Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Übung 3

Dipl.-Inform. Leonard Masing
Dr.-Ing. Oliver Sander

Institut für Technik der Informationsverarbeitung (ITIV)

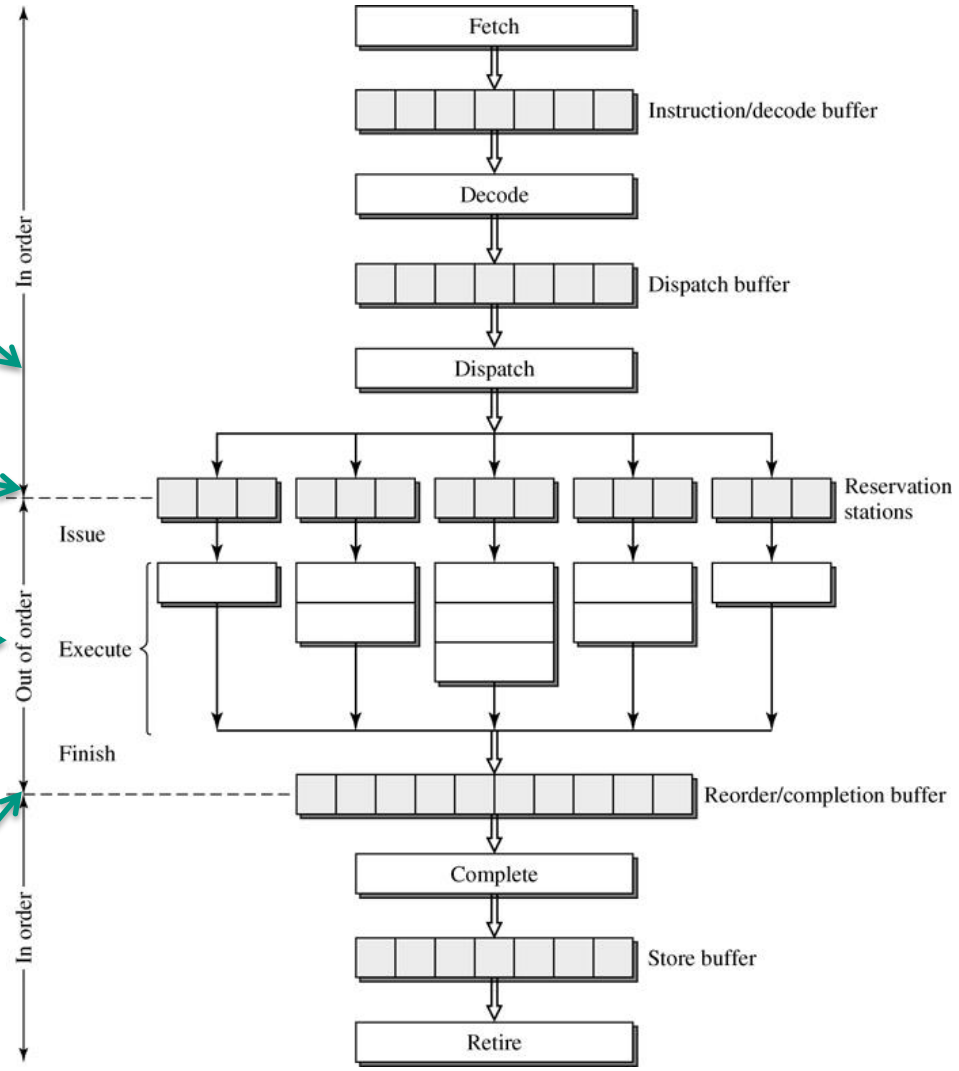
Hardware/Software Co-Design

Agenda

- Wiederholung ausgewählter Themen
- Gruppenarbeit
- Vorstellung der Lösung

2.3.2.2 Dynamisches Scheduling - Prozessorpipeline (I)

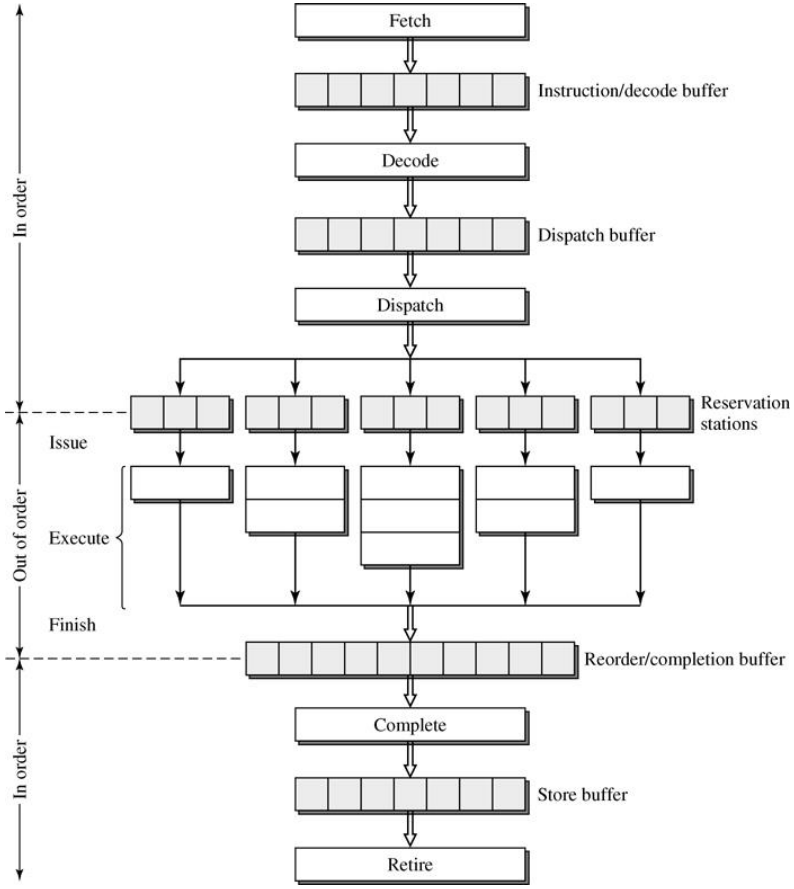
- Prinzipielle Vorgehensweise:
  - Zuweisen von Befehlen an Ausführungseinheiten
    - In-order Issue
    - Out-of-Order Issue
  - Reservation Stations
  - Weiterleiten der Befehle an Funktionseinheiten, wenn Operanden verfügbar sind
  - Ablegen der Befehle in Programmreihenfolge in dem Reorder-Puffer



