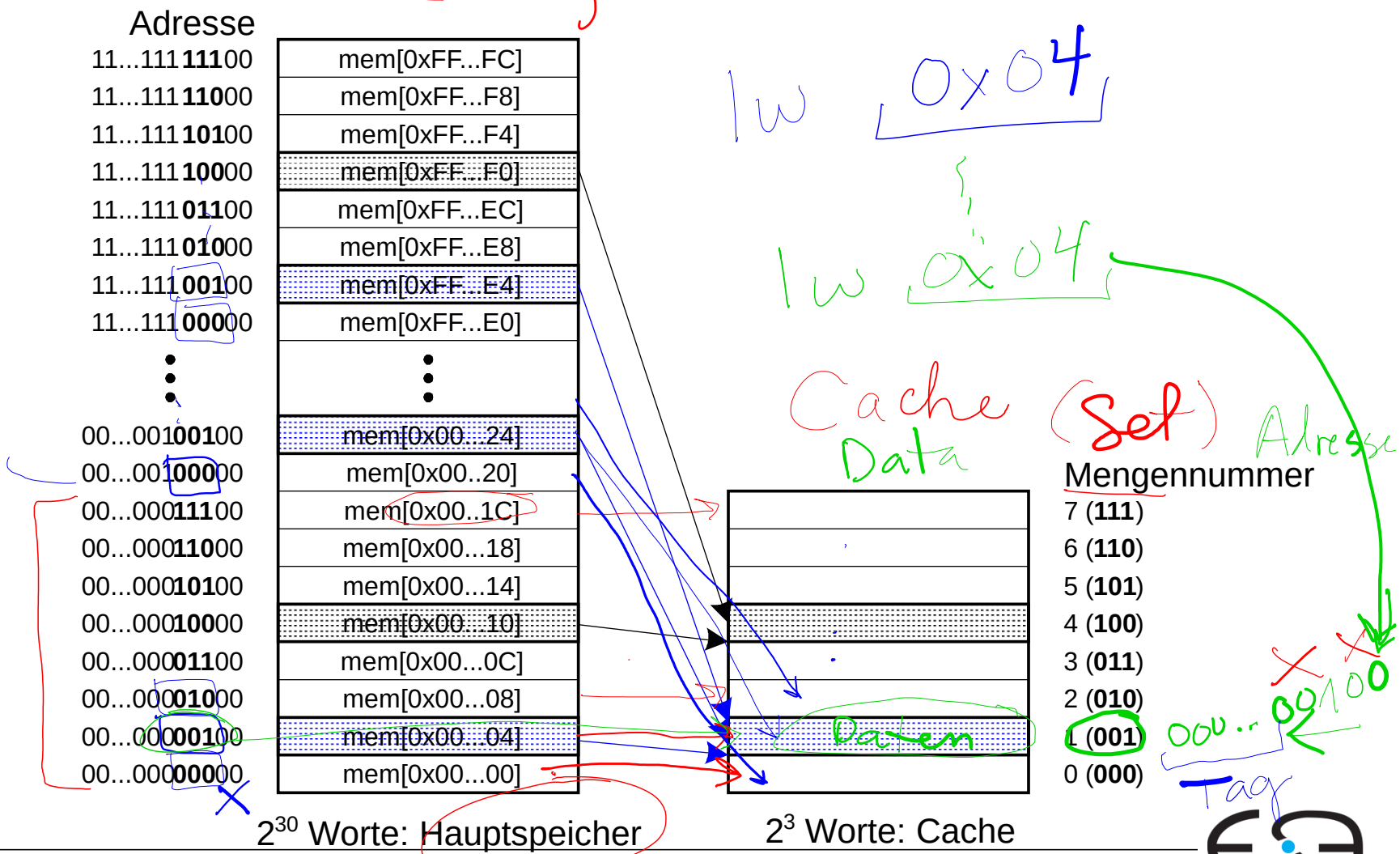
Annahme:

Wir speichern nur 8 Wörter im Cache

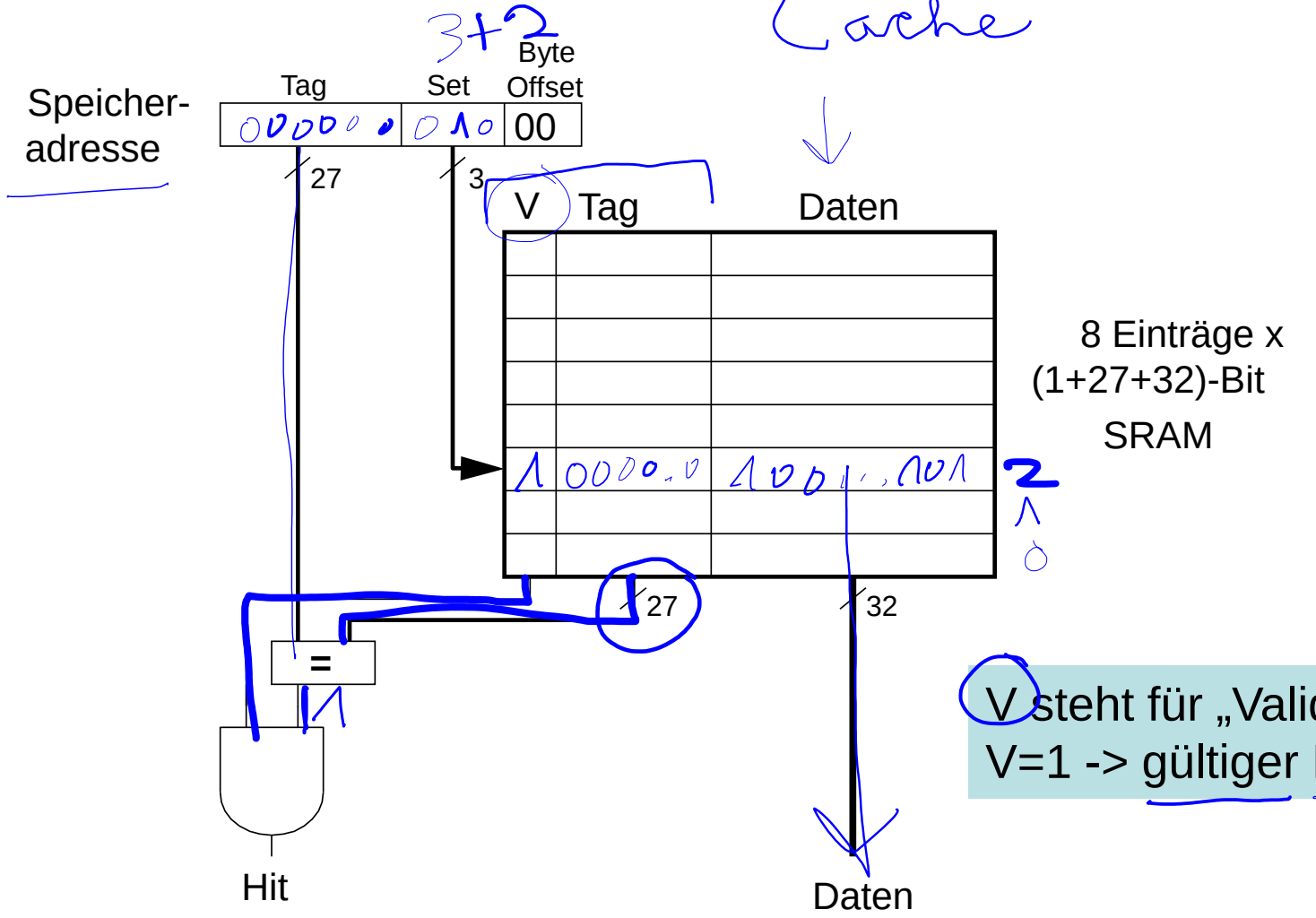
(**Kapazität** = 8 Wörter oder 32 Bytes)

Embedded Systems & Applications

Direktabgebildeter Cache

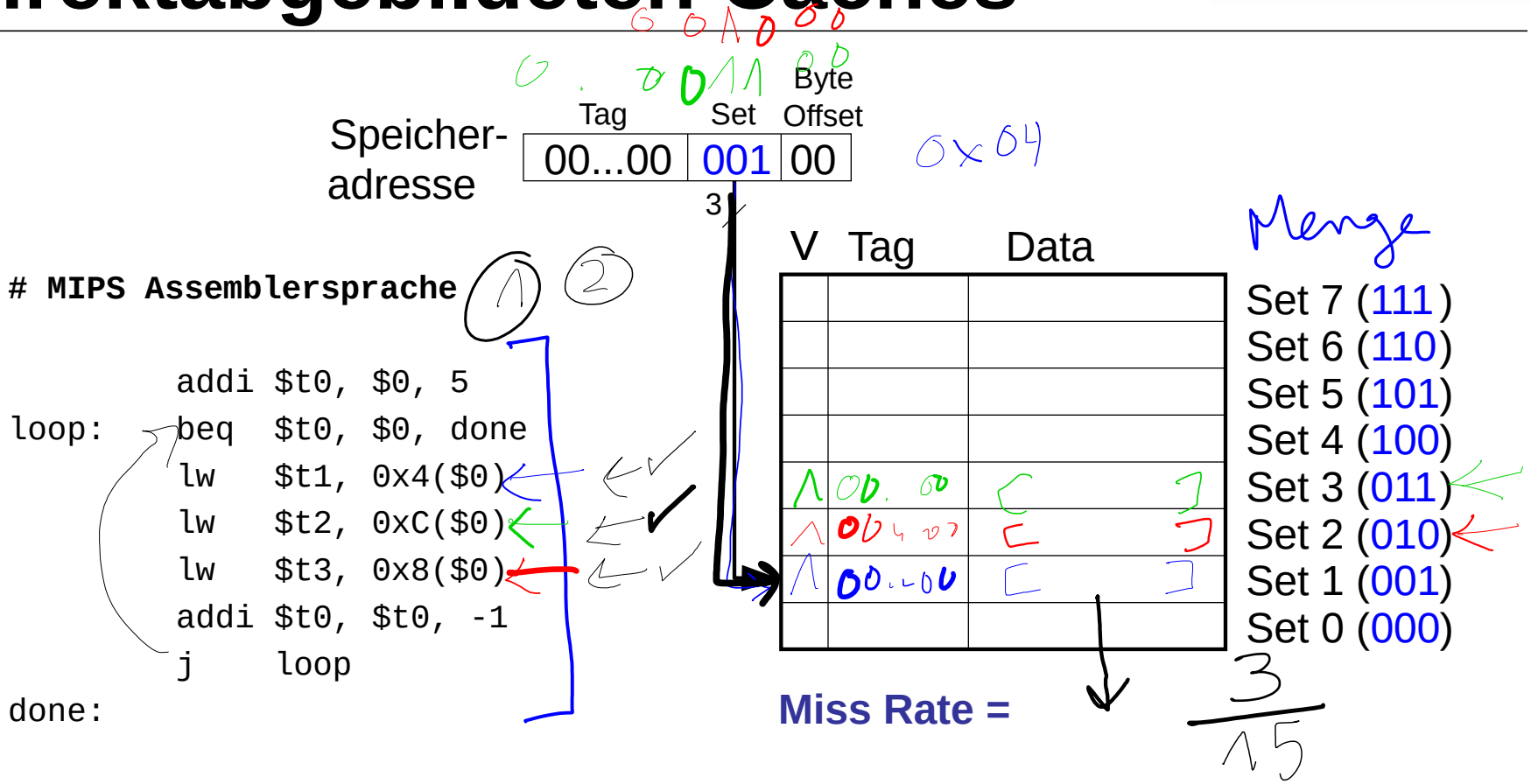


Direktabgebildeter Cache: Hardware



V steht für „Valid“
V=1 -> gültiger Eintrag

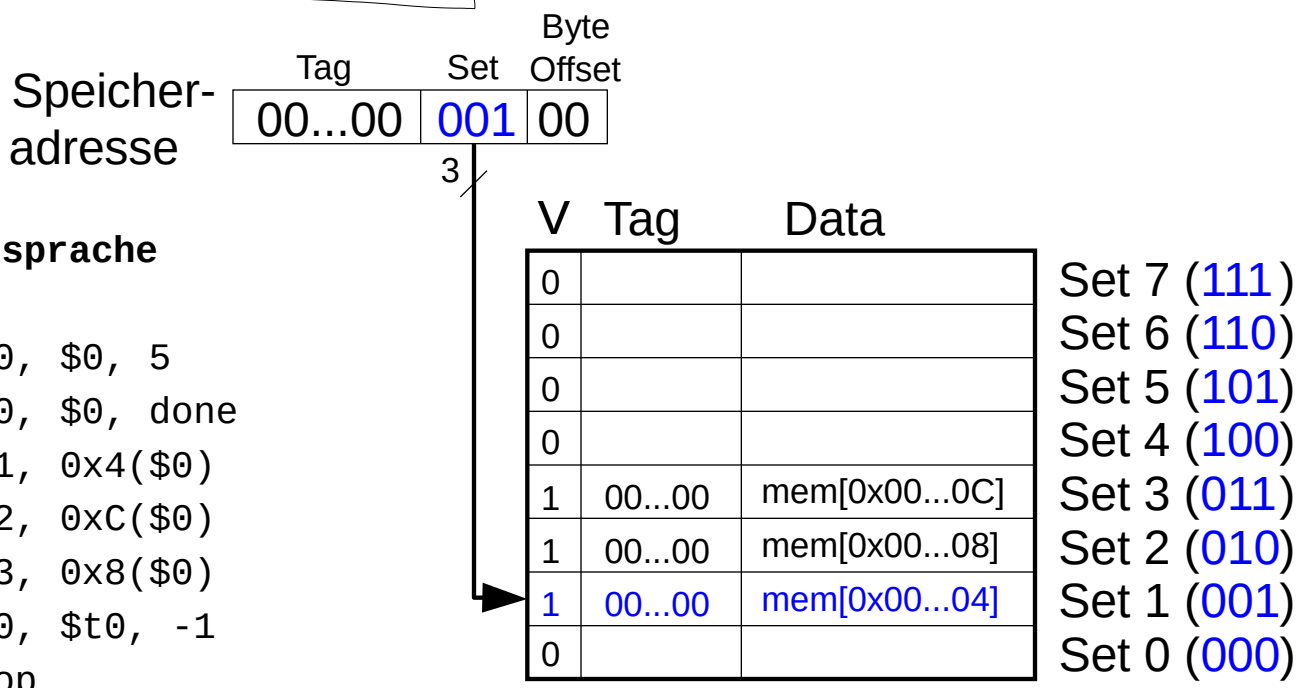
Leistung eines direktabgebildeten Caches



Leistung eines direktabgebildeten Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# MIPS Assemblersprache

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Miss Rate = 3 / 15

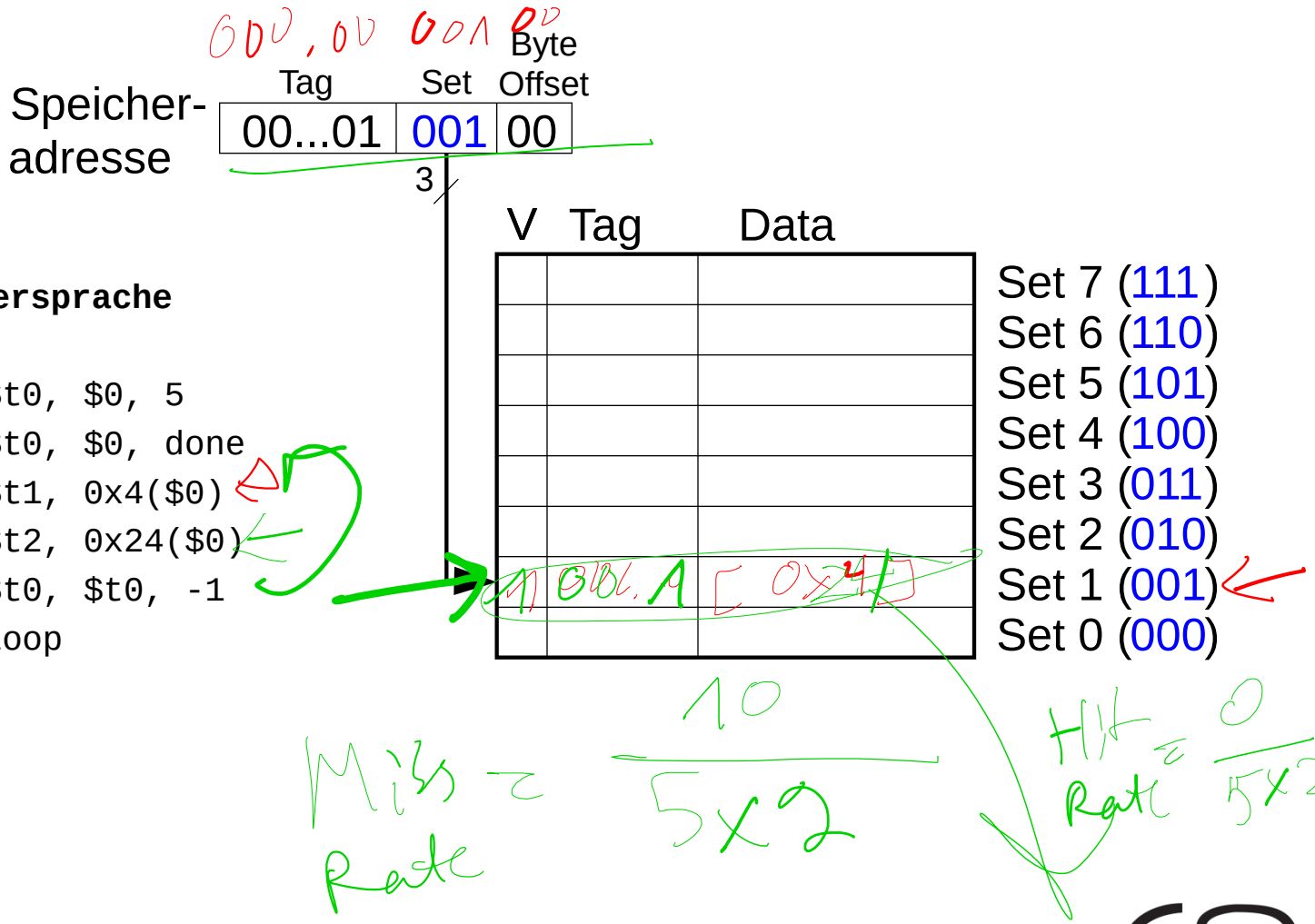
= 20%

Temporale Lokalität

unvermeidbare (*compulsory*) Misses

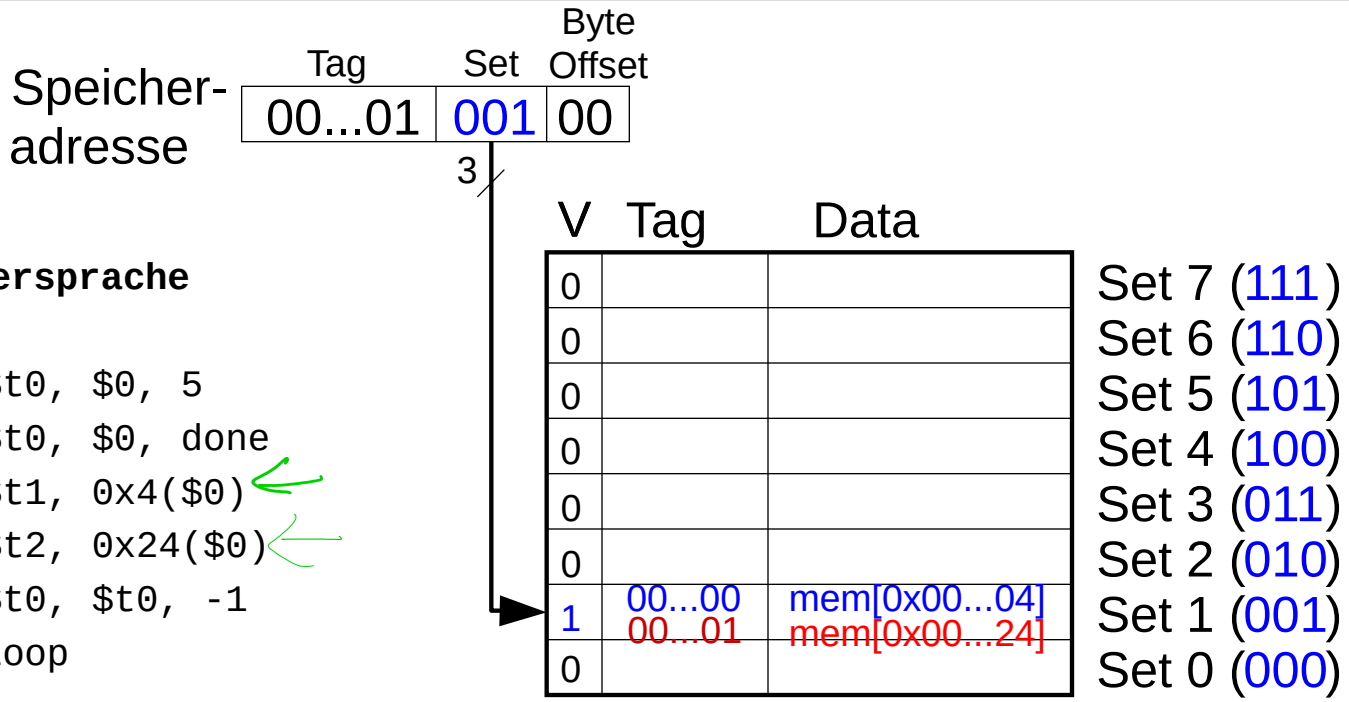
Embedded Systems & Applications

Konflikte bei direktabgebildeten Caches




```
# MIPS Assemblersprache

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Konflikte bei direktabgebildeten Caches



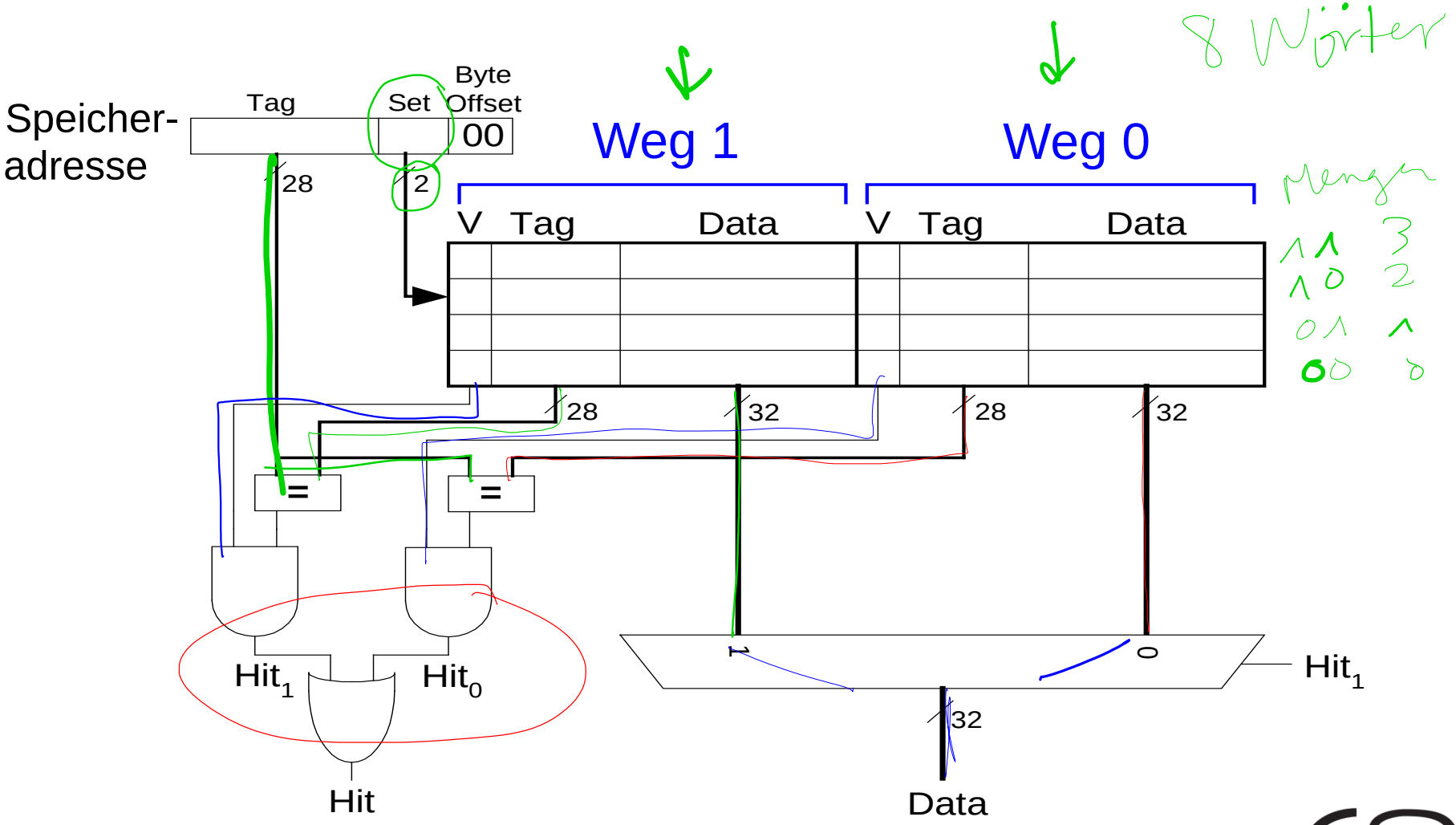

```
# MIPS Assemblersprache

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 10/10

= 100%

Konflikt (Conflict) Misses

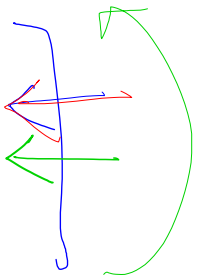
N-Wege Assoziativer Cache

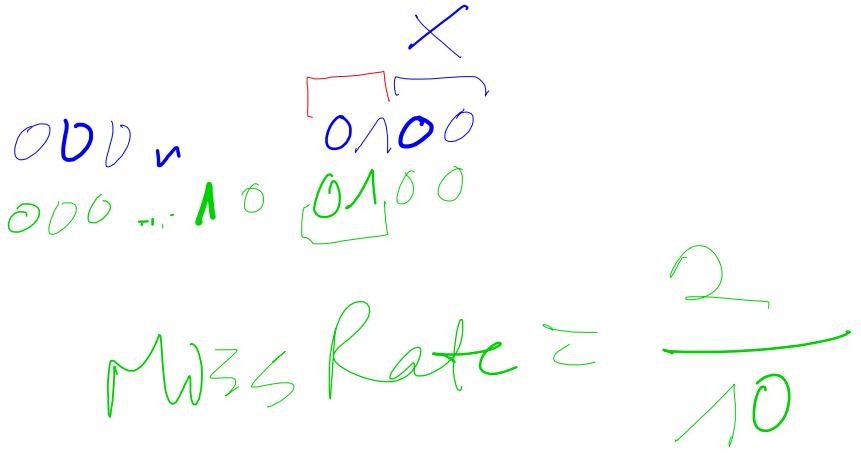


Leistung eines *N*-Wege assoziativen Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# MIPS Assemblersprache

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```





Weg 1			Weg 0		
V	Tag	Data	V	Tag	Data
1	000...10	[0x24]	1	00...0	[0x4]

1 ✓

Embedded Systems & Applications

Leistung eines *N*-Wege assoziativen Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# MIPS Assemblersprache

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 2/10

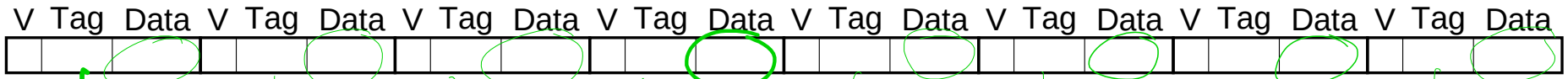
= 20%

Assoziativität reduziert Konflikt-Misses

	Weg 1			Weg 0		
	V	Tag	Data	V	Tag	Data
Set 3	0			0		
Set 2	0			0		
Set 1	1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]
Set 0	0			0		

Embedded Systems & Applications

Vollassoziativer Cache



- Keine Konflikt-Misses
- Kostet viel Hardware

Zeitplan für den Rest des Semesters

TECHNISCHE UNIVERSITÄT DARMSTADT

- **20.06:** - Caches, Virtueller Speicher
- **27.06:** - Evaluation des Kurses (am Anfang der Vorlesung)
  - Ein-/Ausgabegeräten
  - Zusammenfassung
- **04.07:** - Probefragen für die Klausur
  - sollten davor **von selber** durchgearbeitet
- **11.07:** - Vortrag auf Englisch (wird nicht auf der Klausur stehen):
  - Weiterführende Themen der Mikroarchitektur
  - ARM Architektur
- **18.07:** - Klausur, 10 Uhr – 11:30 Uhr
  - wo die stattfindet, wird in der Woche davor durch Moodle bekanntgegeben

Die Sprechstunde von Prof. Harris (Mi., 13:30 – 14:30 Uhr) fällt diese Woche (22.06.) aus.

