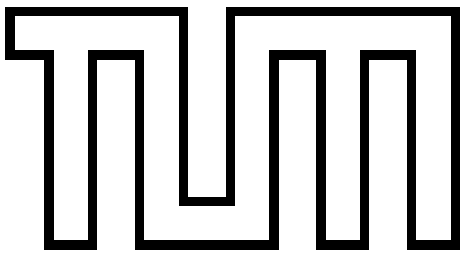


Technische Universität München

Fakultät für Informatik

Forschungs- und Lehreinheit Informatik VIII

Rechnerarchitektur

Wintersemester 2000 / 2001

Prof. E. Jessen

Schriftsatz und Layout: Gerrit Hillebrand

Inhaltsverzeichnis

1 Was ist Rechnerarchitektur ? 3
- 1.1 Begriff . . . 3
- 1.2 Unreife des Gebiets . . . 4
- 1.3 Ziele der Vorlesung . . . 5

2 Allgemeines 6
- 2.1 Funktionseinheiten und Aufträge . . . 6
  - 2.1.1 Bemerkungen . . . 7
- 2.2 von Neumann revisited . . . 8
- 2.3 Interpretation . . . 13
- 2.4 Rechenanlagen . . . 15
- 2.5 Kommunikationsstrukturen . . . 23
- 2.6 Interpretationshierarchie . . . 29
- 2.7 Ausschnittshierarchie . . . 30
- 2.8 Durchsatzmasse . . . 32
- 2.9 Geschichte . . . 36
- 2.10 Aussichten . . . 38
- 2.11 Freiheitsgrade von Rechnern, Klassifikationen . . . 38
- 2.12 Erscheinungsformen,Markt . . . 41

3 Zentralprozessoren 43
- 3.1 Beispiele . . . 43
  - 3.1.1 CISC uns RISC . . . 44
  - 3.1.2 MIPS R 10000/12000 . . . 46
  - 3.1.3 DEC / Compaq Alpha . . . 49
  - 3.1.4 Intel Pentium . . . 53
  - 3.1.5 Intel IA-64, Itanium . . . 56
- 3.2 Grundstrukturen . . . 57
  - 3.2.1 Gliederung des Prozessors auf verschiedenen Ebenen . . . 57
  - 3.2.2 Aktueller Ausschnitt des Prozessors . . . 57
  - 3.2.3 Privilegierung . . . 58
  - 3.2.4 Datentypen . . . 61
- 3.3 Befehle . . . 63
  - 3.3.1 Formate . . . 63
  - 3.3.2 Drei/zwei/eins/null-Adressprinzip . . . 64
  - 3.3.3 Befehlstypen . . . 71
  - 3.3.4 Leitwerksbefehle . . . 72
  - 3.3.5 Rechenwerksbefehle . . . 78
  - 3.3.6 Operationen an höheren Datenstrukturen . . . 80
  - 3.3.7 Ein/Ausgabe-Befehle . . . 80
- 3.4 Adressierung . . . 81
  - 3.4.1 Wert statt Adresse . . . 81
  - 3.4.2 Adressierbares Format . . . 82
  - 3.4.3 Programm-, Prozess-, Maschinenadressen . . . 82
  - 3.4.4 Transformation von Programmadressen in Prozessadressen . . . 82
  - 3.4.5 Tranformation von Prozessadressen in Maschinenadressen . . . 90
  - 3.4.6 Unabhängige Teiladressräume . . . 94

