Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

ITIV

Hardware/Software Co-Design

Übung 3 - Wiederholung

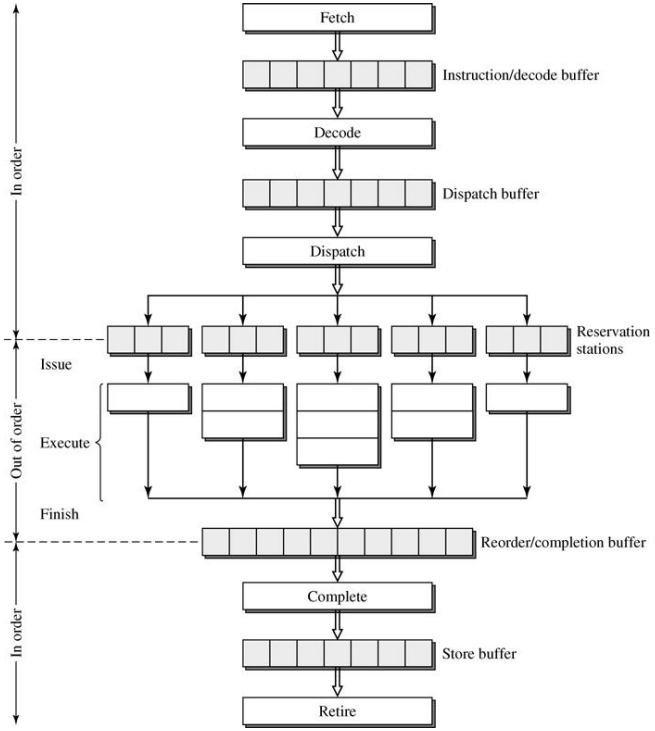
M.Sc. Fabian Lesniak

Agenda

- Wiederholung ausgewählter Themen
  - Superskalarität: Dynamisches Scheduling
  - PowerPC
  - VLIW
  - SIMD

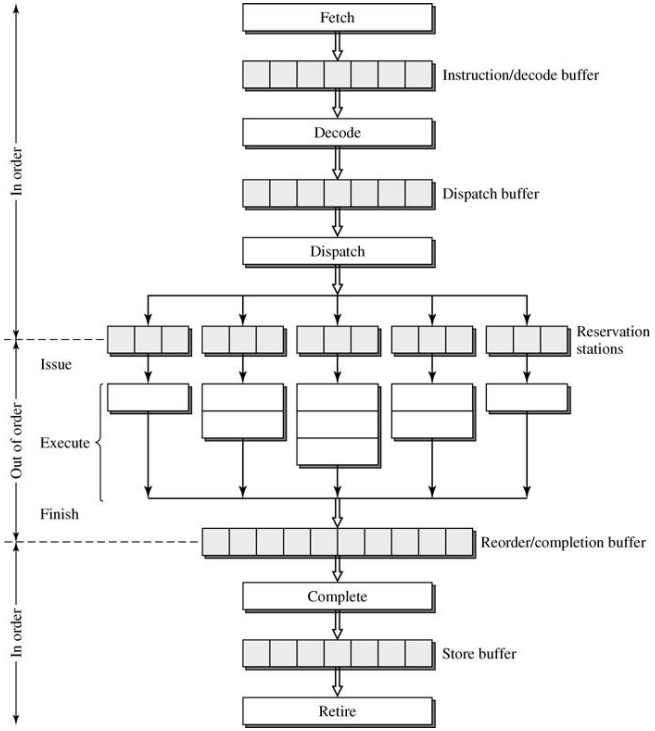
2.3.2.2 Dynamisches Scheduling – Prozessorpipeline (I)

- Prinzipielle Vorgehensweise:
  - Zuweisen von Befehlen an Ausführungseinheiten
    - In-order Issue
    - Out-of-Order Issue
- Reservation Stations
  - Weiterleiten der Befehle an Funktionseinheiten, sobald:
    - Operanden verfügbar sind
    - Funktionseinheit frei ist
- Ablegen der Befehle in Programm-reihenfolge in dem Reorder-Puffer