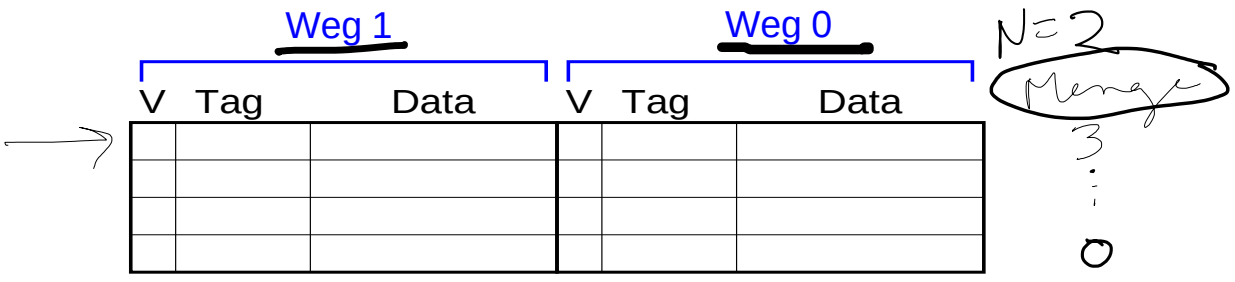
Wie werden Daten gefunden?

TECHNISCHE UNIVERSITÄT DARMSTADT

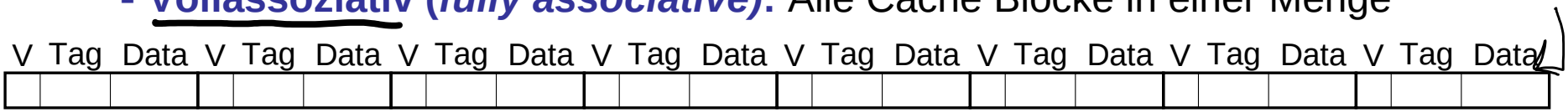
Caches werden klassifiziert nach Anzahl von Blöcken in einer Menge:

- **Direktabgebildet (*direct mapped*):** Ein Block pro Menge

- ***N*-Wege Mengenassoziativ (*N-way set associative*):** *N* Blöcke pro Menge



- **Vollassoziativ (*fully associative*):** Alle Cache Blöcke in einer Menge



Jede Speicheradresse wird genau auf **eine** Menge abgebildet

Embedded Systems & Applications

Caches: Begriffe

TECHNISCHE UNIVERSITÄT DARMSTADT

Kapazität (*capacity, C*):

- Anzahl der im Cache speicherbaren Bytes

Blockgröße (*block size, b*):

- Anzahl der auf einen Satz in den Cache geladenen Bytes

Blockanzahl (*B = C/b*):

- Anzahl von Blöcken im Cache: *B = C/b*

Assoziativitätsgrad (*degree of associativity, N*):

- Anzahl von Blöcken in einer Assoziativitätsmenge (kurz: *Menge)*

Anzahl von Mengen (*S = B/N*):

- Jede Speicheradresse wird auf genau eine Menge abgebildet

Embedded Systems & Applications

Welche Daten werden im Cache gehalten?

TECHNISCHE UNIVERSITÄT DARMSTADT

Basiere Vorhersagen auf bisherigem Verhalten

- **Zeitliche Lokalität**: kopiere gerade benutzte Daten in den Cache. Bei nächster Verwendung werden die Daten im Cache gefunden (*Cache Hit*).
- **Räumliche Lokalität**: kopiere benachbarte Daten auch in den Cache
  - Blockgröße: Anzahl von Bytes, die immer zusammen in den Cache kopiert werden

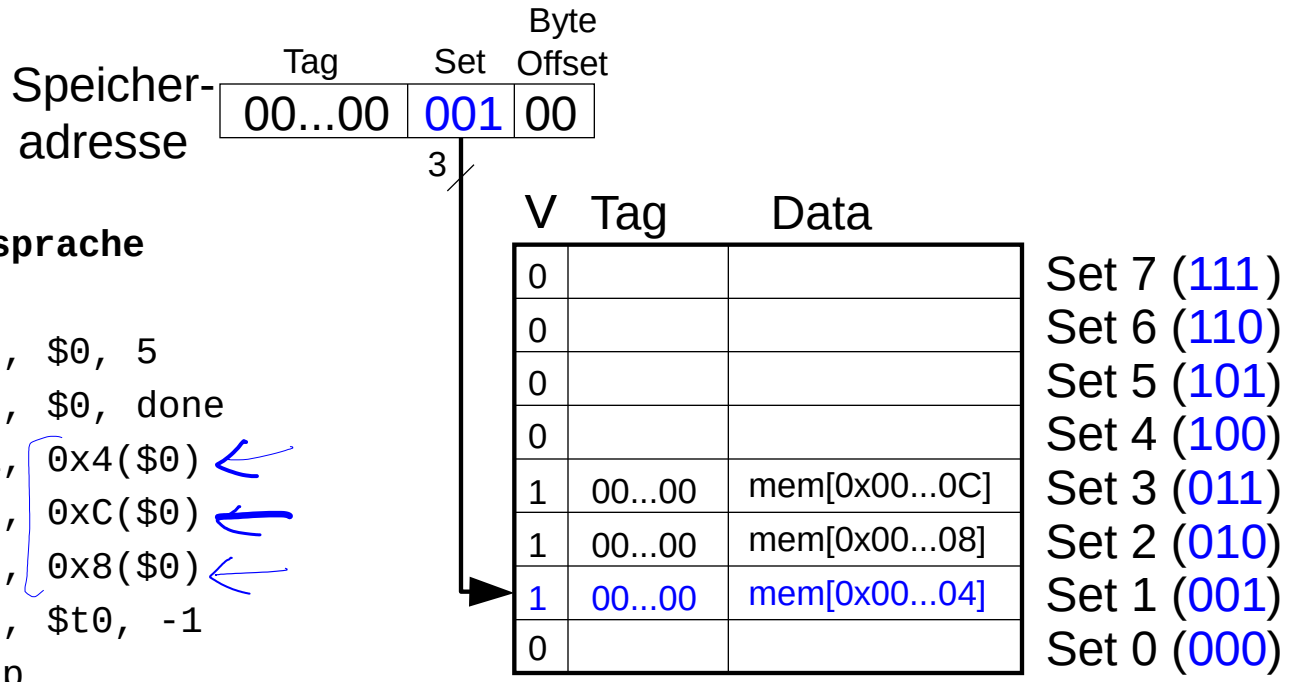
jetzt

Embedded Systems & Applications

Wiederholung: Leistung eines direktabgebildeten Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# MIPS Assemblersprache

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Miss Rate = 3 / 15

= 20%

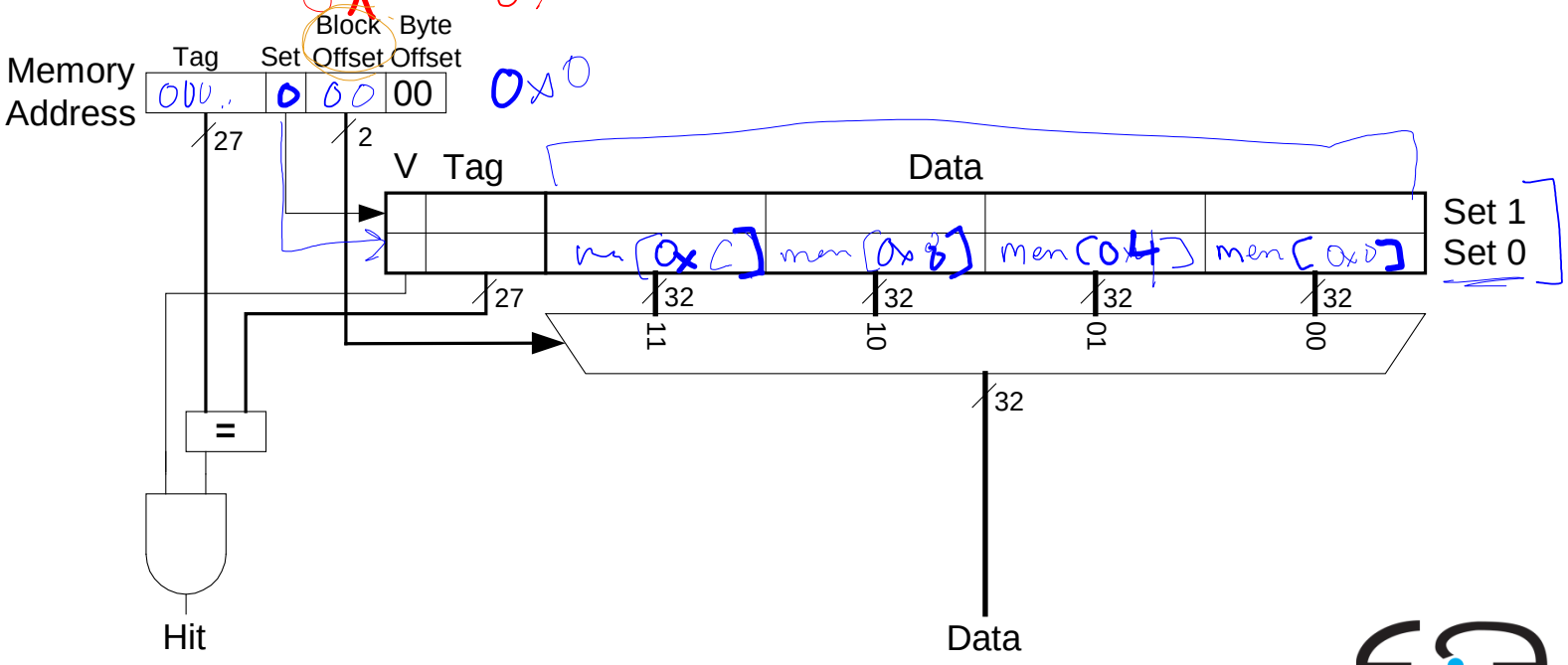
Embedded Systems & Applications

Räumliche Lokalität

TECHNISCHE UNIVERSITÄT DARMSTADT

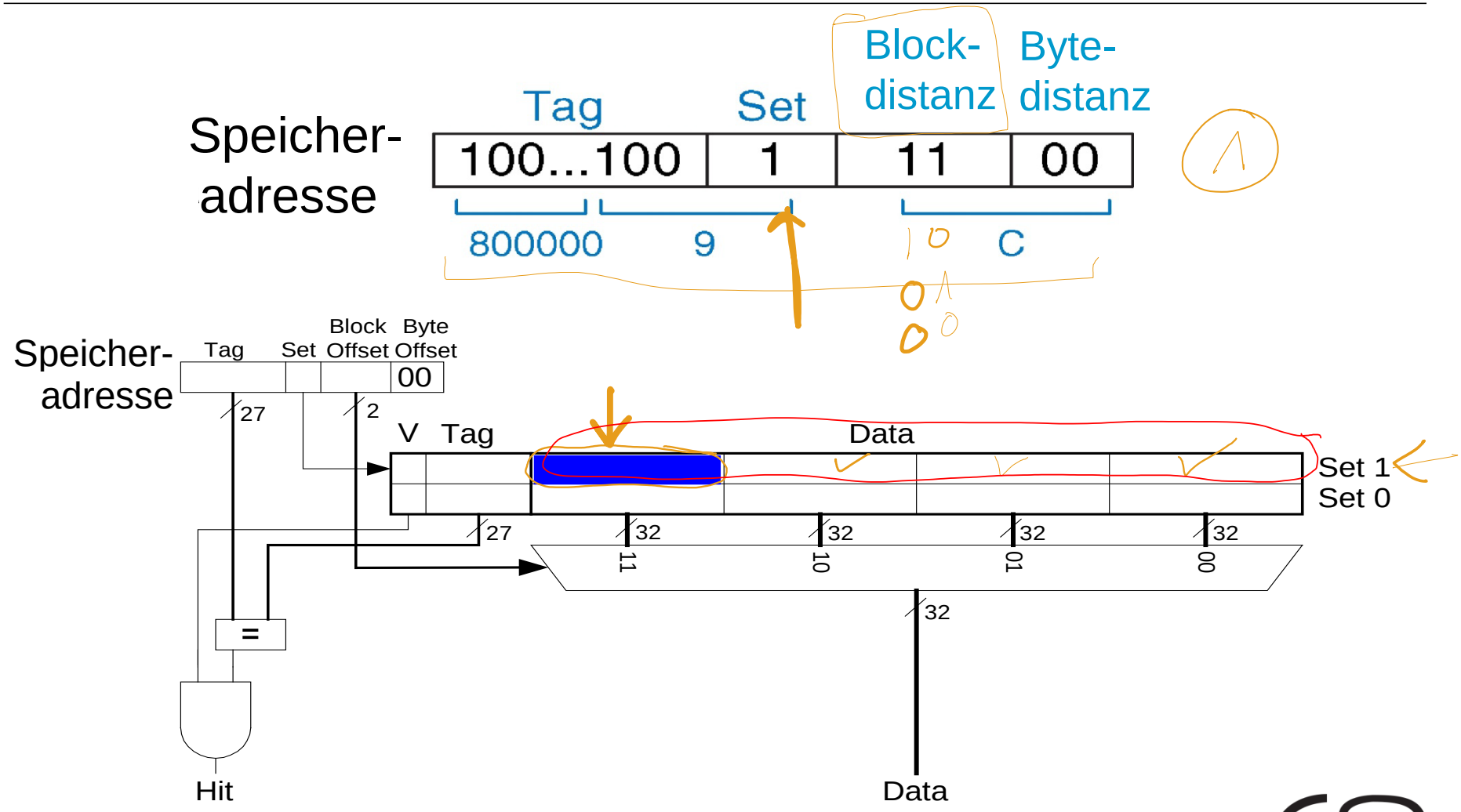
Blockgröße erhöhen:

- Blockgröße, ***b* = 4** Wörter
- ***C*** = 8 Wörter
- **Direktabgebildeter** Cache (1 Block pro Menge)
- Blockanzahl, ***B*** = *C*/*b* = 8/4 = 2



Embedded Systems & Applications

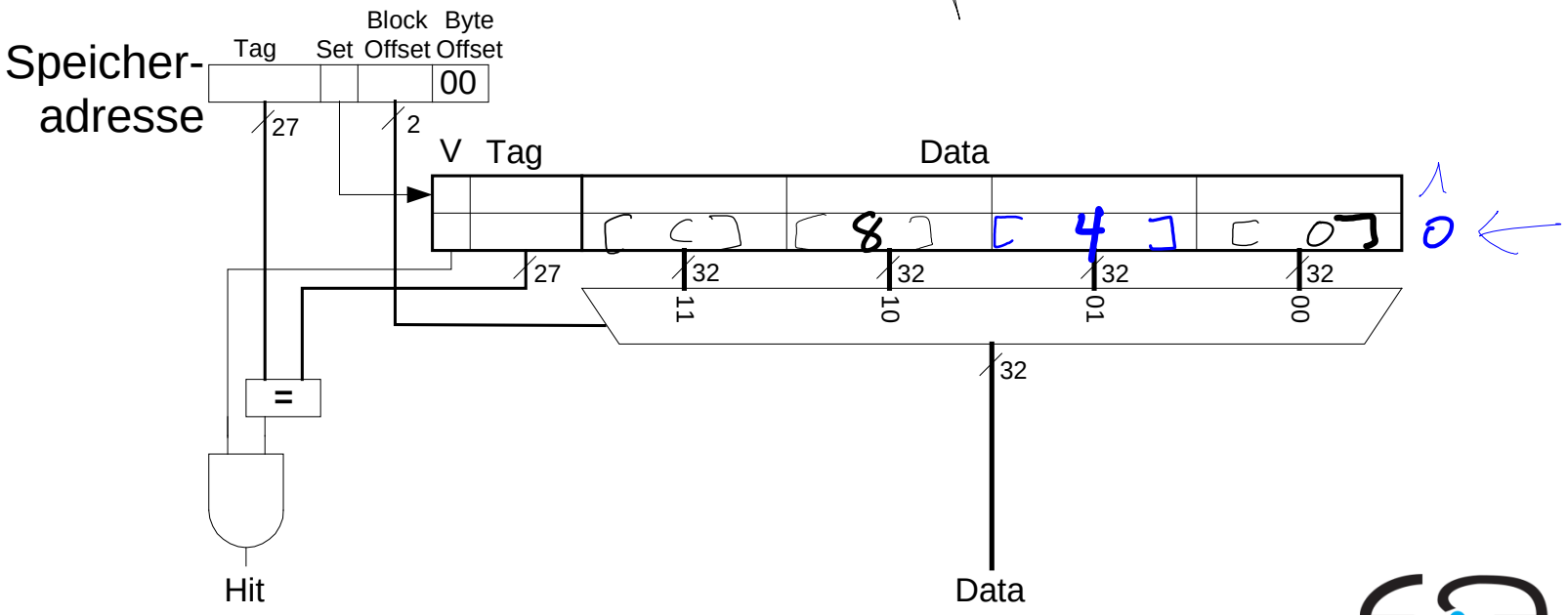
Cache mit erhöhter Blockgröße



Leistung eines direktabgebildten Caches mit erhöhter Blockgröße

TECHNISCHE UNIVERSITÄT DARMSTADT

```
        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Embedded Systems & Applications

Leistung eines direktabgebildten Caches mit erhöhter Blockgröße

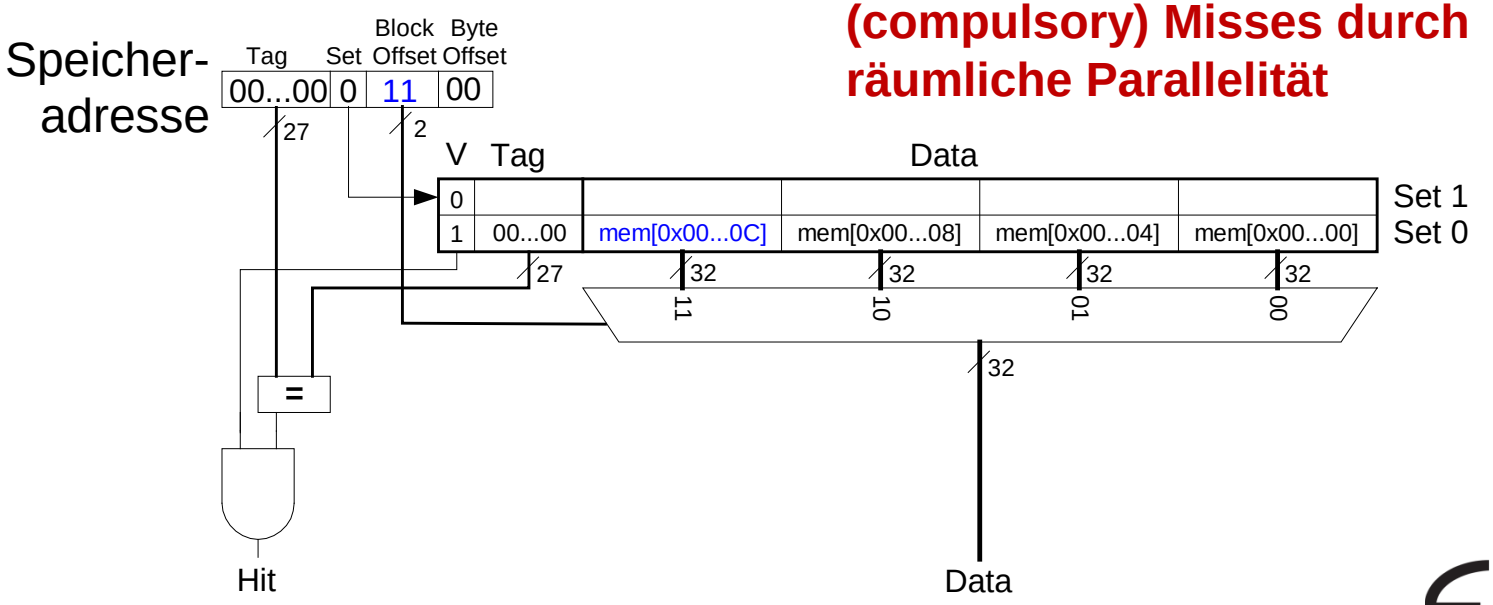
TECHNISCHE UNIVERSITÄT DARMSTADT

```
        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 1/15

= 6.67%

- **Erhöhte Blockgröße**
- **reduziert unvermeidbare (compulsory) Misses durch räumliche Parallelität**



Embedded Systems & Applications

Zusammenfassung: Aufbau von Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kapazität: ***C***
- Blockgröße: ***b***
- Blockanzahl: ***B*** = *C*/*b*
- *Assoziativitätsgrad*: ***N***
- Anzahl von Mengen (Sets): ***S*** = *B*/*N*

Name	Assoziativitätsgrad (*N*)	Anzahl von Mengen (*S* = *B*/*N*)
Direktabgebildete	1	*B*
N-Wege Mengenassoziativ	1 < *N* < *B*	*B* / *N*
Vollassoziativ	*B*	1

Embedded Systems & Applications

Kapazitäts-Misses

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein Cache könnte zu klein sein, alle benötigten Daten auf einmal zu halten

- Wenn der Cache schon voll ist, müssen Daten entfernt werden
  - z.B. Datum *Y* könnte entfernt werden bei Zugriff auf neues Datum *X*
- Ein ***Kapazitäts-Miss*** tritt auf, wenn das Programm dann wieder auf Datum *Y* zugreift

In assoziativen Caches muss entschieden werden, welches Datum entfernt werden soll

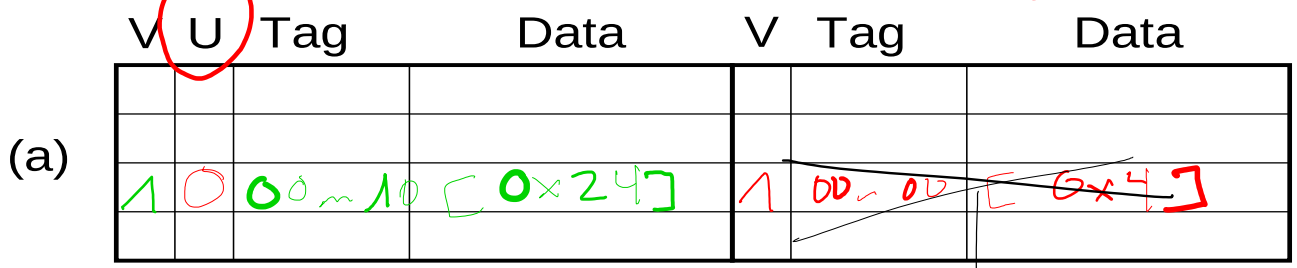
- **Least Recently Used (LRU) Ersatz**: den Block entfernen, den wir am längsten nicht benutzt haben

Embedded Systems & Applications

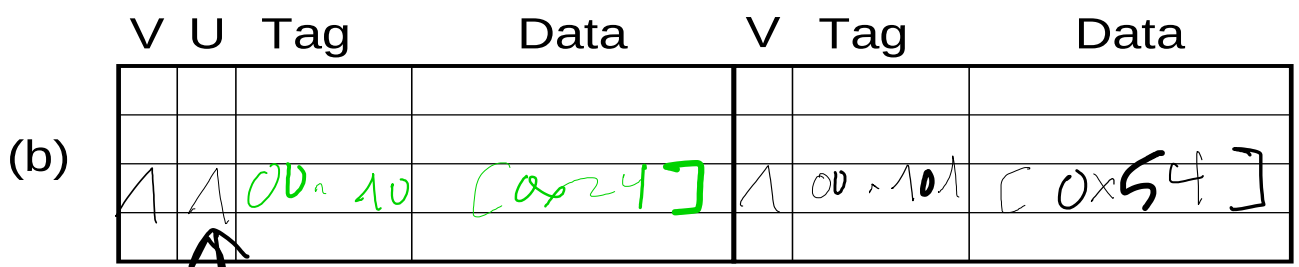
Ersetzen mit LRU-Schema

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# MIPS Assemblersprache
lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```



Mengennummer
3 (**11**)
2 (**10**)
1 (**01**)
0 (**00**)



Mengennummer
3 (**11**)
2 (**10**)
1 (**01**)
0 (**00**)