3.4.7 Zugriffsschutz im Hauptspeicher ("Speicherschutz") . . . 97
3.5 Unterbrechungen (*Interrupts*) . . . 97
3.6 Leitwerk . . . 101
3.7 Rechenwerk und arithmetische Operationen . . . 102
3.7.1 Festpunkt- und Gleitpunktzahlen . . . 103
3.7.2 Positive und negative Festpunktzahlen, verallgemeinerte Addition . . . 104
3.7.3 Multiplikation und Division von Festpunktzahlen . . . 104
3.7.4 Gleitpunktoperationen . . . 105
3.7.5 Aufbau von Rechenwerken . . . 106
3.8 Operationssteuerung . . . 107
3.8.1 Ad-hoc-Steuerungen (*Hardwired Control*) . . . 108
3.8.2 Mikroprogrammierte Operationensteuerung: Grundstruktur . . . 110
3.8.3 Erste Modifikation: Statusmultiplexer . . . 110
3.8.4 Zweite Modifikation: Mikro-Befehlszähler . . . 110
3.8.5 Mikroprogrammsteuerwerk . . . 112
3.8.6 Horizontale Mikroprogrammierung . . . 113
3.8.7 Vertikale Mikroprogrammierung . . . 114
3.8.8 Rückblick . . . 114
3.8.9 Ausführung fremder Befehlssätze . . . 115
3.9 Cache Memory und Pipelining . . . 116
3.9.1 Cache Memory . . . 117
3.10 Unkonventionelle Rechner . . . 128

4 Hauptspeicher und Verkehrswege 133
4.1 Speichertechnik . . . 133
4.2 Zugang der CPU zum Hauptspeicher . . . 133
4.3 Hauptspeicherzugang über Bus . . . 138

5 Periphere Speicher 141
5.1 Bauformen und Technologie der peripheren Speicher mit bewegtem Medium . . . 141
5.2 Magnetischer Plattenspeicher . . . 142
5.3 Optische periphere Speicher . . . 143

6 Ein/Ausgabe-Werk 144
6.1 Ein/Ausgabe über Zentralprozessor . . . 144
6.2 Ein/Ausgabe über System-/Peripherie-Bus . . . 145
6.3 Ein/Ausgänge über eigene Speicherzugänge . . . 146

7 Mehrprozessoranlagen 148

Kapitel 1

Was ist Rechnerarchitektur ?

1.1 Begriff

vergl. [BODE HÄNDLER 80]

Verdächtig, daß Rechner-Leute ein Wort mit ebenso vielen ästhetischen wie technisch-funktionalen Bezügen schätzen, und dann auch noch in zwei Bedeutungen:

> architecture (Blaauw, 1964): *"...attributes of a system as seen by the programmer, i.e. the conceptional structure and functional behavior as distinct from the organisation and data flow and control, the logical design and the physical implemetation"*

Blaauw war einer der Konzepterfinder der IBM 360-Familie.

Gemeint ist also die Gesamtheit der dem Nutzer zugänglichen Funktionen, unabhängig von interner Struktur, Organisation, physischer Verwirklichung. Dies ist eine wesentliche Denkform der Informatik und Voraussetzung für die Konstruktion von Systemen !
Der sehr wichtige Blaauwsche "Architektur"begriff ist häufiger im amerikanischen (*architecture*) als im deutschen anzutreffen. Er wird gelegentlich auch als

> Funktionalität (*functionality*)

bezeichnet. Vergleiche sehr viel enger: [BAUER GOOS 82,I,S.63] Funktionalität: Charakter der Abbildung (Stelligkeit und Sorten der Argumente und des Funktionswertes)

Architektur (Funktionalität) ist ein auf allen Schichten anwendbarer Begriff: Sprache, CPU, Rechenwerk, ALU, Addierer, Schalter, ...

Meist aber wird mit "Architektur" auch das interne Prinzip der Verwirklichung der Funktionalität gemeint z.B. [GILOI W 91]:

> ***Rechnerarchitektur*** *ist bestimmt durch ein Operationsprinzip für die Hardware und die Struktur ihres Aufbaus aus den einzelnen Hardwarebetriebsmitteln*
>
> ***Operationsprinzip*** *definiert das funktionelle Verhalten der Architektur durch Festlegung einer Informationsstruktur und einer Kontrollstruktur.*
>
> ***Informationsstruktur*** *... eine Menge von abstrakten Datentypen (Objekte, aber ohne Vererbung, seit Anfang der 70er)*
>
> ***Kontrollstruktur*** *... Algorithmen für die Interpretation und Transformation der Informationskomponenten der Maschine...*

Struktur des Aufbaus [STONE H 75]:

> *The study of computer architecture is the study of the organisation and interconnections of components of computer systems. Computer architects construct computers from basic building*

> *blocks such as memories, arithmetic units and buses. From these building blocks the computer architect can construct anyone of a number of different types of computers, ranging from the smallest hand-held pocket-calculator to the largest ultrafast supercomputer...*

Diese Sichtweise der Architektur ist berechtigt, da Taschenrechner und Großrechner prinzipiell aus den gleichen Komponenten bestehen. Man kann dies mit der Konstruktion eines Gebäudes vergleichen, das aus Geschossen, Dach, Wänden, Fenstern, Türen etc. besteht.

Auffassung von Bode (1990)

Rechnerarchitektur

- allgemeine Strukturlehre mit ihren Hilfsmitteln
- ingenieurwissenschaftliche Disziplin, die bestehende und zukünftige Rechenanlagen beschreibt, vergleicht, beurteilt, verbessert und entwirft
- betrachtet dabei Aufbau und Eigenschaften des Ganzen (Rechenanlage), seiner Teile (Komponenten) und seiner Verbindungen (Globalstruktur, Infrastruktur)

Der Architekturbegriff in Systemen meint oft Mengen (Klassen) von Funktionseinheiten, jeweils nach Blaauws Konzept beschrieben:

> Funktionalität
> Schnittstellen (z.B. "Dienste", d.h. zulässige Aufträge)

z.B. Kommunikationsarchitektur, Datenbankarchitektur

1.2 Unreife des Gebiets

Die Rechnerentwicklung war bisher weit überwiegend technologiegetrieben.
Es gibt verhältnismäßig wenige grundlegende Strukturprinzipien, die immer wieder variiert und neu kombiniert werden ($\rightarrow$ 2.9)
Innovationssucht: Firmen und Akademiker versuchen oftmals, Strukturen als neu zu verkaufen, die (oft unter anderer Bezeichnung) bekannt sind:

- z.B. virtueller Speicher: Güntsch 1957, Ferranti 1963, IBM 1967 und IBM 1972 "NEU"
- z.B. RISC Kombination der vertikalen Mikromaschine (1965), Pipelining (1962) und compilerbestimmter Ablaufsteuerung (in Mikroassemblern, etwa 1974)

Eine Rolle spielen hierbei Buzzwords (Reizwörter, schräg zur bestehenden Nomenklatur), die aus Ängstlichkeit überall nachgesprochen werden.
Es bestehen abgeschottete Kulturkreise (z.B. Micro/ Mainframe/ Supercomputer) mit Mißtrauen und der Weigerung zu lernen. In diesen Kreisen werden oft groteske Defizite freudig ertragen, z.B. :

- Benutzeroberflächen bei Superrechnern
- Struktur von PC-Prozessoren
- Preise von Mainframes, zu teuer
- PC-Betriebssysteme

Akademische Rechnerarchitekturen verfolgen Jahrzehnte Konzepte, die sich (zu unrecht?) nicht auf dem Markt durchsetzen (non-von Kampagne):

- Datenflussrechner
- Reduktionsrechner
- Datentyp / objektorientierte Rechner

- sprachorientierte Rechner

Diese Konzepte sind inzwischen bis auf Reste größtenteils wieder verschwunden. Man vergleiche dieses Gebiet einmal mit der Compilertechnik !

1.3 Ziele der Vorlesung

- Grundausstattung eines jeden Informatikers in Rechnerarchitektur
- Verständnis der Funktion konventioneller Rechner (99,99%)
- Einfluß von Betrieb und Anwendung auf Rechner, Bewertungskriterien
- Verständnis für technologisch-strukturelle Wechselwirkungen
- Grundlagen für andere Gebiete:
  - Betriebssysteme
  - verteilte Systeme
  - Systemprogrammierung
  - Mikroprogrammierung
  - Mikroprozessoren
  - Übersetzer
  - quantitative Modelle von Rechensystemen
  - Prozessrechner
  - Entwurf von Rechenanlagen
  - und andere
- insbesondere natürlich für die weiterführenden Vorlesungen der Informatik II (technische Informatik) und die Praktika !