Leonard Masing
Hardware/Software Co-Design

2.3.2.2 Dynamisches Scheduling - Prozessorpipeline (II)

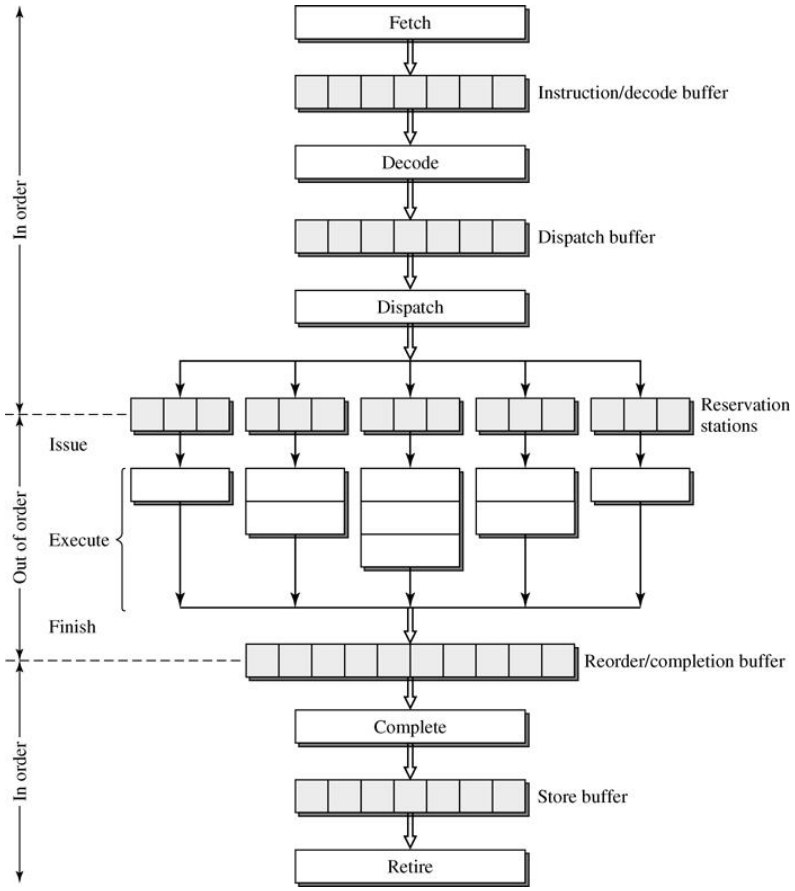
- IF Phase
  - Holen mehrerer Befehle in den Hole-Puffer
  - Verwaltung der Komponenten für die Sprungzielvorhersage
- ID Phase
  - Dekodierung der Befehle
  - Umbenennung von Registern
  - Abbildung von logischen Registern auf physikalische Prozessorregister
  - Schreiben der Befehle ins Befehlsregister
  - Erkennen und Auflösen von Konflikten aufgrund von Daten- und Strukturabhängigkeiten



Leonard Masing
Hardware/Software Co-Design

2.3.2.2 Dynamisches Scheduling – Dispatcher

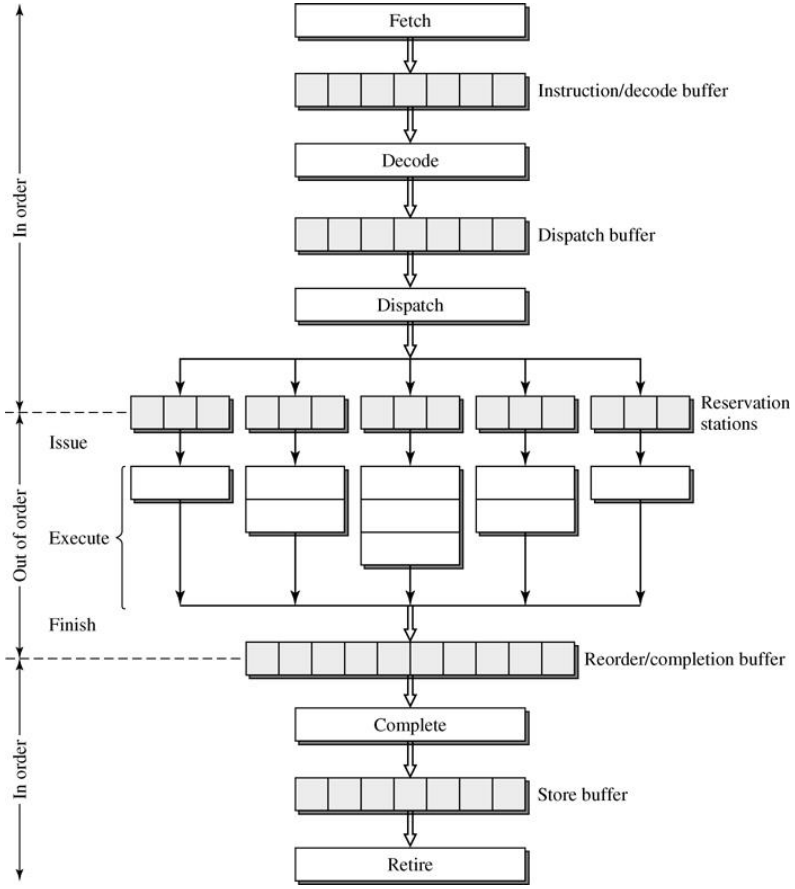
- Eine Instruktion wird aus der Reservation Station an die Funktionseinheit weitergegeben, wenn alle Operanden verfügbar sind
- Der Befehl wird in der Funktionseinheit ausgeführt
- Wenn alle Operanden verfügbar sind und die Funktionseinheit nicht besetzt ist, wird die Instruktion sofort zur Ausführung angestoßen
- Eine Instruktion kann sich null oder mehrere Zyklen in der Reservation Station aufhalten
- Dispatch und Ausführen erfolgt nicht in der sequentiellen Programmordnung