Embedded Systems & Applications

Ersetzen mit LRU-Schema


```
# MIPS Assemblersprache
lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```

MR = 100 %

(a)

	Weg 1				Weg 0			
	V	U	Tag	Data	V	Tag	Data	Mengennummer
	0	0			0			(11)
	0	0			0			(10)
	1	0	00...010	mem[0x00...24]	1	00...000	mem[0x00...04]	(01)
	0	0			0			(00)

(b)

	Weg 1				Weg 0			
	V	U	Tag	Data	V	Tag	Data	Mengennummer
	0	0			0			(11)
	0	0			0			(10)
	1	1	00...010	mem[0x00...24]	1	00...101	mem[0x00...54]	(01)
	0	0			0			(00)

Arten von Cache Misses

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Unvermeidbare (Compulsory):** beim ersten Zugriff auf Daten
- **Kapazität:** der Cache ist zu klein, um alle benötigten Daten zu halten
- **Konflikt:** mehrere Daten müssten sich am gleichen Ort in Cache befinden

Miss-Kosten (Miss penalty): die benötigte Zeit, um die Daten aus unteren Ebenen der Speicher-Hierarchie zu holen

Embedded Systems & Applications

Zusammenfassung von Caches

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Daten werden im Cache gehalten?

- Kürzlich zuvor gebrauchte Daten (zeitliche Lokalität)
- Benachbart liegende Daten (räumliche Lokalität, Verwendung von größeren Blöcken)

Wie werden die Daten gefunden?

- Datenadresse bestimmt:
  - Menge
  - Wort innerhalb eines Blocks ← Block Offset
- In assoziativen Caches, Daten können in einem von mehreren Wegen sein

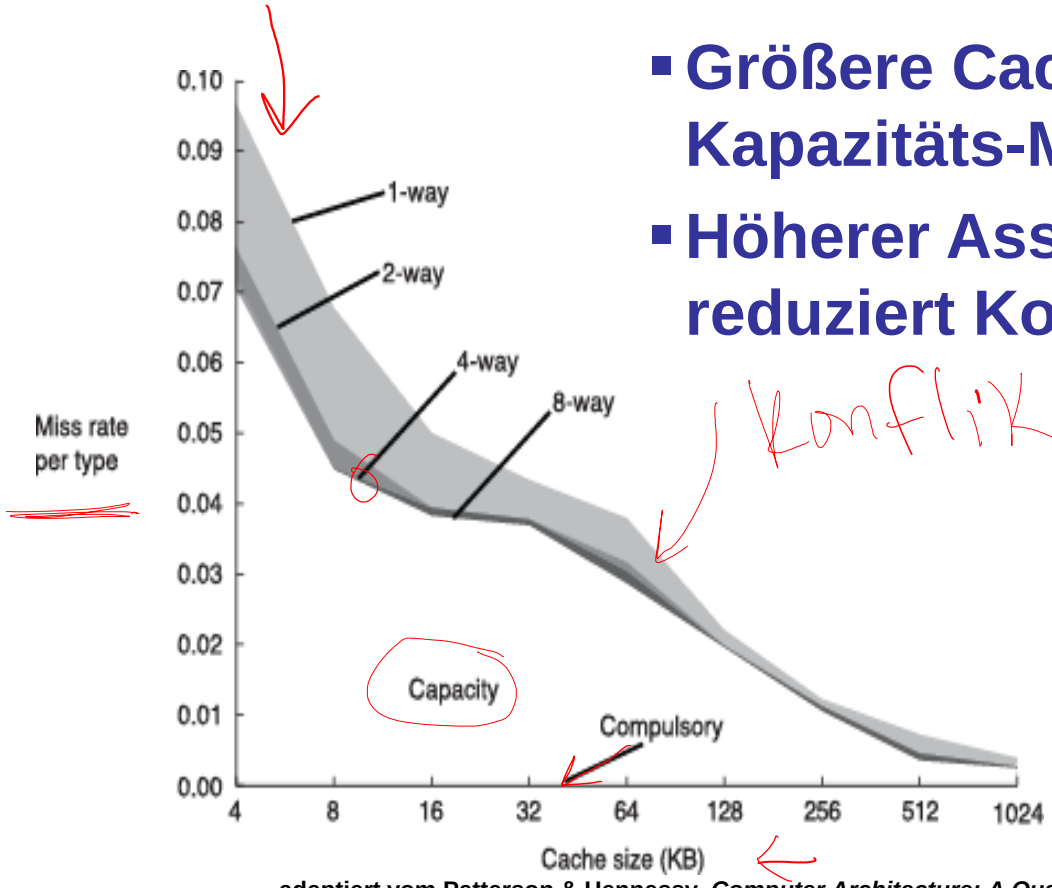
Wie werden Daten ersetzt?

LRU-Schemes

- Der älteste nicht gebrauchte Weg einer Menge wird ersetzt

Embedded Systems & Applications

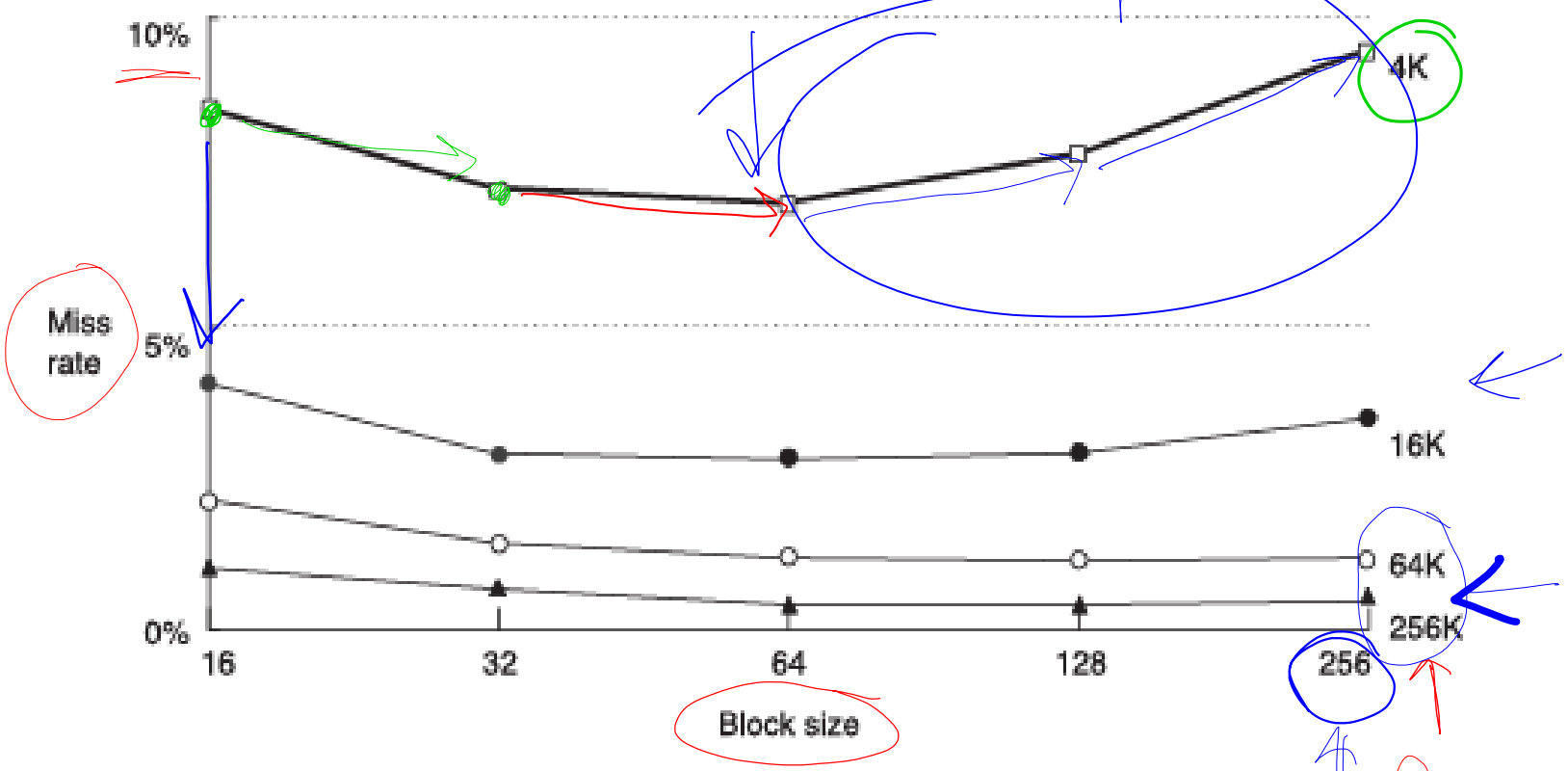
Miss Rate von Daten



- **Größere Caches reduzieren Kapazitäts-Misses**
- **Höherer Assoziativitätsgrad reduziert Konflikt-Misses**

adaptiert vom Patterson & Hennessy, *Computer Architecture: A Quantitative Approach*

Miss Rate von Daten



- Größere Blöcke reduzieren unvermeidbare (compulsory) Misses
- Größere Blöcke steigern Konflikt-Misses

Multi-Level Caches

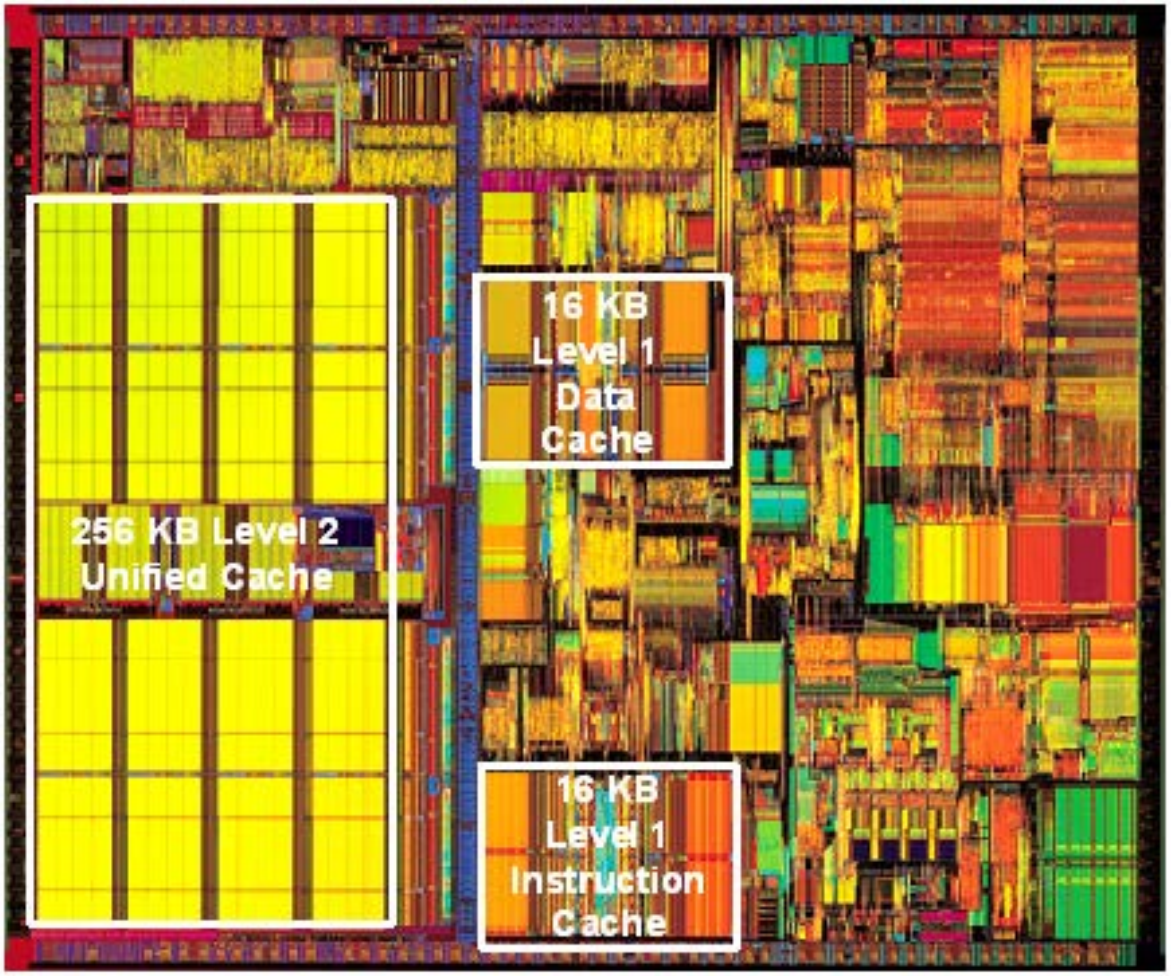
TECHNISCHE UNIVERSITÄT DARMSTADT

Größere Caches erreichen niedrigere Miss Rates, aber haben längere Zugriffszeiten

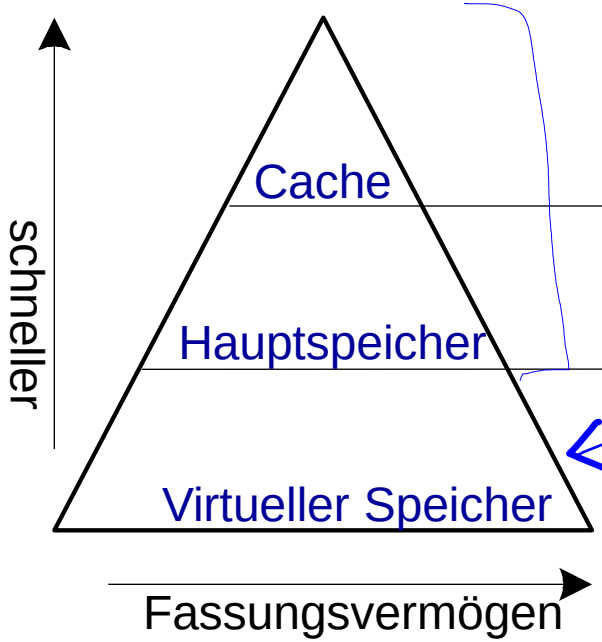
- Wir können die **Cachehierarchie erweitern**:
  - **Level 1** (L1): klein aber schnell
    - z.B. 16 KB, 1 Takt
  - **Level 2** (L2): größer aber langsamer
    - z.B. 256 KB, 2-6 Takte
  - ...wären auch weitere Ebenen möglich
    - Heute: bis zu **L4** im praktischen Einsatz

Embedded Systems & Applications

Intel Pentium III Chip



Speicherhierarchie



Technologie	Kosten/GB	Zugriffszeit
SRAM	~ $10,000	~ 1 ns
DRAM	~ $10	~ 10-50 ns
Festplatte	~ $0.1	~ 100,000 ns
SSD	~ $1	~ 10,000,000 ns

- **Physikalischer Speicher:** DRAM (Hauptspeicher)
- **Virtueller Speicher:** Festplatte

Langsam, gross, günstig

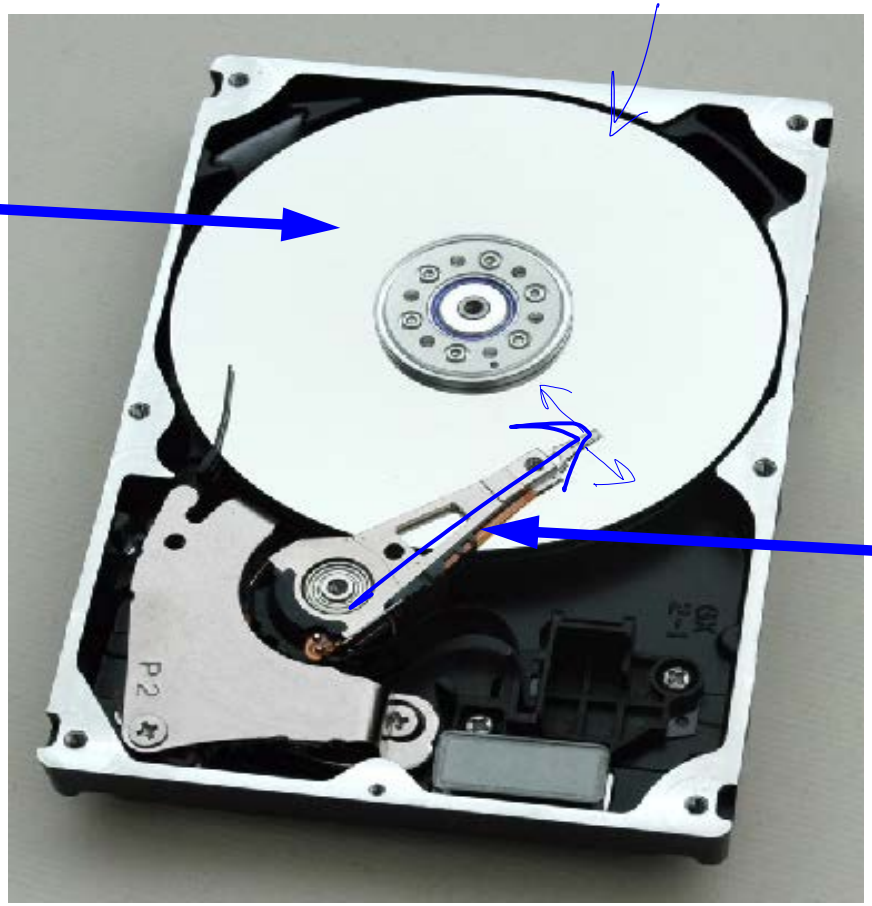
Virtueller Speicher: Festplatte

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bietet die Illusion, dass der Speicher größer ist
  - … ohne die Kosten von DRAM
- Hauptspeicher (DRAM) wirkt als *Cache* für die Festplatte

Die Festplatte

TECHNISCHE UNIVERSITÄT DARMSTADT



SSD
→ schreiben
langsam

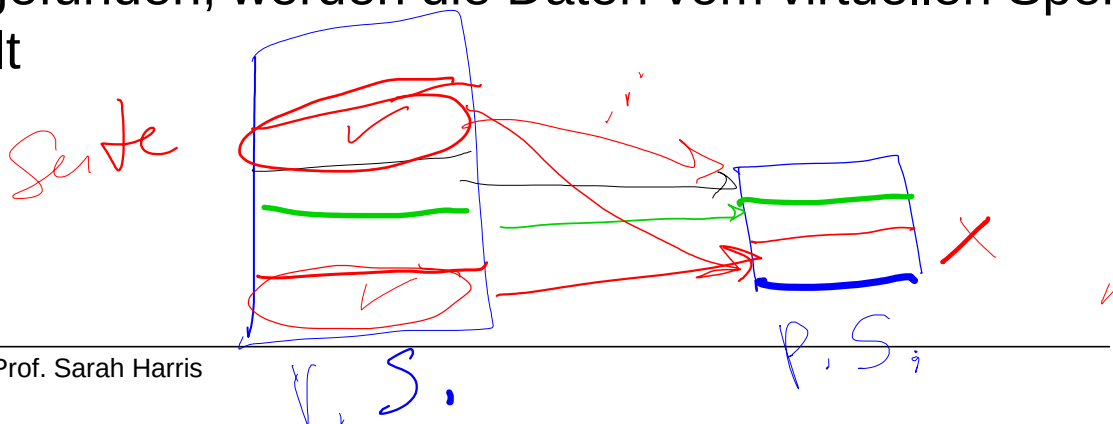
Es dauert Millisekunden um den Zeiger zu bewegen

Embedded Systems & Applications

Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jedes Programm benutzt **virtuelle Adressen**
- Alle Daten stehen im virtuellen Speicher
- Hauptspeicher (physikalischer Speicher) wirkt als Cache für virtuellen Speicher
  - Eine Untermenge der Daten des virtuellen Speichers steht im Hauptspeicher
  - CPU versucht, die Daten erstmals im Hauptspeicher zu finden
  - Wenn dort nicht gefunden, werden die Daten vom virtuellen Speicher (Festplatte) geholt



Embedded Systems & Applications

Vergleich der Terminologie: Cache / Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

Hauptspeicher (physikalischer Speicher) wirkt als Cache für virtuellen Speicher

Cache	Virtueller Speicher
Block	Seite (Page)
Blockgröße	Seitengröße
Blockdistanz (Block Offset)	Seitendistanz (Page Offset)
Miss	Seitenfehler (Page Miss)
Tag	Virtuelle Seitennummer

Embedded Systems & Applications

Virtueller Speicher: Begriffe

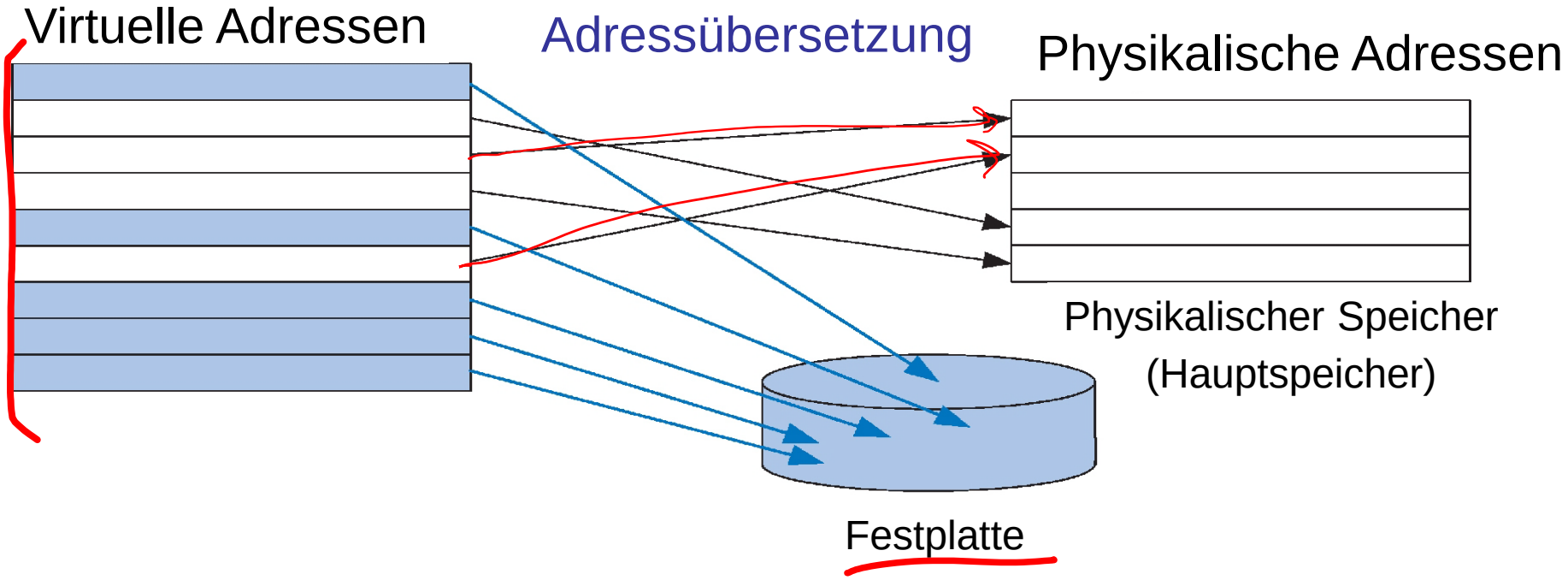
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Seitengröße:** Anzahl der Bytes, die auf ein Mal von Festplatte zum Hauptspeicher geliefert wird

- **Adressübersetzung:** die Hauptspeicheradresse (physikalische Adresse, PA) eines Datums wird aus der virtuellen Adresse (VA) bestimmt

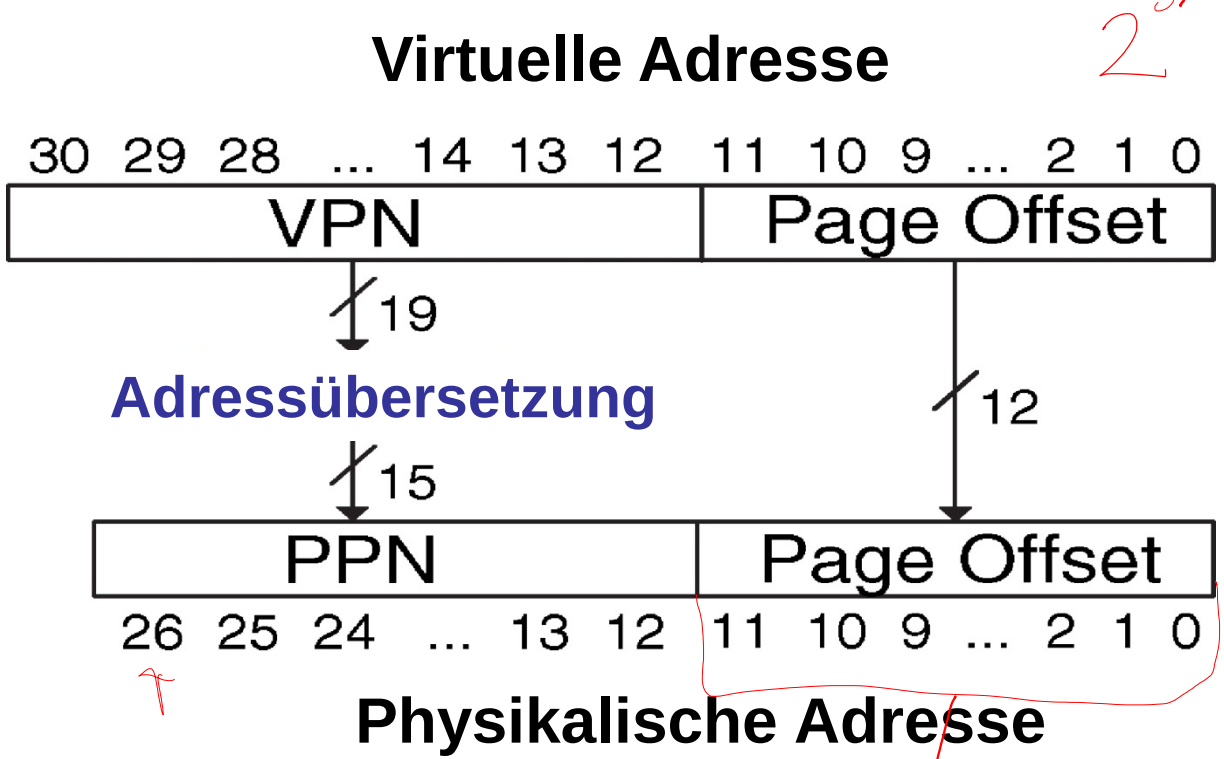
- **Seitentabelle (Page Table):** Tabelle für die Adressübersetzung

Virtuelle und Physikalische Adressen



- Die meisten Zugriffe werden **im Hauptspeicher** (i.d.R. GBs an physikalischem Speicher) gefunden werden
- Aber Programme haben die große Kapazität der Festplatte (i.d.R. TBs) als virtuellem Speicher zur Verfügung

Adressübersetzung



Beispiel: Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

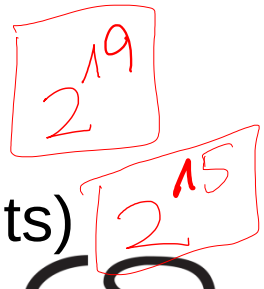
- **System:**
  - Größe des virtuellen Speichers: 2 GB = 2^{31} Bytes
  - Größe des physikalischen Speichers: 128 MB = 2^{27} Bytes
  - Seitengröße (Page Size): 4 KB = 2^{12} Bytes

Embedded Systems & Applications

Beispiel: Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

- **System:**
  - Größe des virtuellen Speichers: 2 GB = 2^{31} Bytes
  - Größe des physikalischen Speichers: 128 MB = 2^{27} Bytes
  - Seitengröße (Page Size): 4 KB = 2^{12} Bytes
- **Organisation:**
  - Virtuelle Adresse: **31** Bits
  - Physikalische Adresse: **27** Bits
  - Seitendistanz (Page Offset): **12** Bits
  - # virtuelle Seiten = $2^{31}/2^{12}$ = $\mathbf{2^{19}}$ (VSN = 19 Bits)
  - # physikalische Seiten = $2^{27}/2^{12}$ = $\mathbf{2^{15}}$ (PSN = 15 Bits)

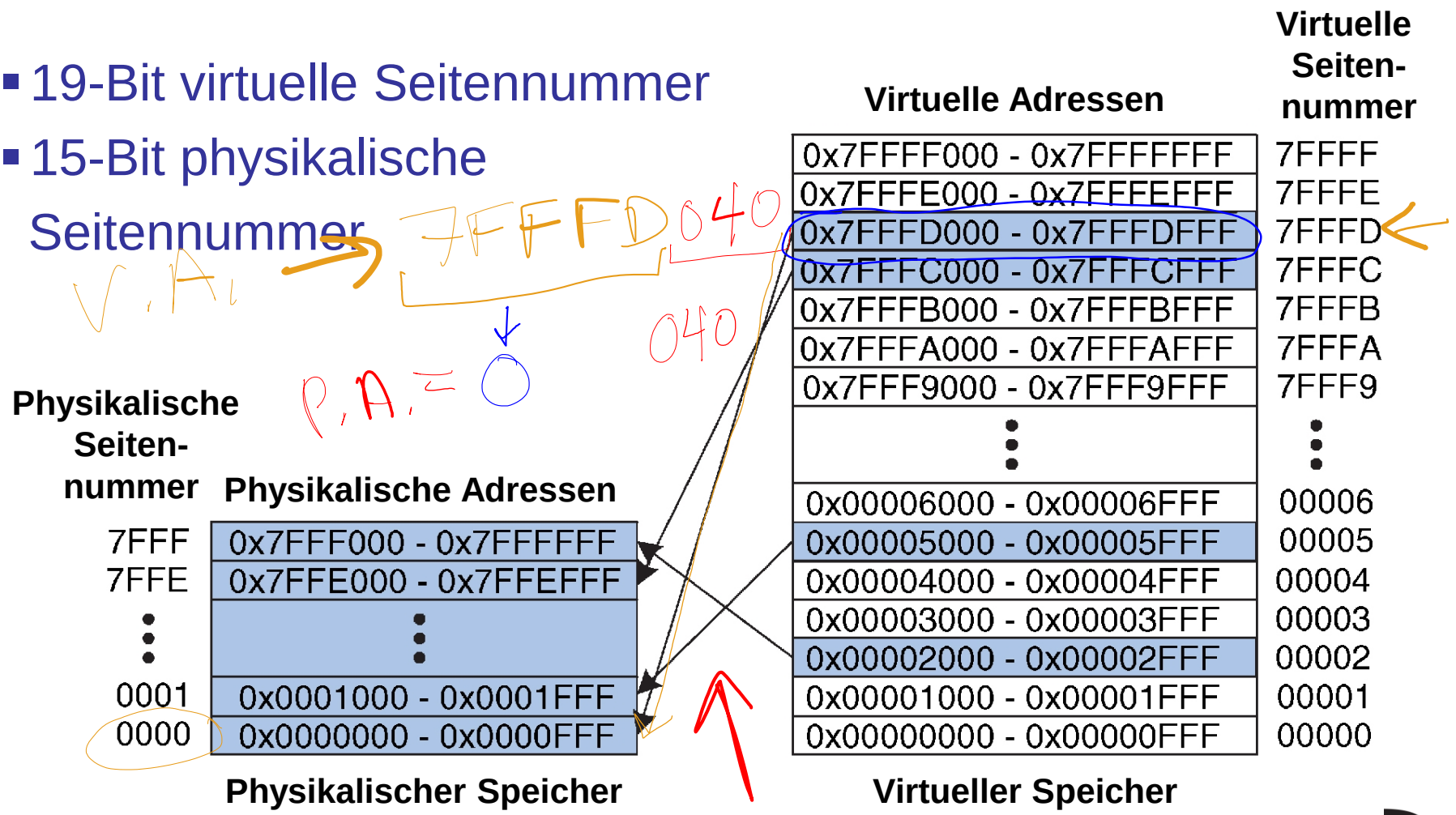


Embedded Systems & Applications

Beispiel: Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

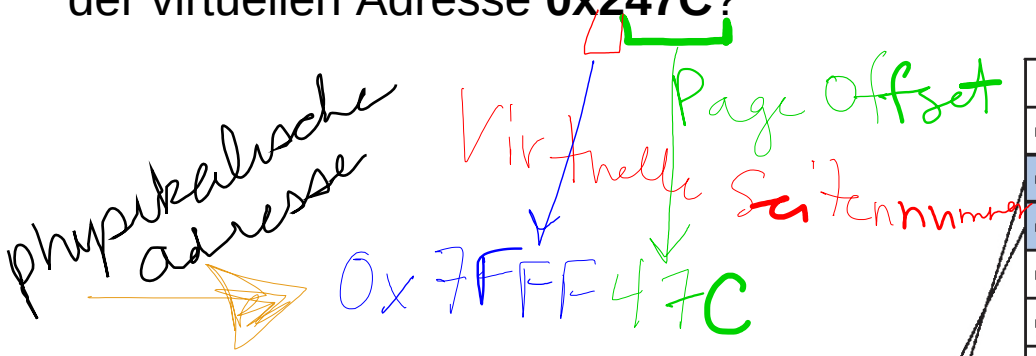
- 19-Bit virtuelle Seitennummer
- 15-Bit physikalische Seitennummer



Embedded Systems & Applications

Beispiel: Virtueller Speicher

Welche physikalische Adresse entspricht der virtuellen Adresse **0x247C**?



Virtuelle Adressen

	Virtuelle Seiten-nummer
0x7FFFF000 - 0x7FFFFFFF	7FFFF
0x7FFFE000 - 0x7FFFEFFF	7FFFE
0x7FFFD000 - 0x7FFFDFFF	7FFFD
0x7FFFC000 - 0x7FFFCFFF	7FFFC
0x7FFFB000 - 0x7FFFBFFF	7FFFB
0x7FFFA000 - 0x7FFFAFFF	7FFFA
0x7FFF9000 - 0x7FFF9FFF	7FFF9
⋮	⋮
0x00006000 - 0x00006FFF	00006
0x00005000 - 0x00005FFF	00005
0x00004000 - 0x00004FFF	00004
0x00003000 - 0x00003FFF	00003
0x00002000 - 0x00002FFF	00002
0x00001000 - 0x00001FFF	00001
0x00000000 - 0x00000FFF	00000

Virtueller Speicher

Physikalische Seiten-nummer	**Physikalische Adressen**
7FFF	0x7FFF000 - 0x7FFFFFF
7FFE	0x7FFE000 - 0x7FFEFFF
⋮	⋮
0001	0x0001000 - 0x0001FFF
0000	0x0000000 - 0x0000FFF

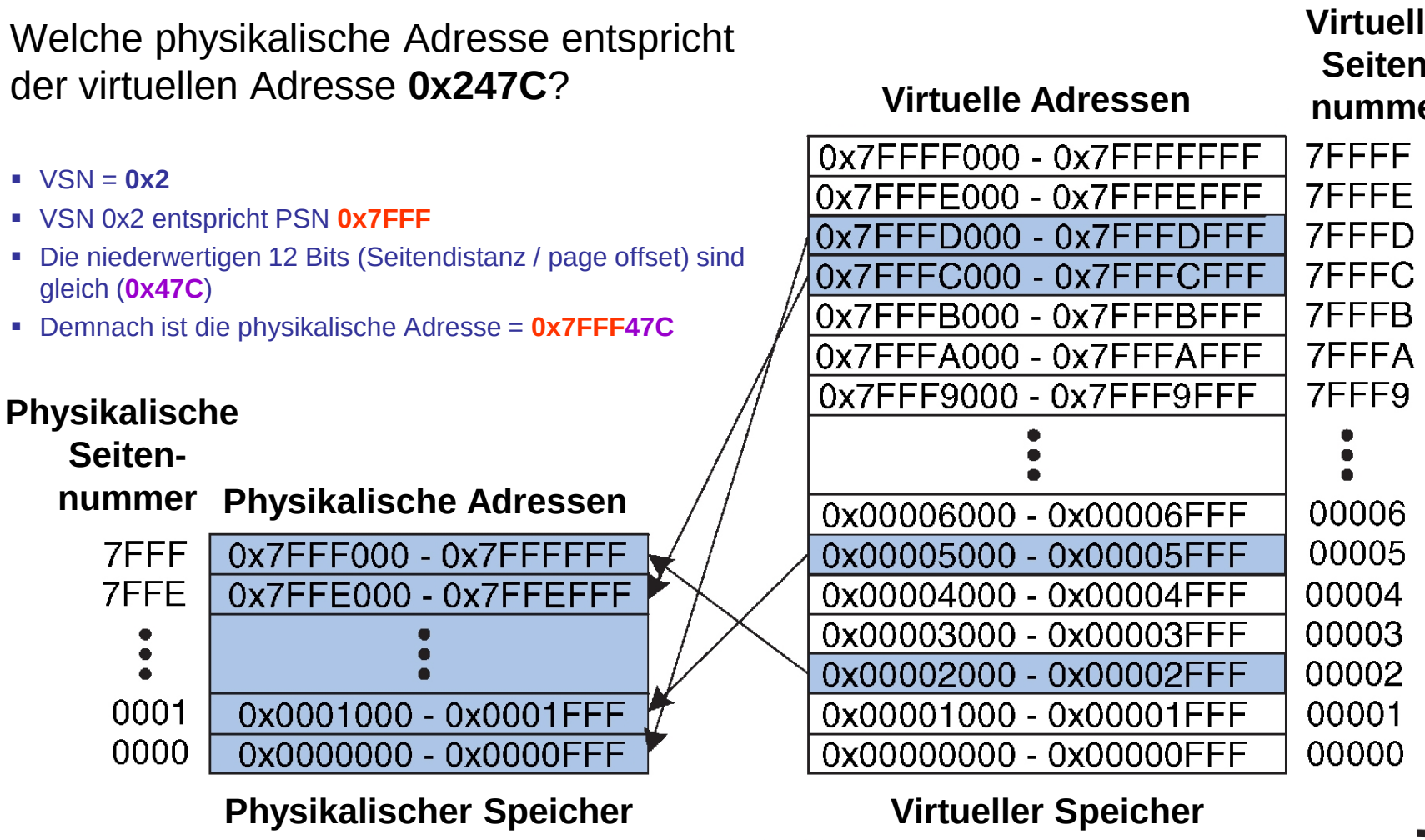
Physikalischer Speicher

Beispiel: Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche physikalische Adresse entspricht der virtuellen Adresse **0x247C**?

- VSN = **0x2**
- VSN 0x2 entspricht PSN **0x7FFF**
- Die niederwertigen 12 Bits (Seitendistanz / page offset) sind gleich (**0x47C**)
- Demnach ist die physikalische Adresse = **0x7FFF47C**



Embedded Systems & Applications

Wie übersetzen wir Adressen?

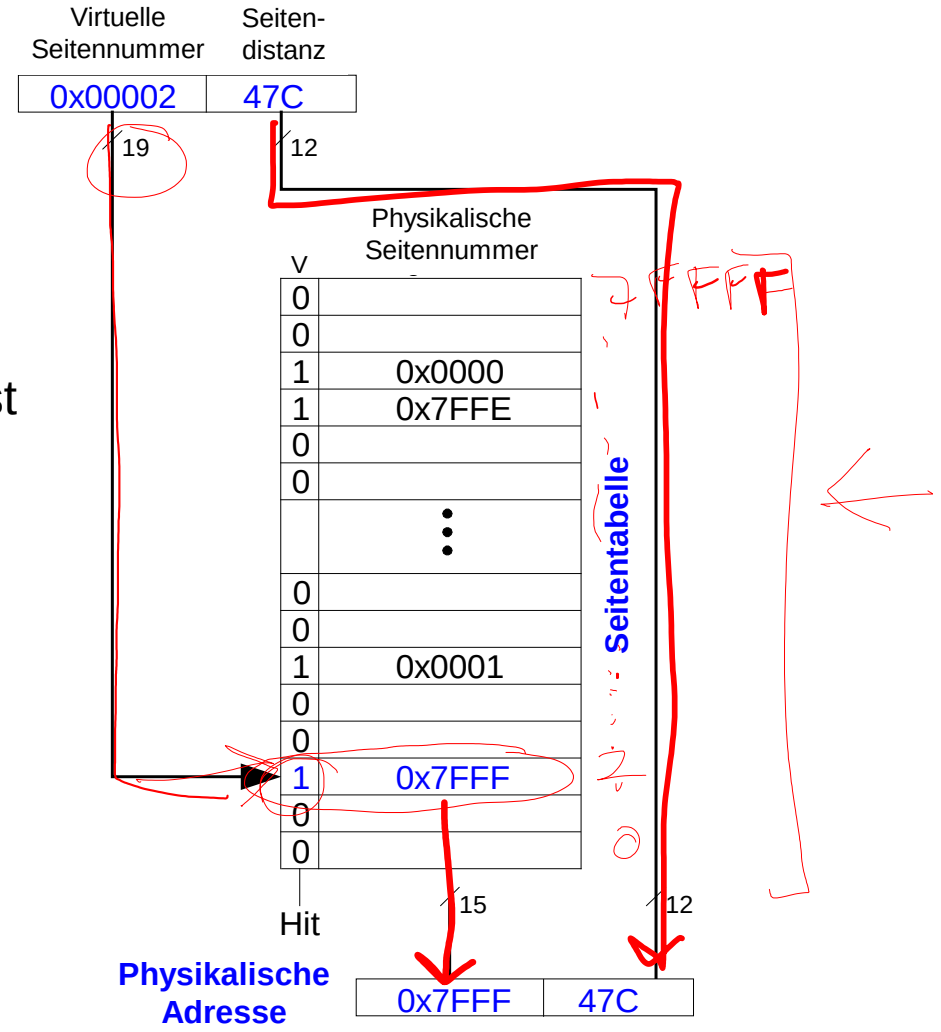
TECHNISCHE UNIVERSITÄT DARMSTADT

Seitentabelle (Page table)

- Jede virtuelle Speicherseite hat einen Eintrag in der Tabelle
- Jeder Eintrag besteht aus:
  - **Gültigkeits-Bit (Valid bit):** 1 wenn die virtuelle Seite im physikalischen Speicher steht, sonst 0 (sie müsste von der Festplatte geholt werden)
  - **Physikalische Seitennummer:** wo im Hauptspeicher die Seite steht

Embedded Systems & Applications

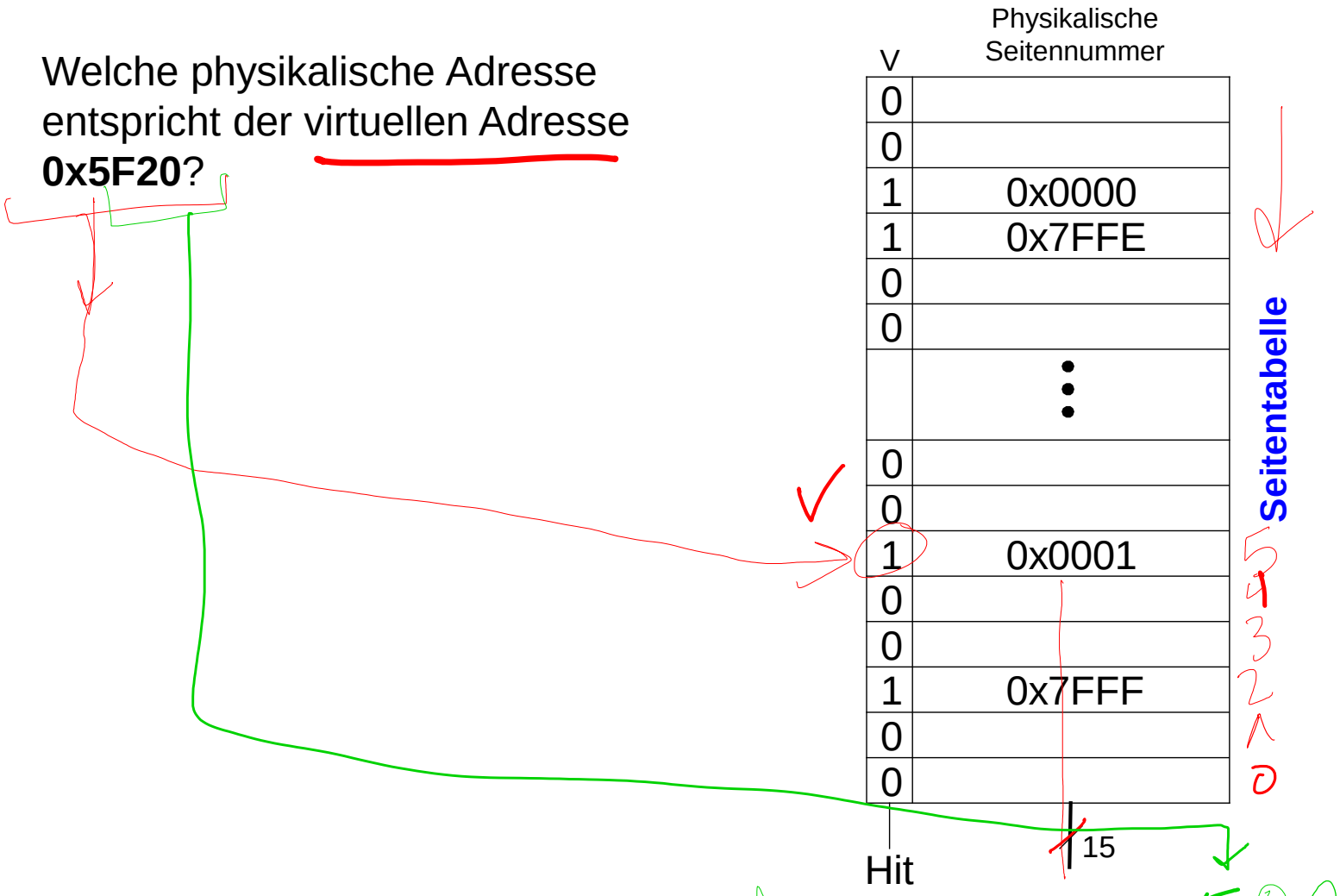
Beispiel: Seitentabelle (Page Table)



VSN (Virtuelle Seitennummer) ist der **Index** in der Seitentabelle

Beispiel 1: Seitentabelle

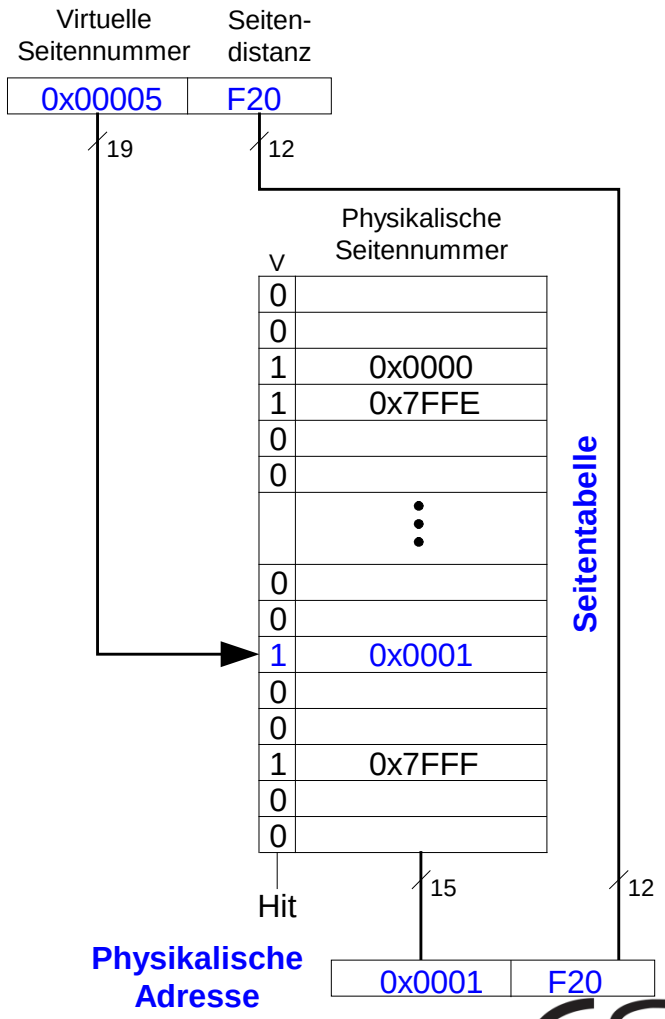
Welche physikalische Adresse entspricht der virtuellen Adresse **0x5F20**?



Beispiel 1: Seitentabelle

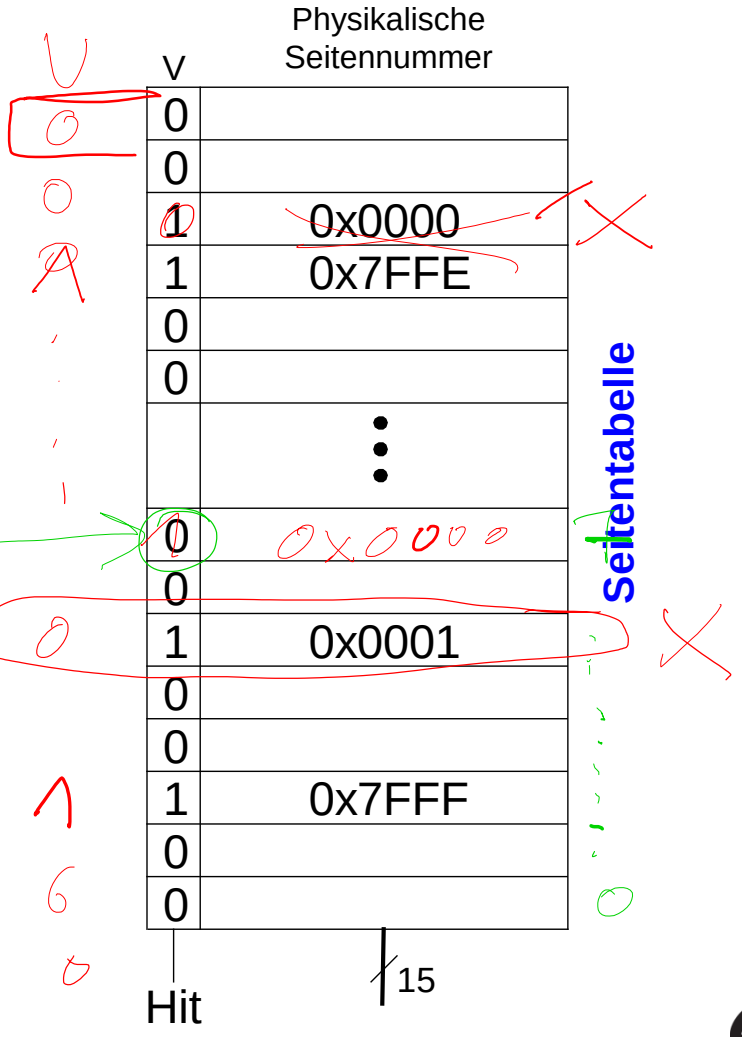
