Einführung in die technische Informatik

Christopher Kruegel chris@auto.tuwien.ac.at

http://www.auto.tuwien.ac.at/~chris

Betriebssysteme

- Aufgaben
  - Management von Ressourcen
  - Präsentation einer einheitlichen Schnittstelle für Anwendungen
- 4 wichtige Gebiete
  - Prozessmanagement
  - *Speichermanagement*
  - Dateisystem
  - Eingabe und Ausgabe
- System Calls
  - Funktionen, die ein Betriebssystem den Anwendungen zur Verfügung stellt

Speicherverwaltung

- Ideal
  - Speicher ist groß
  - Speicher ist schnell (niedrige Zugriffszeiten)

- Praxis
  - Speicherhierarchie
    - schneller Cache
    - mittlerer Hauptspeicher
    - langsamer Plattenspeicher

  - physikalischer Speicherplatz muss unter Prozessen aufgeteilt werden

Speicherverwaltung

- das Betriebssystem stellt jedem Prozess exklusiv einen riesigen Speicherbereich zur Verfügung
  - üblicherweise, das Maximum des adressierbaren Bereichs
  - bei 32-bit Architektur, d.h., 32-bit Adressen → 4 Gbyte
  - dieser Speicher ist allerdings nicht wirklich vorhanden, also nur virtuell
  - ➢ virtuelle Adressen

- physikalischer Speicher ist begrenzt und muss unter allen Prozessen aufgeteilt werden
  - ➢ physikalische Adressen

Speicherverwaltung

Lösung des Problems

- Aufteilen des virtuellen Adressraums in kleinere Stücke
- diese Stücke können nicht mehr weiter unterteilt werden, und werden als ganzes in den physikalischen Teil geladen

- 2 Probleme
  1. Wie unterteile ich den virtuellen Adressraum?
  2. Wie bekomme ich die richtige physikalische Adresse, wenn ich eine virtuelle Adresse gegeben habe?

Speicherverwaltung

1. Problem: Aufteilung des virtuellen Adressraums

- 2 Möglichkeiten

1. Aufteilung in unterschiedlich grosse Teile, die direkt auf Teile des Programms abgebildet werden können (z.B., Codebereich, Stack)
   - Aufteilung erfolgt nicht transparent
   - Teile (segments) sind nicht gleich groß
   - ➢ Segmentierung

2. Aufteilung in gleich grosse Teile, die vom Programm unabhängig sind
   - Aufteilung erfolgt transparent
   - Teile (pages, frames) sind gleich groß
   - ➢ Paging

Speicherverwaltung

2. Problem: Umsetzen einer virtuellen Adresse in die passende physikalische Adresse

- jede virtuelle Adresse liegt genau in einem Segment oder in einer Page
- virtuelle Adresse kann angegeben werden als
  - Start-Adresse des entsprechenden Segments (der entsprechenden Page) *plus* ein Offset (Abstand zur Start-Adresse)

- Beispiel mit Paging
  - Gegeben ist eine virtuelle Adresse = 0x1234 $(4660)_{10}$
  - Pages sind 0x100 $(256)_{10}$ groß
  - wie lautet die Start-Adresse und der Offset?

Speicherverwaltung

- Beispiel mit Paging (Lösung)

1. Page-Nummer berechnen
   0x1234 / 0x100 = 0x12 $(18)_{10}$
2. Start-Adresse berechnen
   0x100 * 0x12 = 0x1200 $(4608)_{10}$
3. Offset berechnen
   0x1234 - 0x1200 = 0x34 $(52)_{10}$

0x1234 = (0x100 * 0x12) + 0x34
Start-Adresse Offset

Speicherverwaltung

- Was bringt diese Darstellung als Start-Adresse und Offset?

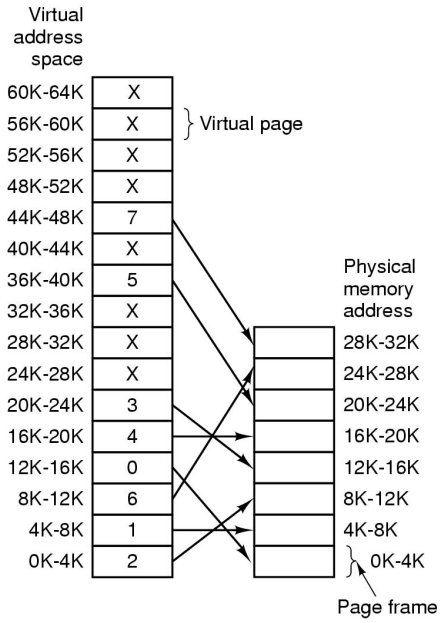
  Wenn ein Segment (oder Page) in den physikalischen Speicher geladen ist (beginnend bei Adresse A), dann muss nur die virtuelle Start-Adresse durch die physikalische Start-Adresse A ersetzte werden, um die Umsetzung zu erledigen.

