RheinlandPfalz

# Materialien

zum

# Lehrplan Informatik

Grundfach

in der Oberstufe des Gymnasiums

*(Mainzer Studienstufe)*

# Rechnerarchitekturen

Landesmedienzentrum

Satz:
Gregor Noll, Erpel am Rhein

Druck und Vertrieb:
Landesmedienzentrum Rheinland-Pfalz
Hofstraße 257c, D-56077 Koblenz-Ehrenbreitstein
Telefon (0261) 97 02-3 00
Telefax (0261) 97 02-3 62

Juni 2001

# Vorwort

Das hier vorgelegte Material zum Thema „Rechnerarchitekturen" versucht, die sich fortwährend weiterentwickelnden Konstruktionselemente von Computern auf zeitlose Grundkomponenten und Architekturen zu reduzieren.

Dabei wird die Entwicklung vom „elementaren" Von-Neumann-Rechner bis hin zu Vektorrechnern und Multiprozessorsystemen ausführlich diskutiert und mit vielen Abbildungen veranschaulicht.

Vorrangiges Ziel ist es, Hintergrundwissen für Lehrerinnen und Lehrer bereit zu stellen. Für einen unmittelbaren unterrichtlichen Einsatz müssen weitergehende methodische Aufbereitungen erfolgen. Hierzu findet sich ein erster Ansatz in der begleitenden Hypertextpräsentation von M. Spengler, die über unsere Homepage http://informatikag.bildung-rp.de auf dem Bildungsserver von Rheinland-Pfalz online eingesehen werden kann. Die Präsentation steht dort auch als Acrobat-File im pdf-Format zum Download bereit.

Entwurf und Herstellung von Rechenmaschinen waren schon zu den Zeiten von Pascal und Leibniz Herausforderungen, die nicht nur auf die Entwickler einen ungeheuren Reiz ausstrahlten. Daran hat sich im Zeitalter des Computers nichts geändert. Wir sind davon überzeugt, dass sich diese Faszination auch im Informatikunterricht fruchtbringend vermitteln lässt und möchten Lehrerinnen und Lehrer mit unserem Material zu entsprechenden Unterrichtssequenzen ermuntern. Über Rückmeldungen würden wir uns sehr freuen.

Arbeitsgruppe Informatik
des Landesmedienzentrums Rheinland-Pfalz

# Rechnerarchitekturen

**Herbert Drumm**

unter Mitarbeit der Arbeitsgruppe Informatik
des Landesmedienzentrums Rheinland-Pfalz

**Mitglieder der Arbeitsgruppe**

Dr. Klaus-Peter Becker

Dr. Herbert Drumm

Josef Glöckler

Gregor Noll

Werner Rockenbach

Mario Spengler

# Inhaltsverzeichnis

*Computer sind schnell,*
*gute Gedanken sind langsam.*

*Peter Bamm*





# 1 Einleitung

Der Informatikunterricht muss ein Verständnis der Prinzipien des Von-Neumann-Rechners erzeugen, da dieser die Basis der heute von Schülerinnen und Schülern benutzten Computer darstellt. Dazu gibt es im Lehrplan mehrere Ansatzpunkte, insbesondere im Rahmen des Kapitels „Technische Grundlagen". Darüber hinaus bietet sich die Möglichkeit einer umfassenderen Behandlung, Vertiefung und Erweiterung im Themenkreis „Aktuelle Entwicklungen". Hier stehen Weiterentwicklungen des Von-Neumann-Rechners, andere Rechnerarchitekturen und eventuell auch andere Rechnerkonzepte im Mittelpunkt. Sie sind nicht nur zu erwähnen, sondern sollten ausführlich behandelt werden.

Eine unterrichtliche Realisation des Themas „Rechnerarchitekturen" kann auf verschiedene Weise erfolgen:

- Nach Assemblerstrukturen, der Abarbeitung von Assemblerbefehlen und des grundlegenden Rechnerprinzips werden die Rechnerarchitekturen als Vertiefung und Weiterführung behandelt.
- Die rechnerbezogenen Assemblerhintergründe werden zunächst ausgespart und später als Einstieg in das Thema „Rechnerarchitekturen" benutzt.
- Zuerst wird die Von-Neumann-Architektur behandelt und daraus die Assemblergrundlagen abgeleitet. Daran schließt sich die Darstellung neuerer Hardwareentwicklungen an.

Es bleibt natürlich Aufgabe des Lehrers oder der Lehrerin, diese oder ähnliche Wege für den eigenen Unterricht konkret zu gestalten. Das Ziel der vorliegenden Materialien ist es, dazu grundlegende Fachinformationen zu liefern. Zunächst wird das Von-Neumann-Prinzip mit seiner Rechnerstruktur und die Abarbeitung eines Assemblerbefehls dargestellt. Es schließen sich Weiterentwicklungen der Von-Neumann-Architektur und einige daraus entstandene neuere Architekturen an. Andere Konzepte – etwa „Neuronale Netze" – werden nicht behandelt.

# 2 Der Von-Neumann-Rechner

Die Grundidee von Neumanns besteht in der Äquivalenz von Daten und Programmen, die in einem einzigen Speicher abgelegt sind, und erst bei Ablauf entsprechend interpretiert werden.

In seiner Grundstruktur muss der Rechner ein Steuerwerk und ein Rechenwerk enthalten, die zusammen die CPU bilden, sowie ein Speicherwerk und ein Ein-/Ausgabewerk. Diese Teile sind durch Bussysteme verbunden. Dabei werden auf dem Datenbus gemäß dem Grundprinzip dieses Rechners sowohl Daten als auch Befehle, zwischen denen hier ja nicht unterschieden wird, transportiert. Im folgenden Blockschaltbild ist dieser Aufbau skizziert, wobei auch die Richtung des Informationsflusses auf den Bussen angegeben ist.

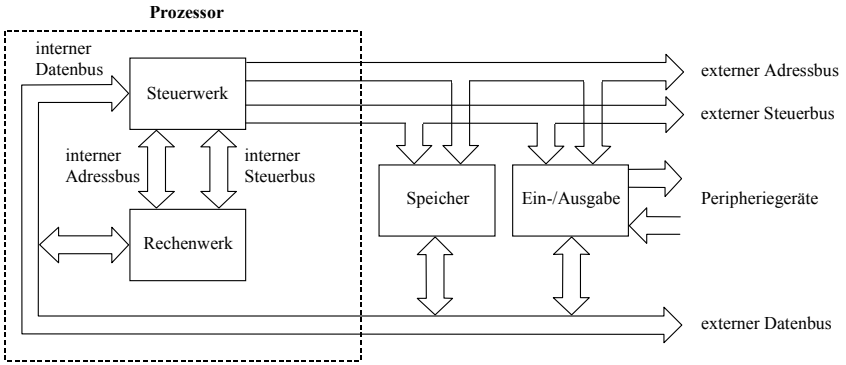



*Bild 1*

Eine genauere Untersuchung der einzelnen Werke und des Informationsflusses führt zum nächsten Blockschaltbild (*Bild 2*), in dem auch die prinzipielle Abarbeitung eines Befehls zu erkennen ist. Mit A sind dabei Adress-, mit B Befehls-, mit D Daten- und mit S Steuerleitungen bezeichnet.

Das Steuerwerk regelt den Ablauf eines Programms, das aus einer Folge von Maschinenbefehlen mit den dazugehörenden Daten besteht, durch Abarbeitung der einzelnen Maschinenbefehle. Dazu wird in der Fetch-Phase der Befehl, auf den der Befehlszähler (PC) gerade zeigt, aus dem Speicher in das Befehlsregister (IR) geholt und dekodiert. Dann schaltet das Steuerwerk gemäß dem durch seinen Operationscode festgelegten Maschinenbefehl auf den entsprechenden Steueralgorithmus (Mikroprogramm) um, der den Befehl explizit ausführt (Execute-Phase). Der Transport der dabei benötigten Daten vom Speicher in das Rechenwerk wird ebenfalls über das Steuerwerk geregelt.

Das Rechenwerk enthält einen Registerblock, der zur schnelleren Bearbeitung die Operanden und Ergebnisse aufnimmt, sowie ein steuerbares Schaltnetz, die Arithmetic Logic Unit (ALU), die eigentliche Recheneinheit.

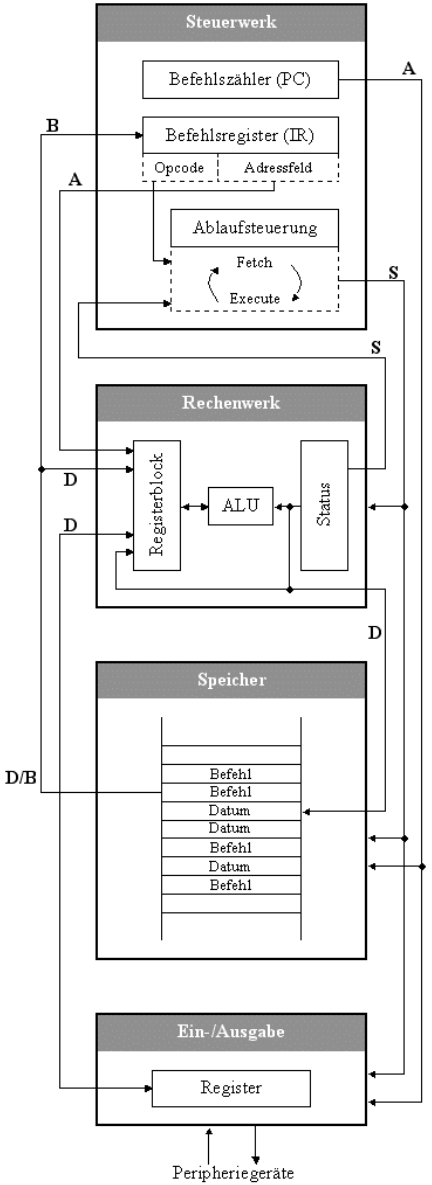



*Bild 2*

Die prinzipielle Abarbeitung eines Assemblerbefehls bei einer solchen Rechnerarchitektur ist im folgenden Struktogramm nochmals verdeutlicht.

