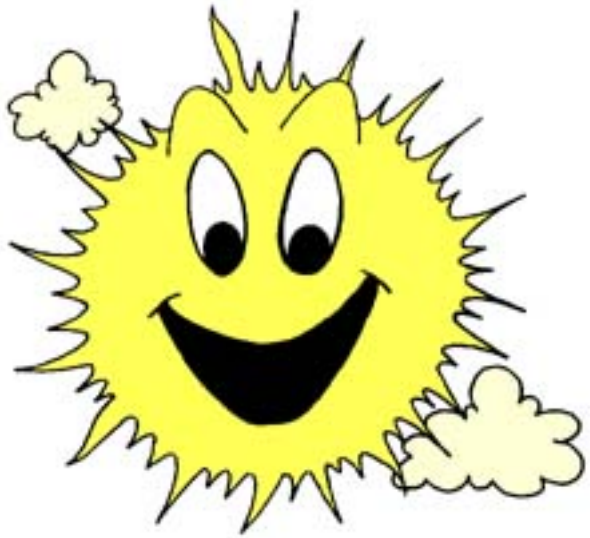
Wiederholung der 9. Vorlesung



6.1.7.1 Bus-Zyklen

Die zeitliche Abfolge von Signalen auf den Bussen zum Transfer eines Datenwortes zwischen den Einheiten des Rechners wird als **Buszyklus** bezeichnet.

Für jeden Buszyklus gibt es immer genau einen **Bus-Master**, der die logische und zeitliche Abfolge der Signale beim Transfer steuert. Üblicherweise ist die CPU der Bus-Master.

Lesezyklus:

- CPU (Master): → Kontr.Signale + Adresse
- Speicher: Daten von Adresse → Datenbus
- CPU: deakt. Kontr.Sign.

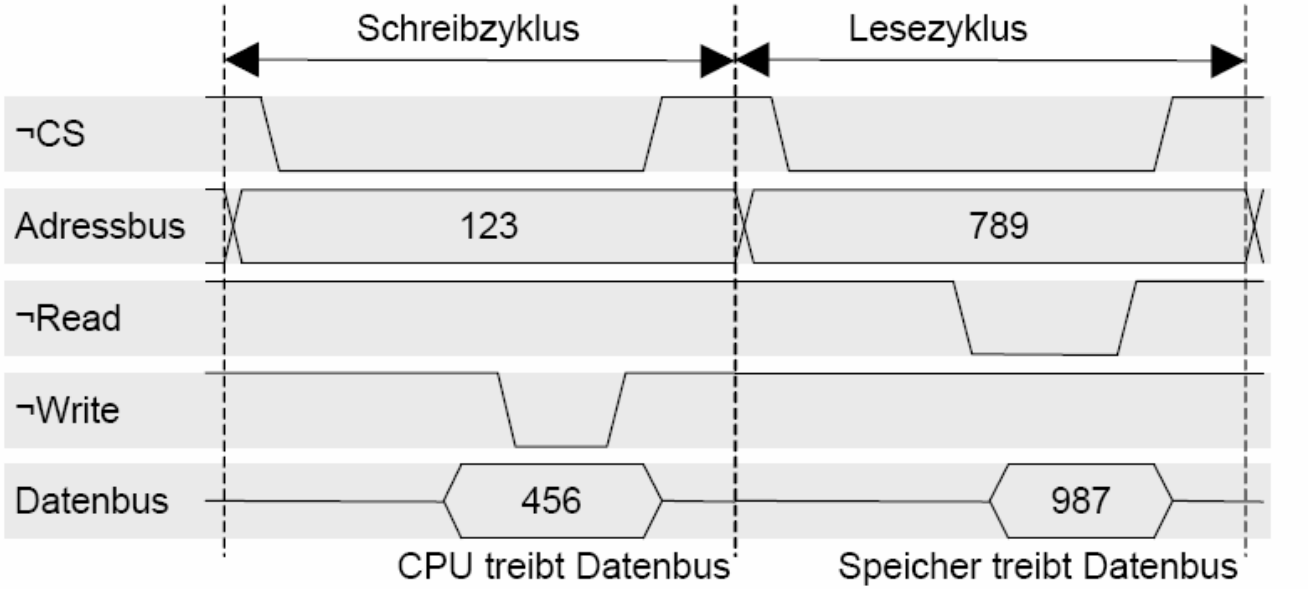
Speicher (Quelle) → CPU (Senke)

Schreibzyklus:

- CPU (Master): → Kontr.Signale + Adresse
- CPU: Daten → Datenbus
- Speicher: Daten ← DB
- CPU: deakt. Kontr.Sign.

CPU (Quelle) → Speicher (Senke)

6.1.7.1 Buszyklen: Synchroner Schreib-Zyklus

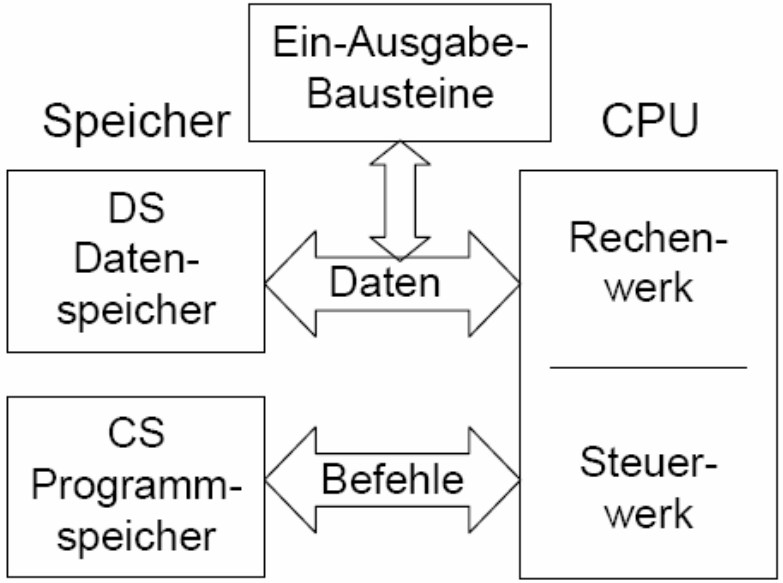


1. CPU legt Adresse an
2. CPU aktiviert ¬CS, Speicher wertet Adresse aus, überprüft Zuständigkeit.
3. CPU legt Daten an
4. CPU aktiviert ¬Write,
5. Speicher übernimmt Daten in Adresse
6. CPU deaktiviert ¬Write, Speicher muss Daten übernommen haben.
7. CPU deaktiviert Datenbus, ¬CS