- Vorteil
  - alle Adressen eines Segments (einer Page) können mit einer Operation umgewandelt werden
- Fragen
  - Wie bekomme ich die physikalische Start-Adresse, wenn ich die virtuelle Start-Adresse habe?
  - Wer genau macht diese Umsetzung?

Speicherverwaltung

- Paging
  - virtuelle Adressen sind in Pages unterteilt
  - typischerweise zwischen 512 Bytes und 4 KBytes groß
  - physikalischer Speicher ist in gleich grosse Page Frames (oder Frames) unterteilt

Speicherverwaltung



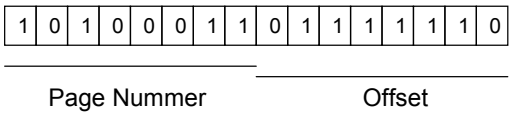
Speicherverwaltung

- Pages haben eine Zweierpotenz Größe
- Grund
  - einfache Berechnung der Page Nummer
  - virtuelle Adresse zerfällt in Page Nummer und Offset
  - Page Nummer einfach aus Adresse ablesen

Speicherverwaltung

- Beispiel
  - 16-bit virtuelle Adresse, Page ist 0x100 = 256 (2^8) Bytes groß
  - ➢ letzten 8 Bits der Adresse sind Offset
  - ➢ ersten 8 Bits der Adresse sind Page Nummer

  - virtuelle Adresse 0xA37E



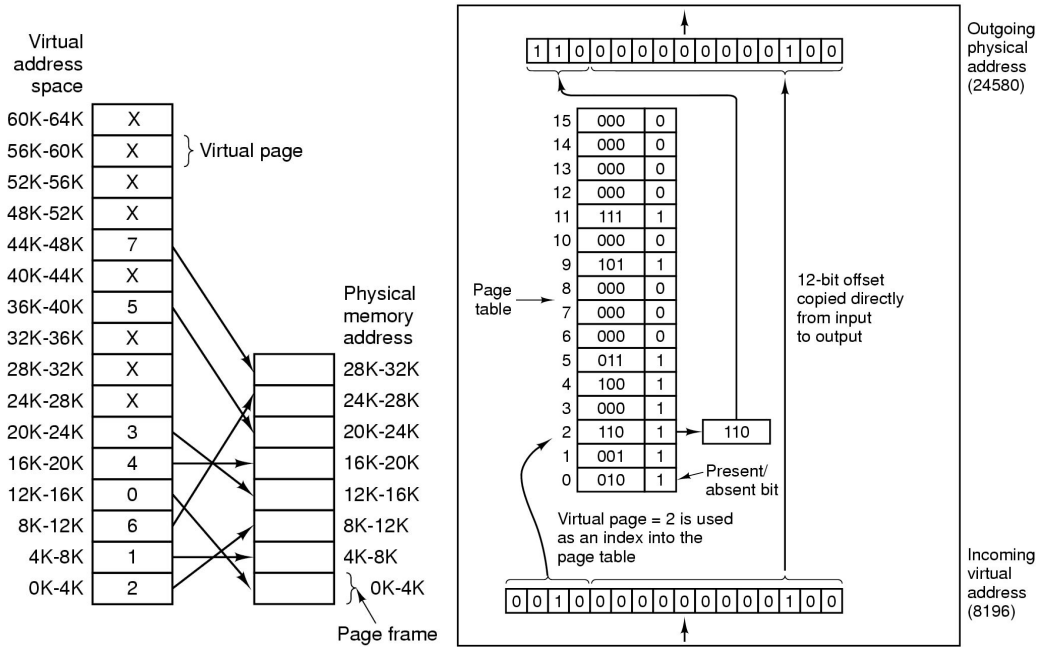
  - Page Nummer = 0xA3, Offset = 0x7E

Speicherverwaltung

- Frage
  - Wie bekomme ich physikalische Start-Adresse, wenn ich die virtuelle Start-Adresse habe?