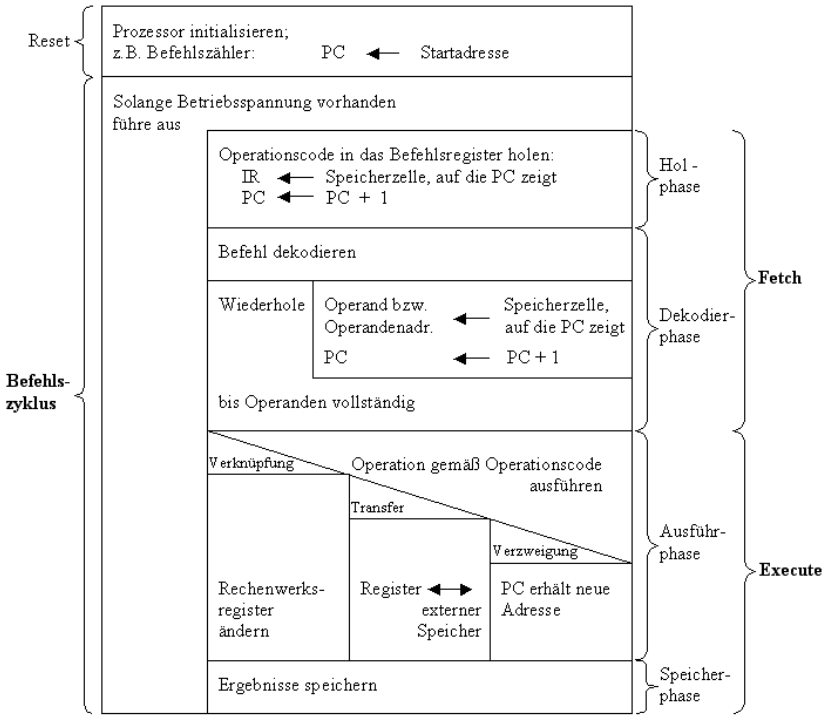



*Bild 3*



# 3 Weiterentwicklungen, Überblick

Das Von-Neumann-Prinzip führt einerseits zu einer relativ einfachen Rechnerstruktur, hat andererseits aber den Nachteil, dass alle Befehle und Daten nacheinander über das Bussystem transportiert und von einem einzigen Prozessor gesteuert und verarbeitet werden müssen („Von-Neumann-Flaschenhals").

Es gibt eine Reihe von Ansätzen zur Erhöhung der Effizienz dieses Rechners bzw. zur Entwicklung weiterführender Architekturen. Sie basieren alle neben der Erhöhung der Leistungsfähigkeit von Bauteilen auf einer Vermehrung und Parallelisierung von Komponenten.

Zur Gruppierung der dabei auftretenden Rechnertypen wird die bekannte – wenn auch nicht unumstrittene – Flynnsche Klassifikation der Rechnerarchitekturen benutzt. Sie fußt auf der Anzahl der Befehle und Daten, die eine Maschine gleichzeitig bearbeitet und unterscheidet demnach 4 Klassen:

- **SISD** (**S**ingle **I**nstruction – **S**ingle **D**ata): hierhin gehören alle sequentiellen Rechner (skalare Einprozessorsysteme), insbesondere die Von-Neumann-Rechner mit allen effizienzsteigernden Maßnahmen.
- **SIMD** (**S**ingle **I**nstruction – **M**ultiple **D**ata): in diese Klasse gehören die Rechner, die mit einem Befehl mehrere Daten parallel bearbeiten, also die Vektorrechner und die Felder von Rechenelementen.
- **MISD** (**M**ultiple **I**nstructions – **S**ingle **D**ata): hierhin gehören die Rechner, die zur Berechnung eines Datenwertes schon mehrere Befehle benötigen, d. h. dass der Prozessor dem Von-Neumann-Prozessor noch unterlegen wäre. Daher gibt es in dieser Klasse in der Praxis keinen Vertreter.
- **MIMD** (**M**ultiple **I**nstructions – **M**ultiple **D**ata): in diese Klasse gehören die Rechner, die parallel mehrere Befehle und mehrere Daten bearbeiten, also alle Multiprozessorsysteme und die Rechner mit mehreren Datenprozessoren.

Im Folgenden werden die drei relevanten Klassen näher untersucht, insbesondere im Hinblick auf Verbesserungen gegenüber dem ursprünglichen Von-Neumann-Rechner.

Eine graphische Aufbereitung für den Unterrichtseinsatz findet sich in der Hypertext-Präsentation von Mario Spengler (s. Abschnitt 8).

Der nachstehende Überblick soll die Orientierung erleichtern:

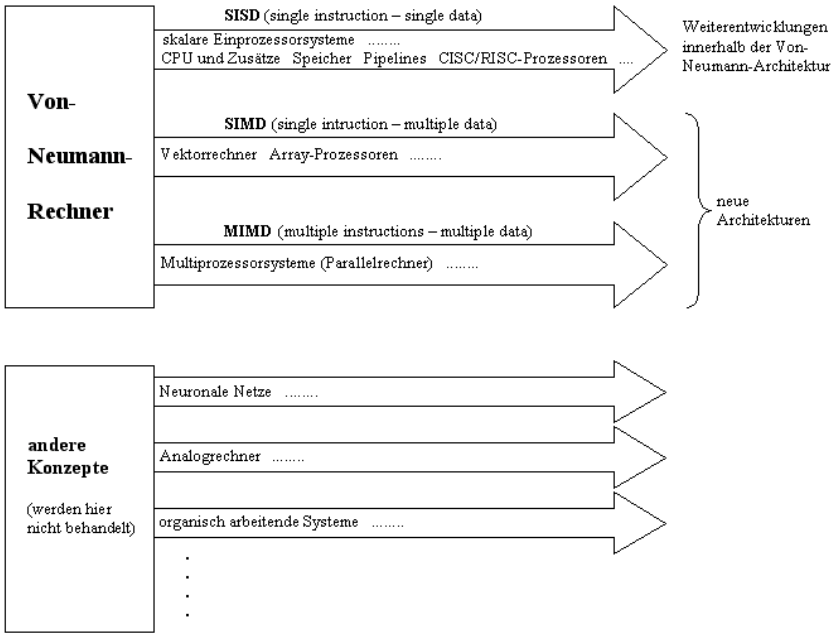



*Bild 4*



# 4 SISD-Architektur

## 4.1 Leistungssteigerung bei skalaren Einprozessorsystemen

Die Leistung eines solchen Prozessors, beispielsweise gemessen in MIPS (**M**illion **I**nstructions **P**er **S**econd), hat in den vergangenen Jahren ständig zugenommen (etwa 50% jährlich) und wird voraussichtlich noch einige Zeit so weitersteigen:

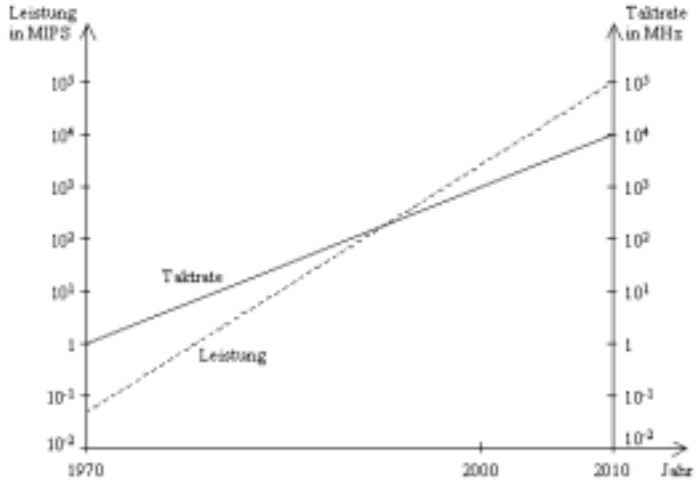



*Bild 5*

Zu dieser Graphik seien zwei Anmerkungen gemacht: Zum einen ist es klar, dass die Prozessorleistung mit der Taktfrequenz steigt. Dabei bedeutet eine Frequenz von 1 GHz, dass ein Taktzyklus 10 ns dauert, eine Zeit, in der Licht eine Strecke von 30 cm zurücklegt. Auf Grund der Schaltverzögerung der elektronischen Bauteile beträgt die Signalgeschwindigkeit in einem Chip zur Zeit nur etwa 10% der Lichtgeschwindigkeit. Daher liegt für diesen Fall die maximale Signallaufstrecke und damit die maximale Chipgröße bei 3 cm. Eine Erhöhung der Taktfrequenz muss also einhergehen mit einer weiteren Miniaturisierung und Verdichtung der Chips sowie der Entwicklung schnellerer Bauteile. Ebenso müssen die Bussysteme schneller, kürzer und breiter werden. Doch erreichen alle diese Maßnahmen langsam ihre Grenzen. Insbesondere ist der Miniaturisierung dadurch, dass die Bauteile immer mehrere Atome groß sein müssen, ein absolutes Limit gesetzt.