6.2 Harvard-Architektur

Harvard - Architektur

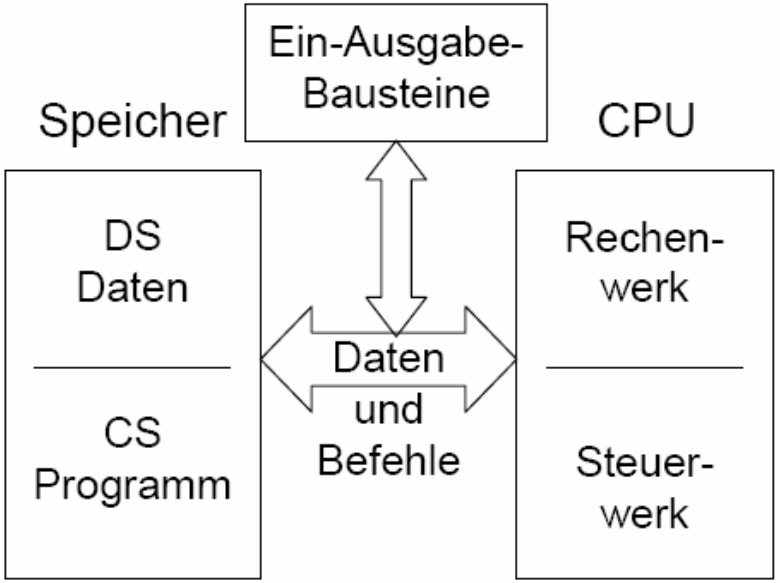
Befehle und Daten in getrennten Speichern



- Je ein Befehls- und Datenbus
- Schneller gleichzeitiger Zugriff auf Code und Daten
- Hauptanwendung: Signalprozessoren

Von Neumann – Architektur (Princeton Architektur)

Befehle und Daten im gemeinsamen Speicher



- Nur ein Bus für Befehle und Daten
- Flexible Aufteilung zwischen Code und Daten
- Hauptanwendung: Allgemeine Computer

➢ Gemeinsames Kennzeichen: Sequentielle Abarbeitung des Programms / der Befehle
➢ Heute sind häufig Mischformen anzutreffen

6.3 CISC – RISC (Complex ↔ Reduced Instruction Set)

Im Laufe der Jahre waren den ursprünglich einfachen Maschinebefehlssätzen immer mehr spezialisierte (komplexe) Befehle hinzugefügt worden (CISC).

Man wollte Hochsprachenkonstrukte besser unterstützen:

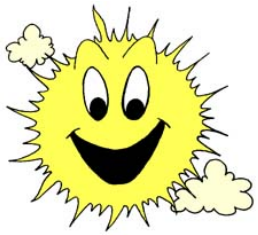
Befehlsbreite nimmt zu → Befehls-Phase braucht mehr Zyklen → Prozessor aufwendiger und langsamer.

Schließen der „semantischen Lücke“: Fähigkeit Hochsprachen ←→ Maschinen

- Komplexe Assembler → komplexe Compiler
- Hochsprachenbefehl passt u.U nicht zu Assembler

→ Erkenntnis der 70er: Hochsprachen benutzen überwiegend nur sehr einfache Befehle, komplexe nur sehr selten.
→ **RISC**-Prozessoren