Leonard Masing
Hardware/Software Co-Design

2.3.2.2 Dynamisches Scheduling - Reservation Stations

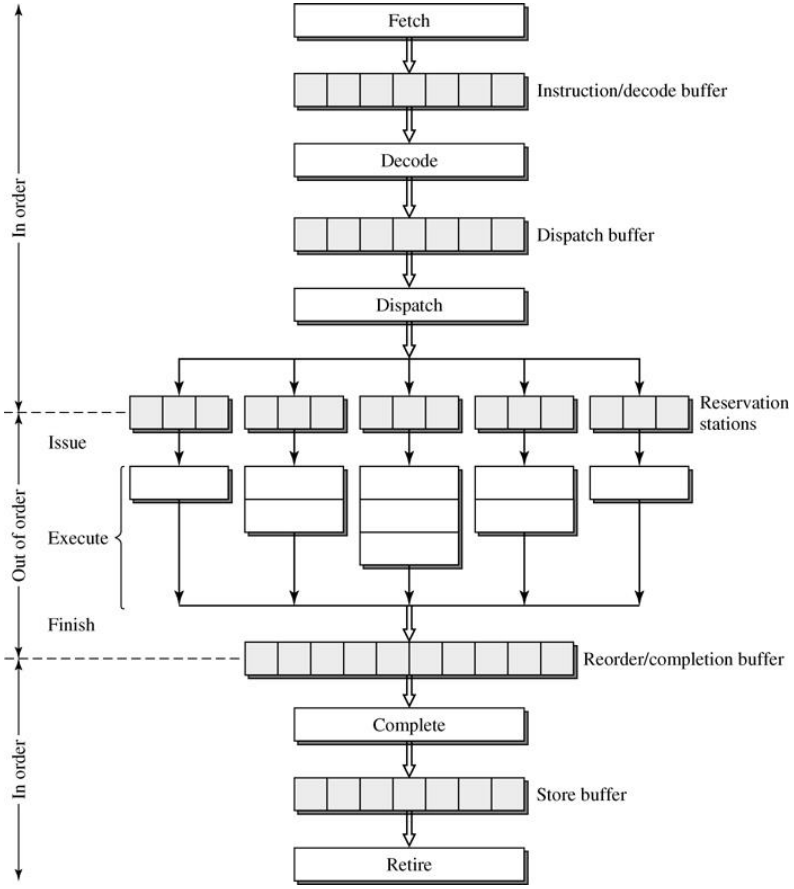
- Puffer für eine Instruktion mit ihrem Operanden
  - (siehe Tomasulo-Algorithmus, Hennessy/Patterson Buch)
- Puffer mit ein oder mehreren Einträgen. Jeder Eintrag kann eine Instruktion mit ihrem Operanden enthalten
- Konfliktauflösung mit Hilfe von Reservation Stations
- Befehle warten in Reservation Station bis Operand verfügbar ist



Leonard Masing
Hardware/Software Co-Design

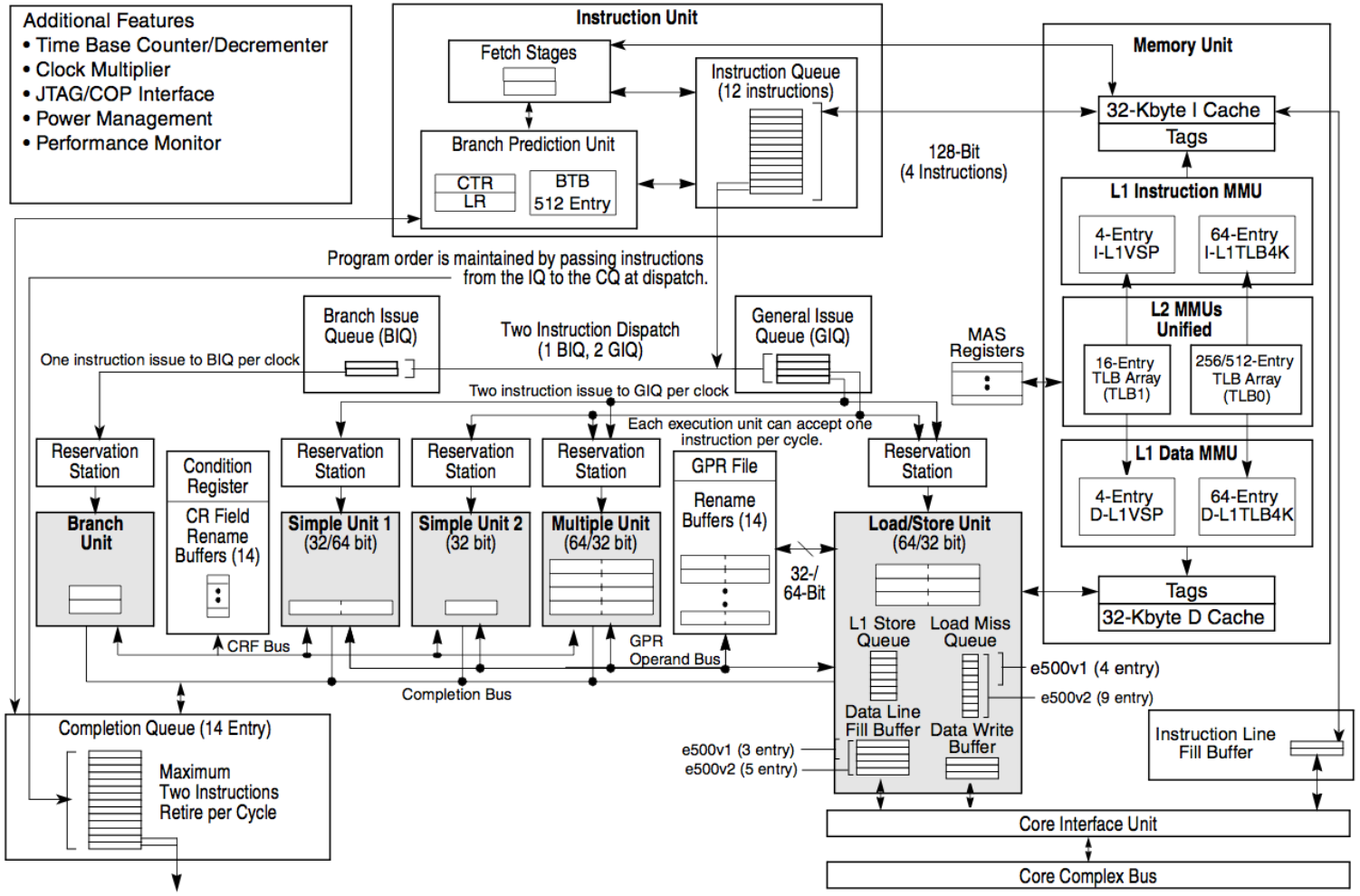
2.3.2.2 Dynamisches Scheduling - Completion-Unit