Zum anderen ersieht man aus der Graphik, dass die Rechenleistung der Prozessoren schneller steigt als die Taktfrequenz, mittlerweile sogar mehr als ein Befehl pro Taktzyklus abgearbeitet wird. Gründe für diesen günstigen Verlauf ergeben sich aus den folgenden Überlegungen.

Eine einfache Berechnungsformel für die Leistung ist

$$Leistung = \frac{f \cdot N_P}{A_\mu + A_m} \text{ [MIPS]}$$

Hierbei ist:

f die Taktfrequenz des Prozessors in MHz;
$N_p$ die Anzahl der voneinander unabhängigen Datenprozessen im Prozessor;
$A_\mu$ die mittlere Zahl von Mikroinstruktionen pro Maschinenbefehl;
$A_m$ die Anzahl der Taktzyklen für einen Speicherzugriff.

Bei gängigen CISC-Prozessoren (s. Kap. 4.5) betragen die mittleren Werte für $A_\mu$ etwa 5 - 7, für $A_m$ etwa 2 - 5. Damit ergeben sich für eine Taktfrequenz von 300 MHz Leistungen im Bereich von 25 MIPS bis 50 MIPS ($N_p$=1).

Bei modernen Prozessoren, insbesondere bei RISC-Prozessoren (s. Kap. 4.5), versucht man diese Größen nahe an ihren theoretischen Grenzwert zu bringen, d. h. dass dann bei jedem Takt ein vollständiger Maschinenbefehl ausgeführt wird und auch die Speicherzugriffszeit nur einen Taktzyklus dauert.

Eine entscheidende Verbesserung des Wertes von $A_m$ erhält man durch Einführung eines Cache-Speichers (s. Kap. 4.3) auf dem Prozessorchip. Der theoretische Wert 1 kann nicht ganz erreicht werden, da stets mit einer gewissen Häufigkeit Fälle auftreten, bei denen die benötigte Größe nicht im Cache-Speicher ist, und somit ein Hauptspeicherzugriff nötig wird. Durch geschickte Verwaltung und Belegung des Cache-Speichers lässt sich der Wert 1 aber fast erreichen.

Auch für $A_\mu$ wird mittlerweile der theoretisch mögliche Wert 1 fast erreicht. Dazu beschränkt man sich einerseits auf eine reduzierte Anzahl relativ einfacher Maschinenbefehle, die nicht mehr mikroprogrammiert sondern festverdrahtet sind (RISC-Prozessoren). Andererseits muss man Pipeline-Verfahren (s. Kap. 4.4) benutzen und zwar sowohl bei Befehlsausführung als auch bei längeren arithmetischen Operationen. Aber auch bei Pipeline-Verfahren kann der theoretische Wert nicht ganz erreicht werden, da das Füllen der Pipeline eine Verzögerung bedingt.

Eine weitere Verbesserung kann man erzielen, wenn man die Befehlsausführung und den Speicherzugriff zeitlich überlappt.

Mit all diesen Maßnahmen erhält man die angestrebte optimale Rechnerleistung für solche skalare Prozessoren. Dabei gibt es mittlerweile keine großen Leistungsunterschiede mehr zwischen modernen CISC- und RISC-Prozessoren.

Weiterhin existieren superskalare Prozessoren, bei denen der Prozessor einige Funktionseinheiten wie z. B. Datenprozessoren oder Pipelines mehrfach (interne parallele Verarbeitung möglich) oder auch höher getaktet enthält, so dass die MIPS-Leistung nochmals gesteigert wird. Es können auf diese Weise sogar mehrere Befehle pro Taktzyklus abgearbeitet werden.



## 4.2 Zusätze innerhalb der Von-Neumann-Architektur Die Harvard-Architektur

Durch Übertragung von zeitraubenden, oft benötigten Funktionen auf zusätzliche Hardware kann der Prozessor entlastet und für andere Aufgaben freigestellt werden. Man erreicht so eine gewisse Parallelisierung der Arbeit und eine Steigerung der Prozessorleistung. Da diese Elemente weitgehend bekannt sind, genügt hier eine kurze Zusammenstellung:

- **Mathematischer Coprozessor**: Dieser entlastet die CPU bei der Ausführung von Gleitkommaoperationen.
- **Direkter Speicherzugriff**: Der Prozessor kann einen DMA-Controller (**D**irect **M**emory **A**ccess) anstoßen, der dann selbstständig, ohne weitere Prozessorintervention, ganze Datenblöcke direkt zwischen dem Hauptspeicher und einem Ein-/Ausgabegerät überträgt.
- **Ein-/Ausgabeprozessoren**: Diese können zusätzlich zu einem DMA-Controller auch mehrere Ein-/Ausgabegeräte bedienen oder auf Zustandsänderungen der Geräte reagieren, also programmierte Ein-/Ausgabeoperationen durchführen.
- **Harvard-Architektur**: Im Gegensatz zur Von-Neumann-Architektur (Princeton-Architektur) werden hier getrennte Speicher für Befehle und Daten benutzt und auch getrennte Befehls- und Datenbusse. Der Aufbau ist komplizierter, aber man gewinnt Geschwindigkeit auf Grund des parallelisierten Transportes von Befehlen und Daten. Diese Architektur, die z. T. schon bei den ersten Computern verwendet wurde, gewinnt derzeit wieder an Bedeutung, wobei zur Begrenzung des Aufwandes meist ein gemeinsamer Hauptspeicher, aber getrennte Cache-Speicher für Befehle und Daten benutzt werden.

## 4.3 Cache-Speicher

In der folgenden Darstellung ist ein Überblick über verschiedene Speichertypen gegeben samt der durch ihre Geschwindigkeit bedingten Hierarchie.

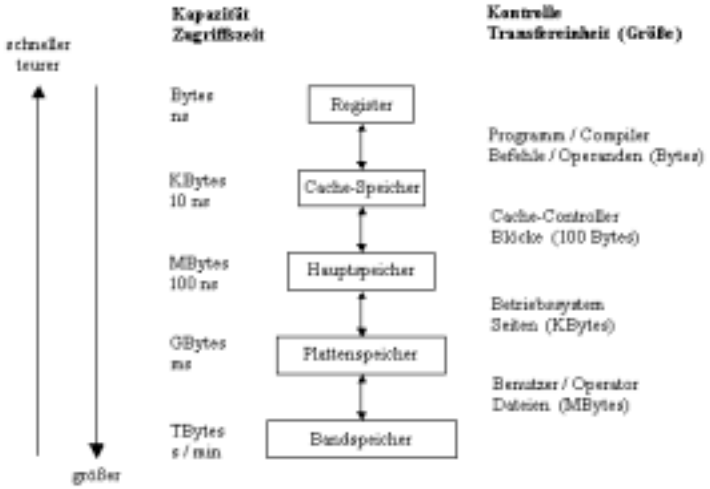



*Bild 6*

Für moderne Prozessoren bildet die Zugriffszeit auf den Hauptspeicher eine große Verzögerung, so dass man mittlerweile einen deutlich schnelleren, aber auch deutlich teureren Cache-Speicher dazwischenschaltet, der räumlich nahe am Prozessor liegt bzw. in den Prozessorchip integriert ist. Heute hat man oft mehrere Hierarchiestufen von Cache-Speichern, von denen dann der Speicher der höchsten Stufe auf dem Prozessorchip sitzt. Dies alles kann in Form einer Princeton- oder einer Harvard-Architektur geschehen:

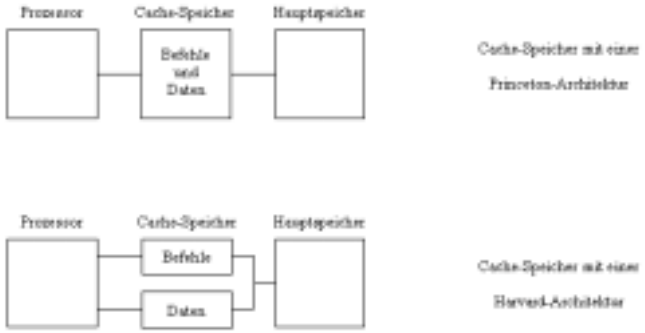



*Bild 7*



Der Cache-Speicher ist wesentlich kleiner als der Hauptspeicher – oft 256 KByte oder 512 KByte – und kann normalerweise nur Teile eines Programms aufnehmen. Für die Effizienz ist daher seine Organisation von großer Bedeutung: es muss möglich sein, schnell zu erkennen, ob der benötigte Programmblock im Cache-Speicher vorhanden ist, ihn gegebenenfalls schnell nachzuladen und dabei – falls der Speicher voll ist – möglichst einen nur selten benutzten Block zu überschreiben. Entscheidend ist auch das lokale Verhalten des Programms und der Daten, von dem es abhängt, ob das zeitaufwendige Nachladen nur selten vorkommt. Unter lokalem Verhalten versteht man die Tatsache, dass oft viele aufeinander folgende Befehle oder Daten eines Programms in einem engen Speicherbereich liegen. Eine diesbezügliche Optimierung, die eigentlich vom Compiler zu leisten wäre, ist für eine Effizienzsteigerung unerlässlich, bedarf aber noch großer Anstrengungen. Programme, die wenig Sprünge enthalten, sind oft lokal.