Ende der Wiederholung



6.4 RISC (Reduced Instruction Set)

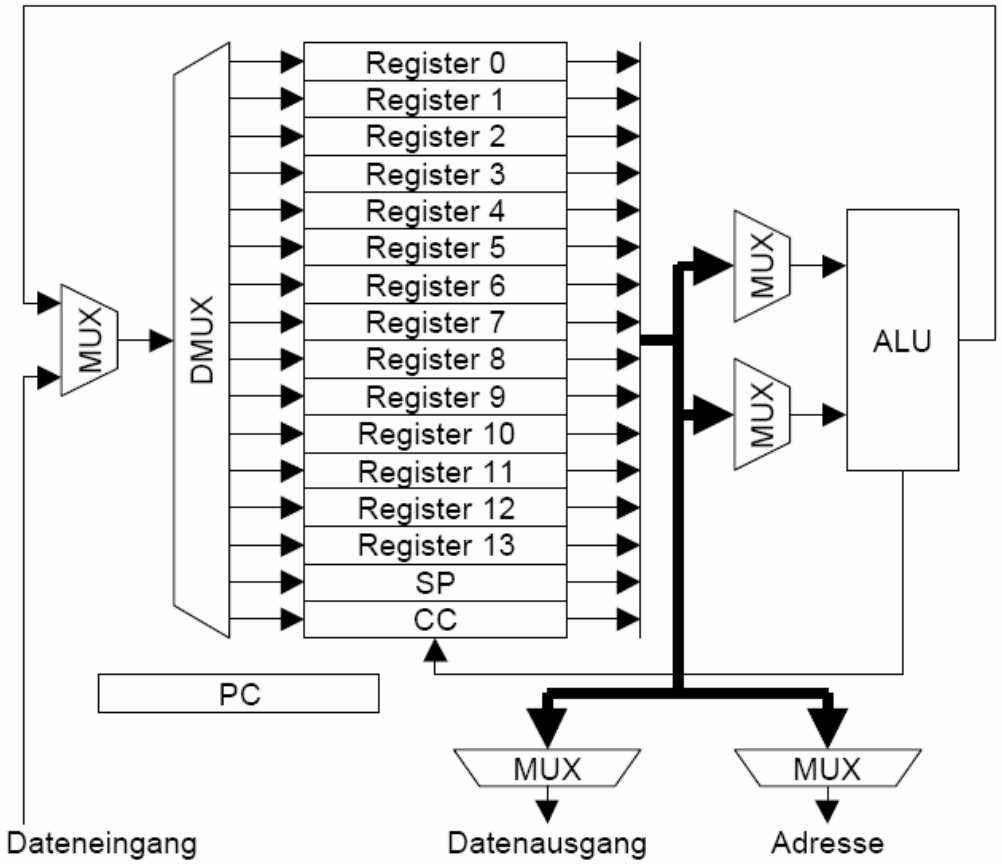
Informatik

Zitat *Antoine de Saint-Exupéry*:
„*Perfektion ist nicht erreicht, wenn nichts mehr hinzugefügt werden kann, sondern wenn nichts mehr übrig ist, was man weglassen kann*.“

Kennzeichen von RISC-Prozessoren:

- Großer Registersatz lokal im Prozessor:
  - Speicherung und schneller Zugriff auf Operanden.
  - Registerarithmetik
- Adressierung
  - Unmittelbar (immediate): Operand in Instruktion selbst gespeichert.
  - Register indirekt: Register enthält Speicher-Adresse des Operanden.
- Feste Länge der Instruktionen
- Eingeschränkter Befehlssatz
- Pipelining

6.4 RISC : Register – ALU – Systembus



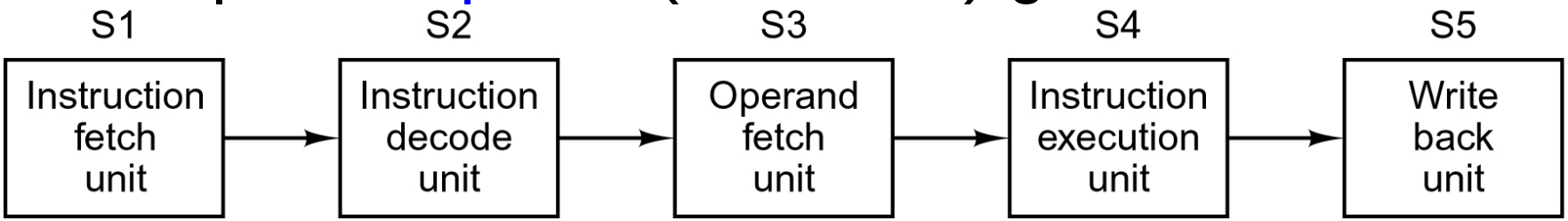
- Jedes Register kann Operand für ALU sein
- Jedes R. kann Ergebnis aufnehmen
- Jede R. kann Adresse liefern bei Schreib/Lese-Operationen
- Jedes R. kann Wert auf Datenbus legen bei Schreib-Operationen

6.4 Pipelining (Fließbandverabeitung)

Informatik

Das Prinzip des Vorauslesens (prefetch) schon länger bekannt (IBM 1959). Spaltet Ausführung in zwei Teile: Abruf und Ausführung.

Konzept der Pipeline (Fließband) geht noch weiter:



Stufen (steps) der Pipeline

S1: Abruf Instruktion aus Speicher

S2: dekodiere Instruktion, bestimme Typ und Operanden

S3: Suche Operanden, Abruf aus Register oder Speicher

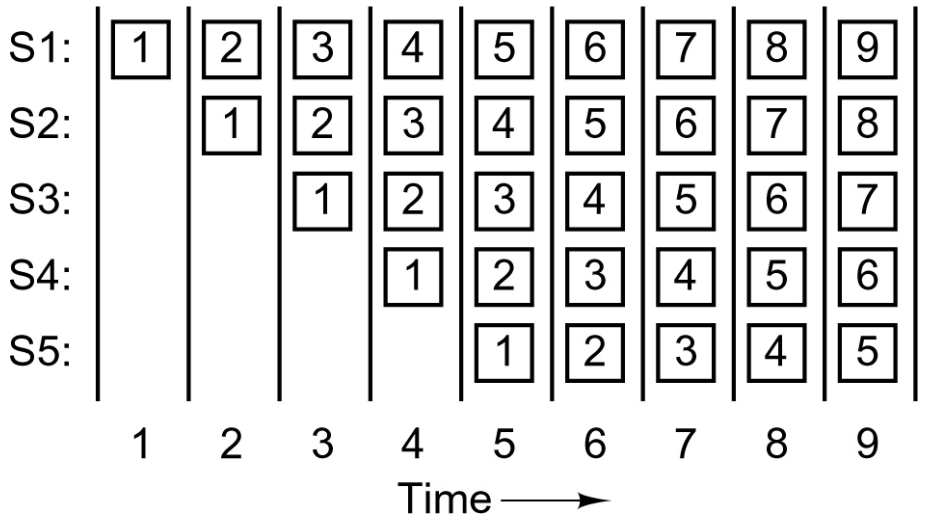
S4: Operation ausführen

S5: Schreibe Ergebnis in Register

6.4 Beispiel Großbäckerei

Kuchenbacken und versandfertiges Verpacken:
Langes Fließband, fünf Arbeiter: alle 10 Sekunden

1. Leere Kuchenschachtel auf Fließband stellen
2. Kuchen hineinlegen
3. Schachtel schließen und versiegeln
4. Etikett auf Schachtel kleben
5. Schachtel von Fließband nehmen und zum Versand auf Palette stellen



Analogie:

- Fließband = Pipeline
- 10 Sek = Taktfrequenz, Zykluszeit T
- Arbeiter = Verarbeitungseinheit, Stufe
- Latenzzeit n×T = 50 Sec
- Bandbreite 1 Kuchen/10 Sec

6.4 Hazards – Konflikte

Die durch die versetzte Ausführung der Instruktionen entstehenden Konflikte werden als Hazards bezeichnet. Man unterscheidet:

- **Struktureller Hazard**:
  Mehrere Teiloperationen unterschiedlicher Instruktionen müssen auf gleiche Ressourcen zugreifen.
- **Datenhazard**:
  Ergebnis einer Instruktion liegt noch nicht vor, wenn sie von einer nachfolgenden Instruktion benötigt wird.
- **Kontrollhazard**:
  Instruktionen direkt hinter einem Sprungbefehl werden noch in die Pipeline geladen, bevor der Sprung ausgeführt wird.