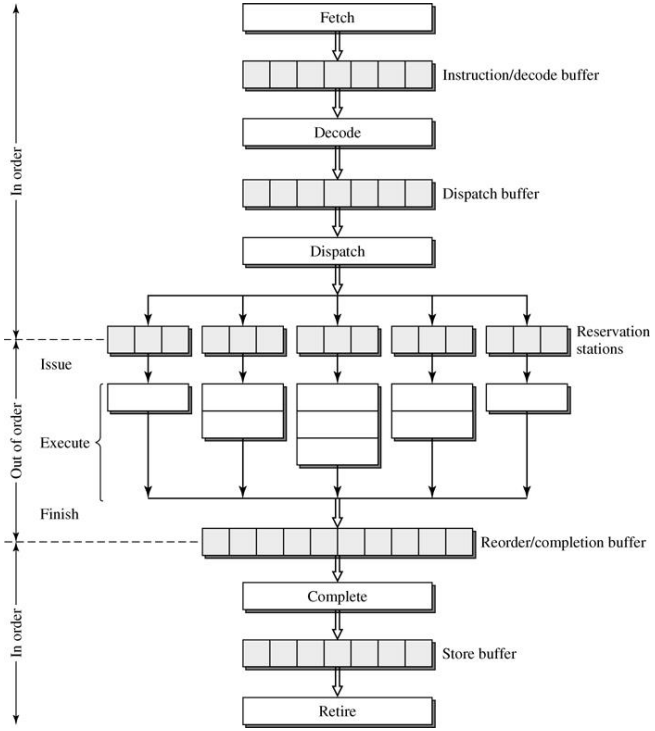
2.3.2.2 Dynamisches Scheduling – Reservation Stations

- 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



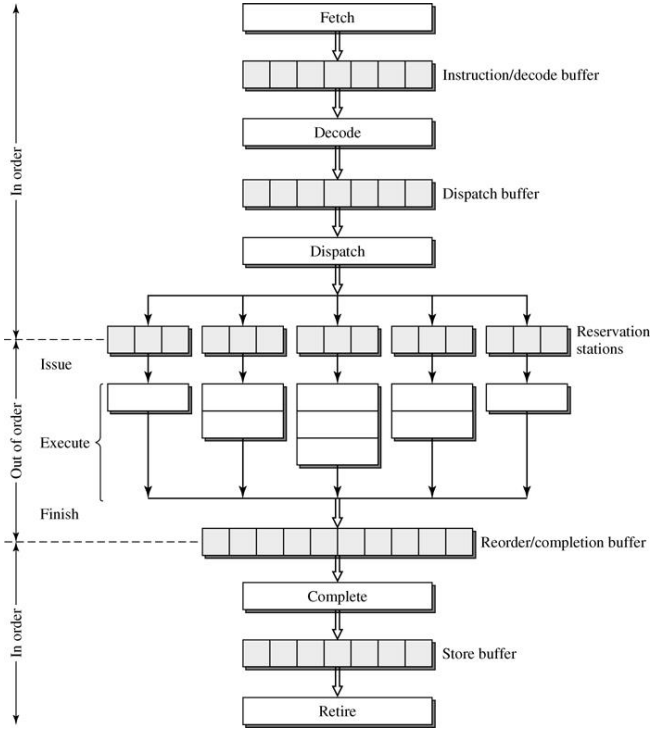
2.3.2.2 Dynamisches Scheduling – Prozessorpipeline (II)