## 4.4 Pipeline-Prozessoren

Oft werden in einem Programm die gleichen Operationen mit einer Vielzahl von Operanden ausgeführt, z. B. bei Vektor- oder Matrizenrechnungen. Eine Möglichkeit, diese Rechnungen zu beschleunigen, besteht darin, die Rechenoperation in mehrere Stufen aufzuteilen, die durch Zwischenspeicher entkoppelt werden. So kann z. B. die zweite Stufe mit dem Zwischenergebnis weiterrechnen, während die erste Stufe schon den nächsten Operanden bearbeitet. Bei solchen „gefüllten" mehrstufigen Pipelines kann auf diese Weise viel Zeit gespart werden. Dabei hängt die Taktfrequenz der Pipeline von der langsamsten Stufe ab.

Längere arithmetische Operationen muss man zur Verarbeitung mit Pipelines in Teiloperationen zerlegen, die sich zeitlich verschränkt mit entsprechender Hardware ausführen lassen. Solche Pipelines können für eine bestimmte Aufgabe festgelegt sein oder aber durch Mikroprogrammierung für verschiedene Operationen konfiguriert werden. Das folgende Blockschaltbild zeigt eine solche Pipeline für arithmetische Operationen.

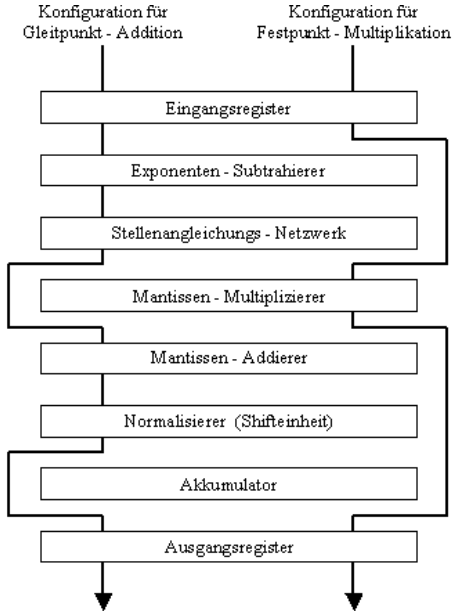



*Bild 8*

Neben arithmetischen Pipelines im Rechenwerk gibt es auch Befehls-Pipelines im Steuerwerk zur Verkürzung des Befehlszyklus. Z. B. kann die Befehlsabarbeitung in 2 Stufen aufgespalten werden: Fetch und Execute. Durch Anwendung des Pipeline-Verfahrens lässt sich der Durchsatz dann fast verdoppeln:



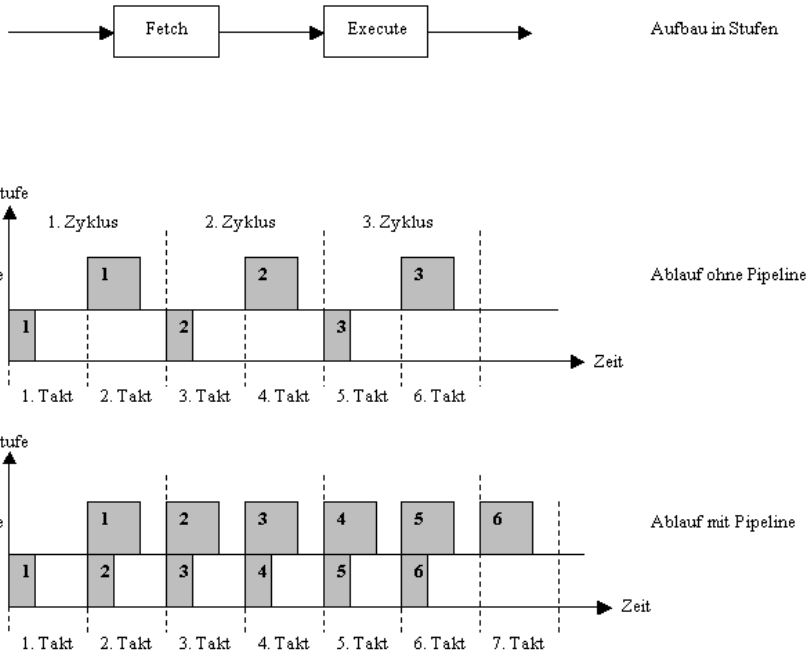



*Bild 9*

Solche Pipelines gibt es sowohl in kleineren als auch in Supercomputern, wobei zur Effizienzsteigerung Programme „vektorisiert" werden müssen.

Im Folgenden ist ein Beispiel für eine Befehlsausführungs-Pipeline gegeben. Sie setzt voraus, dass für jede der angegebenen 4 Phasen eine autonome Bearbeitungseinheit zur Verfügung steht und dass die Ausführphase nur ein Taktzyklus lang ist ( *Bild 10* ).


*Bild 10*

Außerdem ist ein Sprungbefehl vorhanden, der den Ablauf verzögert, da die Sprungbedingung während der Ausführungsphase berechnet wird ( *Bild 11* ). Auch hierfür gibt es Optimierungen.

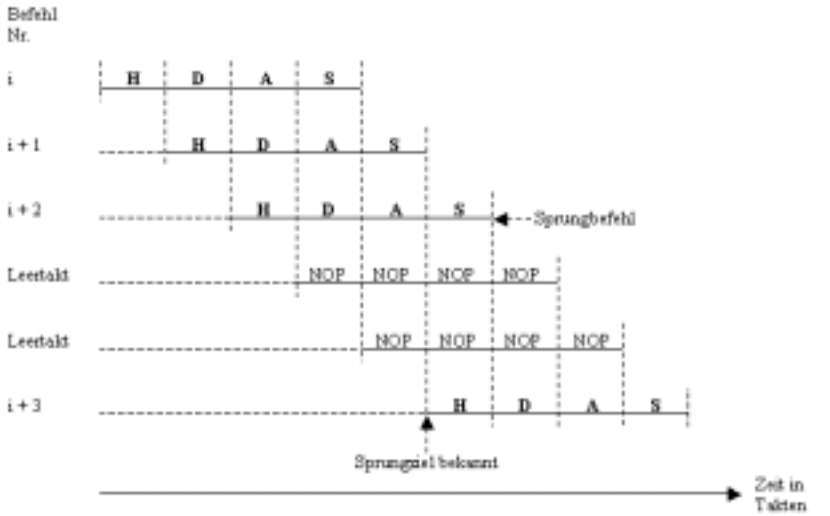



*Bild 11*

Eine Steigerung des Pipeline-Verfahrens benutzt man bei Superpipeline-Prozessoren: hier werden die Befehle in eine große Anzahl von zeitlich etwa gleich langen Teilschritten zerlegt, die so schnell zu bearbeiten sind, dass man die Taktfrequenz der Pipeline erhöhen kann. Nach der Füllphase ergibt sich damit ein um das Taktverhältnis vergrößerter Befehlsdurchsatz.



## 4.5 CISC-/RISC-Prozessoren

CISC-Prozessoren (**C**omplex **I**nstruction **S**et **C**omputer) sind dadurch gekennzeichnet, dass sie viele und mächtige Assemblerbefehle mit Hilfe einer aufwendigen Mikroprogrammierung bearbeiten können. Somit wird die Assemblerprogrammierung erleichtert und es entstehen kurze Maschinenprogramme. Auch kann ein einheitlicher Befehlssatz leicht auf verschiedene Hardware implementiert werden. Allerdings steigt die Laufzeit auf Grund der umfangreichen Mikroprogramme.

Da bei CISC-Prozessoren die überwiegende Anzahl von Befehlen kaum benutzt wird, geht man wieder verstärkt zu Prozessoren über, die nur eine sehr eingeschränkte Anzahl von Befehlen erlauben, deren Ausführung aber fest verdrahtet ist, so dass keine Mikroprogrammierung benötigt wird: die RISC-Prozessoren (**R**educed **I**nstruction **S**et **C**omputer). Bei ihnen werden dadurch die Assemblerprogramme zwar umfangreicher und maschinenabhängiger, doch deutlich schneller ausgeführt. Als wichtigstes Ziel wird eine Verarbeitungsrate von einem Befehl pro Taktzyklus angestrebt. Pipelines und optimierende Compiler bieten sich zur Unterstützung dieser Architektur an, was andererseits aber auch zu einer Verwischung der Grenzen zwischen den beiden Konzepten führt.

# 5 SIMD-Architektur

## 5.1 Parallelarbeit

Voraussetzung für die sinnvolle Nutzung paralleler Elemente in einer Rechnerarchitektur ist einerseits das Vorhandensein von Parallelität in einem Programm (z. B. Daten, Befehle) und andererseits die Aufbereitung von Daten und Programmelementen im Hinblick auf die Rechnerstruktur. Dabei ist die Abhängigkeit der Daten untereinander von großer Bedeutung. Eine solche Programmoptimierung sollte vom Compiler durchgeführt werden. Dafür gibt es u. a. neue Konzepte und Programmiersprachen. Trotzdem stellt dieser Bereich noch ein Haupthindernis für die effektive Nutzung der Parallelarbeit dar. Hier soll anhand von zwei Beispielen ein erster Eindruck der Prinzipien gegeben werden.