Das Assemblerprogramm muss gewährleisten, dass ein Operand erst dann verwendet wird, nachdem er geladen ist. Dies ist Aufgabe des Assembler-Programmierers oder des Compilers.

6.4 Hazard – Beispiel

Unter Speicheradresse 200 sei eine Zahl abgespeichert. Diese Zahl soll um den konstanten Wert 5 erhöht werden.

	T1	T2	T3	T4	T5	T6	T7
Instruktion holen	1	2	3	4			
Instruktion ausführen		1	2	3	4		
Speicherzugriff					3	3	3
					Konflikt!!		Operand verfügbar

```
1: LD R2,#5          ; Lade R2 mit dem konstanten Wert 5
2: LD R1,#200        ; Lade R1 mit dem konstanten Wert 200
                       (Adresse der Zahl)
3: LD R1,(R1)        ; Lade Operand aus Sp. (Reg.-indirekte
                       Adressierung mit R1) nach R1
4: ADD R1,R1,R2      ; Zum Wert aus dem Speicher konstanten Wert
                       5 addieren (Konflikt !!!)
```

6.4 Hazard – Beispiel (2)

Informatik

Zweiter Versuch eines Assembler Programms:

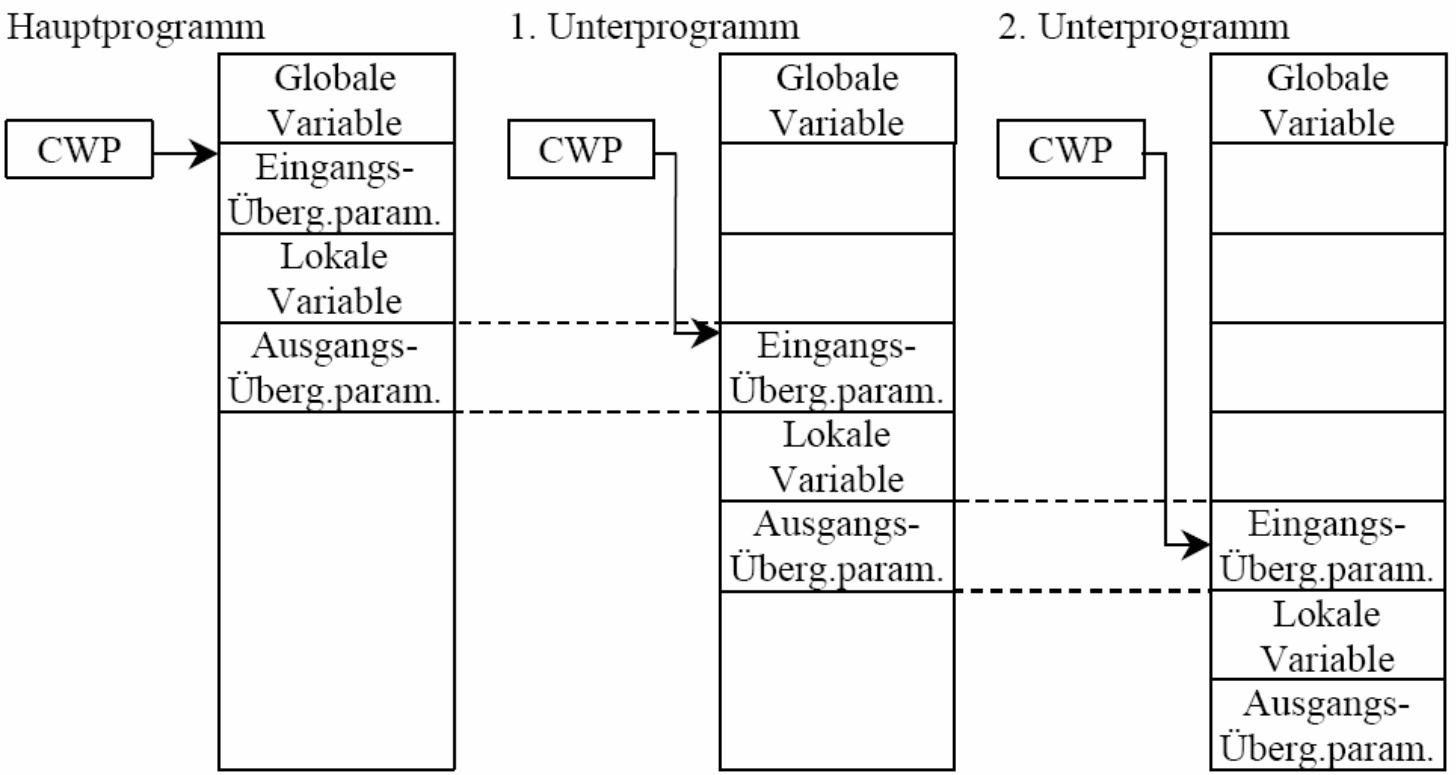
```
A: LD R1,#200        ; Lade R1 mit dem konstanten
                       Wert 200 (Adresse der Zahl)
B: LD R1,(R1)        ; Lade Operand aus Sp. (Reg.-indirekte
                       Adressierung mit R1) nach R1
C: LD R2,#5          ; Lade R2 mit dem konstanten Wert 5
D: NOP               ; Nop-Befehl (No Operation)
E: ADD R1,R1,R2      ; Zum Wert aus dem Speicher
                       konstanten Wert 5 addieren
```

	T1	T2	T3	T4	T5	T6	T7
Instruktion holen	A	B	C	D	E		
Instruktion ausführen		A	B	C	D	E	
Speicherzugriff				B	B	B	
						Operand verfügbar	

Operand steht jetzt genau zum richtigen Zeitpunkt zur Verfügung. →

6.4 Überlappende Registerfenster

Wenn ein Programm in ein Unterprogramm verzweigt, oder ein Interrupt verarbeitet wird, müssen Übergabeparameter bereitgestellt sowie alle Register gerettet (gespeichert) und später wieder restauriert (geladen) werden. → **zeitintensiv**!