Welche physikalische Adresse entspricht der virtuellen Adresse **0x5F20**?

- VSN = **5**
- Eintrag auf Index 5 in der Seitentabelle zeigt, dass VSN 5 in der physikalischen Seite **1** steht
- Die physikalische Adresse ist damit: **0x1F20**



Beispiel 2: Seitentabelle

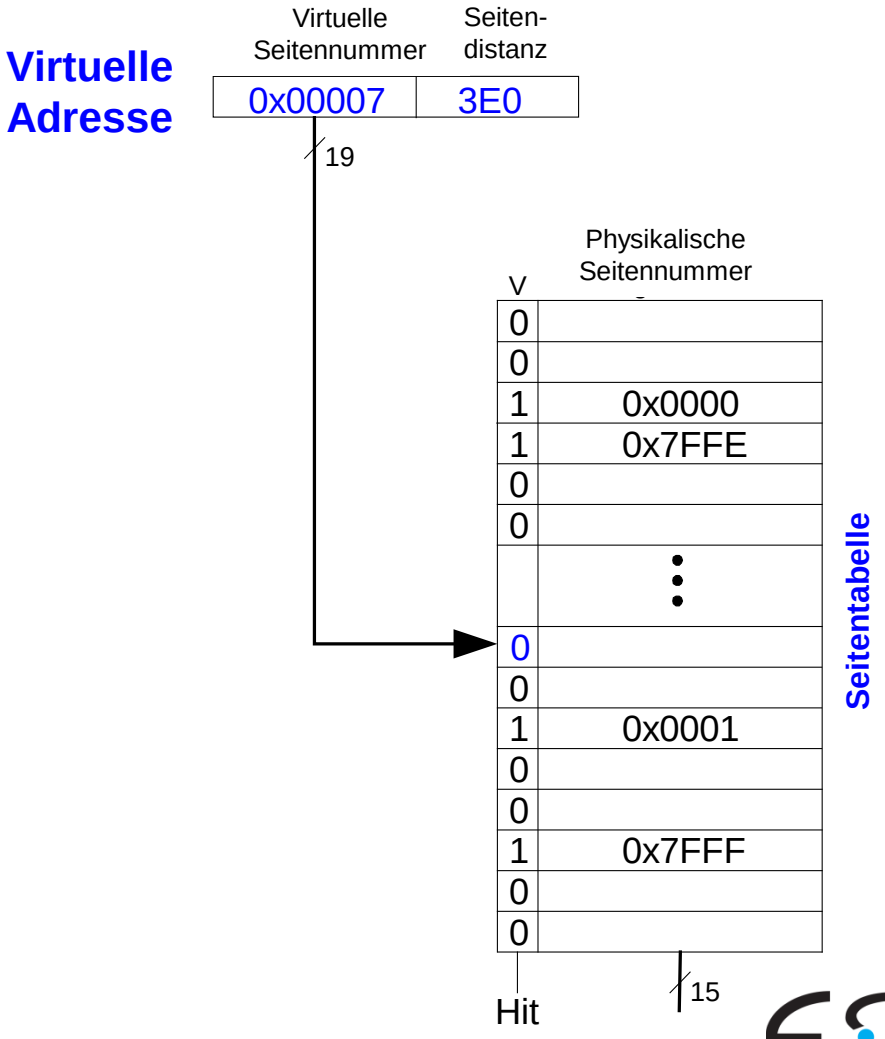
Welche physikalische Adresse entspricht der virtuellen Adresse **0x73E0**?



Beispiel 2: Seitentabelle

Welche physikalische Adresse entspricht der virtuellen Adresse **0x73E0**?

- VSN = **7**
- Eintrag auf Index 7 in der Seitentabelle ist nicht gültig (V=0), die Seite ist also nicht im physikalischen Speicher präsent
- Die virtuelle Seite muss in den physikalischen Speicher von der Festplatte geholt werden (paged)
  - Historisch verwendet: “Swapped”
  - Damalige Bedeutung: Prozesse können nur im **Ganzen** eingelagert / ausgelagert werden
  - Paging erlaubt dagegen das Einlagern / Auslagern **einzelner** Speicherseiten



Herausforderungen beim Aufbau von Seitentabellen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Die Seitentabelle ist gross**
  - steht häufig auch im physikalischen Speicher (und braucht dort Platz!)
- Alle Lese-/Schreibbefehle fordern **2 Hauptspeicherzugriffe**
  - einmal auf die Übersetzung (Zugriff auf Seitentabelle)
  - einmal auf die eigentlichen Daten (nach der Übersetzung)
- **Halbiert den Speicherdurchsatz**
  - *Kann man aber schlauer anstellen …*

Embedded Systems & Applications

Translation Lookaside Buffer (TLB)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein kleiner Cache für die neuesten Übersetzungen
- Die meisten Speicherzugriffe werden jetzt nur einen Hauptspeicherzugriff erfordern

Embedded Systems & Applications

Translation Lookaside Buffer (TLB)

TECHNISCHE UNIVERSITÄT DARMSTADT

Seitentabellenzugriffe haben hohe zeitliche Lokalität, da:

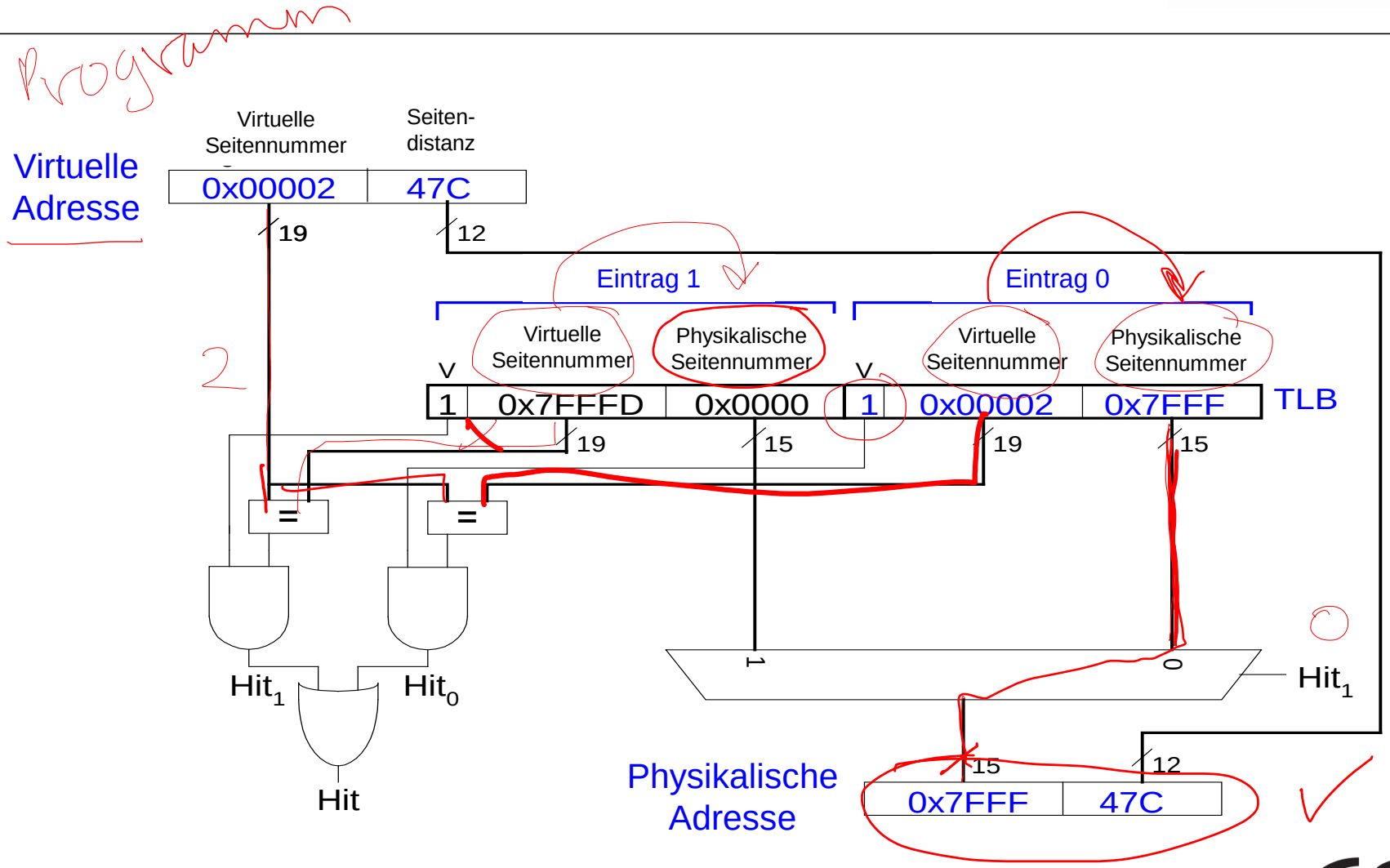
- Ursprüngliche Datenzugriffe haben selber hohe zeitliche und räumliche Lokalität
- Die Seitengröße ist (relativ) groß, aufeinanderfolgende Lese-/Schreibbefehle werden oftmals auf die gleiche Seite zugreifen

TLB

- **Klein:**
  - Zugriff <= 1 Takt
  - Typischerweise 16 - 512 Einträge
  - Vollassoziativ
  - Typischerweise > 99 % Hit Rates
- Die meisten der Lese-/Schreibbefehle erfordern damit nur **einen Hauptspeicherzugriff**

Embedded Systems & Applications

Beispiel: TLB mit zwei Einträgen



Zeitplan für den Rest des Semesters

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Heute:** - Evaluation des Kurses
  - Ein-/Ausgabegeräten
  - Zusammenfassung

- **04.07:** - Probefragen für die Klausur (werden ab 29.06.2016 auf der Webseite stehen)
  - sollten davor **von selber** durchgearbeitet
- **11.07:** - Vortrag auf Englisch (wird nicht auf der Klausur stehen):
  - Weiterführende Themen der Mikroarchitektur
  - ARM Architektur
- **18.07:** - Klausur, 10 Uhr – 11:30 Uhr
  - wo die stattfindet, wird in der Woche davor durch Moodle bekanntgegeben

Evaluation der Vorlesung Rechnerorganisation

TECHNISCHE UNIVERSITÄT DARMSTADT

Organisatorisch:

Ich brauche 2 Studierende, die alle Bögen zur Fachschaft Informatik zurückbringen werden:

- Gebäude S2/02 (Piloty) Raum D120

Embedded Systems & Applications

Evaluation der Vorlesung Rechnerorganisation

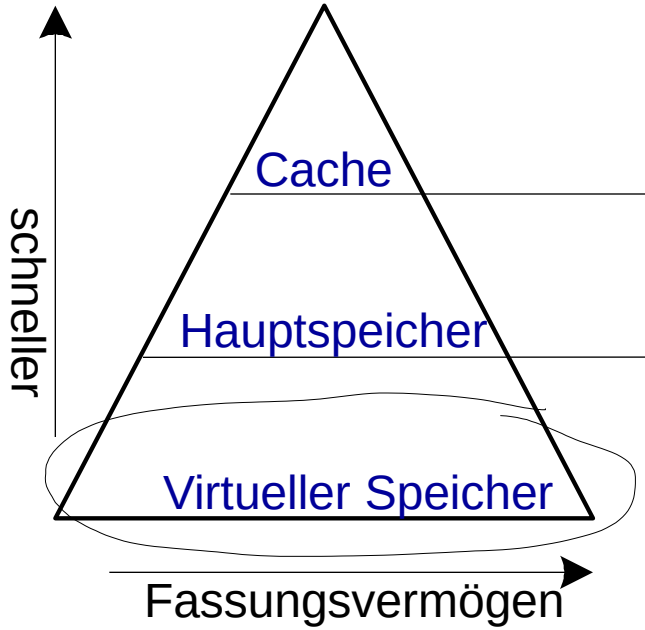
TECHNISCHE UNIVERSITÄT DARMSTADT

Teil 5 (Freie Fragen):

5.1. Ein vorlesungsbegleitendes Praktikum (3 CP) wäre für das Verständnis der Vorlesung hilfreich.

5.2. Ich wäre bereit mehrere Stunden (3-5) wöchentlich für ein vorlesungsbegleitendes Praktikum aufzuwenden.

Speicherhierarchie



Technologie	Kosten/GB	Zugriffszeit
SRAM	~ $10,000	~ 1 ns
DRAM	~ $10	~ 10-50 ns
Festplatte	~ $0.1	~ 10,000,000 ns
SSD	~ $1	~ 100,000 ns

- **Physikalischer Speicher:** DRAM (Hauptspeicher)
- **Virtueller Speicher:** Festplatte

Langsam, gross, günstig

Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

Jedes Programm erzeugt **virtuelle Adressen**

- Der ganze Bereich der virtuellen Adressen steht auf der Festplatte
- Eine Untermenge der virtuellen Adressen steht im Hauptspeicher
- Der Prozessor erzeugt virtuelle Adressen und muss diese in physikalische Adressen übersetzen (durch die TLB und Seitentabelle)
- Daten, die im DRAM nicht gefunden sind, müssen von der Festplatte geholt werden

Embedded Systems & Applications

Speicherschutz (Memory Protection)

TECHNISCHE UNIVERSITÄT DARMSTADT

Hier vereinfacht angenommen: ein Programm ist ein *Prozess*

Jeder Prozess hat seine **eigene Seitentabelle**, damit gilt:

- Jeder Prozess kann **den ganzen Adressraum** benutzen – ohne sich Sorgen machen zu müssen welche Adressen die anderen Prozesse benutzen
  - Zwei Prozesse können die *gleiche virtuelle* Adresse für zwei *verschiedene* Variablen verwenden
  - Ein Prozess muss nicht die Adressvergabe eines anderen Prozesses berücksichtigen
- Ein Prozess kann **nur auf die physikalischen Seiten zugreifen**, die in seiner **eigenen Seitentabelle** stehen
  - Damit kann der Speicher anderer Prozessen nicht überschrieben werden
  - Ein Prozess (z.B. Virus) kann den Speicher eines anderen Prozesses nicht korrumpieren

Embedded Systems & Applications

Zusammenfassung vom virtuellem Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

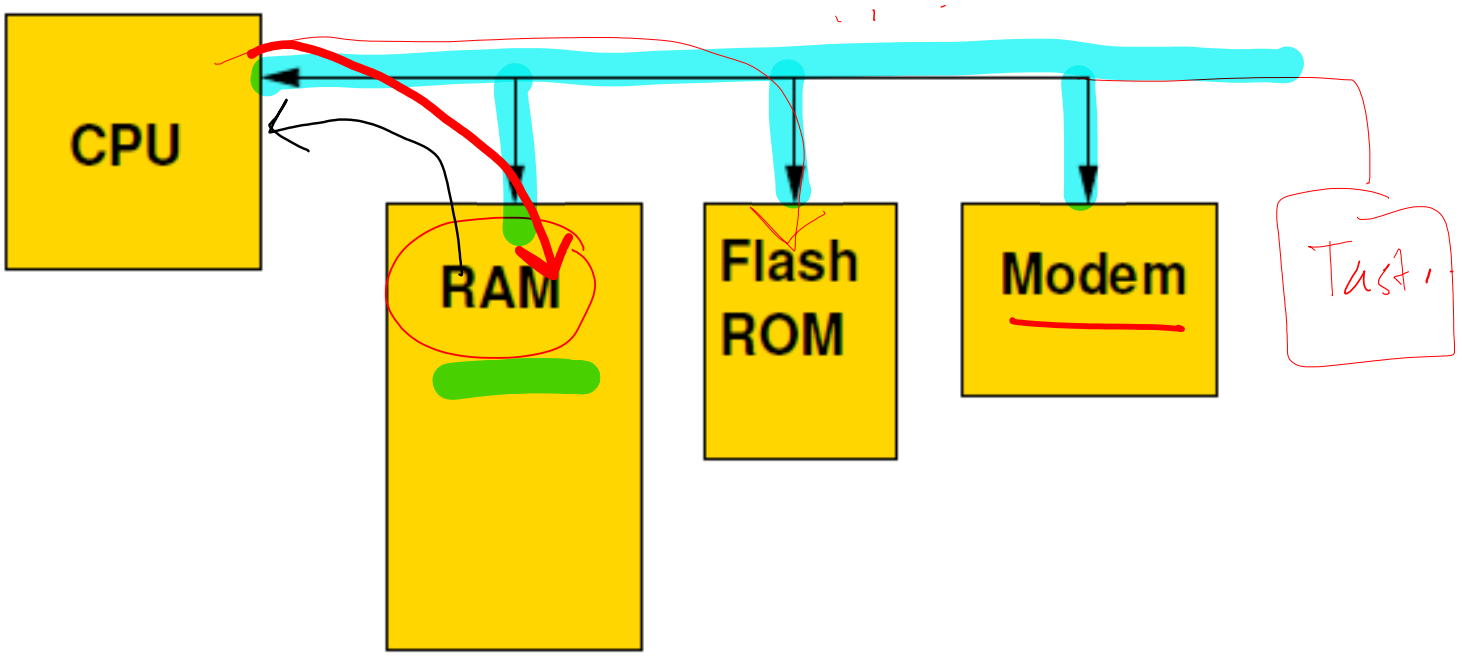
- Virtueller Speicher erhöht die zugängliche **Speicherkapazität**
- Eine **Untermenge** der virtuellen Seiten wird im physikalischen Speicher gehalten
- Eine **Seitentabelle** zeigt an, wo die virtuellen Seiten im physikalischen Speicher stehen – und erlaubt so die **Adressübersetzung**
- Ein **TLB** beschleunigt die Adressübersetzung
- Da Prozesse ihre eigenen Seitentabellen haben, erlaubt virtueller Speicher auch **Speicherschutz** (**memory protection**)

Embedded Systems & Applications

Ein-/Ausgabegeräten (I/O Devices)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verschiedene Untereinheiten
- Kommunikation untereinander
  - durch einen gemeinsamen Bus



Embedded Systems & Applications

Zugriff auf Ein-/Ausgabegeräten

TECHNISCHE UNIVERSITÄT DARMSTADT

2 Vorgehensweise:

1. **Getrennter Adressbereich** für Bus: Speicher, Ein-/Ausgabegeräte
2. **Speichereinblendung von Ein-/Ausgabegeräten** (Memory-mapped I/O):
   - Ein-/Ausgabegeräte benutzen den gleichen Adressbereich wie Speicher: Blende Busadressen in normalen Adressraum ein.
   - Dabei kann der Prozessor die normalen Lade-/Schreibbefehlen (**`lw`**, **`sw`**) benutzen um den Speicher **oder** die Ein-/Ausgabegeräte zuzugreifen

Hardware

Embedded Systems & Applications

Zugriff auf Ein-/Ausgabegeräten: Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

2 Vorgehensweise:

1. **Getrennter Adressbereich** für Bus: braucht Spezielle Instruktionen (programmed I/O)

   Beispiel: `swio $s1, 0x60($0)`

   # Schreibt den Inhalt des **`$s1`** Registers an I/O Adresse 60

   Beispiel: `lwio $t3, 0x40($0)`

   # Liest I/O Adresse 40 und schreibt das Ergebnis in Register **`$t3`**

2. **Speichereinblendung von Ein-/Ausgabegeräten** (Memory-mapped I/O):

   Beispiel: `sw $s1, 0x28($0)`

   Beispiel: `lw $t3, 0x52($0)`

Embedded Systems & Applications

Zugriff auf Ein-/Ausgabegeräten

TECHNISCHE UNIVERSITÄT DARMSTADT

Nachteil des getrennten Adressbereichs:

- Zusätzliche Instruktionen
- Ungünstig: Instruktionsbits sind rares Gut

Memory-mapped I/O: Heute überwiegend verwendet

Embedded Systems & Applications

Speichereinblendung von Ein-/Ausgabegeräten

TECHNISCHE UNIVERSITÄT DARMSTADT

Prozessoren greifen auf **Ein-/Ausgabegeräte** (z.B. Tastaturen, Monitore, und Drucker) zu genau **wie auf Daten im Speicher**

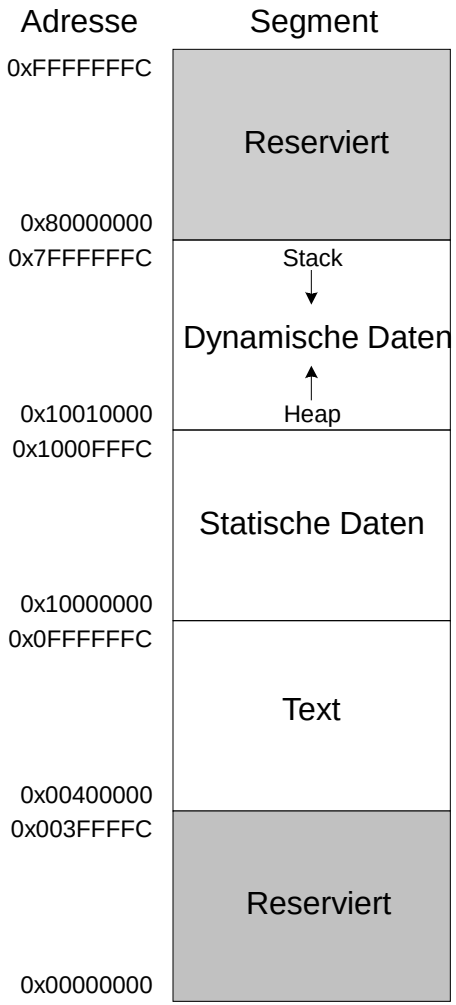
- Ein oder mehrere Adressen sind einem Gerät zugewiesen
- Wenn auf diese Adressen zugegriffen wird, werden die Daten vom **Gerät** gelesen oder geschrieben (statt vom Speicher!)

- Eine Untermenge der Adressen sind den Ein-/Ausgabegeräten zugewiesen,
  - z.B., Adressen 0xF0000000 bis 0xFFFFFFFF könnten dafür reserviert sein
  - Konkrete Zuordnung hängt von Rechnerarchitektur und Betriebssystem ab
    - Wird oftmals beim Starten des Betriebssystems ausgegeben:

```
…
uhci_hcd 0000:00:1d.1: UHCI Host Controller
uhci_hcd 0000:00:1d.1: new USB bus registered, assigned bus number 7
uhci_hcd 0000:00:1d.1: irq 19, io base 0x0000d400
usb usb7: New USB device found, idVendor=1d6b, idProduct=0001
…
```


Embedded Systems & Applications

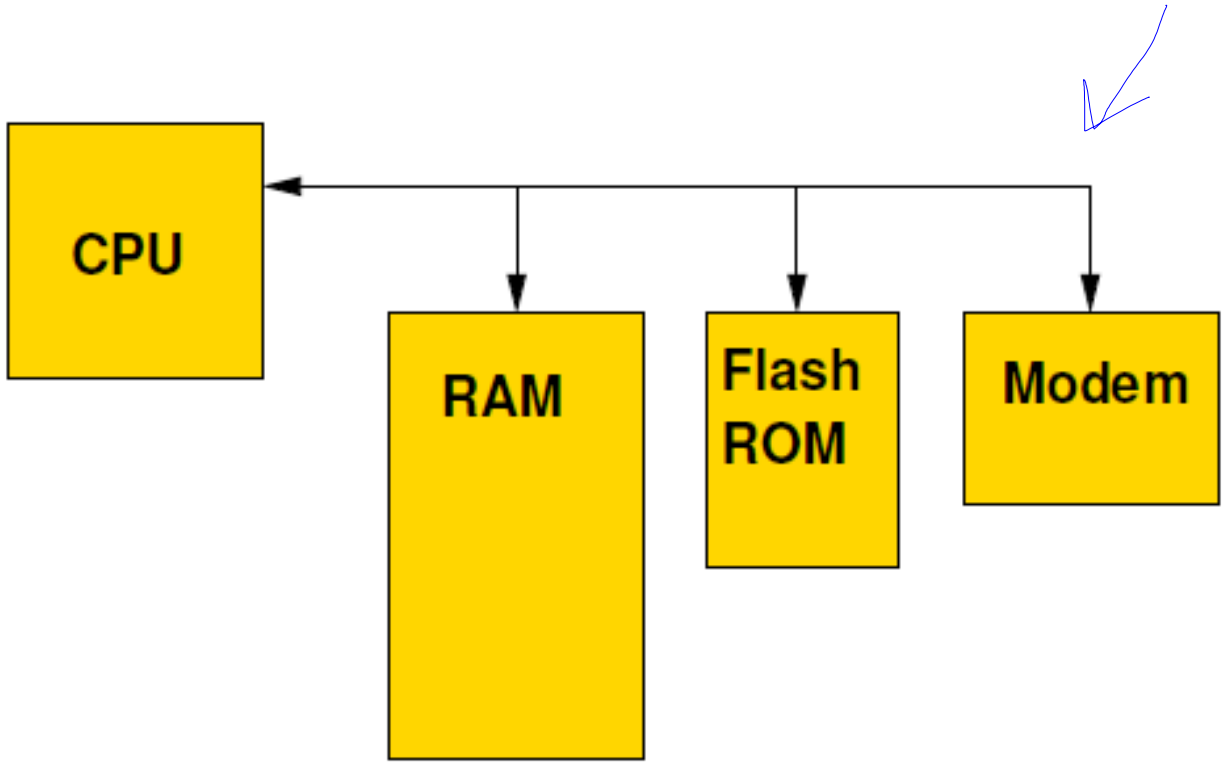
MIPS Speicherorganisation



Speichereinblendung von Ein-/Ausgabegeräten

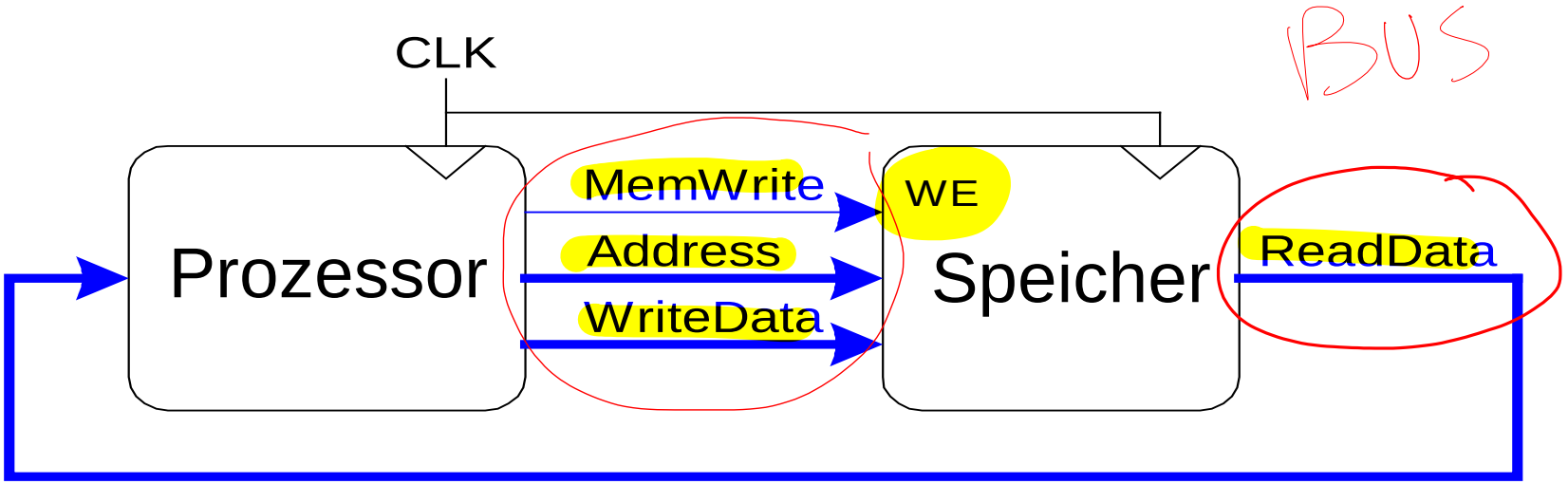
TECHNISCHE UNIVERSITÄT DARMSTADT

Benutzen den gleichen Bus wie der Speicher

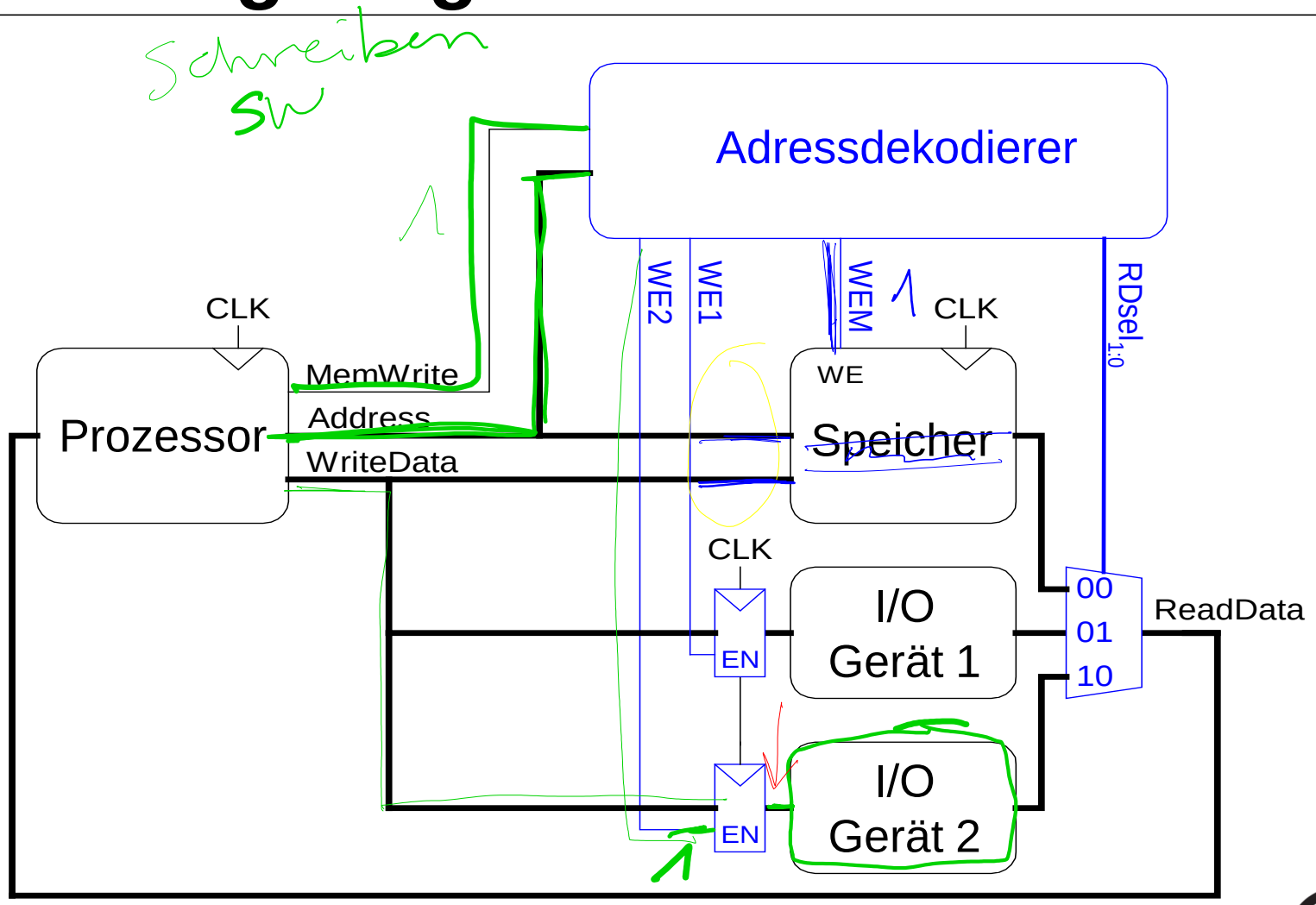


Embedded Systems & Applications

Speicherschnittstelle



Erweiterung der Hardware für Ein-/Ausgabegeräte



Hardware für Ein-/Ausgabegeräte

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Adressdekodierer:**
  - Dekodiert die Adresse um festzustellen, auf welches Gerät (E/A-Gerät oder Speicher) der Prozessor zugreifen möchte
- **Ein-/Ausgaberegister:**
  - Beim **Schreiben** auf ein Gerät: Enthält Werte, die der Prozessor zu den Ein-/Ausgabegeräten schreibt
- **Multiplexer:**
  - Beim **Lesen** von einem Gerät: Wählt zwischen Geräten und Speicher eine Quelle für die Daten aus, die dem Prozessor geschickt werden

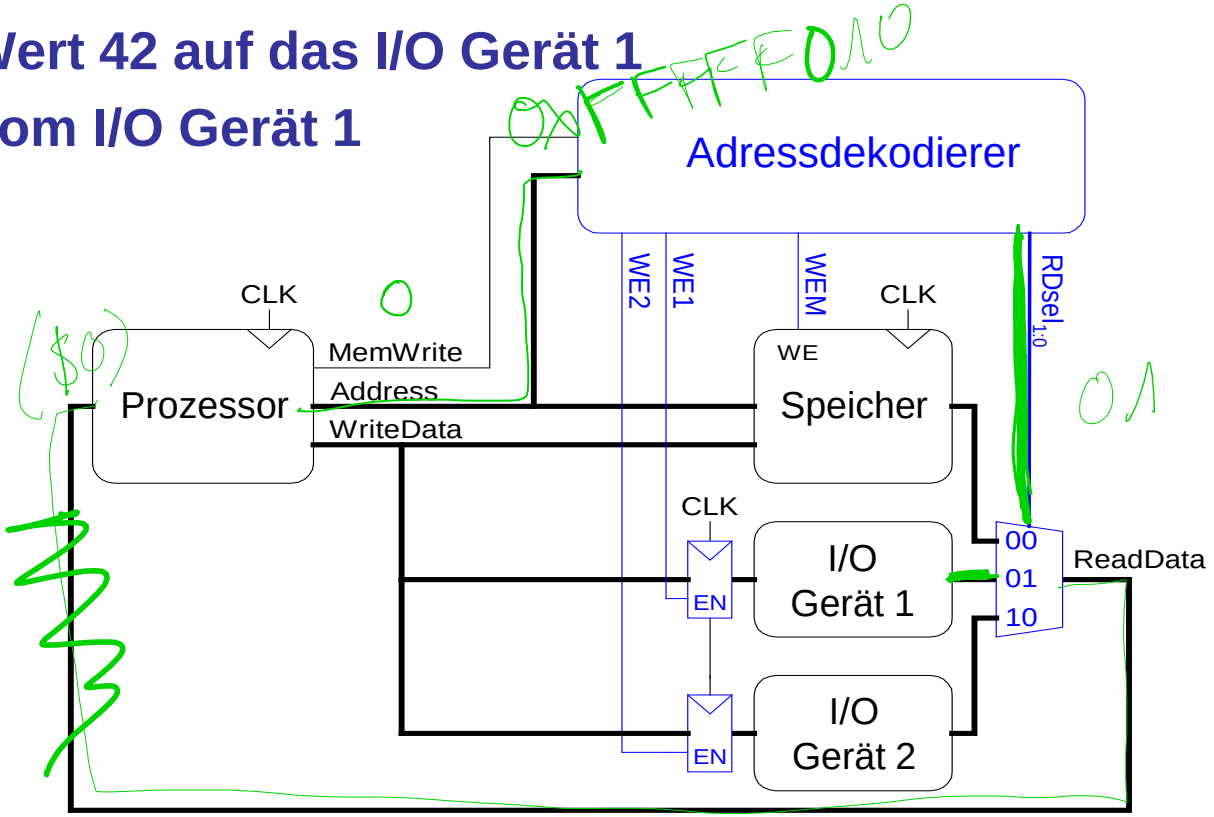
Embedded Systems & Applications

Beispiel: Zugriff auf Ein-/Ausgabegeräte

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: dem Ein-/Ausgabegerät 1 ist die Adresse 0xFFFFF010 zugewiesen

- **Übertragen Sie den Wert 42 auf das I/O Gerät 1**
- **Lesen Sie den Wert vom I/O Gerät 1 ins Register $t3**



Embedded Systems & Applications

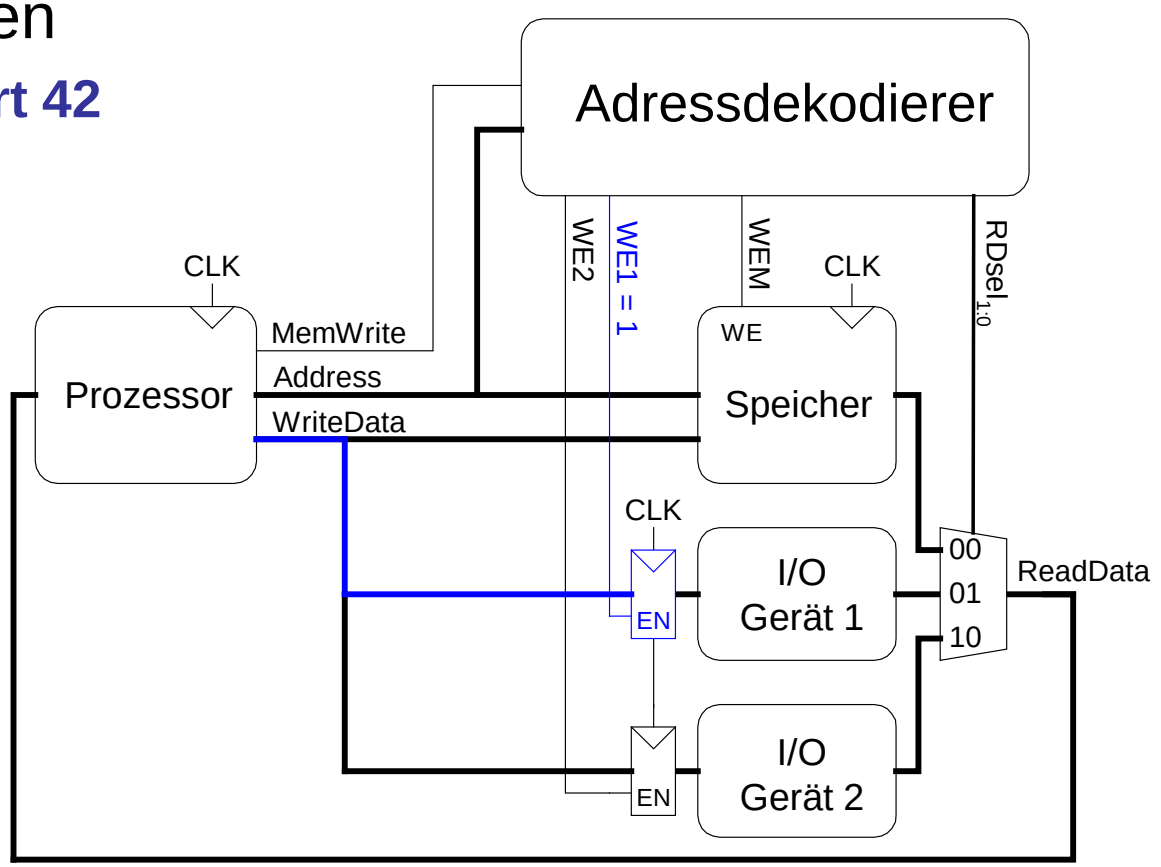
Beispiel: Zugriff auf Ein-/Ausgabegeräte

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: dem Ein-/Ausgabegerät 1 ist die Adresse 0xFFFFF010 zugewiesen

- **Übertragen Sie den Wert 42 auf das I/O Gerät 1**