- Eine Instruktion beendet ihre Ausführung, wenn das Ergebnis für nachfolgende Befehle bereit steht (Forwarding-Puffer)
- Completion: Befehl ist vollständig
- Erfolgt unabhängig von der Programmordnung
- Aktualisierung des Zustands des Rückordnungspuffers (Reorder-Puffer)
  - Es kann eine Unterbrechung angezeigt sein
  - Es kann ein unvollständiger Befehl angezeigt werden, der von einer Spekulation abhängt



Leonard Masing
Hardware/Software Co-Design

2.3.2.2 Beispiel - PowerPC e500 core



2.3.2.3 Very Long Instruction Word (VLIW) (I)

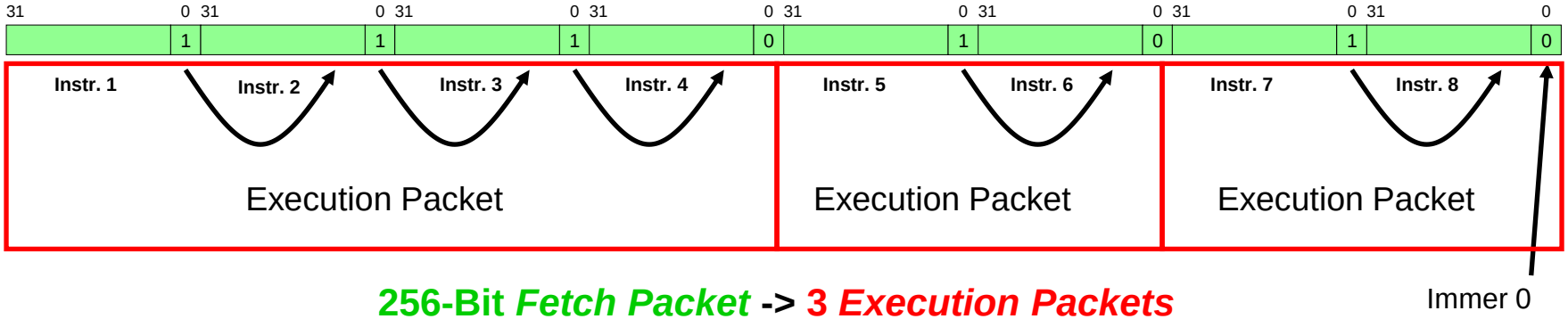
- VLIW = Very Long Instruction Word
- Architekturtechnik, bei der ein Compiler eine feste Anzahl von einfachen, voneinander unabhängigen Befehlen zu einem Befehlspaket zusammenpackt und in einem Maschinenbefehlswort meist fester Länge speichert.
- Das Maschinenbefehlsformat eines VLIW-Befehlspakets kann mehrere hundert Bits lang sein, in der Praxis sind dies zwischen 128 und 1024 Bits.
- Alle Befehle innerhalb eines VLIW-Befehlspakets müssen unabhängig voneinander sein und eigene Opcodes und Operandenbezeichner enthalten.

2.3.2.3 Very Long Instruction Word (VLIW) (II)

- Die Anzahl der Befehle in einem VLIW-Befehlspaket ist in der Regel fest.
  - Wenn die volle Bandbreite eines VLIW-Befehlspakets nicht ausgenutzt werden kann, muss es mit Leerbefehlen aufgefüllt werden.

- Problem bei VLIW-Architekturen:
  - volle Parallelität nicht immer ausnutzbar Befehlswort teilweise unnötig lang
  - Neuere VLIW-Architekturen sind in der Lage, durch ein komprimiertes Befehlsformat auf das Auffüllen mit Leerbefehlen zu verzichten.

- Lösung: Instruction-Packing: Instruktionswort enthält mehrere Teil-Instruktionen

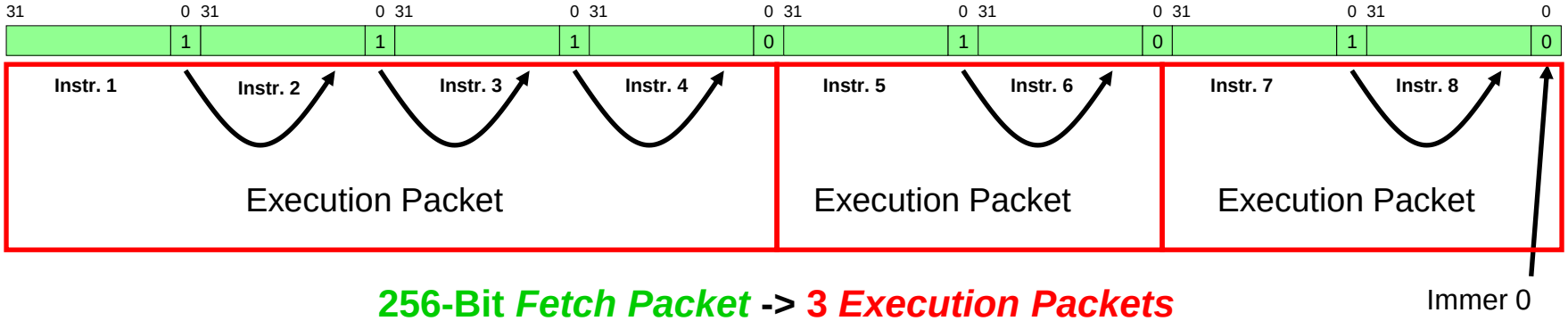


256-Bit *Fetch Packet* -> 3 *Execution Packets*

Leonard Masing
Hardware/Software Co-Design

2.3.2.3 Very Long Instruction Word (VLIW) (III)

- Befehlswort enthält bis zu 8 32-Bit-Befehle (je 5 ns-Zyklus) = Befehlspaket
  - ein Bit gibt an, ob nächster Befehl zum selben Befehlspaket gehört,
  - erlaubt Befehle unterschiedlicher Bitbreite
- Teil-Instruktionen bedienen sich nebenläufiger Funktions-Einheiten.
- Komplizierte Compiler/Code-Generatoren
  - Scheduling muss konfliktfrei für die Ausführung geplant sein.