- Page Table
  - virtuelle Start-Adresse wird nicht benötigt
  - Page Nummer reicht aus
  - Page Table speichert für jede Page, wo der entsprechende Page Frame im Speicher liegt
  - außerdem, ein Bit (present bit), welches angibt, ob die Page überhaupt geladen ist
  - falls auf eine Page zugegriffen wird, die nicht im physikalischen Speicher liegt → *page fault*

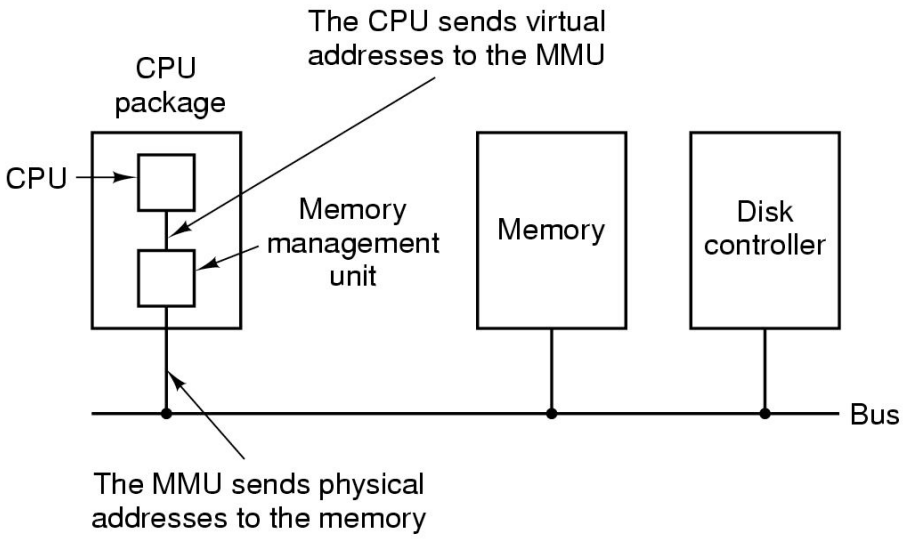
Speicherverwaltung



Speicherverwaltung

- Frage
  - Wer macht die Umsetzung?
- Memory Management Unit (MMU)
  - Hardware Baustein, der vom Betriebssystem entsprechend geladen wird
  - eine Page Table pro Prozess ist notwendig
  - muss bei jedem Context Switch passend geladen werden
- Page Tables brauchen einen Eintrag pro Page
  - kann sehr viel werden
  - 32-bit Adressraum mit 4 KByte Pages ergibt 2^{20} Page Table Einträge
  - Teile müssen in den Hauptspeicher ausgelagert werden
  - Page Table Hierarchie

Speicherverwaltung



Speicherverwaltung

- Beispiel
  - Umsetzen einer virtuellen Adresse in die entsprechende physikalische Adresse
  - gegeben ist Page Table, Page Größe, und virtuelle Adresse
  - gefragt ist die physikalische Adresse

Speicherverwaltung

- Angabe
  - Page Größe ist 4 KBytes, virtuelle Adressen haben 32-bit, physikalische Adressen haben 24-bit

Page Nummer	Frame Nummer
0x00	0x7C
0x01	0x8A
0x02	(not present)

virtuelle Adressen a) 0x000001BCD

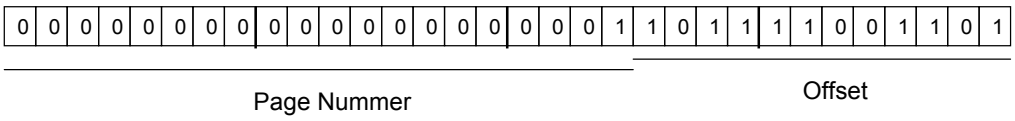
b) 0x000002FFE

Speicherverwaltung

- 1. Schritt

Zerlegen der virtuellen Adresse in Page Nummer und Offset
4 KByte Pages bedeutet 12 Bits Offset (weil 4096 = 2^{12}),
daher ist die Page Nummer 20 Bit groß

a) 0x 00 00 1B CD



Page Nummer = 0x01
Offset = 0xBCD

Speicherverwaltung

- 2. Schritt

Page Nummer = 0x01
Offset = 0xBCD

Nachschlagen der entsprechenden Frame Nummer in der Page Table

Page Nummer	Frame Nummer
0x00	0x7C
0x01	0x8A
0x02	(not present)