Überlappende Registerfenster mit Zeiger CWP (CWP: current window pointer) sparen Zeit.

6.4 CISC ←→ RISC

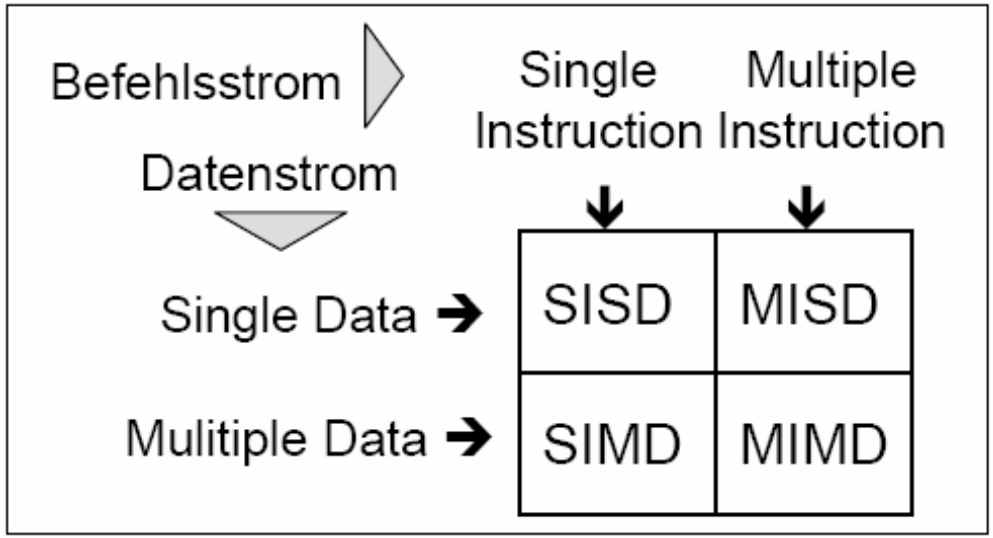
CISC	RISC
Komplexe Instruktionen, Ausführung in mehreren Taktzyklen.	Einfache Instruktionen, Ausführung in einem Taktzyklus.
Jede Instruktion kann auf den Speicher zugreifen.	Nur Lade- und Speicherbefehle greifen auf den Speicher zu.
Kein oder wenig Pipelining.	Intensives Pipelining.
Instruktionen werden von Mikroprogramm interpretiert.	Instruktionen werden durch festverdrahtete Hardware ausgeführt.
Instruktionsformat variabler Länge.	Instruktionen alle mit fester Länge.
Viele Instruktionen und Adressierungsarten.	Wenige Instruktionen und Adressierungsarten.
Die Komplexität liegt im Mikroprogramm.	Die Komplexität liegt im Compiler.
Einfacher Registersatz.	Mehrere Registersätze.

Fazit RISC:

- Häufigste Befehle einer Hochsprache → einfache Befehle in Maschinensprache
- Feste Instruktionslänge – feste Bearbeitungszeit
- Festverdrahtete Logik
- Komplexität: µProgramm- → Programmebene
- Minimierung Speicherzugriffe → viele Register
- Compiler für korrekte Beschickung der Pipeline verantwortlich.

6.5 Struktur nach Befehls- und Datenstrom

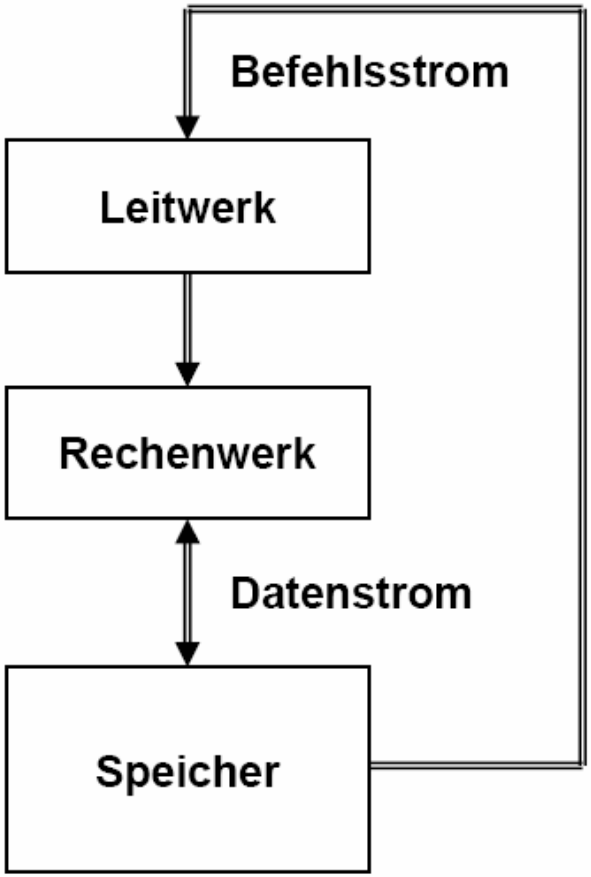
- **Single Instruction – Single Data** (SISD)
  Je einen sequenziell abzuarbeitenden Befehls und Datenstrom
- **Single Instruktion – Multiple Data** (SIMD)
  Ein sequenzieller Befehlsstrom, mehrfach paralleler Datenstrom
- **Multiple Instruction – Single Data** (MISD)
  Mehrerer Steuerwerke steuern einen Datenstrom
- **Multiple Instruction – Multiple Data** (MIMD)
  Sowohl Befehle, als auch Daten werden parallel abgearbeitet.