```
addi $t0, $0, 42
sw $t0, 0xF010($0)
```



Embedded Systems & Applications

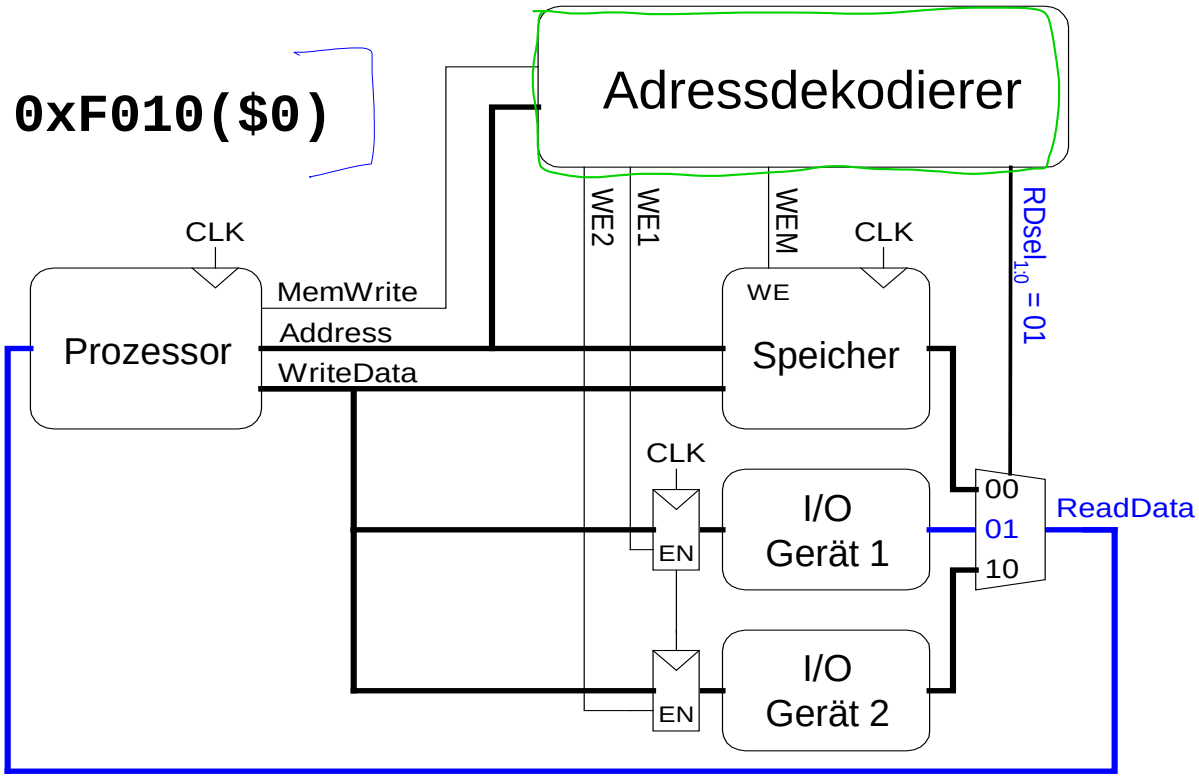
Beispiel: Zugriff auf Ein-/Ausgabegeräte

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: dem Ein-/Ausgabegerät 1 ist die Adresse 0xFFFFF010 zugewiesen

- **Lesen Sie den Wert vom I/O Gerät 1 ins Register $t3**