256-Bit *Fetch Packet* -> 3 *Execution Packets*

2.3.2.4 Single Instruction Multiple Data (SIMD)

- Gleiche Instruktion auf mehreren Daten
  - Multimediaanwendungen
    - Viele Daten gleichzeitig
    - Viele Parallelisierungsmöglichkeiten
- Reduziert Zyklenzahl für gleiche Menge von Daten
- Vektorisierung von Daten
  - Parallele Berechnung der Vektoren
  - Speichern der einzelnen Daten in Register
- Schwer zu implementieren
  - Schwer automatisch Code zu finden, der SIMD gut ausnutzt
    - Code von Hand schreiben/optimieren

2.3.2.4 SIMD – Multimedia-Erweiterungen

- Multimedia-Anwendungen
  - z.B.: Audio/Video Playback, DVD, Videokonferenzen
  - 8/16-Bit Datentypen (z.B.: Pixeldarstellung in Bildverarbeitung)
  - viele arithmetische Operationen (Multiplikation, Addition)
  - große Datenmengen, große I/O Bandbreite
  - viel Datenparallelität Single Instruction Multiple Data (SIMD)

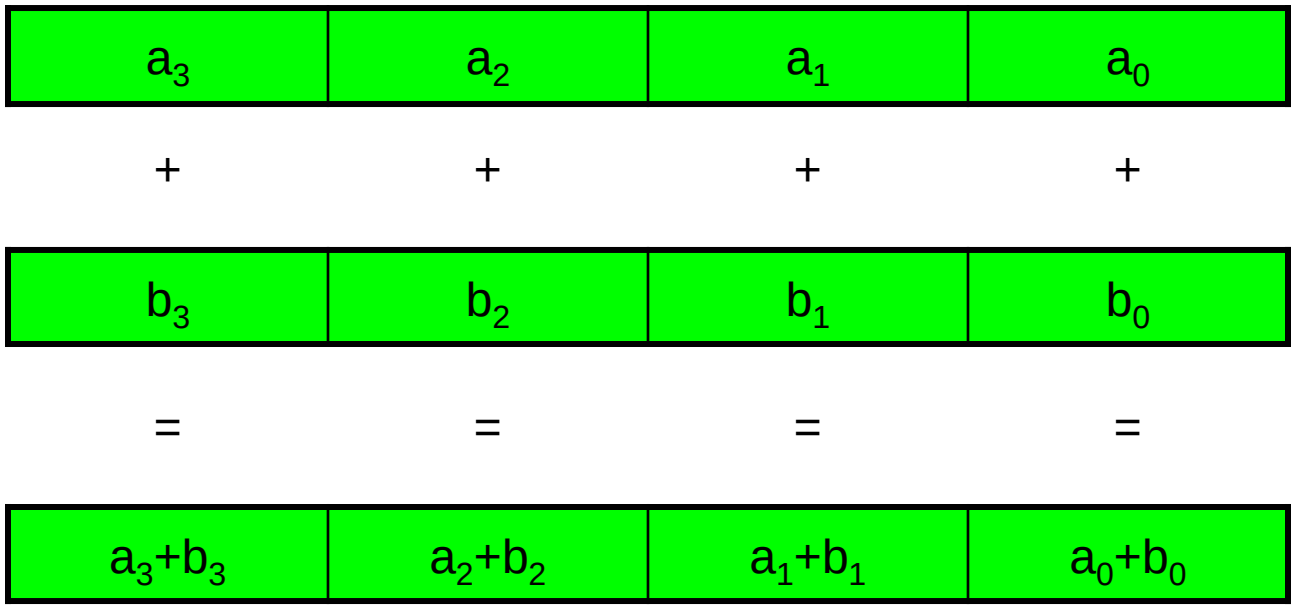
- Sub-Word Execution Model
  - 32/64-Bit Register und ALUs in kleinere Einheiten auftrennen
  - Instruktionen arbeiten parallel auf den Einheiten -> SIMD

- Beispiele:
  - MMX (Intel), VIS (UltraSparc), MDMX (MIPS), MAX-2 (HP)

2.3.2.4 SIMD – Sub-Word Execution Model

- Beispiel 1: Addition
  - Instruktionen parallel auf 4 Ausführungseinheiten

- ADD R3 := R1, R2

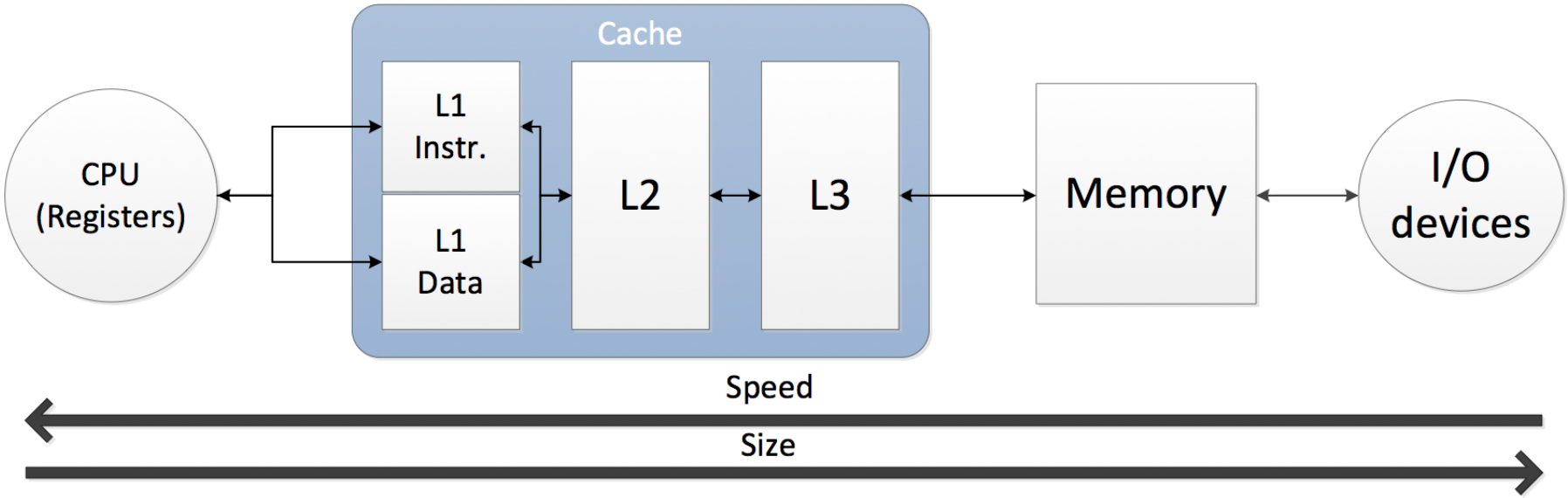


Leonard Masing
Hardware/Software Co-Design

2.3.2.5 Caches - Hierarchie

- In der Regel besitzt ein Rechner einen getrennten Cache für Instruktionen (Instruktionscache) und für Daten (Datencache)