Parallelität auf der Ebene der Elementaroperationen und der arithmetischen Ausdrücke ist vom Compiler leicht zu erkennen. In der Baumdarstellung des Ausdrucks sieht man, dass im Prinzip alle Operationen einer Ebene gleichzeitig, diejenigen auf verschiedenen Ebenen nur sequentiell ausgeführt werden können. Die Optimierung besteht nun darin, durch Umorganisation einen äquivalenten Verarbeitungsbaum mit möglichst geringer Höhe zu erhalten. Im folgenden Bild ist ein Beispiel für einen Ausdruck gegeben:

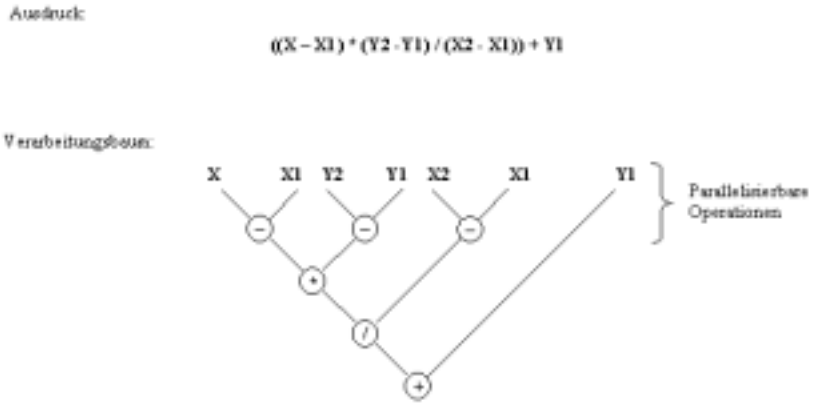



*Bild 12*

Im zweiten Beispiel soll die Parallelisierung einer Schleife dargestellt werden:

```
for i := 1 to 8 do
  begin
    C[i] := A[i] * B[i];
    E[i] := C[i] + D[i];
    F[i] := C[i] - D[i]
  end;
```



Der Ablauf lässt sich in dem folgenden Datenflussgraphen darstellen:

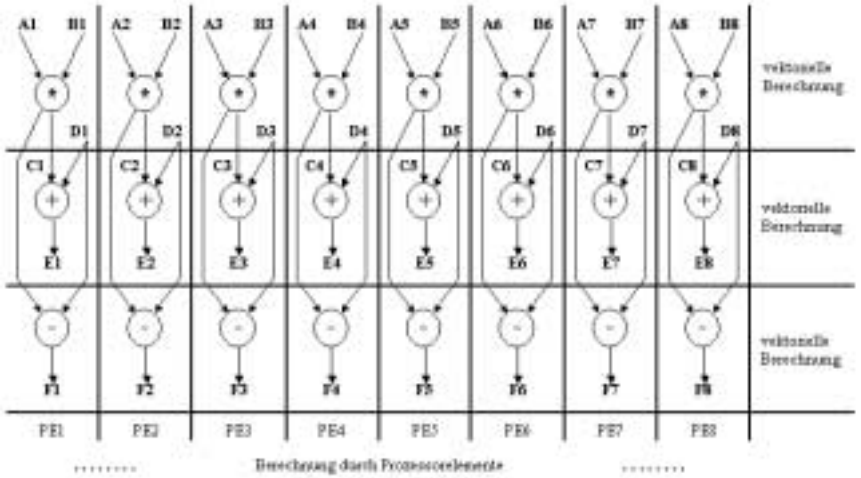



*Bild 13*

Die Ausführung entlang der horizontalen Streifen entspricht dabei einer Vektorisierung, die entlang der vertikalen Streifen einer Iteration.

Die erste Version lässt sich mit einer Vektormaschine realisieren, bei der verschiedene spezialisierte Funktionselemente zu einer der Aufgabe entsprechenden Pipeline konfiguriert werden können (multifunktionaler Pipeline-Prozessor). Insbesondere eignen sich dafür die Gleitpunkt-Operationen.

Die zweite Version führt zu einer Anordnung von Rechenelementen, von denen jedes Einzelne alle Operationen ausführen kann und die über eine gemeinsame Steuereinheit verbunden sind.

Das nächste Bild zeigt einen Vergleich der beiden Prinzipien am Beispiel der Vektoroperation $D = A * B + C$, wobei $A*B$ eine komponentenweise Multiplikation mit vektoriellem Ergebnis darstellt.

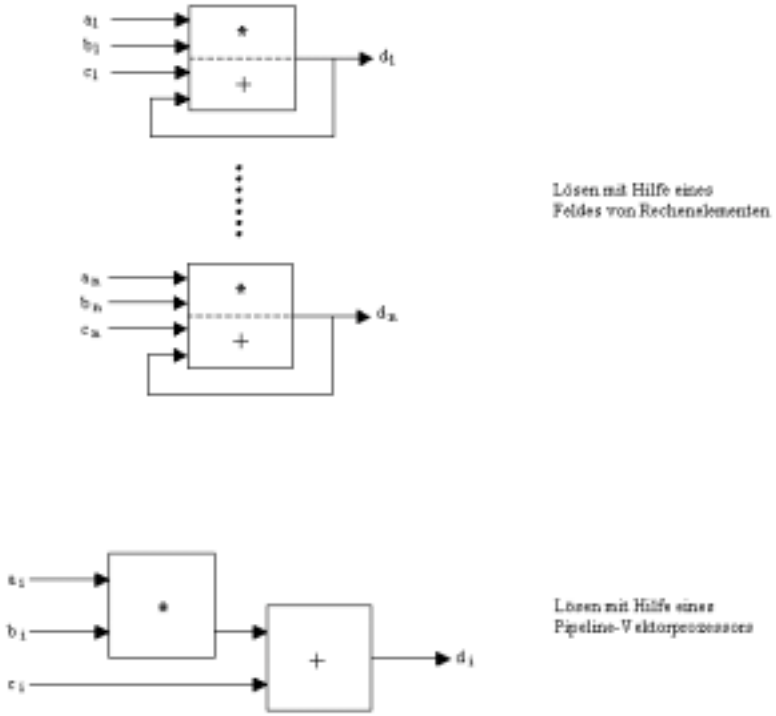



*Bild 14*



## 5.2 Vektorrechner

Die nächste Darstellung zeigt das Blockschaltbild einer Vektormaschine:

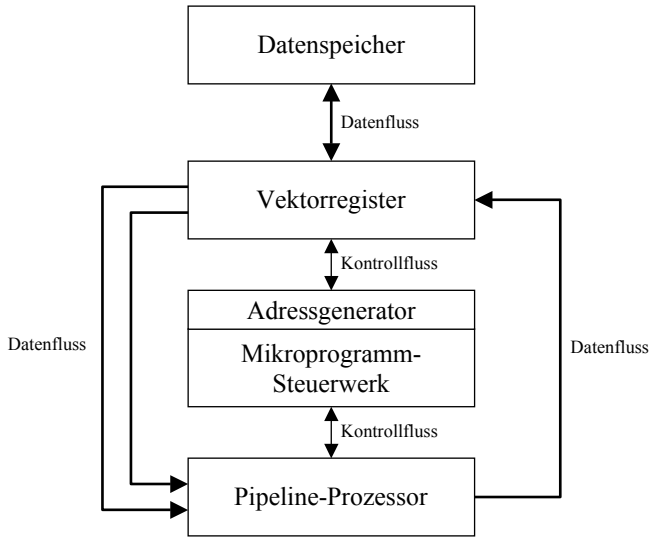



*Bild 15*

Zum Beispiel enthält die CRAY-1, ein Superrechner dieser Klasse, acht Vektorregister, die jeweils einen Vektor mit bis zu 64 Komponenten aufnehmen können.

Die in einem Programm zu bearbeitenden Vektoren sollen einerseits möglichst viele Komponenten haben, damit die Maschine ihre ganze Leistungsfähigkeit ausspielen kann. Andererseits müssen diese Vektoren aufgrund der beschränkten Länge der Vektorregister der Maschine intern unterteilt werden. An diesen Unterteilungsstellen gibt es durch das Nachladen der nächsten Komponenten Leistungseinbrüche. Bei der CRAY-1 muss dies also jeweils bei Vielfachen von 64 Komponenten geschehen, was im folgenden Bild auch zu erkennen ist:

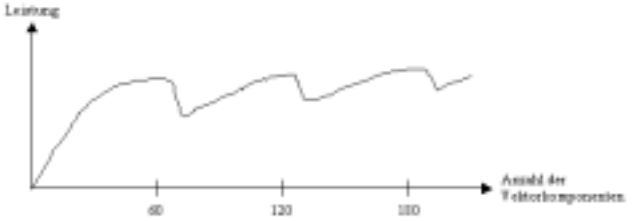



*Bild 16*

## 5.3 Array-Prozessoren (Feldrechner)

Eine andere Möglichkeit zur Beschleunigung paralleler Prozesse besteht in der Vervielfachung von Prozessorelementen, die eigene Speicher haben können. Diese parallelen Rechenwerke werden von einem gemeinsamen Steuerwerk geregelt. Ein Verbindungsnetzwerk dient zum Datenaustausch zwischen den Verarbeitungselementen.

Auch hierbei müssen die Programme entsprechend optimiert werden, um eine möglichst hohe Effektivität zu erhalten. Diese Optimierung sowie das Verbindungsnetzwerk schaffen große Probleme.