Literaturhinweise:
Flynn, M.J.: Computer Architecture. Jones and Bartlet, 1995

Flynn, M.J.: Some Computer Organizations and their Effectiveness. IEEE Transactions on Computers 21, 1972, pp 948 - 960

6.5 SISD-Rechner



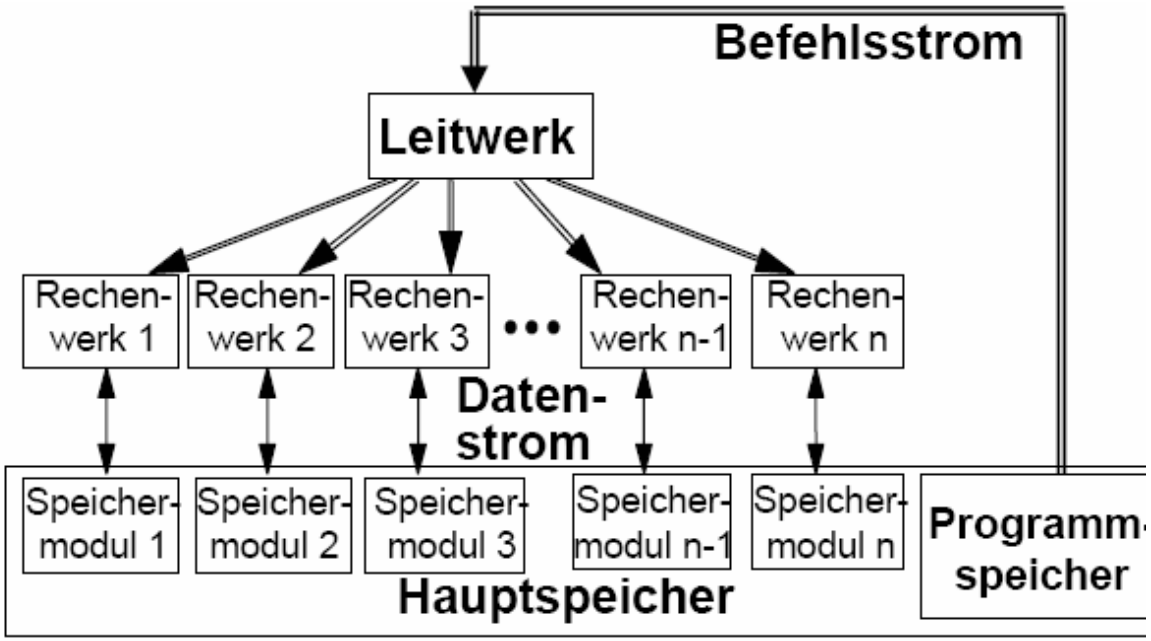
Informationsfluss beim SISD-Rechner

Die meisten Einzelrechner sind SISD-Rechner:

- CISC- und RISC-Rechner
- Die meisten µProcessoren (Intel ≤ 80486)
- Fast alle Mainframe-Computer
- Fast alle Minicomputer der 80er
- Pipeline-Processoren

Beim echten SISD-Rechner werden auch alle I/O-Vorgänge vom Prozessor selbst erledigt, ggf. Mit Hilfe intelligenter Controller.

6.5 SIMD-Rechner

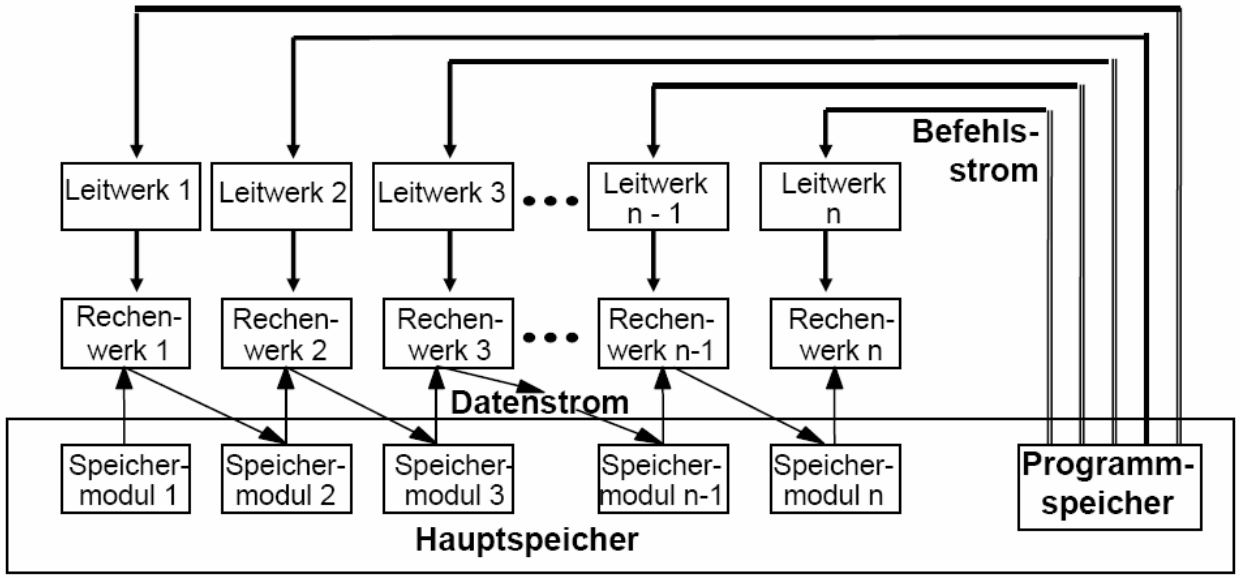


- **Vektorrechner** mit parall. Rechenwerk und parall. Speicher-zugriff.
- **VLIW** (very long instruction word, 5-10 Operationen). Spezialprocessoren für (De-) Komprimierung von Bilddaten.
- **MMX**-Einheit von Intel-Prozessoren

Nur einen Befehlsstrom steuert mehrere parallele Datenströme:

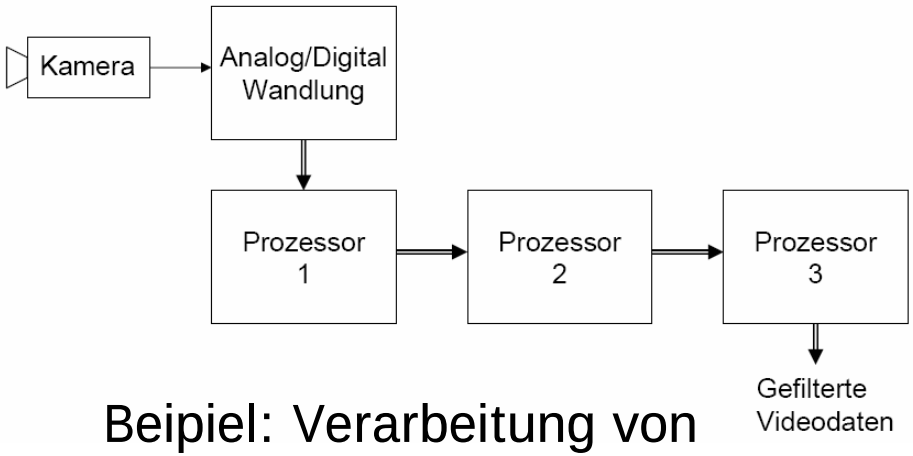
- Nur ein Leitwerk zur Befehlsentschlüsselung
- Benutzung mehrerer paralleler Rechenwerke. Verteilung Befehlsinformationen „Instruction Broadcasting“
- Jede Einheit verfügt über bidirektionalen Zugriffspfad zum Hauptspeicher (Erweiterung der von Neumann Struktur)

6.5 MISD-Rechner



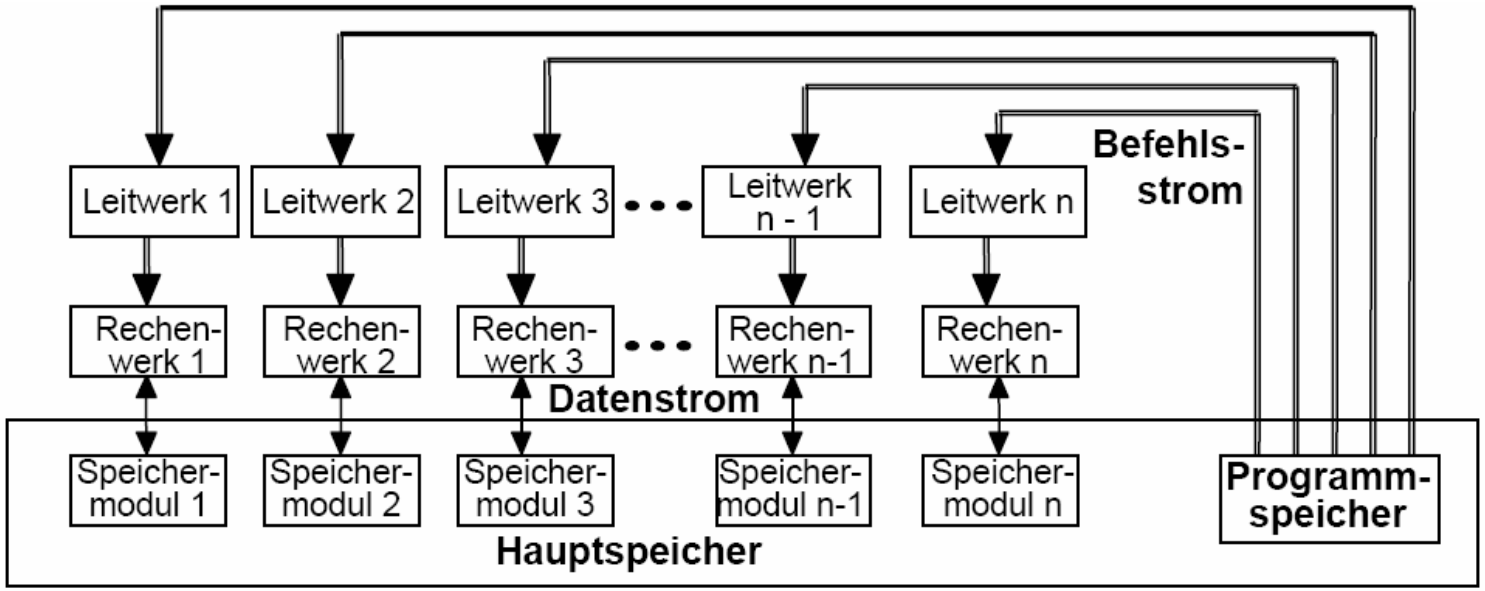
Mehrere Rechenwerke bearbeiten eine Datenstrom:

- Gesamtaufgabe wird in Teilschritte zerlegt
- Softwaretechnisch umgesetzt in den „pipe“-Befehlen in UNIX Betriebssystemen: Ein Prozess versorgt den nächsten mit Daten.



Beipiel: Verarbeitung von Videodaten

6.5 MIMD-Rechner



Parallele Bearbeitung von Befehlen und Daten:

- Leitwerk, das die Rechenwerke steuert, bezeichnet man als „Prozessor“
- Anwendungen:
  - Mit Video- und Soundkarten etc. hat moderner Computer erhebliche Parallelität implementiert.
  - Hauptprozessor selbst (ab Pentium und AMD K5) ist hoch parallel.
  - PC‘s und Server mit mehreren CPU
  - Parallelrechner, Supercomputer ...