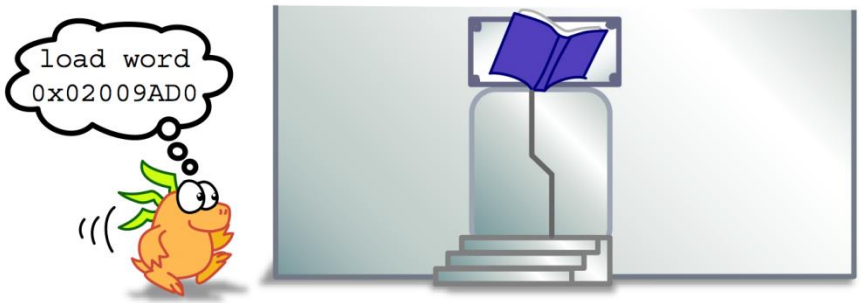
- Cache Level: Level 1 bis Level N bezeichnet die Cache Speicher beginnend mit dem schnellsten Speicherzugriff



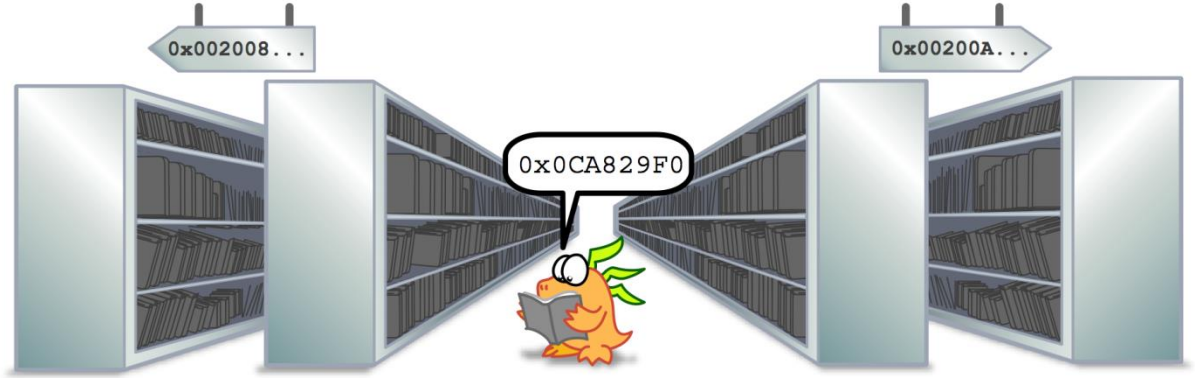
© Bachelorarbeit Maximilian Braun, ITIV 2012

2.3.2.5 Caches

For computers, memory accesses are like going to the library,



Finding the necessary information in the page of a book,



http://csillustrated.berkeley.edu/PDFs/handouts/

2.3.2.5 Caches

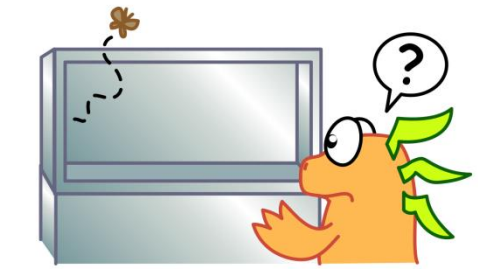
And going back home to do the work involving that information.

While computers don't mind going back and forth like this for data, it usually means users have to do a lot of waiting.

Fortunately for users, computers have caches, which is the equivalent of keeping copies of the books needed on a shelf near the workspace. Through a number of mechanisms, caches give the illusion of being able to access memory very quickly!

http://csillustrated.berkeley.edu/PDFs/handouts/

2.3.2.5 Cache Misses

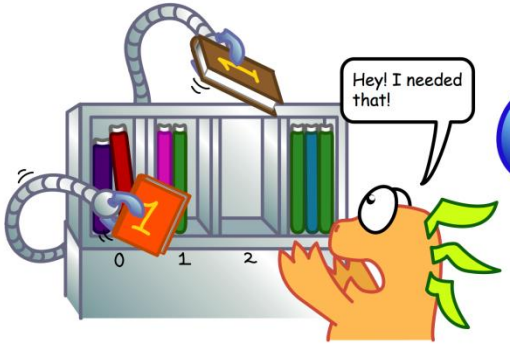


Compulsory

Compulsory misses happen when a block is referenced for the first time. The computer can't get a block that doesn't exist yet!

Capacity

The block is not in the cache because there is no space in the cache for it. Caches are of finite size, after all.

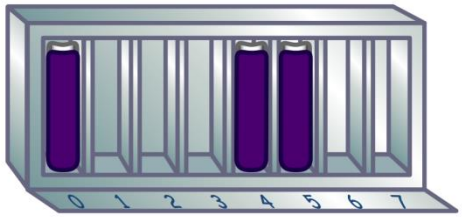


Conflict

These types of misses happen only in direct-mapped and set-associative caches. Multiple blocks can be mapped to a set, forcing evictions when the set is full.

http://csillustrated.berkeley.edu/PDFs/handouts/

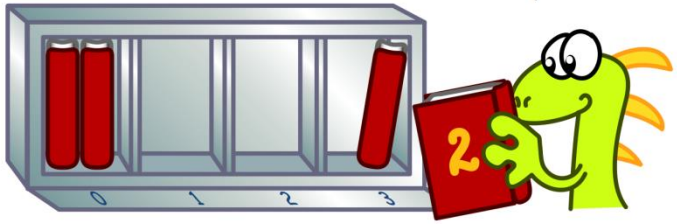
2.3.2.5 Cache Assoziativität



Direct Mapped

Tag	Index	Offset

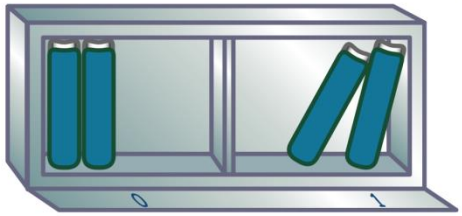
A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.