Das folgende Blockschaltbild soll eine solche Anordnung von Rechenelementen nochmals verdeutlichen:

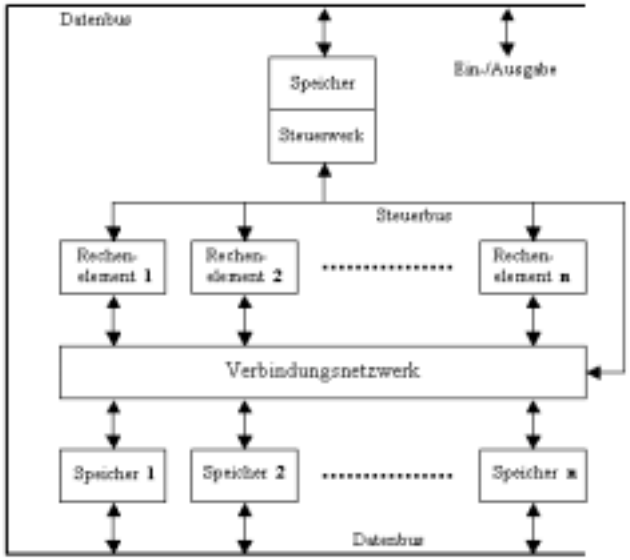



*Bild 17*



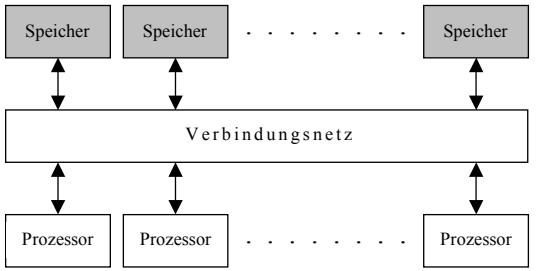
# 6 MIMD-Architektur

## 6.1 Multiprozessorsysteme

Die hier betrachteten Multiprozessorsysteme bestehen aus einer Anzahl von Universalprozessoren, die unabhängig voneinander parallel arbeiten können und die auch als zentrale Recheneinheit in einem Einprozessorsystem einsetzbar wären. Dadurch unterscheiden sich diese Parallelrechner sowohl von denjenigen Mehrprozessorsystemen, bei denen eine zentrale Recheneinheit durch Koprozessoren für spezielle Aufgaben entlastet wird, als auch von Anordnungen von Rechenelementen, da diese Rechenelemente keine autonomen Prozessoren sind, sondern von außen gesteuerte Funktionseinheiten. Die einzelnen Prozessoren mit ihrem lokalen Speicher nennt man Knoten.

Aufgrund ihres physikalischen Aufbaus kann man die Parallelrechner in zwei Hauptgruppen unterteilen:

*Systeme mit gemeinsamem Speicher für alle Knoten (speichergekoppelte Systeme)*

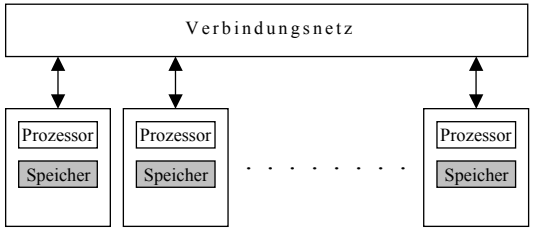



System mit gemeinsamem Speicher

*Bild 18*

Zu den Systemen mit gemeinsamem Speicher zählt man auch die Architekturen, bei denen die Prozessoren zusätzlich zu dem allen gemeinsamen Speicher noch jeweils einen lokalen Speicher besitzen. Die Programme mit den Daten befinden sich in dem zentralen Speicher, der zur Verbesserung des Zugriffs in autonome Speicherbänke aufgeteilt sein kann.

*Systeme mit verteiltem Speicher, bei denen es nur die lokalen Knotenspeicher gibt*




System mit verteiltem Speicher

*Bild 19*

Bei den Systemen mit verteiltem Speicher unterscheidet man, ob ein Prozessor nur Zugriff auf seinen eigenen Knotenspeicher hat (nachrichtenorientiertes System, da die Kommunikation zwischen den Knoten nur durch Austausch von Nachrichten geschehen kann) oder ob er auch auf die Speicher der anderen Prozessoren zugreifen kann (System mit verteiltem gemeinsamen Speicher, nämlich der Gesamtheit aller Knotenspeicher).

In beiden Fällen müssen die Programme mit den Daten so in Teile zerlegt und über die Speicher verteilt sein, dass für jeden Prozessor möglichst derjenige Teil, den er zu bearbeiten hat, in seinem lokalen Speicher steht.

Ein weiterer wesentlicher Unterschied der beiden Gruppen liegt in der Skalierbarkeit der Systeme, d. h. in der Möglichkeit, mit den gleichen Hard- und Softwarekomponenten Konfigurationen beliebiger Größe aufzubauen. Systeme mit zentralem gemeinsamen Speicher sind nicht skalierbar, da der Speicher bei zunehmender Prozessorzahl nicht mitwächst. Er wird so zum Flaschenhals, der die Anzahl der Prozessoren begrenzt (heute maximal etwa 30). Systeme mit verteiltem Speicher sind dagegen skalierbar und können fast beliebig groß werden, da der Speicher an die Prozessoren gekoppelt ist und mit ihrer Anzahl zunimmt. Es gibt Systeme mit mehreren Tausend Prozessoren.

Ein Vorteil des zentralen Speichers liegt darin, dass dieser Speicher einen globalen Adressraum darstellt und somit die Möglichkeit des Programmierens im herkömmlichen Stil und mit den konventionellen Sprachen zulässt. Bei den Systemen mit verteiltem Speicher muss der Benutzer das Programm in einzelne Prozesse aufteilen und auch die Kommunikation zwischen diesen Prozessen angeben. Dazu sind entsprechende Erweiterungen für die Programmiersprachen nötig. Compiler, die dem Benutzer diese Aufgabe wenigstens erleichtern, sind erst in der Entwicklung.

Bei der Programmierung eines solchen Rechners sind eine Reihe von Festlegungen zu treffen wie z. B.

- die parallel auszuführenden Programmeinheiten und die Art der Parallelisierung (Granularität, Programm- oder Datenparallelität),
- die Organisation der Kommunikation zwischen den parallelen Programmeinheiten (Kommunikationsprotokoll), wobei auch die Kommunikationszeit (Zeit für den Aufbau der Verbindung, die Datenübertragung, die Absicherung der Kommunikation usw.) eine wichtige Rolle spielt und
- die Gewährleistung der Korrektheit des parallelen Ablaufs (Koordination, Synchronisation).

Dabei versucht man möglichst unabhängig von den Realisierungsdetails einer realen Maschine Festlegungen für allgemeine Fälle anzugeben, z. B. für den Nachrichtenaustausch bzw. den globalen Adressraum bei verteiltem gemeinsamen Speicher.



## 6.2 Verbindungsnetze

Ein wesentlicher Teil des Parallelrechners ist das Verbindungsnetz zwischen den Prozessoren, Speichern und Knoten, über das alle Kommunikationen und Datenaustausche laufen. Dabei muss zwischen jedem Sender und jedem Empfänger eine Verbindung herstellbar sein. Es wurde eine Vielzahl von Netztopologien entwickelt, von denen im Folgenden einige kurz angesprochen werden.

Zuvor soll aber noch eine Aufgliederung der Verbindungsnetze, unabhängig von ihrer speziellen Topologie, nach einigen Hauptmerkmalen angegeben werden:

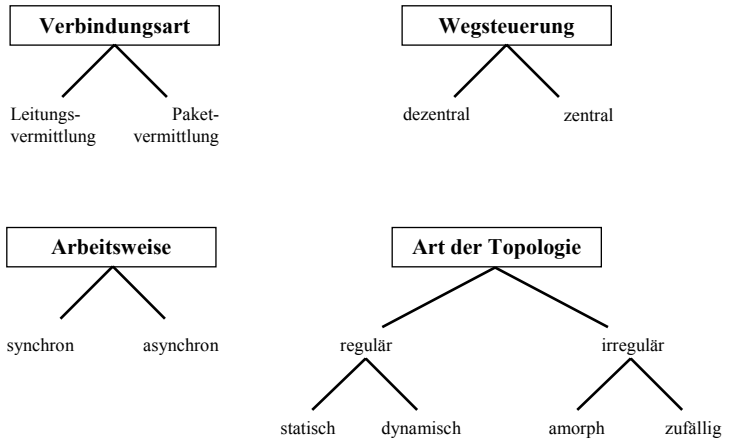



*Bild 20*

Die Datenübertragung in einem Verbindungsnetz kann mit Hilfe von Leitungs- oder Paketvermittlung stattfinden. Im ersten Fall wird für die Dauer der Übertragung eine feste Verbindung aufgebaut, ähnlich wie beim Telefonieren. Diese kann die gleichzeitige Herstellung weiterer Verbindungen stark einschränken und damit das Netz teilweise blockieren. Bei der Paketvermittlung, bei der kleine Nachrichtenpakete (auch Teile), die zu verschiedenen logischen Verbindungen gehören können, gleichzeitig durch das Netz gesandt werden, passiert dies nicht. Allerdings müssen die Vermittlungsknoten des Netzes in der Lage sein, die durchlaufenden Pakete zwischenzuspeichern.

Arbeitet ein Verbindungsnetzes synchron, so übertragen alle Sender und Empfänger zu festen, zentralgetakteten Zeitpunkten. Bei großen, räumlich ausgedehnten Netzen wird dies aufgrund der Laufzeitunterschiede und der damit verbundenen Taktverschiebungen immer schwieriger. Daher muss die synchrone Arbeitsweise, obwohl sie einfacher zu realisieren ist, dann durch die asynchrone ersetzt werden, bei der jeder Sender seine Übertragungsanforderungen zu jedem beliebigen Zeitpunkt an das Netz stellen kann.