- 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



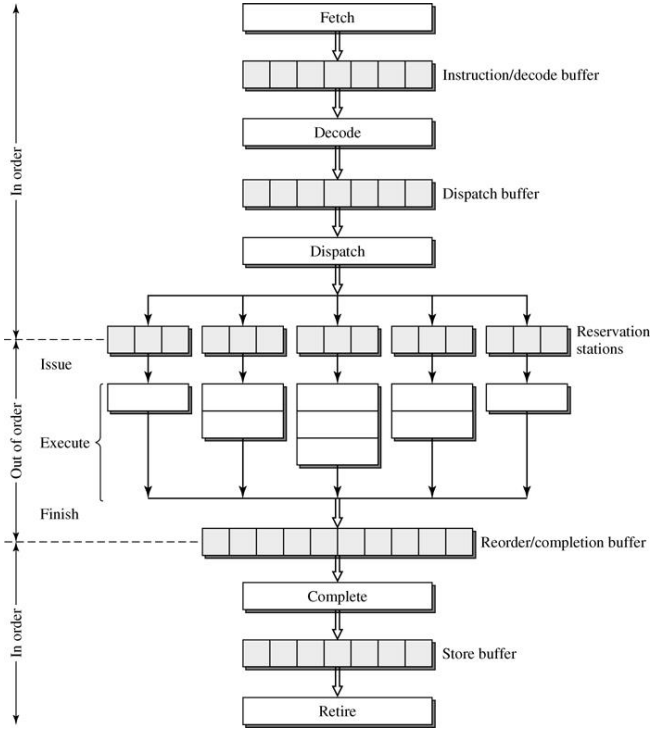
2.3.2.2 Dynamisches Scheduling – Dispatcher/Reservation Stations