```
lw $t3, 0xF010($0)
```



Embedded Systems & Applications

Aufbau des Adressdekodierers

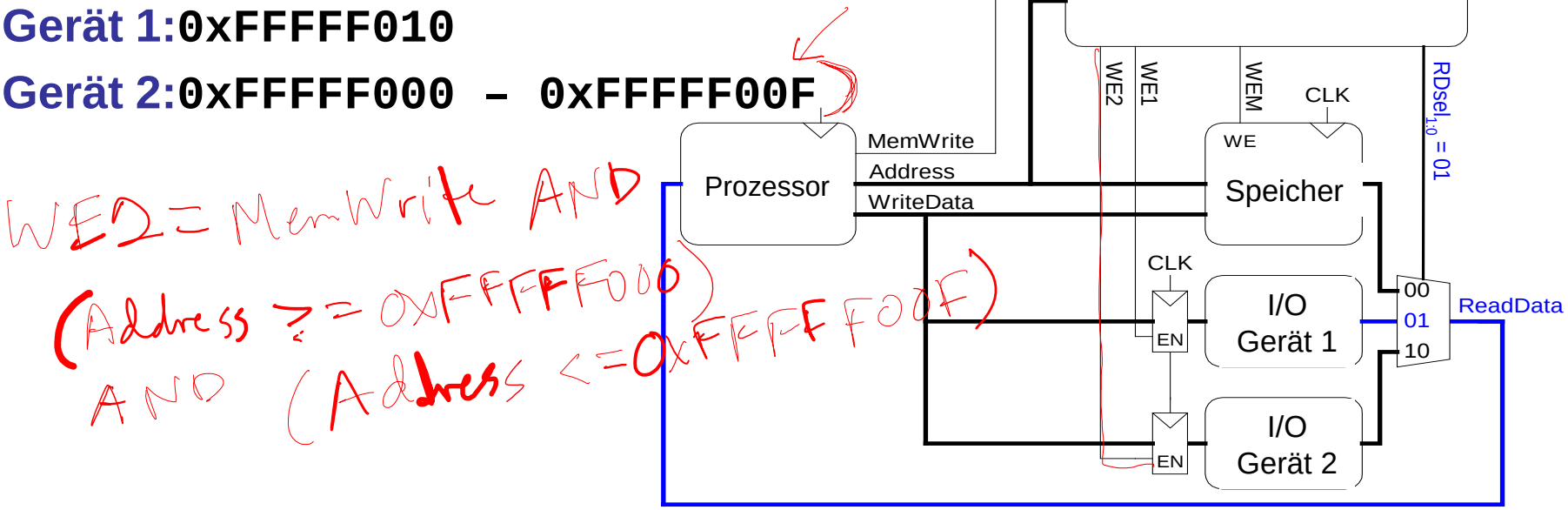
Adressbereiche:

Speicher: **0x00000000 – 0x9FFFFFFF**

I/O: **0xF0000000 – 0xFFFFFFFF**

Gerät 1: **0xFFFFF010**

Gerät 2: **0xFFFFF000 – 0xFFFFF00F**



WE2 = MemWrite AND
(Address >= 0xFFFFF000
AND (Address <= 0xFFFFF00F)

Aufbau des Adressdekodierers

Adressbereiche:

Speicher: **0x00000000 – 0x9FFFFFFF**

I/O: **0xF0000000 – 0xFFFFFFFF**

Gerät 1: **0xFFFFF010**

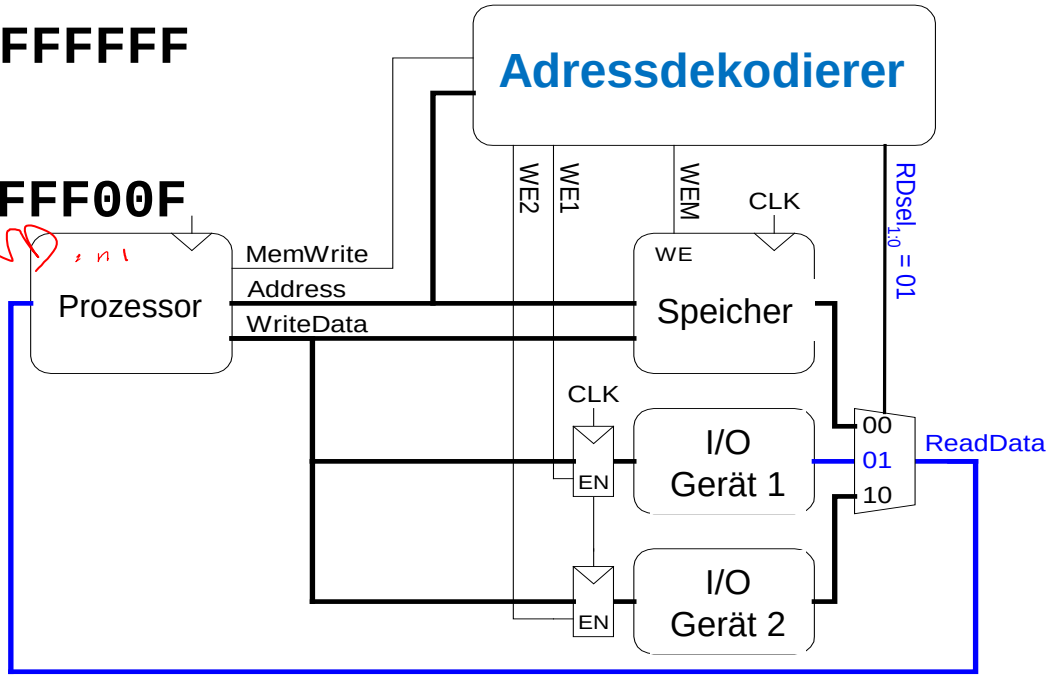
Gerät 2: **0xFFFFF000 – 0xFFFFF00F**

Naive Ansatz:

WE1 = (Address == 0xFFFFF010)

WE2 = (Address >= 0xFFFFF000) AND
(Address <= 0xFFFFF00F)

WEM= (Address >= 0x00000000) AND
(Address <= 0x9FFFFFFF)



Aufbau des Adressdekodierers

Adressbereiche:

Speicher: 0x00000000 – 0x9FFFFFFF

I/O: 0xF0000000 – 0xFFFFFFFF

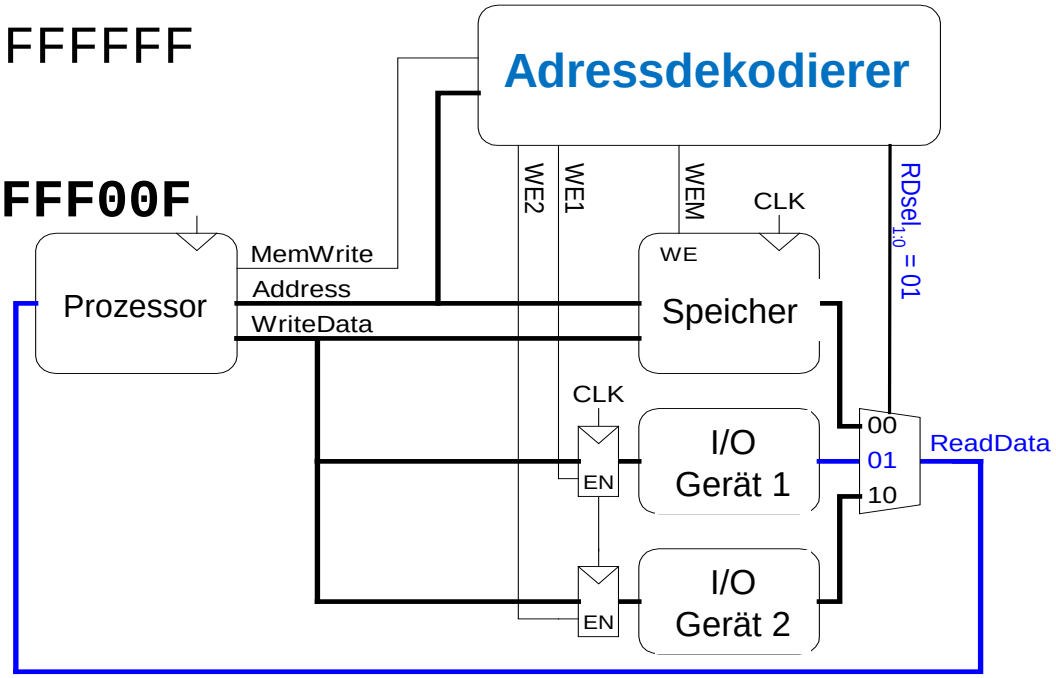
Gerät 1: 0xFFFFF010

Gerät 2: 0xFFFFF000 – 0xFFFFF00F

Besserer Ansatz:

SELIO = Address[31] AND Address[29]

SELM = ~(SELIO)



Aufbau des Adressdekodierers

TECHNISCHE UNIVERSITÄT DARMSTADT

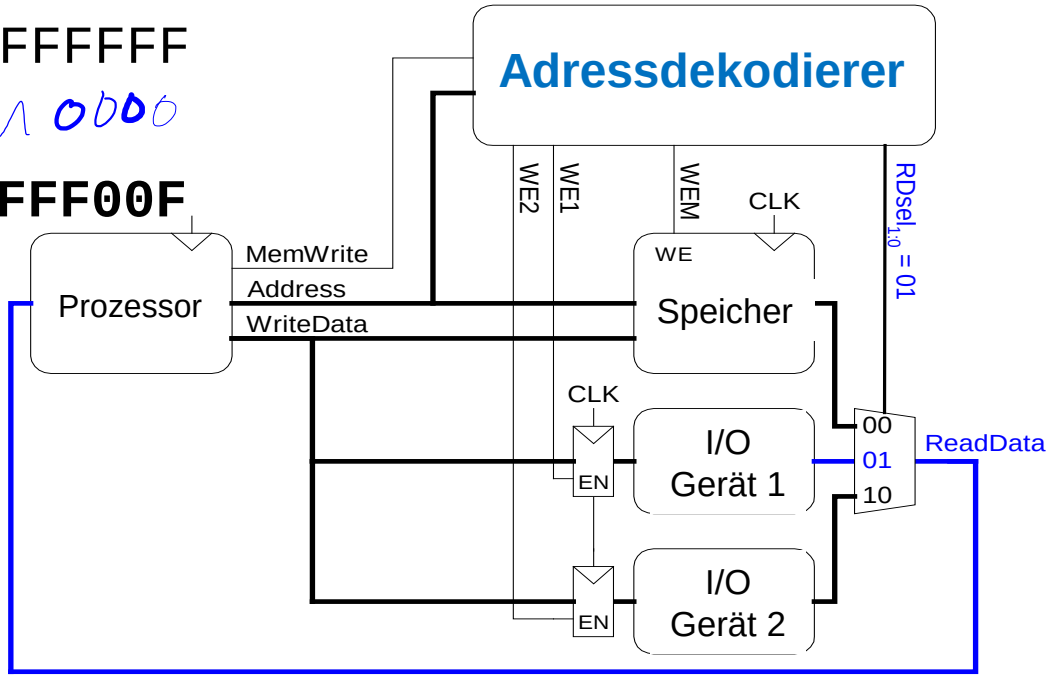
Adressbereiche:

Speicher: 0x00000000 – 0x9FFFFFFF

I/O: 0xF0000000 – 0xFFFFFFFF

Gerät 1: 0xFFFFF010

Gerät 2: 0xFFFFF000 – 0xFFFFF00F



Besserer Ansatz:

SELIO = Address[31] AND Address[29]

SEL1 = SELIO AND Address[4]

SEL2 = SELIO AND ~Address[4]

WE1 = SEL1 AND MemWrite

WE2 = SEL2 AND MemWrite

SELM = ~(SELIO)

WEM = SELM AND MemWrite

Embedded Systems & Applications

Aufbau des Adressdekodierers

Adressbereiche:

Speicher: `0x00000000 – 0x9FFFFFFF`

I/O: `0xF0000000 – 0xFFFFFFFF`

Gerät 1:`0xFFFFF010`

Gerät 2:`0xFFFFF000 – 0xFFFFF00F`

Besserer Ansatz:

SELIO = Address[31] AND Address[29]

SEL1 = SELIO AND Address[4]

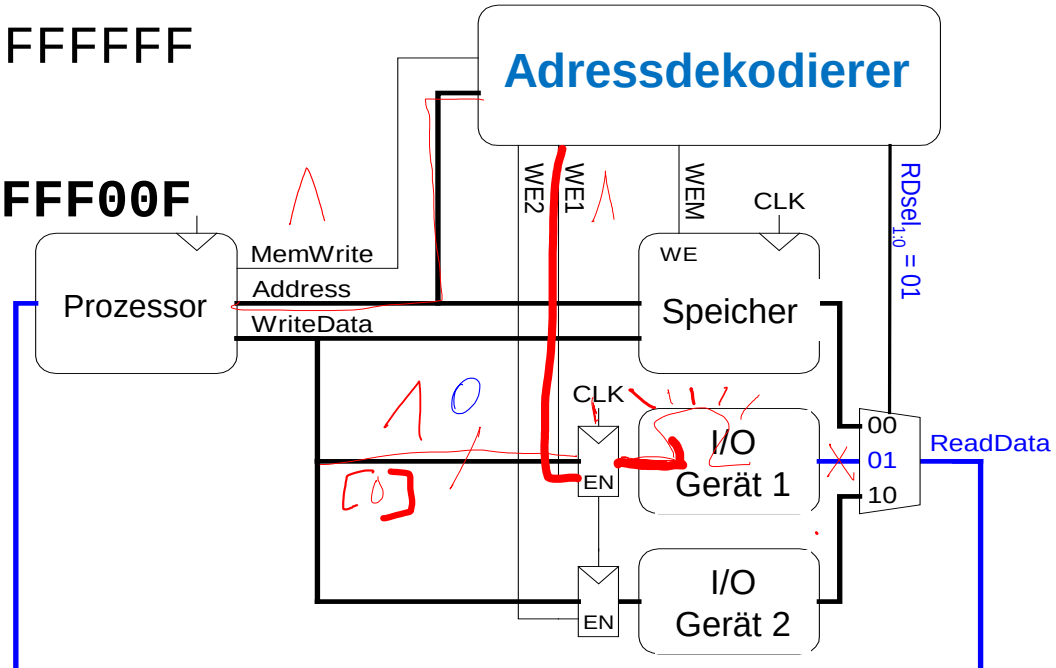
SEL2 = SELIO AND ~Address[4]

WE1 = SEL1 AND MemWrite

WE2 = SEL2 AND MemWrite

SELM = ~(SELIO)

WEM = SELM AND MemWrite



if (SEL1) **ReadData** = 01

else if (SEL2) **ReadData** = 10

else **ReadData** = 00

Idee von Bussen

TECHNISCHE UNIVERSITÄT DARMSTADT

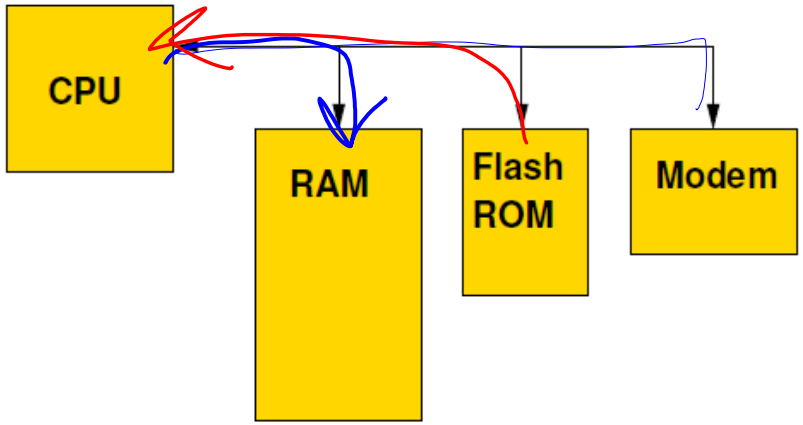
Gemeinsam genutztes Medium

Vorteile:

- Einfache Realisierung
- Wenig Chip-Fläche

Nachteile:

- Nur eine Verbindung gleichzeitig (Engpass – bottleneck)



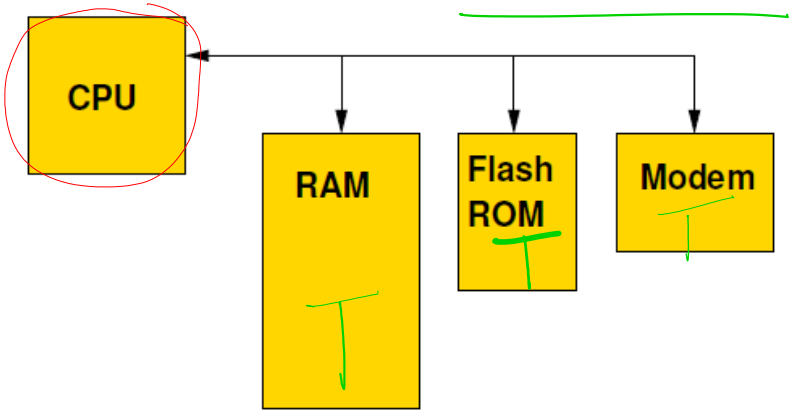
Embedded Systems & Applications

Busorganisation: Initiator/Master und Targets/Slaves

TECHNISCHE UNIVERSITÄT DARMSTADT

Bus ist Gemeinsame Verbindungen zwischen Komponenten

- Maximal ein **Initiator** / Master gleichzeitig
  - Veranlasst Aktivitäten auf Bus
- Ein oder mehrere **Targets** / Slaves
  - Reagieren auf Aktivitäten auf dem Bus
- **Grundlegende Transaktionen** auf dem Bus
  - Initiator fordert Daten von Target an: Lesezugriff
  - Initiator überträgt Daten zum Target: Schreibzugriff



Embedded Systems & Applications

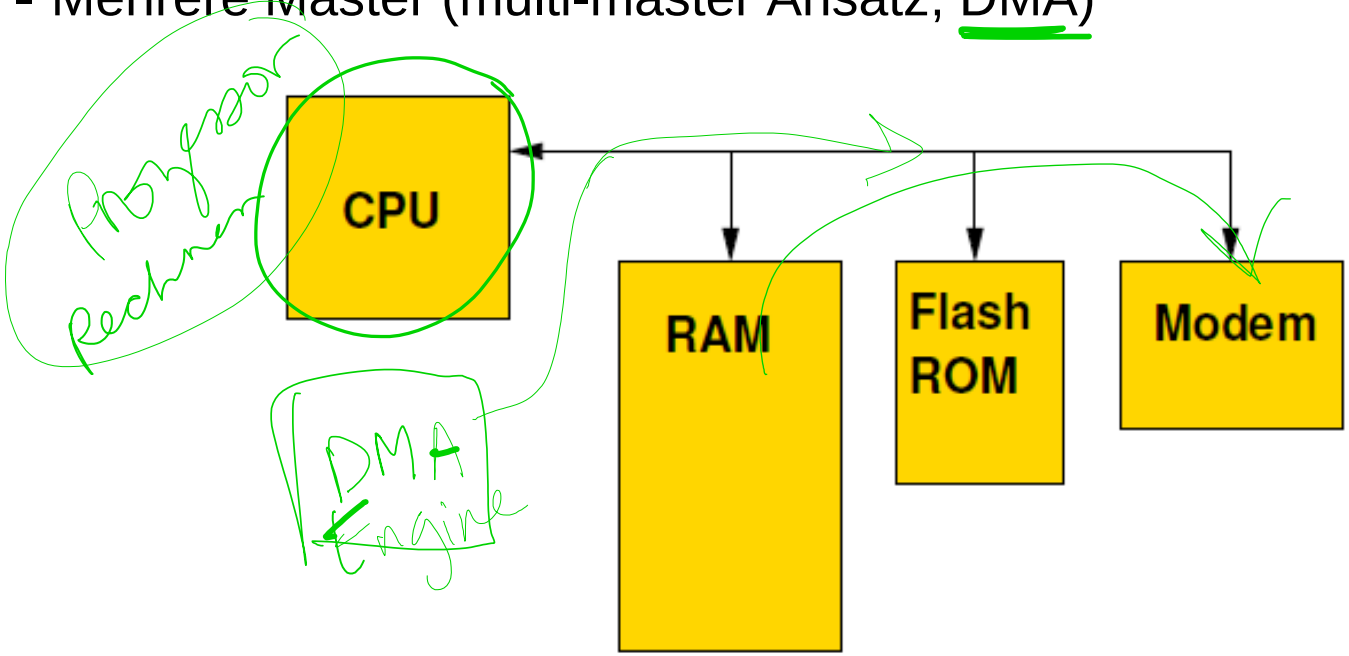
Busorganisation

In einfachen Systemen

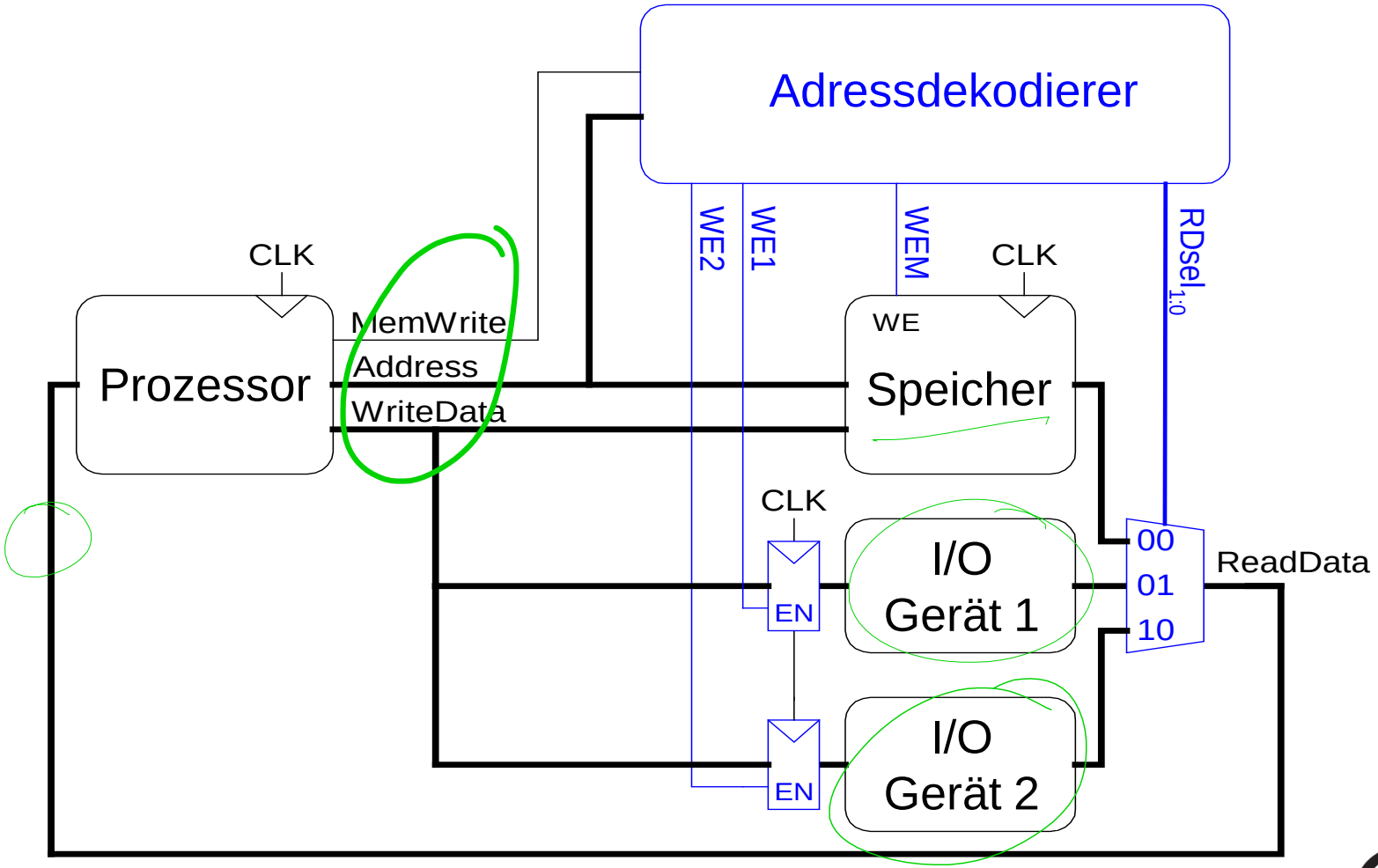
- Nur ein Master, häufig die CPU

In leistungsstärkeren Systemen

- Mehrere Master (multi-master Ansatz, DMA)



Wiederholung: Hardware für Ein-/Ausgabegeräte



Zusammenfassung der Veranstaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

Lehrstoff:

- Rechnerarchitektur – Assemblersprache (**Kapitel 6**)
- Mikroarchitektur – Aufbau eines Prozessors (**Kapitel 7**)
- Speichersysteme (**Kapitel 8**)

Eine zusammenfassung der Hauptthemen folgt, aber Sie sollten **alle die Folien** der Kapiteln genau anschauen als Vorbereitung für die Klausur.

Kapitel 6: Themen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Assembler-Sprache
- Maschinensprache
- Programmierung
- Adressierungsmodi
- Compilieren, Assemblieren und Linken
- Dies und Das: Pseudobefehle, Ausnahmebehandlung, Gleitkommadarstellung, u.s.w.

Embedded Systems & Applications

Assemblersprache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Rechnen: `add`, `sub`, `and`, `or`, `slt`, `sll`, `sllv`, `addi`, u.s.w.
- Speicherzugriffe: `lw`, `lh`, `lb`, `sw`, `sh`, `sb`
- Sprung-/Verzweigungsbefehle: `beq`, `bne`, `j`, `jal`, `jr`, u.s.w.

MIPS Machinensprache: Befehlsformate

TECHNISCHE UNIVERSITÄT DARMSTADT

R-Typ

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Typ

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

J-Typ

op	addr
6 bits	26 bits

Embedded Systems & Applications

Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Häufige Konstrukte:

- if/else-Anweisungen
- while-Schleifen
- for-Schleifen
- Feld (Array) zugriffe
- Prozeduraufrufe
- Stack

Embedded Systems & Applications

Adressierungsarten

TECHNISCHE UNIVERSITÄT DARMSTADT

Operanden für Befehle:

- Aus einem Register

  Beispiel: `add $s0, $t1, $s7`

- Direktwert aus Instruktion

  Beispiel: `addi $s3, $0, -28`

- Relativ zu einer Basisadresse

  Beispiel: `lw $t0, 7($t2)`

- Relativ zum Programmzähler (PC + 4)

  Beispiel: `beq $s0, $t4, Loop`

- Pseudodirekt

  Beispiel: `j  done`

Embedded Systems & Applications

Kapitel 7: Themen

TECHNISCHE UNIVERSITÄT DARMSTADT

Microarchitektur

- Analyse der Rechenleistung
- Ein-Takt-Prozessor
- Mehrtakt-Prozessor
- Pipeline-Prozessor
- Ausnahmebehandlung

Embedded Systems & Applications

Rechenleistung eines Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausführungszeit eines Programms

Ausführungszeit = (#Instruktionen)(Takte/Instruktion)(Sekunden/Takt)

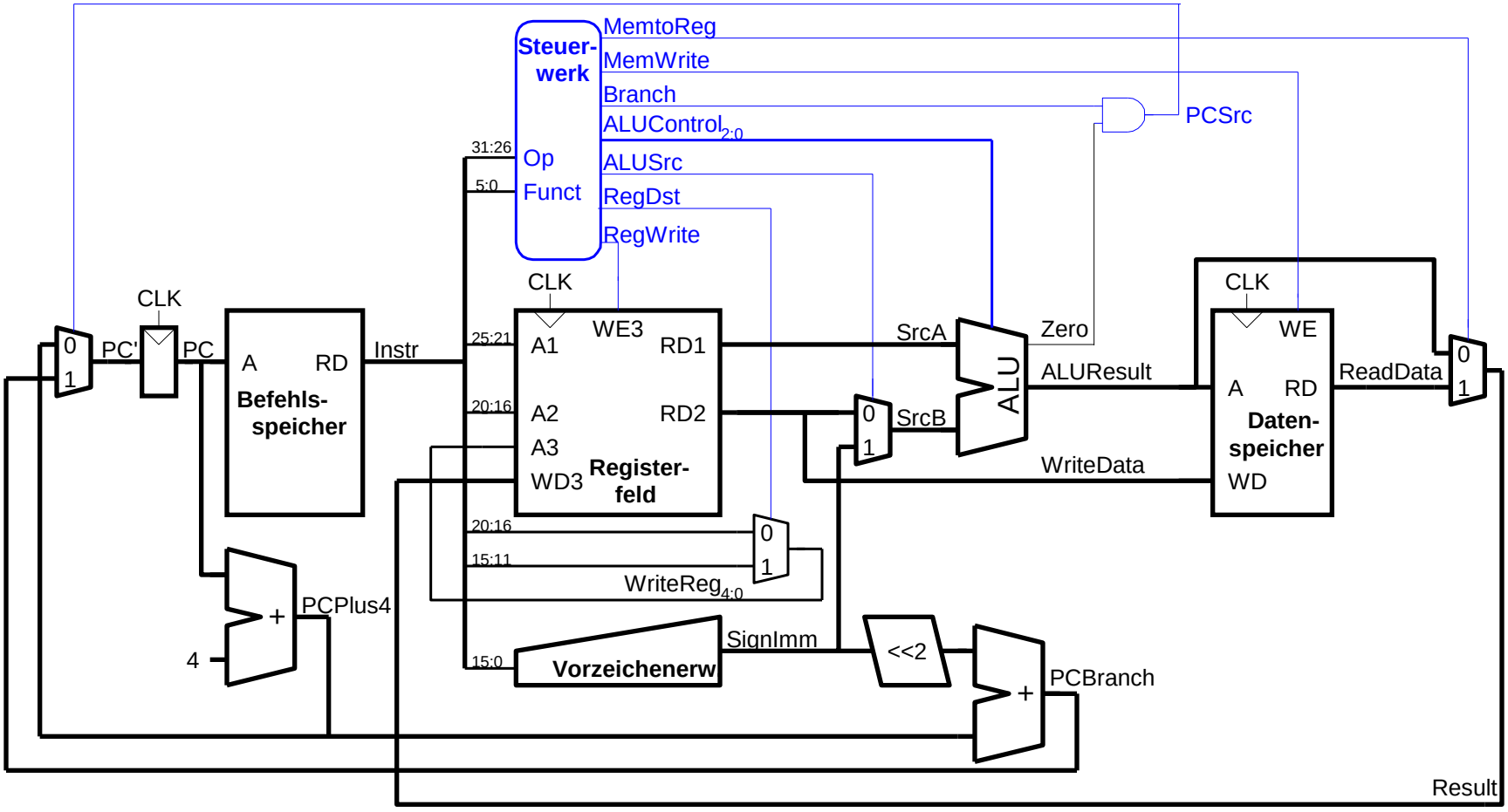
= #Instruktionen x CPI x T_c

Definitionen:

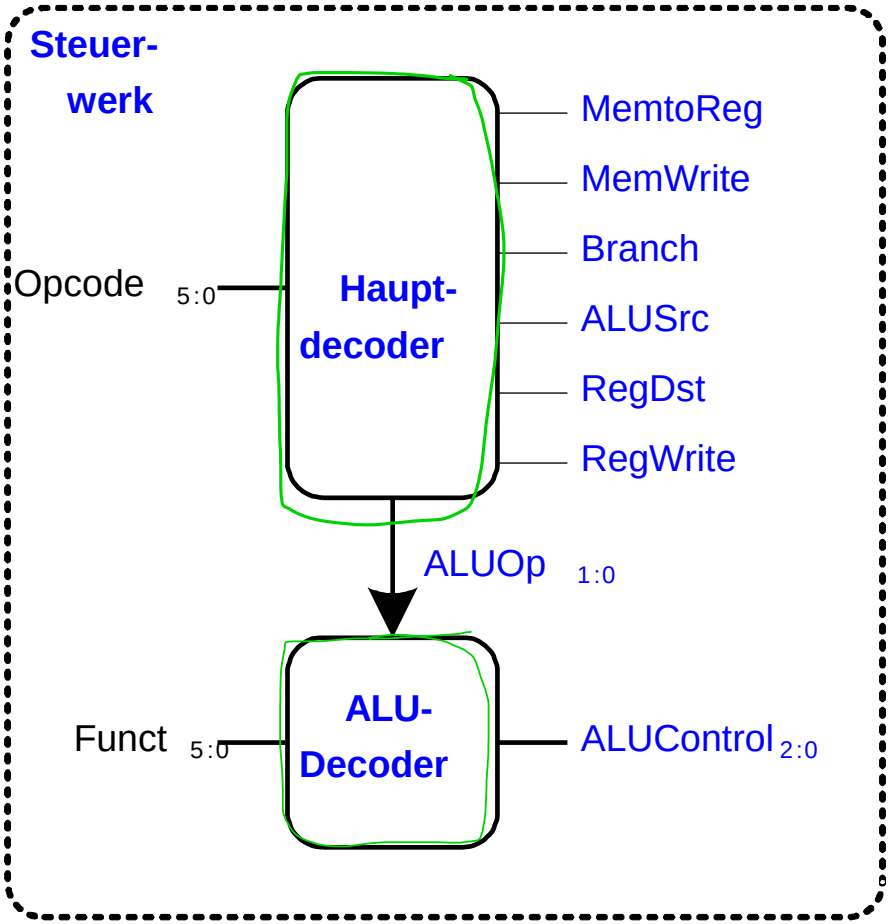
- Takte/Instruktion = **CPI** (*cycles per instruction*)
- Sekunden/Takt = **Taktperiode**

Embedded Systems & Applications

MIPS Ein-Takt-Prozessor



Steuerwerk


Steuer-
werk
Opcode
5:0
Haupt-
decoder
MemtoReg
MemWrite
Branch
ALUSrc
RegDst
RegWrite
ALUOp
1:0
Funct
5:0
ALU-
Decoder
ALUControl
2:0

Steuerwerk: ALU-Decoder

$ALUOp_{1:0}$	Bedeutung
00	Addiere
01	Subtrahiere
10	Werte Funct-Feld aus
11	unbenutzt

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (add)	010 (Add)
1X	100010 (sub)	110 (Subtract)
1X	100100 (and)	000 (And)
1X	100101 (or)	001 (Or)
1X	101010 (slt)	111 (SLT)

Steuerwerk: Hauptdecoder

TECHNISCHE UNIVERSITÄT DARMSTADT