Die asynchrone Arbeitsweise ebenso wie die Skalierbarkeit bei Rechnern mit verteiltem Speicher benötigen eine dezentrale Wegsteuerung. Unter Wegsteuerung versteht man den Aufbau eines

Verbindungsweges durch das Netz. Bei der dezentralen Wegsteuerung muss die Wegangabe im Kopf der zu übertragenden Nachricht stehen, im zentralen Fall wird sie unabhängig von der zu übertragenden Information zum Wegaufbau genutzt.

Während die irregulären Netztopologien in der Praxis unbedeutend sind, setzt man eine Vielfalt von regulären Topologien ein. Die statischen Verbindungsnetze bestehen aus festen Leitungsverbindungen, die in mehreren Dimensionen von jedem Schaltknoten des Netzes ausgehen können. Schalteinrichtungen gibt es nur am Anfang und Ende der Verbindung. Ein dynamisches Verbindungsnetz enthält eine Vielzahl von Schaltknoten, die durch die Wegsteuerung so gesetzt werden, dass ein bestimmter Verbindungsweg - eventuell über mehrere Stufen hinweg - entsteht. Die Schalter können auch innerhalb des Verbindungsweges liegen. Statische Verbindungsnetze enthalten also relativ viele Leitungen mit wenigen Schaltern, bei dynamischen Netzen ist es umgekehrt. Beim heutigen Stand der Technik sind Leitungen mit Steckern wesentlich teurer als Schalter, die sich auf hochintegrierten Chips herstellen lassen. Daher sind die dynamischen Netze in der Regel die heute angemessenere Lösung.



## 6.3 Verbindungstopologien

Das nächste Bild gibt einen zuordnenden Überblick über eine Reihe bekannter regulärer Verbindungstopologien.

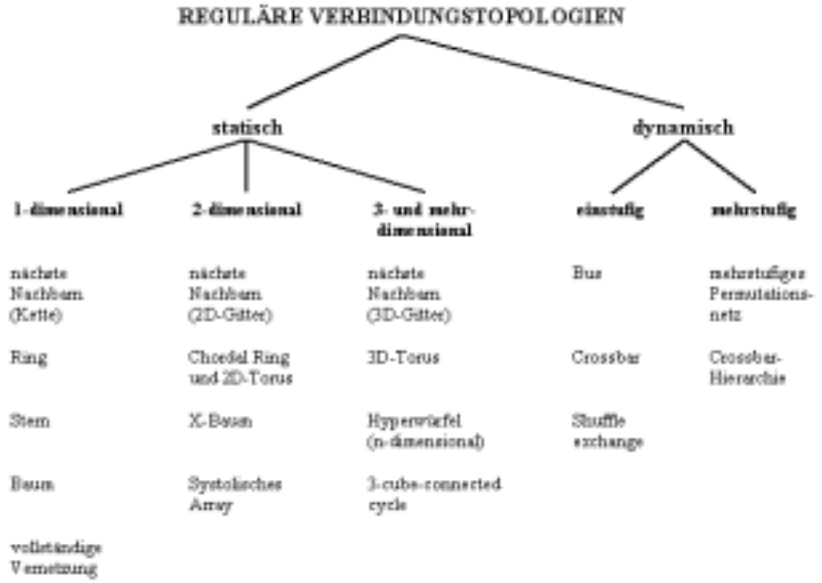



*Bild 21*

Einige der wichtigsten Topologien sind im Bild 22 auf der nachfolgenden Seite illustriert.

Wesentliche, praktisch bedeutsame Verbindungsnetze, auf die im Folgenden nochmals kurz eingegangen wird, sind:

- Busse,
- Ringe,
- Gitter,
- Hyperwürfel,
- Crossbar-Hierarchien.

Auch in Parallelrechnern, insbesondere bei kleinen Systemen, werden Busse oft weiterhin für die verschiedensten Aufgaben benutzt, wie z. B. den Datentransport, den Nachrichtenverkehr oder die Synchronisation. Sie sind oft für die spezielle Anwendung optimiert. Busparameter sind u. a. die Datenrate, die Wortbreite, das Übertragungsverfahren (synchron, asynchron) und die Hierarchiestufen (lokal, global).

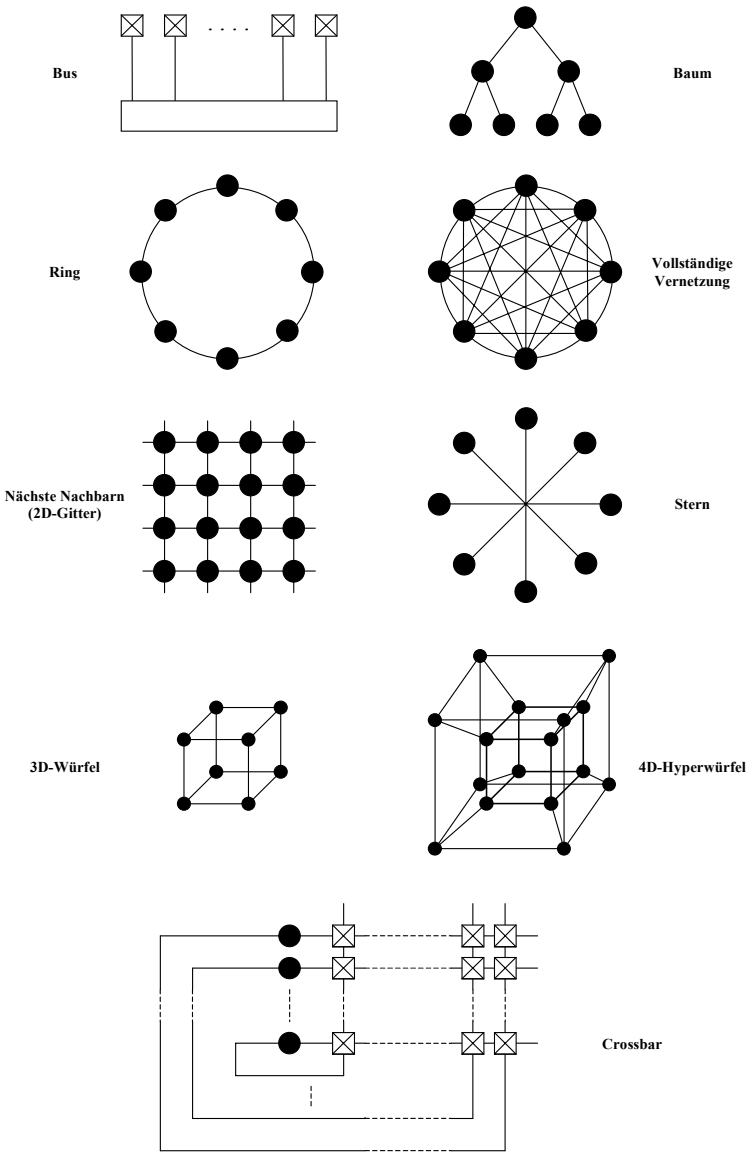



*Bild 22*



Die Geschwindigkeit eines Bussystems wird in erster Linie durch die Signallaufzeit in den Leitungen begrenzt, so dass ein Bus nur dann eine schnelle Verbindungsstruktur darstellt, wenn er kurz ist. In den Knoten eines Parallelrechners werden Busse viel benutzt, früher auch bei Systemen mit gemeinsamem Speicher zur Verbindung zwischen den Prozessoren und dem Speicher (Parallelbus). Hier setzt sich aber immer mehr die Crossbar-Verbindung durch, die eine höhere Übertragungsgeschwindigkeit erlaubt. Sie wird weiter unten beschrieben.

Ringe sind meist als Tokenringe realisiert. Mit dem Begriff Token bezeichnet man hier ein besonderes Bitmuster auf dem Ring, an dem ein Knoten z. B. erkennen kann, ob der Ring frei ist und er somit senden, d. h. Daten paketweise auf den Ring legen kann. Im Kopfpaket gibt er Ziel- und Senderadresse an sowie weitere Leitinformationen. Erkennt ein Knoten seine Adresse als Zielangabe, kopiert er die Daten in einen Puffer und versieht das Schwanzpaket mit einer entsprechenden Information. Kommen die Daten wieder an den Sender zurück, nimmt dieser sie aus dem Ring. Er kann nicht sofort wieder senden, so dass jeder Knoten eine faire Sendechance hat. Wenn die Kapazität des Rings es zulässt, können mehrere Informationen gleichzeitig auf dem Ring kreisen, die durch verschiedene Token unterschieden werden.

Eine beliebte Verbindungsform stellt die zweidimensionale Gitterverbindung dar. Sie ist beliebig skalierbar und kostengünstig, da die Zahl von 4 Verbindungskanälen je Knoten recht gering ist und die Kosten eines Netzes im Wesentlichen durch die Zahl der Verbindungen bestimmt werden. Weiterhin lässt sich dieses Gitter vollständig auf einer Platine realisieren und dadurch ohne Kabel herstellen. Allerdings überschneiden sich bei großen Knotenentfernungen oft Wege, so dass dann ein ungünstiges Blockierungsverhalten vorliegt. Eine sinnvoller Einsatz eines solchen Gitters ist daher nur dann gegeben, wenn der Rechner hauptsächlich für Anwendungen eingesetzt wird, bei denen die Kommunikation eine starke Lokalität aufweist.