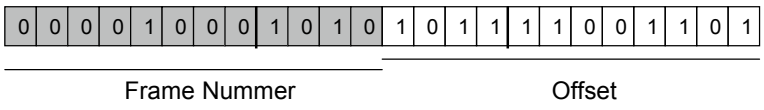
Frame Nummer = 0x8A

Speicherverwaltung

- 3. Schritt

Page Nummer = 0x01
Offset = 0xBCD
Frame Nummer = 0x8A

Zusammensetzen von Frame Nummer und Offset (24 Bit)



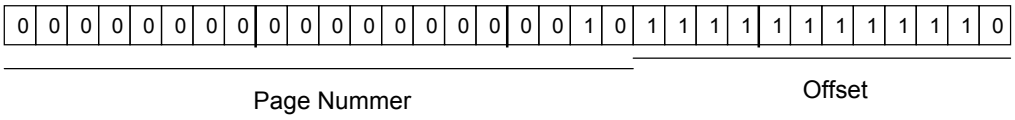
physikalische Adresse = 0x 08 AB CD

Speicherverwaltung

- 1. Schritt

Zerlegen der virtuellen Adresse in Page Nummer und Offset
4 KByte Pages bedeuten 12 Bits Offset (4096 = 2^{12}),
daher ist die Page Nummer 20 Bit groß

a) 0x 00 00 2F FE



Page Nummer = 0x02
Offset = 0xFFE

Speicherverwaltung

- 2. Schritt

Page Nummer = 0x02
Offset = 0xFFE

Nachschlagen der entsprechenden Frame Nummer in der Page Table

Page Nummer	Frame Nummer
0x00	0x7C
0x01	0x8A
0x02	(not present)

Frame Nummer = (not present) → *page fault*

Speicherverwaltung

- Was passiert bei einem *page fault*?
  - Betriebssystem lädt Page in ein freies Frame im Speicher
  - passt Page Table entsprechend an
  - setzt danach Anwendung fort
- Problem
  - was passiert, wenn alle Frames belegt sind
  - ein existierendes Frame muss überschrieben werden
  - veränderte Frames müssten zurückgeschrieben werden
  - daher, besser unmodifizierte Frames nehmen
- Entscheidung
  - ➢ page replacement algorithm

Speicherverwaltung

- Page replacement algorithm
  - Ziel
    - Minimiere die Anzahl der page faults
  - Optimal
    - ersetze die Page, die am weitesten in der Zukunft gebraucht wird
    - unmöglich, aber gut fuer Vergleiche
  - FIFO
    - first-in, first-out
    - ersetze älteste Page
  - LRU
    - least-recently-used
    - ersetze die am längsten nicht gebrauchte Page

Speicherverwaltung

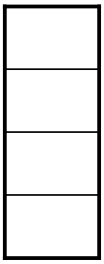
- Beispiel
  - gegeben ist die Anzahl der Frames, die benützt werden können, sowie eine Reihenfolge von Zugriffen (*reference string*) auf die Pages
  - gefragt sind die Entscheidungen des page replacement algorithm, d.h., welche Pages befinden sich nach jedem Zugriff im Speicher (in den Frames)

- Angabe
  - zu verwenden ist LRU
  - 4 Frames sind verfügbar
  - reference string

    0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1



Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

P

0

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

P

2
0

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
		P														

1
2
0

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
			P													

3
1
2
0

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
				P												

5
3
1
2

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
					P											

4
5
3
1

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
						P										

6
4
5
3

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
6
4
5

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
								P								

7
3
6
4

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

4
7
3
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

7
4
3
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
7
4
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
7
4
6

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
													P			

5
3
7
4

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

5
3
7
4

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1

3
5
7
4

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet;
das am längsten nicht benutzte Frame ist ganz unten

Speicherverwaltung

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
																P

1
3
5
7

Speicher
Pages sind nach der Reihenfolge des letzten Zugriffs geordnet; das am längsten nicht benutzte Frame ist ganz unten