2-Way Set Associative

Tag	Index	Offset

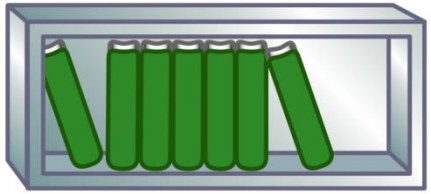
This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.



4-Way Set Associative

Tag	Index	Offset

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.



Fully Associative

Tag	Offset

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!

http://csillustrated.berkeley.edu/PDFs/handouts/

Arbeitsphase

- Aufgabe 2.05: Superskalarität
  - Superskalare Out-of-Order (OoO) Pipeline
- Aufgabe 2.06: VLIW
  - Codestück auf VLIW Prozessor mit drei parallelen Ausführungseinheiten
- Aufgabe 2.07: SIMD
  - MMX-Befehle
    - Bonusaufgabe
- Aufgabe 2.08: Cache
  - 4 Zeilen mit 4 Bytes pro Zeile
  - Direct-mapped, 2-Wege Assoziativ und Voll-Assoziativ

Aufgabe 2.05: Superskalarität

- Gegeben ist eine Superskalare Out-of-Order Pipeline, wie sie in der Vorlesung vorgestellt wurde.
  a) Was macht der Dispatcher?
  b) In welchem Abschnitt der Pipeline findet ein Out-of-Order Ausführung und in welchem Abschnitt eine In-Order Ausführung statt?
  c) Welche Aufgaben erfüllt die Completion Unit?
  d) Wie werden Pipelinekonflikte durch Namensabhängigkeiten verhindert?
  e) Wie werden Pipelinekonflikte durch Datenabhängigkeiten verhindert?
  f) Warum lohnt sich der ganze Aufwand für eine OOO-Pipeline überhaupt?

Lösung Aufgabe 2.05: Superskalarität

- Gegeben ist eine Superskalare Out-of-Order Pipeline, wie sie in der Vorlesung vorgestellt wurde.

a) Was macht der Dispatcher?

- Je nach Implementierung ist der Dispatcher für Dispatch alleine oder Dispatch und Issue zuständig. Dispatch ist das Verteilen der Befehle an die verschiedenen Funktionseinheiten. Bei Issue wird ein Befehl zur Ausführung angestoßen. Dabei muss überprüft werden, ob auch alle Operanden eines Befehles zur Verfügung stehen.

b) In welchem Abschnitt der Pipeline findet ein Out-of-Order Ausführung und in welchem Abschnitt eine In-Order Ausführung statt?

- Die Out-of-Order Ausführung findet in allen Stufen zwischen der oder den Reservation Stations und dem Reorder Buffer statt. Die Befehle werden In-Order in eine Reservation Station geschrieben können aber Out-of-Order gestartet werden. Umgedreht verhält es sich mit dem Reorder Buffer. Die Befehle werden Out-of-Order geschrieben, dürfen Ihn aber nur In-Order verlassen.

Lösung Aufgabe 2.05: Superskalarität

c) Welche Aufgaben erfüllt die Completion Unit?

- Die Completion-Unit beinhaltet den Reorder Buffer. Sie nimmt die Befehle der Ausführungseinheiten Out-of-Order entgegen und aktualisiert den Zustand im Reorder-Buffer. Sie überprüft, ob der Befehl wegen eines Interrupts, Exception oder spekulative Ausführung verworfen werden muss.

d) Wie werden Pipelinekonflikte durch Namensabhängigkeiten verhindert?

- Durch Register Renaming wird verhindert, dass Konflikte durch Namensabhängigkeiten während der Out-of-Order Ausführung entstehen können. Register mit gleichen Namen, die keine Datenabhängigkeiten haben, werden einem anderen internen Register zugewiesen. Das Zurückschreiben der Ergebniswerte (WB oder Retire) findet am Ende wieder In-Order statt und ist somit ebenfalls Namenskonfliktfrei.

Lösung Aufgabe 2.05: Superskalarität

e) Wie werden Pipelinekonflikte durch Datenabhängigkeiten verhindert

- Datenabhängigkeiten werden erkannt und beim Laden der Operanden wird nicht der Wert geladen sondern ein Verweiß auf ein internes Register vermerkt. Dieses interne Register enthält ein Valid-Bit, das gesetzt wird, sobald der Befehle berechnet ist. In der Reservation Station warten Befehl so lange, bis beide Operanden verfügbar sind. Erst dann wird der Befehl zur Ausführung angestoßen.

f) Warum lohnt sich der ganze Aufwand für eine OOO-Pipeline überhaupt?

- Eine superskalare Out-of-Order Pipeline kann gut separate unterschiedliche lange Ausführungspipelines bedienen und dynamisch auf unvorhersehbare Ereignisse reagieren (z.B. Cache Miss beim Speicherzugriff). So können mehrere andere Befehle abgearbeitet werden, während z.B. eine lange Floating-Point Division berechnet wird. Bei einer skalaren In-Order Pipeline müssten Befehle mit kurzen Ausführungszeiten immer auf lange warten.