Instruktion	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-Typ	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (`add`)	010 (Add)
1X	100010 (`sub`)	110 (Subtract)
1X	100100 (`and`)	000 (And)
1X	100101 (`or`)	001 (Or)
1X	101010 (`slt`)	111 (SLT)

$ALUOp_{1:0}$	Bedeutung
00	Addiere
01	Subtrahiere
10	Werte Funct-Feld aus
11	unbenutzt

Embedded Systems & Applications

Rechenleistung des Ein-Takt-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

- **CPI = 1**
- **Kritischer Pfad:**

$$T_c = t_{pcq\_PC} + t_{mem} + \max(t_{RFread}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$

In vielen Implementierungen: Kritischer Pfad durch

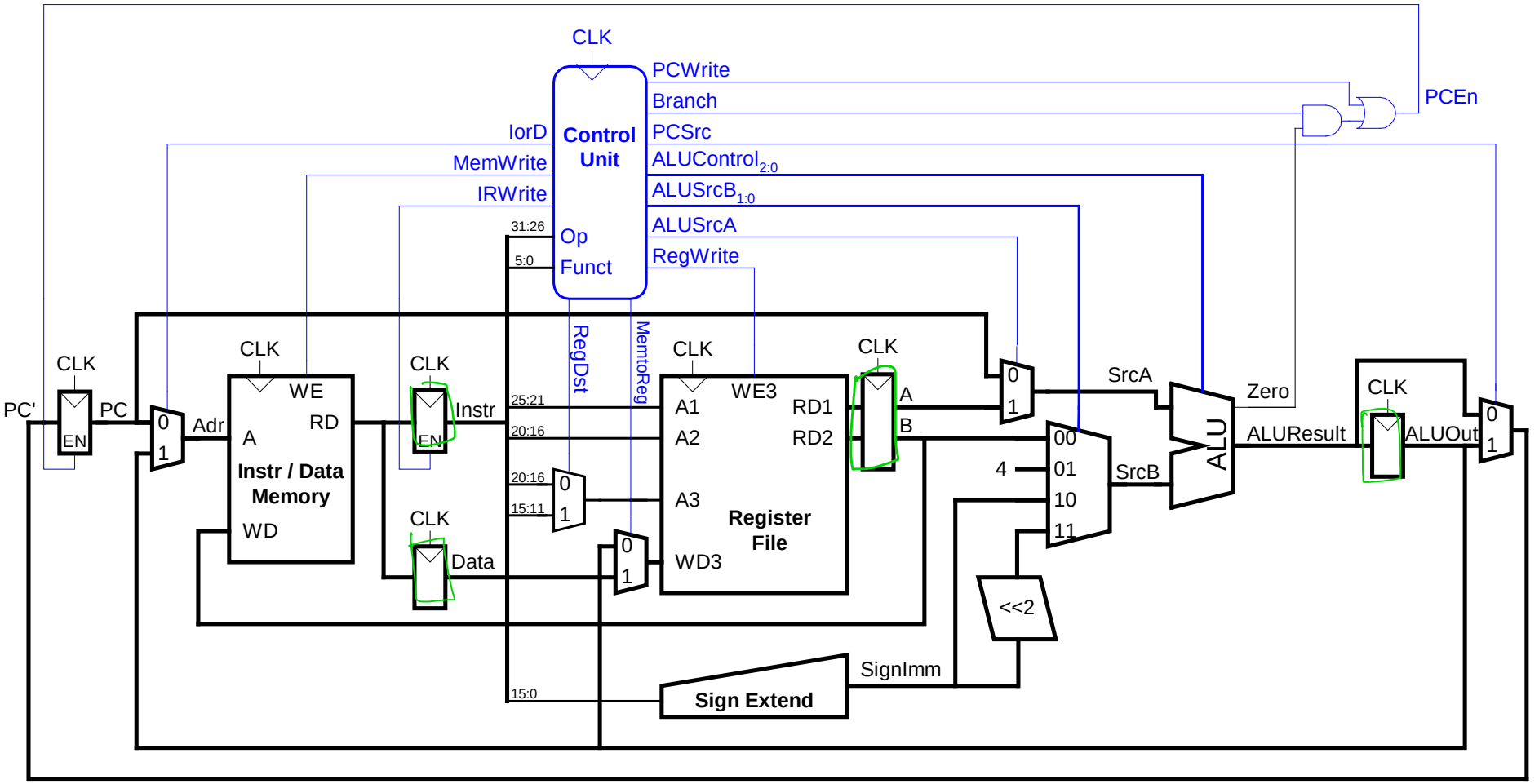
- Speicher, ALU, Registerfeld

Damit:

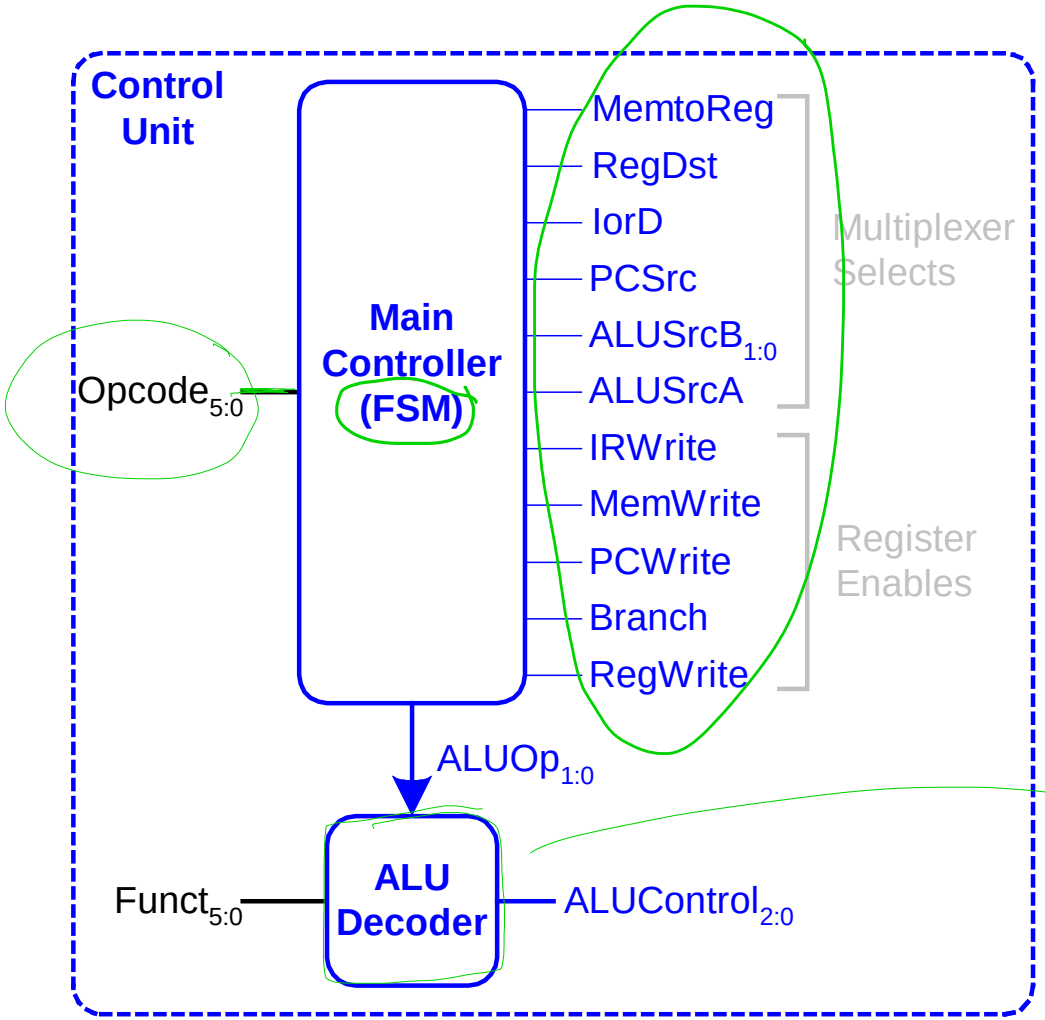
- $\mathbf{T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}}$

Embedded Systems & Applications

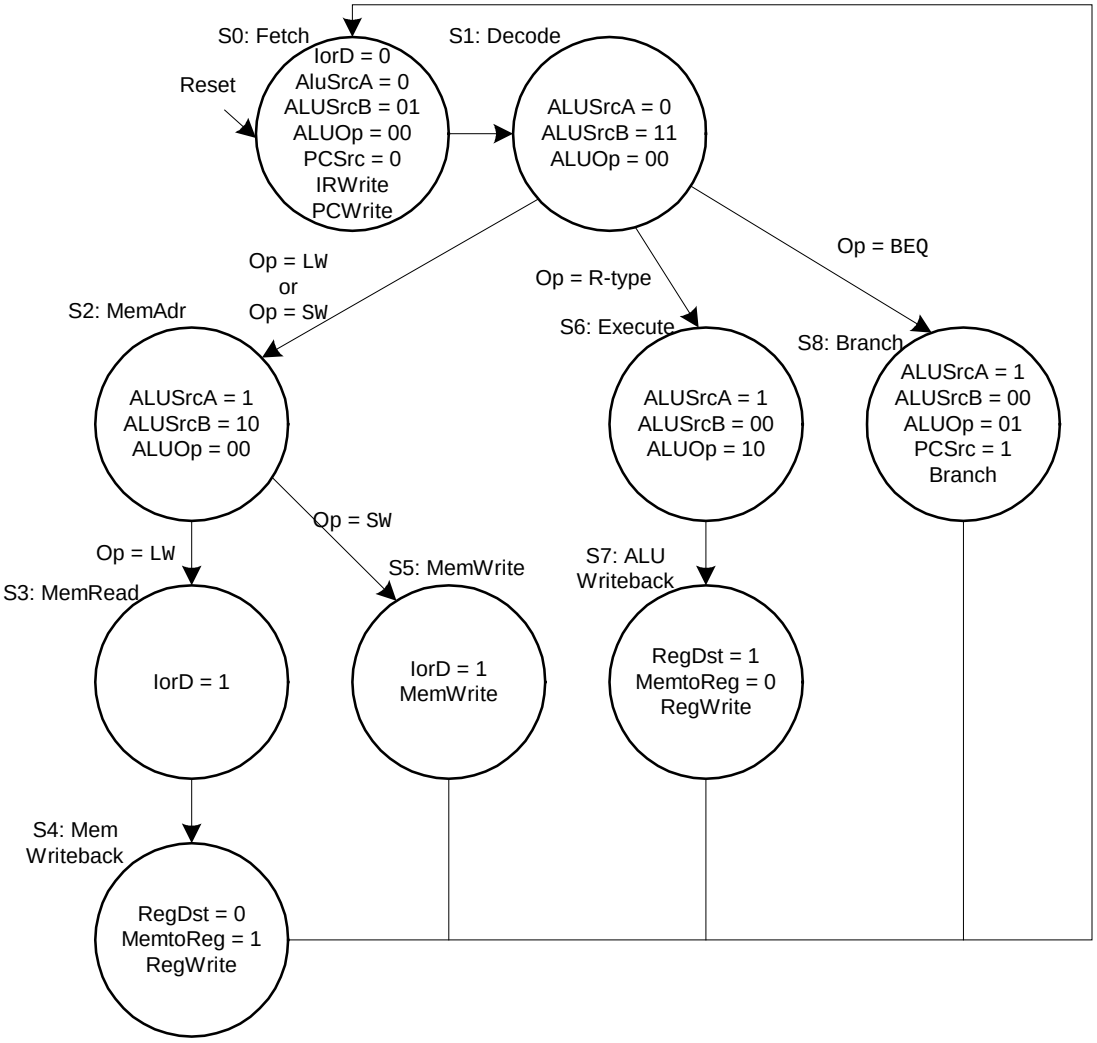
MIPS Mehrtaktprozessor



Steuerwerk Mehrtaktprozessor



Hauptsteuerwerk für Mehrtakt-CPU



FSM

Rechenleistung des Mehrtaktprozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Instruktionen benötigen unterschiedliche viele Takte:

- 3 Takte : `beq`, `j`
- 4 Takte : R-Typ, `sw`, `addi`
- 5 Takte : `lw`

CPI wird bestimmt als gewichteter Durchschnitt

SPECint 2000 Benchmark:

- 25% Laden
- 10% Speichern
- 11% Verzweigungen
- 2% Sprünge
- 52% R-Typ

Durchschnittliche CPI = (0,11 + 0,02)(3) + (0,52 + 0,10)(4) + (0,25)(5) = 4,12

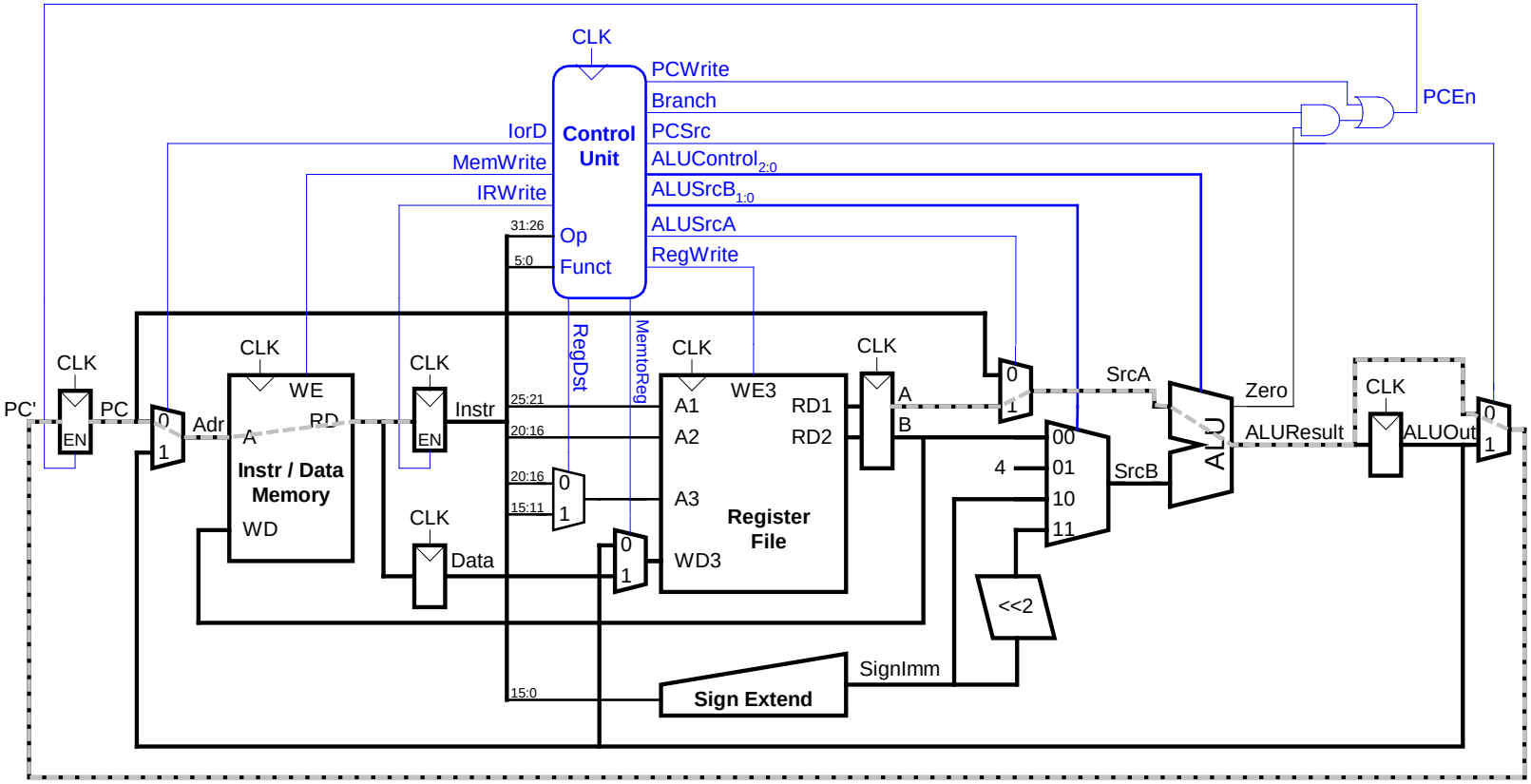
Embedded Systems & Applications

Rechenleistung des Mehrtaktprozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

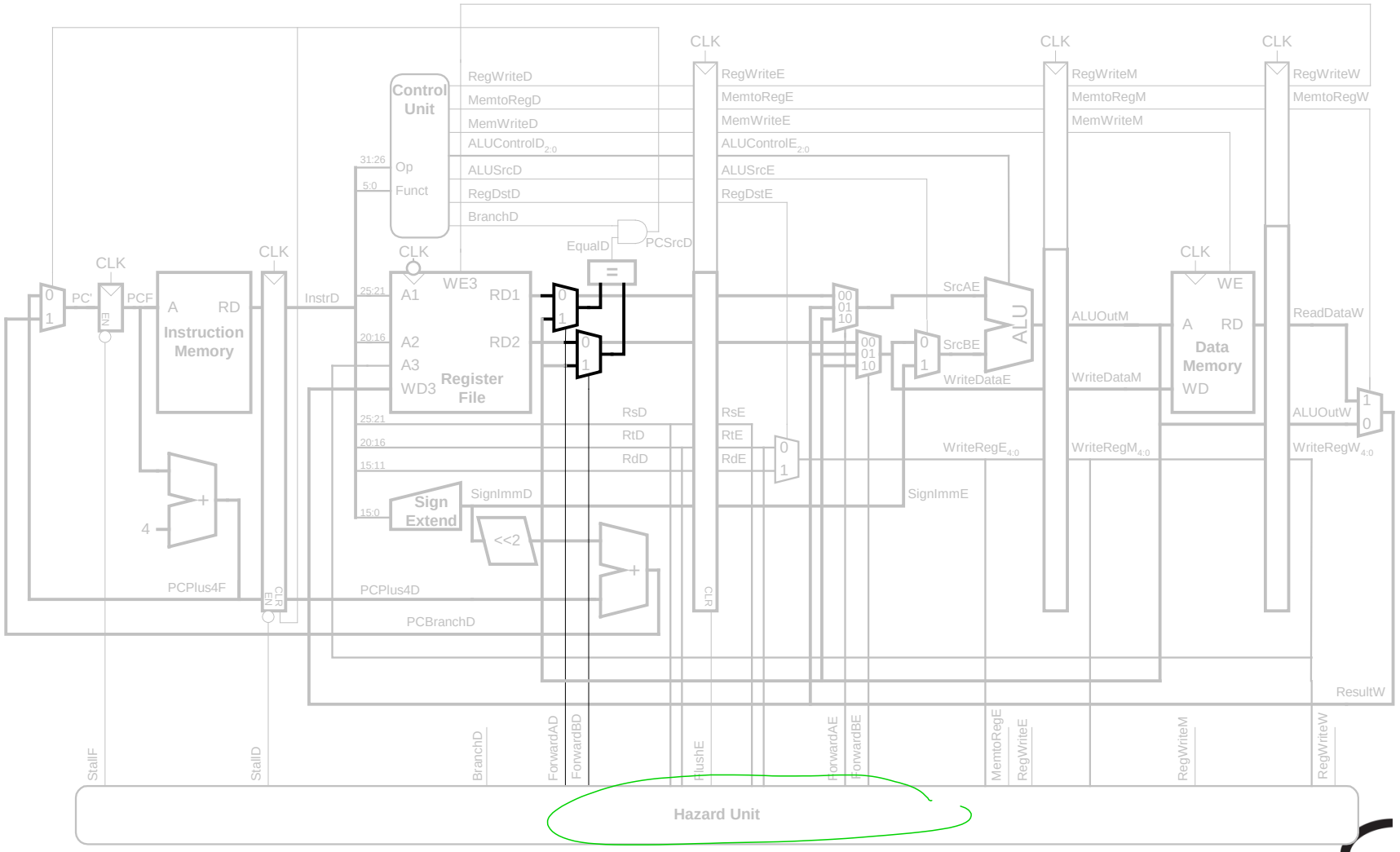
Kritischer Pfad :

$$T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$



Embedded Systems & Applications

MIPS Pipeline Prozessor



Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Idealerweise: CPI = 1

- SPECint 2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

- Annahmen:
  - 40% der geladenen Daten werden gleich in der nächsten Instruktion gebraucht
  - 25% aller Verzweigungen werden falsch vorhergesagt
  - Alle Sprünge erzeugen eine zu entfernende (*flush*) Instruktion

- **Wie hoch ist der durchschnittliche CPI-Wert?**
  - Lade/Verzweigungsinstruktionen haben CPI = 1 ohne Stall, = 2 mit Stall. Daher:
  - CPI_{lw} = 1 (0,6) + 2 (0,4) = 1,4
  - CPI_{beq} = 1 (0,75) + 2 (0,25) = 1,25
  - Also:

Durchschnittliche CPI = (0,25) (1,4) + (0,1) (1,0) + (0,11)(1,25) + (0,02) (2,0) + (0,52)(1,0)

= 1,15

Embedded Systems & Applications

Beispiel: Rechenleistung des Pipelined-Prozessors

TECHNISCHE UNIVERSITÄT DARMSTADT

Kritischer Pfad des Pipelined-Prozessors:

$$
\begin{aligned}
T_c = \max \{ & \\
& t_{pcq} + t_{mem} + t_{setup}, && \text{Fetch} \\
& 2\,(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup}), && \text{Decode} \\
& t_{pcq} + t_{mux} + t_{mux} + t_{mux} + t_{ALU} + t_{setup}, && \text{Execute} \\
& t_{pcq} + t_{memwrite} + t_{setup}, && \text{Memory} \\
& 2\,(t_{pcq} + t_{mux} + t_{RFwrite}) \} && \text{Writeback}
\end{aligned}
$$

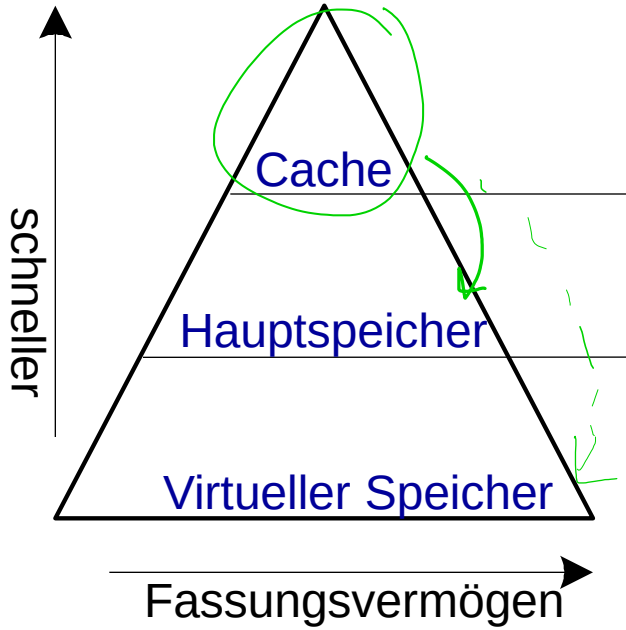
Embedded Systems & Applications

Kapitel 8: Themen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Speichersysteme: Hierarchie
- Leistungsvergleich von Speichersystemen
- Caches
- Virtueller Speicher
- Speichereinblendung von Ein-/Ausgabegeräten

Wiederholung: Speicherhierarchie



Technologie	Kosten/GB	Zugriffszeit
SRAM	~ $10,000	~ 1 ns
DRAM	~ $10	~ 10-50 ns
Festplatte	~ $0.1	~ 10,000,000 ns
SSD	~ $1	~ 100,000 ns

Durch Hierarchie, emuliert das Speichersystem Speicher der **schnell, gross, und günstig** ist.

Leistung eines Speichersystems

Hit-Rate = # hits / # Speicherzugriffe
= 1 – Miss Rate

Miss-Rate (*MR*) = # misses / # Speicherzugriffe
= 1 – Hit Rate

Durchschnittliche Speicherzugriffszeit (*average memory access time, AMAT*): Durchschnittliche Zeit, die der Prozessor braucht, um auf ein Datum zuzugreifen

AMAT = $t_{\text{cache}} + MR_{\text{cache}} (t_{MM} + MR_{MM} t_{VM})$

MM = *main memory*, Hauptspeicher
VM = *virtual memory*, Virtueller Speicher

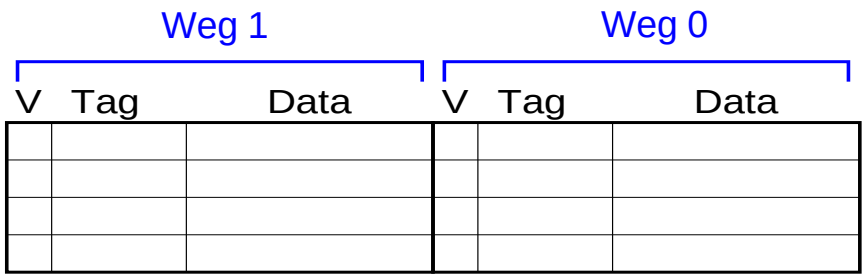
Cache Arten

TECHNISCHE UNIVERSITÄT DARMSTADT

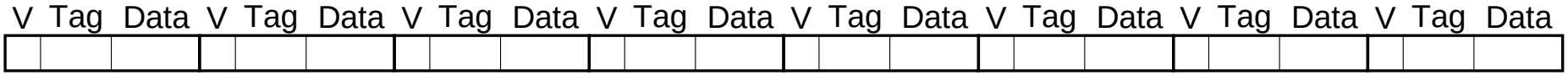
Caches werden klassifiziert nach Anzahl von Blöcken in einer Menge:

- **Direktabgebildet (*direct mapped*):** Ein Block pro Menge

- ***N*-Wege Mengenassoziativ (*N-way set associative*):** *N* Blöcke pro Menge



- **Vollassoziativ (*fully associative*):** Alle Cache Blöcke in einer Menge



Jede Speicheradresse wird genau auf **eine** Menge abgebildet

Embedded Systems & Applications

Zusammenfassung von Caches

- Kapazität: ***C***
- Blockgröße: ***b***
- Blockanzahl: ***B*** = *C*/*b*
- *Assoziativitätsgrad*: ***N***
- Anzahl von Mengen (Sets): ***S*** = *B*/*N*

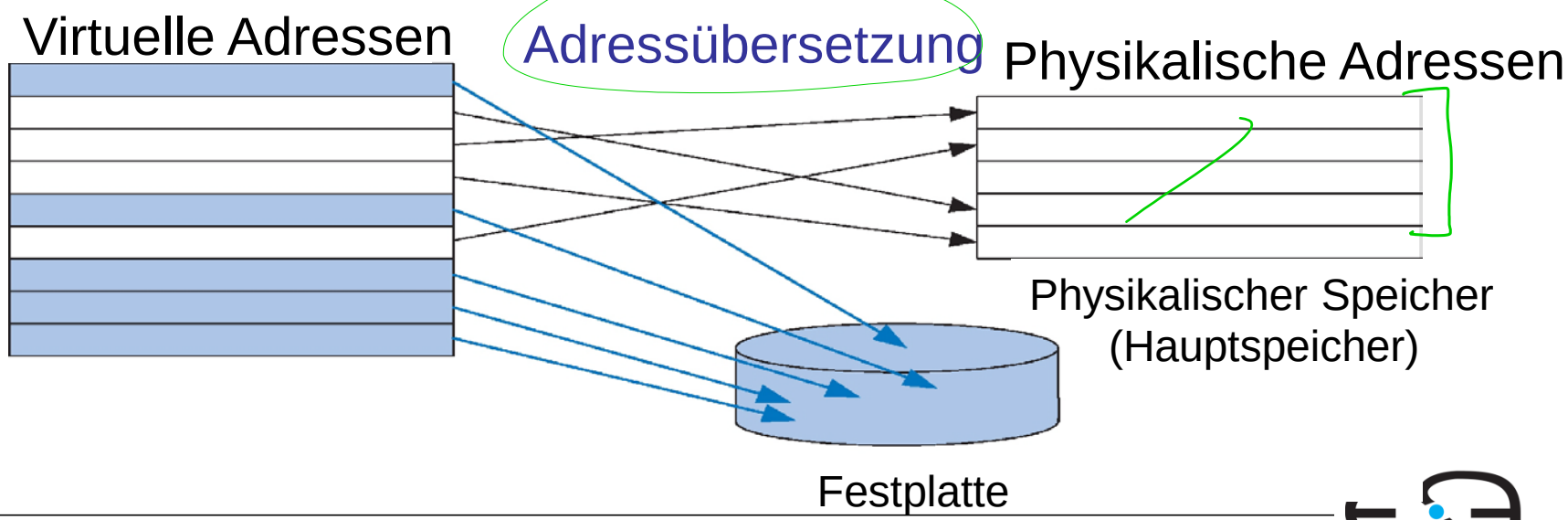
Name	Assoziativitätsgrad (*N*)	Anzahl von Mengen (*S* = *B*/*N*)
Direktabgebildete	1	*B*
N-Wege Mengenassoziativ	1 < *N* < *B*	*B* / *N*
Vollassoziativ	*B*	1

Virtueller Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

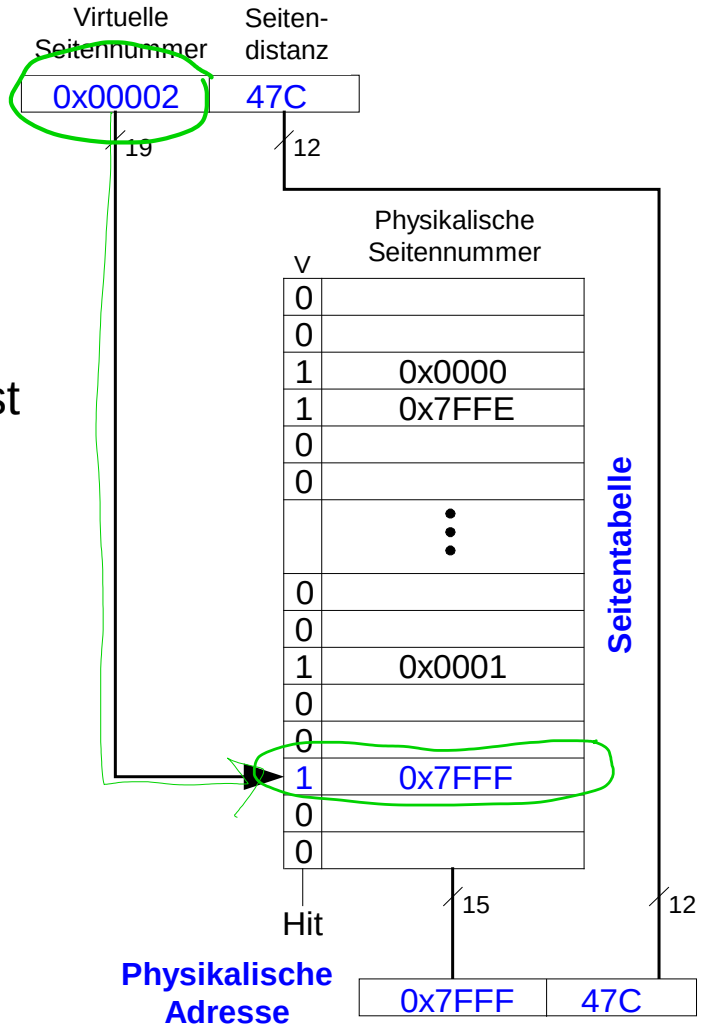
Jedes Programm erzeugt **virtuelle Adressen**

- Der ganze Bereich der virtuellen Adressen steht auf der Festplatte
- Eine Untermenge der virtuellen Adressen steht im Hauptspeicher
- Der Prozessor erzeugt virtuelle Adressen und muss diese in physikalische Adressen übersetzen (TLB, Seitentabelle)
- Daten, die im DRAM nicht gefunden sind, müssen von der Festplatte geholt werden



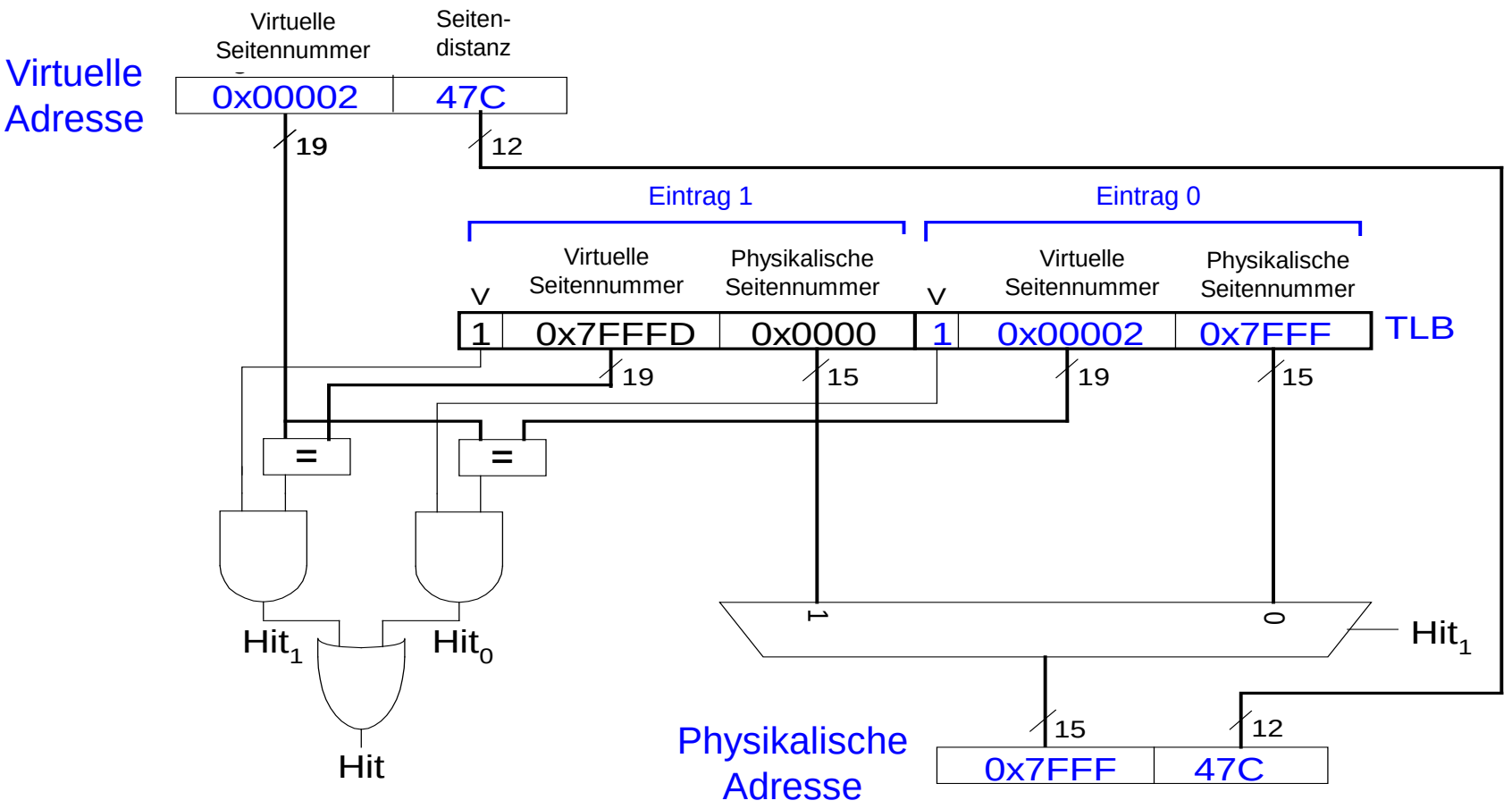
Embedded Systems & Applications

Beispiel: Seitentabelle (Page Table)

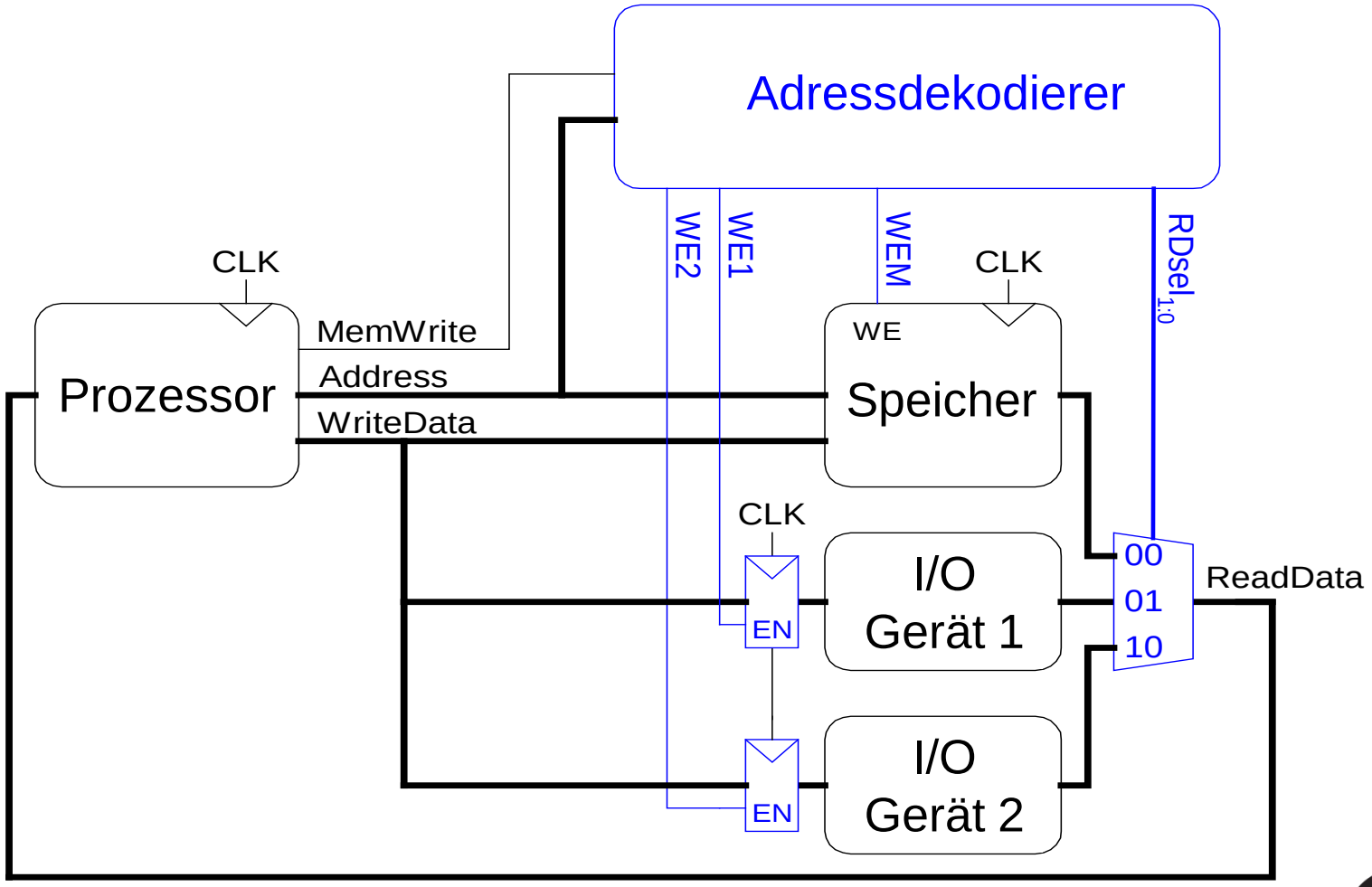


VSN (Virtuelle Seitennummer) ist der **Index** in der Seitentabelle

Beispiel: TLB mit zwei Einträgen



Speichereinblendung von Ein-/Ausgabegeräten



Zusammenfassung der Veranstaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

Lehrstoff:

- Rechnerarchitektur – Assemblersprache (**Kapitel 6**)
- Mikroarchitektur – Aufbau eines Prozessors (**Kapitel 7**)
- Speichersysteme (**Kapitel 8**)

Rechnersysteme und Digitalegeräte werden immer mehr benutzt

- Jetzt haben Sie die Werkzeuge, diese auch zu *verstehen*

  … und auch mit zu entwerfen!

Embedded Systems & Applications