- 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

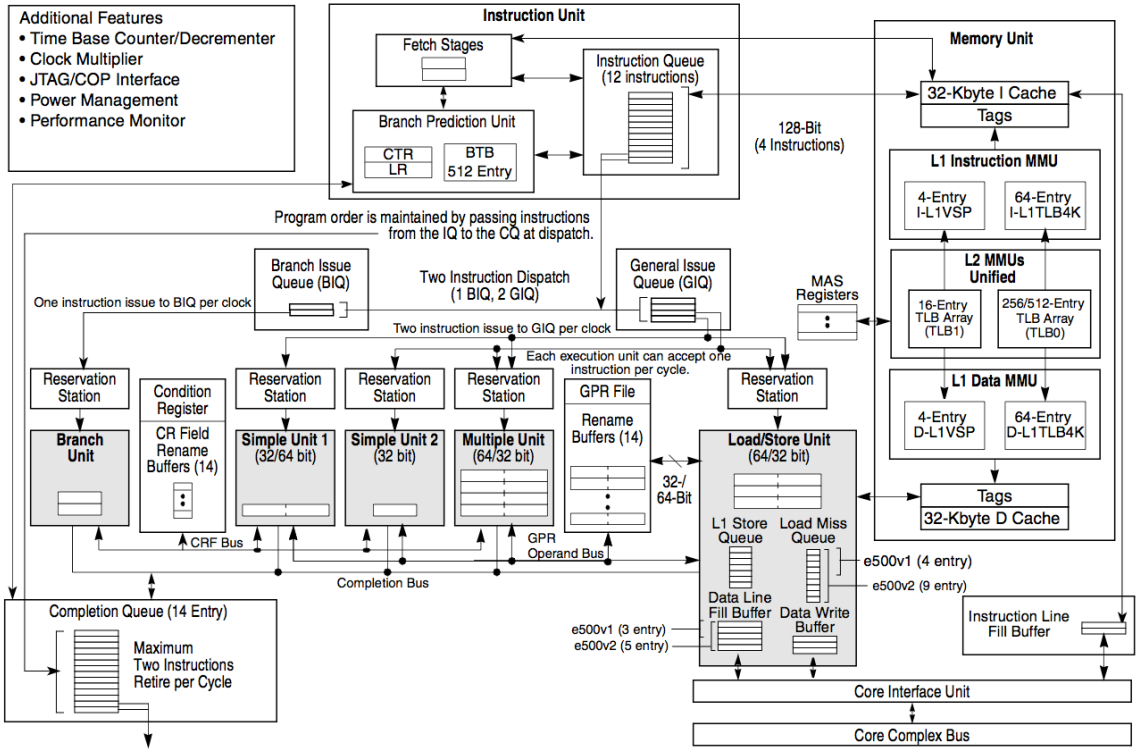


2.3.2.2 Dynamisches Scheduling – Completion-Unit

- 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



2.3.2.2 Beispiel - PowerPC e500 core



2.3.2.3 Very Long Instruction Word (VLIW) (I)

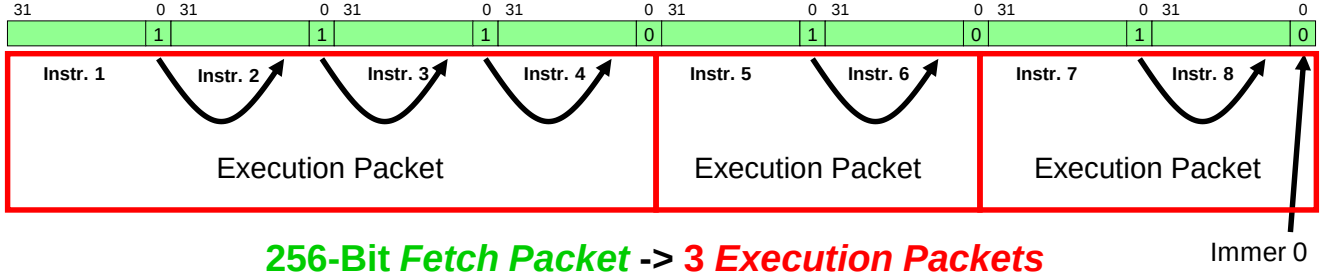
- VLIW = Very Long Instruction Word
- Architekturtechnik, bei der ein Compiler eine feste Anzahl von einfachen, voneinander unabhängigen Befehlen zu einem Befehlspaket zusammenpackt und in einem Maschinenbefehlswort meist fester Länge speichert.
- Das Maschinenbefehlsformat eines VLIW-Befehlspakets kann 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



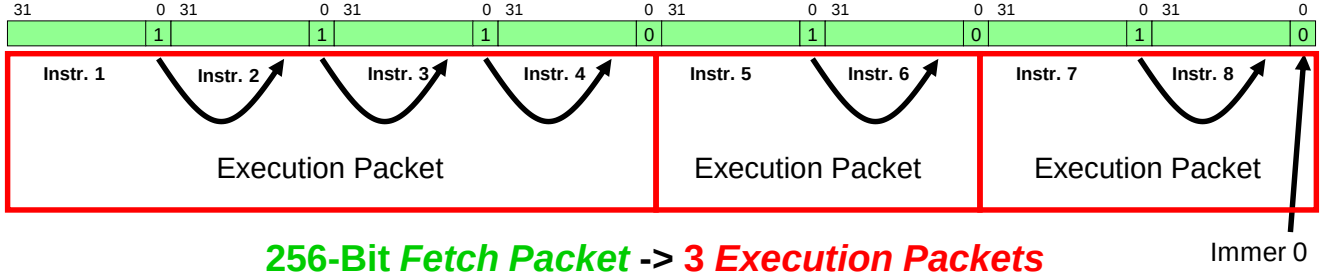
256-Bit *Fetch Packet* -> 3 *Execution Packets*

2.3.2.3 Very Long Instruction Word (VLIW) (II)

- 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 SIMD – Multimedia-Erweiterungen