- Superskalarität
  - Spezielle Einheiten
    - Dispatcher
    - Completion Unit
  - Out of Order Ausführung
  - Register Renaming
  - Datenabhängigkeiten
  - Nutzen

Aufgabe 2.06: VLIW

- Das folgende Codestück soll auf einem VLIW Prozessor mit drei parallelen Ausführungseinheiten (Execution Units) laufen. Geben sie für einen VLIW Prozessor eine möglichst effiziente Befehlsverteilung an, wie sie der Compiler durchführen würde. Die Befehle können beliebig umsortiert werden so lange die Funktionalität identisch bleibt. Man kann annehmen, dass alle Befehle innerhalb eines Taktes berechnet werden. Ein freies Kästchen ist gleichbedeutend mit einem NOP-Befehl.

```
(1)  add   r1, r2, r3      ; r1 = r2 + r3
(2)  sub   r5, r3, r5      ; r5 = r3 – r5
(3)  ld    r3, [r1]        ; Lade r3 mit [r1]
(4)  mul   r3, r3, r3      ; r3 = r3 * r3
(5)  st    [r5], r3        ; Speichere r3 nach [r5]
(6)  ld    r9, [r7]        ; Lade r9 mit [r7]
(7)  ld    r11, [r12]      ; Lade r11 mit [r12]
(8)  add   r11, r11, r12   ; r11 = r11 + r12
(9)  mul   r11, r11, r9    ; r11 = r11 * r9
(10) st    [r12], r11      ; Speichere r11 nach [r12]
```

a) Nehmen Sie an, dass der Prozessor über drei Ausführungseinheiten verfügt, die jeweils alle Befehle ausführen können.

b) Nehmen Sie nun an, dass die ersten beiden Ausführungseinheiten arithmetisch-logische Befehle (add, sub, mul) ausführen kann und die letzte für Load/Store Befehle zuständig ist.

VLIW

Slot 1	Slot 2	Slot 3
(1) add r1, r2, r3	(2) sub r5, r3, r5	(7) ld r11, [r12]
(3) ld r3, [r1]	(6) ld r9, [r7]	(8) add r11, r11, r12
(4) mul r3, r3, r3	(9) mul r11, r11, r9	
(5) st [r5], r3	(10) st [r12], r11	

Slot 1 (ALU)	Slot 2 (ALU)	Slot 3 (LS)
(1) add r1, r2, r3	(2) sub r5, r3, r5	(7) ld r11, [r12]
	(8) add r11, r11, r12	(3) ld r3, [r1]
(4) mul r3, r3, r3		(6) ld r9, [r7]
(9) mul r11, r11, r9		(5) st [r5], r3
		(10) st [r12], r11

- VLIW
  - Parallele Ausführungseinheiten
  - Instruction packing
  - Effiziente Befehlsverteilung
  - Umsortierung der Befehle
    - Hardware Richtlinien beachten

Aufgabe 2.07: SIMD

a) Zwei MMX (64-Bit SIMD) Register mm0 und mm1 enthalten die folgenden Werte. Tragen Sie das Ergebnis der beiden MMX-Befehle PADDUSB und PADDW in die Tabelle ein.

mm0	10	20	00	80	55	33	FF	FF
mm1	01	20	00	80	33	55	00	01
PADDW	11	40	01	00	88	89	00	00
PADDUSB	11	40	00	FF	88	88	FF	FF

Lösung Aufgabe 2.07: SIMD

b) Gegeben ist folgendes Fragment eines C-Programmes:

```
int      i
char     a[8],b[8];      // char = 8-Bit Integer
for (i = 0; i < 8; i++) {
 if (a[i] < b[i]) {
         b[i] = i;
 }
}
```

- Realisieren Sie den C-Code mittels SIMD Assemblerbefehle. Die Registerbreite der SIMD Befehle beträgt 64-Bit und sie können die Register mittels folgender Befehle als Packed-Byte ansprechen. Die Byte-Order (Little oder Big-Endian) sowie Signed/Unsigned kann vernachlässigt werden. Sie können das Ergebnis wahlweise als Assemblerprogramm oder Datenflussgraphen darstellen.

```
PMOV            mm2, 0x0001020304050607
PCMPLTB         mm3, mm0, mm1
PNEG            mm4, mm3
PAND            mm3, mm3, mm2
PAND            mm4, mm4, mm1
POR             mm5, mm3, mm4
```

- SIMD
  - Parallelität ausnutzen
  - Instruktionen einsparen
  - Viele Daten bearbeiten

Aufgabe 2.08: Cache

- Gegeben sei ein Cache mit 4 Cache-Zeilen und 4 Bytes pro Cache-Zeile. Zum Vergleich ist der Cache als Direct-Mapped, 2-Wege Assoziativ und Voll-Assoziativ vorhanden. Die Speicheradresse beträgt 8-Bits.

a) Wie ist die Aufteilung der Speicheradresse bei den Verschiedenen Cache-Typen. Pro Bit der Speicheradresse ist ein Kästchen vorhanden. Jedes Bit kann entweder für den Tag (t), Index (i) oder Byteoffset (o) verwendet werden.

a) Es werden jetzt nacheinander 6 Lesezugriffe auf den jeweiligen Cache-Typen ausgeführt. Die Speicheradressen der Lesezugriffe sind in Dualform in der ersten Spalte vorhanden. In der 2ten Spalte (Hit?) werden Cache-Hits eingetragen. Die restlichen Spalten sind für den Inhalt der 4 Cache-Zeilen vorgesehen. Als Inhalt genügt es den Tag der Cache-Zeile einzutragen. Als Verdrängungsstrategie ist LRU (Least Recently Used) vorgesehen.

Lösung Aufgabe 2.08a: Cache

- Wie ist die Aufteilung der Speicheradresse bei den Verschiedenen Cache-Typen. Pro Bit der Speicheradresse ist ein Kästchen vorhanden. Jedes Bit kann entweder für den Tag (t), Index (i) oder Byteoffset (o) verwendet werden.

Direct-Mapped	t	t	t	t	i	i	o	o
2-Wege Assoziativ	t	t	t	t	t	i	o	o
Voll-Assoziativ	t	t	t	t	t	t	o	o

Lösung Aufgabe 2.08b: Cache

- Es werden jetzt nacheinander 6 Lesezugriffe auf den jeweiligen Cache-Typen ausgeführt. Die Speicheradressen der Lesezugriffe sind in Dualform in der ersten Spalte vorhanden. In der 2ten Spalte (Hit?) werden Cache-Hits eingetragen. Die restlichen Spalten sind für den Inhalt der 4 Cache-Zeilen vorgesehen. Als Inhalt genügt es den Tag der Cache-Zeile einzutragen. Als Verdrängungsstrategie ist LRU (Least Recently Used) vorgesehen.

- Was ist ein Cache?
- Wie sind Offset, Index & Tag aufgeteilt?
- Was ist Assoziativität und wie funktioniert diese?
- Was sind Verdrängungsstrategien?