6.5 Moderne Rechnerarchitekturen

Moderne Rechner können mehrere Befehle gleichzeitig bearbeiten → „Multiple Issue Prozessoren“:

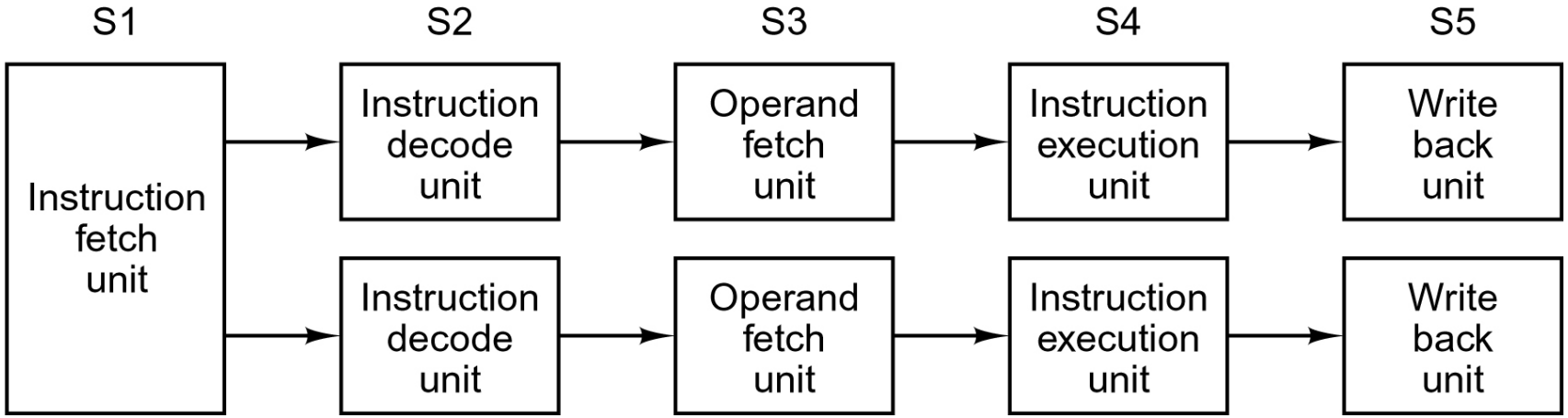
- VLIW – very long instruction word
- Superskalare Architekturen

Beide Typen besitzen mehrere, parallele Verarbeitungspipelines (Arithmetik, Gleitkomma, Logik, Multimedia, etc.), in denen parallel Instruktionen ausgeführt werden. Damit lassen sich mehrere Instruktionen pro Taktzyklus ausführen.

6.5 Superskalare Architekturen

Bei superskalare Architekturen besitzen mehrere Pipelines. Instruktionseinheit ruft mehrere Befehle ab und plaziert jede in eine eigene Pipeline. Jede Pipeline verfügt über eine eigene ALU.

Hazard und Konfliktvermeidung ist Sache des Compilers, nicht der Hardware.



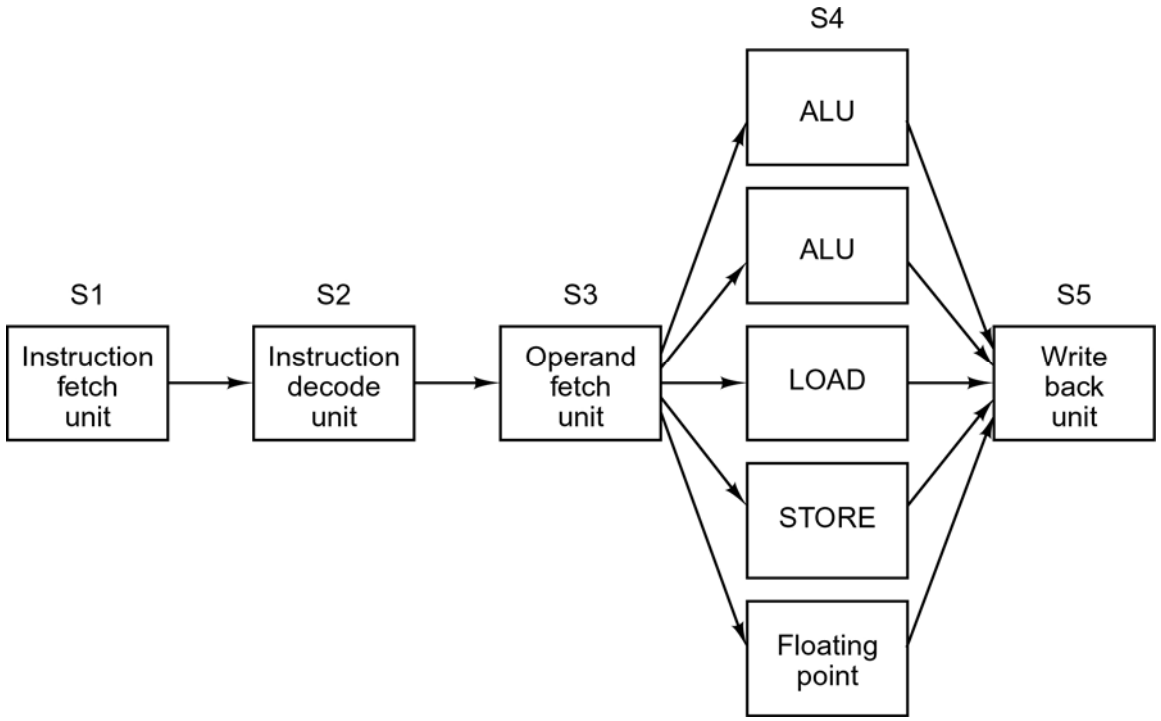
Zwei mal fünfstufige Pipeline mit einer gemeinsamen Instruktions-einheit. Beispiel Pentium

6.5 Superskalare Architekturen (2)

In superskalaren Architekturen kann Stufe S3 wesentlich schneller Instruktionen aufrufen, als S4 sie ausführen.

S4 braucht häufig mehr als einen Taktzyklus, besonders bei Fließkommaoperationen und Speicherzugriff.

Daher kann es in Stufe 4 mehrere ALU‘s geben:



6.5 Superskalare Prozessorfamilien

Hersteller	Prozessorfamilien
Intel	Pentium P5, Pentium P6
AMD	K5, K6, K7
Cyrix	M II, M3
Hewlett Packard	PA-7000, PA-8000
Sun	UltraSPARC
DEC	Alpha 21x64
MIPS Technologies	R1x000
IBM, Motorola, Apple	PowerPC