Cache Speicher

- Cache Speicher
  - schnell, aber teuer und klein
  - nahe beim Prozessor, um häufig benötigte Daten zwischenzuspeichern
- Cache ist transparent
  - sind die angeforderten Daten im Cache (Cache Hit), dann können sie sehr schnell an den Prozessor geliefert werden
  - sind gewünschte Daten nicht im Cache, werden sie aus dem Hauptspeicher nachgeladen (Cache Miss)
  - je seltener auf den Hauptspeicher zurück gegriffen werden muss, desto besser
  - Cache „Hit Rate“ gibt an, welcher Anteil der Zugriffe erfolgreich ist

Cache Speicher

- Kennzahlen
  - Zugriffszeit auf Cache (t_{cache})
  - Zugriffszeit auf Hauptspeicher (t_{memory})
  - Mittlere (effektive) Zugriffszeit auf Cache (t_{eff})

$$t_{eff} = h * t_{cache} + (1 - h) * t_{memory}$$

- Aufbau
  - ein Cache besteht aus mehreren Speicherzellen (cache lines)
  - voll assoziativ
    - Daten von einer bestimmten Adresse können in jeder beliebigen cache line abgelegt werden
  - direct mapping
    - Daten von einer bestimmten Adresse können nur in einer bestimmten cache line abgelegt werden

Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger


```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger

Tag	Data
0x1234	42

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger

Tag	Data
0x1234	42
0x1238	8

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger

Tag	Data
0x1234	42
0x1238	8

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger

Tag	Data
0x1234	42
0x1238	8
0x47A0	50

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger

Tag	Data
0x1234	42
0x1238	8
0x47A0	50
0x1434	17

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Voll assoziativer Cache
  - Daten müssen erst dann ersetzt werden, wenn Cache voll ist
  - flexible replacement Algorithmen möglich
  - Suche nach vorhanden Daten dauert länger

Tag	Data
0x1438	33
0x1238	8
0x47A0	50
0x1434	17

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

Fixed	Tag	Data
0x34		
0x38		
…		
0xA0		

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

Fixed	Tag	Data
0x34	0x12	42
0x38		
…		
0xA0		

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

0x34	0x12	42
0x38	0x12	8
…		
0xA0		
Fixed	Tag	Data

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

0x34	0x12	42
0x38	0x12	8
…		
0xA0		
Fixed	Tag	Data

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

0x34	0x12	42
0x38	0x12	8
…		
0xA0	0x47	50
Fixed	Tag	Data

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

0x34	0x14	17
0x38	0x12	8
…		
0xA0	0x47	50
Fixed	Tag	Data

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```

Cache Speicher

- Direct Mapping
  - Jede Adresse wird auf eine bestimmte cache line abgebildet
  - üblicherweise wird die cache line durch den niederwertigen Teil der Adresse bestimmt
  - kommt ein neuer Datenwert, muss der Alte an dieser Stelle ersetzt werden
  - Suche nach vorhanden Daten sehr schnell

0x34	0x14	17
0x38	0x14	33
…		
0xA0	0x47	50
Fixed	Tag	Data

```
mov 0x1234, %eax
mov 0x1238, %ebx
add %eax, %ebx, %ecx
mov %ecx, 0x47A0
mov 0x1434, %eax
mov 0x1438, %ebx
```


Cache Speicher

- Schreibende Zugriffe
  - gehen zuerst in den Cache
  - Gefahr des Datenverlusts bei Stromausfall
- 2 Strategien
  - write through
    - Daten werden gleichzeitig in den Hauptspeicher geschrieben
    - Daten sind kohärent, aber Vorgang langsam
  - copy back
    - Aktualisierung im Hauptspeicher erfolgt erst, wenn cache line ausgetauscht werden muss

Zusammenfassung

- Betriebssysteme verwalten
  - Prozesse
  - *Speicher*
  - Dateisystem
  - Eingabe und Ausgabe
- Speichermanagement
  - Speicherhierarchie
  - Caches
    - Typen von Caches
  - Hauptspeicher
    - virtuelle Speicherverwaltung

Zusammenfassung

- Virtuelle Speicherverwaltung
  - Segmentation
  - Paging
  - Adressumrechnungen
  - page faults
  - page replace algorithms
- Cache Speicher
  - Arten
    - voll assoziativ
    - direct mapping
  - Schreibzugriff
    - write through
    - copy back