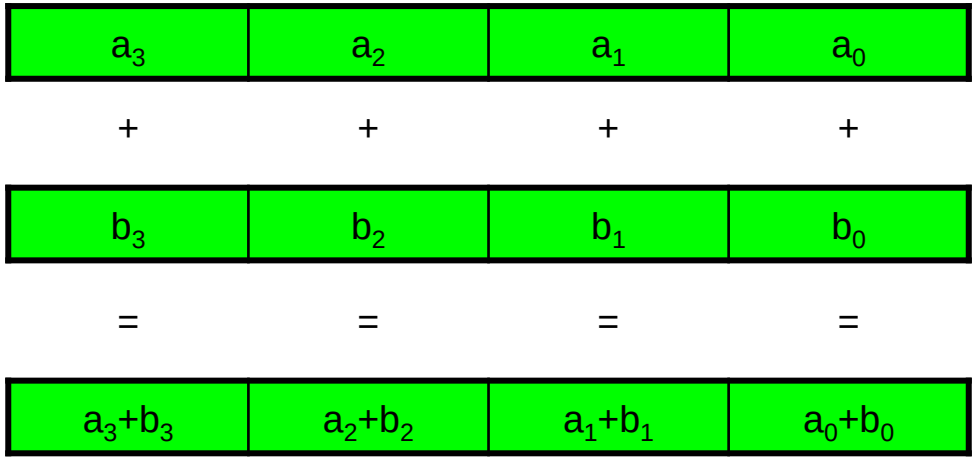
- Single Instruction, Multiple Data (SIMD)
  - Gleiche Instruktion auf mehreren Daten
- Multimedia-Anwendungen
  - z.B.: Audio/Video Playback, DVD, Videokonferenzen
  - 8/16-Bit Datentypen (z.B.: Pixeldarstellung in Bildverarbeitung)
  - viele arithmetische Operationen (Multiplikation, Addition)
  -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

- R1



- R2
- R3

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
    - Knobelaufgabe

Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

ITIV

Hardware/Software Co-Design

Übung 3 - Lösung

M.Sc. Fabian Lesniak

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 eine 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 eine Out-of-Order Ausführung und in welchem Abschnitt eine In-Order Ausführung statt?

- Die Out-of-Order Ausführung findet in allen Stufen zwischen den Reservation Stations und dem Reorder Buffer statt. Die Befehle werden In-Order in die Reservation Stations geschrieben, können aber Out-of-Order gestartet werden. Umgekehrt 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

- Beim Laden der Operanden wird nicht der tatsächliche Wert geladen, sondern ein Verweis auf ein internes Register vermerkt. Dieses interne Register enthält ein Valid-Bit, das gesetzt wird, sobald der Befehl berechnet ist. In der Reservation Station warten Befehl so lange, bis beide Operanden als valid markiert 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 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 können und die dritte 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
    - Hardwarebeschränkungen 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	88	00	00
PADDUSB	11	40	00	FF	88	88	FF	FF

- **PADDUSB** führt eine unsigned (U) Addition (ADD) mit Saturation (S) auf Packed-Bytes (B) aus.
- **PADDW** führt eine Addition (ADD) auf Packed-Words (W) aus.

Lösung Aufgabe 2.07: SIMD

b) Gegeben ist folgendes Fragment eines C-Programmes:

```
int   i;
char  a[8], b[8];        // char = 8-Bit signed 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, 0x0706050403020100
PCMPLTB         mm3, mm0, mm1
PNEG            mm4, mm3
PAND            mm3, mm3, mm2
PAND            mm4, mm4, mm1
POR             mm5, mm3, mm4
Am Ende gilt dann: b = 0x 01 07 05 09 03 02 01 00
```

PMOV mm0, 0x0123456789abcdef	Lädt den hexadezimalen Wert 0x0123456789abcdef in Register mm0
POR mm0, mm1, mm2	Bitweises ODER (mm0 = mm1 | mm2)
PAND mm0, mm1, mm2	Bitweises UND (mm0 = mm1 & mm2)
PNEG mm0, mm1	Bitweise Negation
PCMPGTB mm0, mm1, mm2	Packed-Byte Vergleich zwischen mm1 und mm2. Wenn Packed-Byte in mm1 > mm2 ist, wird in mm0 als Ergebnis 0xFF abgelegt, ansonsten 0x00. Beispiel: MM1= 04 70 45 67 09 0A 00 04h MM2= 03 71 00 20 01 0A 02 0Fh PCMPGTB MM0, MM1, MM2 -> MM0= FF 00 FF FF 00 00 00 00h

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