Lange Zeit galt der Hyperwürfel als zukunftsweisende Netztopologie, da er bei einer $n\log(n)$-Komplexität ($\log(n)$Verbindungskanäle für jeden der $n$ Knoten; maximal $\log(n)$ Zwischenstationen auf dem Weg von einem Knoten zu einem beliebigen anderen) ein sehr gutes Verbindungs- und Blockierungsverhalten aufweist (mehrere mögliche Wege zwischen zwei Knoten). Allerdings sind die Verbindungskanäle sehr aufwendig und damit teuer, außerdem ist der Hyperwürfel nicht skalierbar. Daher verliert er in der Praxis wieder an Bedeutung gegenüber Gittern und Hierarchien von Crossbars.

Bei einer Crossbar-Verbindung kann jeder Sender jeden Empfänger durch Schalterverbindungen ohne Zwischenstufen erreichen, Blockierungen können somit nicht auftreten. Der Nachteil einer $n^2$-Komplexität in der Zahl der Schalter ist heute nicht mehr wesentlich, da die Ausführung auf einem Chip problemlos und billig ist, so dass die Begrenzung jetzt nicht durch die Zahl der Schalter sondern durch die Zahl der Anschlüsse gegeben ist (siehe auch Kapitel 6.2). Größere Systeme baut man als Hierarchie von Crossbars auf, wobei man eine Gruppe von Knoten mit Hilfe eines Crossbar-Chips zu einem Cluster vereint und dann die Cluster wieder durch Crossbars miteinander verbindet. Dabei können allerdings Blockierungen auftreten.

## 6.4 Probleme bei der Parallelisierung

Zum Abschluss soll noch an einem Beispiel gezeigt werden, welche Probleme bei der Parallelisierung von Anweisungen aufgrund der Datenabhängigkeit auftreten können. Gegeben sei der folgende Programmausschnitt:

```
for i := 0 to n-1 do
  begin
    initialize_processor;
    R0 := M[4*i];
    R1 := M[4*i+1];
    R2 := M[4*i+2];
    R3 := M[4*i+3];
    R0 := f(R0,R1,R2,R3);
    M[4*i+7] := R0
  end;
```

Die Elemente M7, M11, M15 usw. werden innerhalb der Schleife berechnet und stehen erst danach für eine weitere Verarbeitung zur Verfügung. Dadurch ergeben sich Abhängigkeiten der Schleifendurchläufe und damit möglicherweise Verzögerungen bei einer parallelen Ausführung.

Im Fall einer direkt umgesetzten Parallelisierung der auftretenden Schleife für 4 bzw. für 2 Prozessoren erhält man:

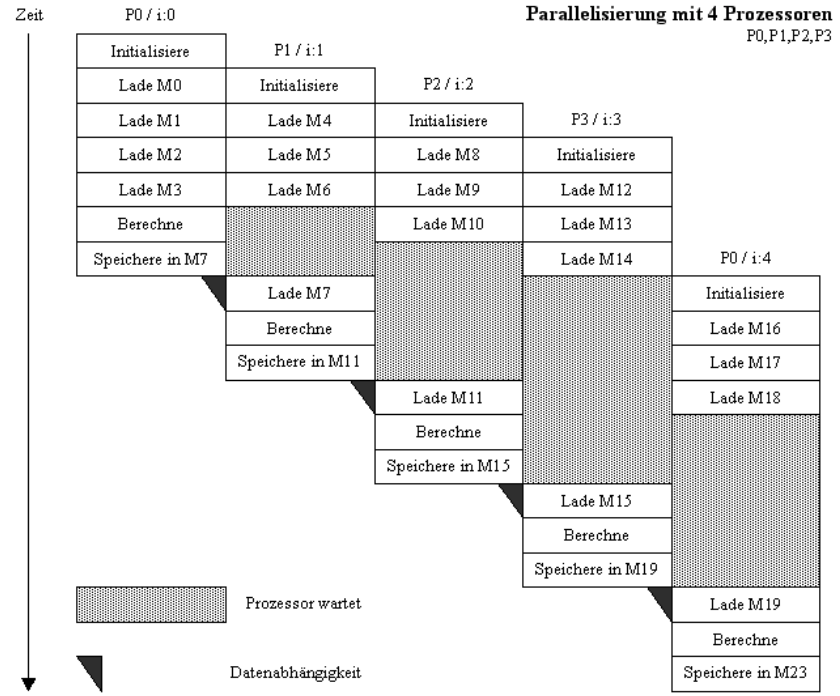



*Bild 23a*



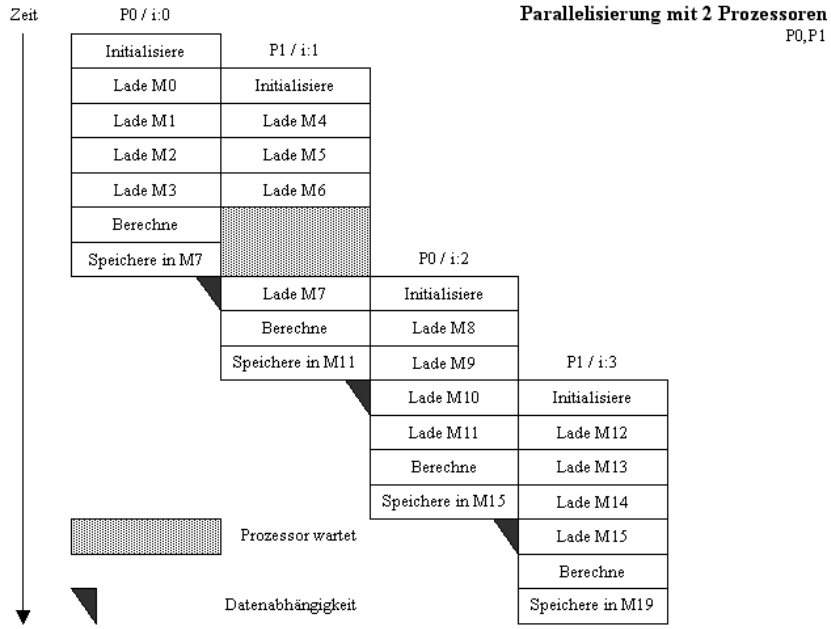



*Bild 23b*

Die Auslastung der gesamten Maschine und damit die Optimierung hängt also in diesem Fall von der Anzahl der eingesetzten Prozessoren ab und ist bei vier kleiner als bei zwei! Daher ist auch der Zeitbedarf bei zwei Prozessoren nur geringfügig größer.

# 7 Literatur

**Eberle**, H.: Architektur moderner RISC-Mikroprozessoren, Informatik Spektrum, Band 20, Heft 5, Oktober 1997, Seiten 259 - 267, Springer-Verlag, 1997

**Giloi,** W. K.: Rechnerarchitektur, 2. Auflage, Springer-Verlag, 1993

**Meyers Lexikonredaktion**; **Puttkamer**, E. von: Wie funktioniert das? Der Computer: Basiswissen über Hardware und Software, 2. Auflage, Meyers Lexikonverlag, 1994

**Oberschelp,** W.; **Vossen**, G.: Rechneraufbau und Rechnerstrukturen, 7. Auflage, R. Oldenbourg Verlag, 1998

**Schiffmann**, W.; **Schmitz**, R.: Technische Informatik 2, Grundlagen der Computertechnik, Springer-Verlag, 1992



# 8 Weblinks

Hypertext-Präsentation „Rechnerarchitekturen" für den Unterrichtseinsatz
http://informatikag.bildung-rp.de/index.html

Rechnerarchitektur: Skript vom Sommersemester 2000 (1.5 MB PostScript), Vorlesungsfolien (7.2 MB PostScript)
http://ls12-www.cs.uni-dortmund.de/edu/scripts.html
*Größe 3 K - 20.12.2000*

Arbeitsblätter im PDF-Format für den Unterricht in Datenverarbeitung
http://www.eduvinet.de/schwinn/html/dav2.html
*Größe 13 K - 29.11.2000 - Autor: Günter Schwinn*

Einführung in die Rechnerarchitektur
http://www.stud.fernuni-hagen.de/q1471341/docs/01704.html
*Größe 10 K - 25.11.2000*

Veröffentlichungen aus dem Fachgebiet Rechnerarchitektur (Rechnerarchitekturen und Parallele Systeme) an der TU Ilmenau
http://www.theoinf.tu-ilmenau.de/ra1/ver/
*Größe 23 K - 22.11.2000*

Vorlesung Rechnerarchitektur, gehalten von: Univ.-Prof. Dr. Peter Marwedel
Erscheinungsjahr: 2000
http://eldorado.uni-dortmund.de:8080/FB4/ls12/lehre/1999/rechnerarchitektur
*Größe 12 K - 19.11.2000*

Vorlesungen: Rechnerarchitektur, 1. Semester
http://www.wi.fh-flensburg.de/wi/riggert/Net-Vorlesung.htm
*Größe 1 K - 4.8.2000*

Vorlesung Rechnerarchitektur I, SS00
http://mufasa.informatik.uni-mannheim.de/lsra/lectures/ss00/vl_ra/vl_ra.html
*Größe 4 K - 18.8.2000*

Von-Neumann-Rechner
http://inf2-www.informatik.unibw-muenchen.de/People/borghoff/slides/info1/tsld052.htm
*Größe 1 K - 16.12.1999*

(Stand: 1. Juni 2001)