Kapitel 2

Allgemeines

2.1 Funktionseinheiten und Aufträge

Ein Vorab-Abschnitt, Grundbegriffe der Analyse von Systemen, von uns für Rechner gebraucht: Funktionseinheiten und Aufträge

Liebhaber lesen : [JESSEN VALK 86] Jessen,E. Valk,R : Rechensysteme - Grundlagen der Modellbildung, Springer, Heidelberg 1986 ca.70,-

Definition 2.1 (Funktionseinheit) *(FE, functional unit): Durch Aufgabe oder Wirkung abgegrenztes Gebilde (DIN 44300).*

Bei uns meist durch die Funktion (→ Blaauws Architekturbegriff)definiert: blackbox, exakte Spezifikation ohne Einsicht in die Realisierung ist anzustreben ! *Rechner, Speicher, Leitung, Transistor,...*

Definition 2.2 (System) *(system): Funktionseinheit, die als aus Funktionseinheiten zusammengesetzt aufgefaßt wird.*

Also keine Blackbox-Perspektive *(Rechnersystem, Betriebssystem, Energieversorgungssystem...)*

Definition 2.3 (Instanz) *Funktionseinheit, deren Aufgabe in der Ausführung bestimmter Handlungen besteht, nicht der Übergabe von Daten*

Definition 2.4 (Kanal) *Funktionseinheit, die der Übergabe von Daten zwischen Instanzen dient*

Speicher, Projektionsfolie, Leitung,... (Ein Ein/Ausgabe Kanal (*input/output-channel*) ist eine Funktionseinheit, die unabhängig vom Zentralprozessor Daten zwischen Hauptspeicher und Peripherie transportiert.)

Definition 2.5 (Auftrag) *(task) Verpflichtung einer Funktionseinheit zu einer Handlung*

Rechenauftrag (job), Speicherauftrag, Übermittlungsauftrag...

Definition 2.6 (Verweilzeit Y) *(sojourn time, delay, latency) eines Auftrages in einer Funktionseinheit: Zeitdauer von Auftragsübergabe/übernahme bis Ende der durch den Auftrag bestimmten Handlung in der Funktionseinheit*

Zugriffszeit (Leseauftrag, Speicher), Programmlaufzeit (Rechenauftrag, Rechensystem) (Rechensystem: Rechner mit betriebsnotwendiger Software)
Die Verweilzeit hängt offenbar ab von: Auftrag, Funktionseinheit (Art, Zustand) aber auch z.B. davon, ob die Funktionseinheit den Auftrag allein oder mit anderen Aufträgen bearbeitet.

Definition 2.7 (Füllung f) *, Auftragsbestand (population; es gibt keine anerkannte Bezeichnung dieses fundamentalen Begriffs): Zahl der Aufträge, die aktuell ihre Verweilzeit in der betrachteten Funktionseinheit verbringen*

Diese ist keineswegs nur 0 oder 1 ! Rechensysteme können viele Rechenaufträge aktuell übernommen, aber nicht erledigt haben. Rechnernetze transportieren viele Nachrichten gleichzeitig, "Pipelining"-Prozessoren führen viele Befehle gleichzeitig aus.
Es ist gleichgültig für unsere Definition, ob sich die Funktionseinheit im zeitlichen Wechsel oder gleichzeitig den bereits übernommenen Aufträgen widmet!

Eine Funktionseinheit heißt frei (*idle*), falls f=0, beschäftigt (*busy*), wenn f>0 gilt. Ist f>1, dann arbeitet sie im Simultanbetrieb; ist stets f≤1, arbeitet sie im seriellen Betrieb.

Wir nehmen nun den Verweilzeitbegriff nochmals auf. Aus guten Gründen ist er brutto, in blackbox-Perspektive gebildet und sagt nichts über eine "Netto"-Bearbeitungszeit aus. Um trotzdem einen Begriff benutzen zu können, der mindestens die Konkurenz anderer Aufträge aus der Bestimmung des Zeitbedarfs für die Auftragserfüllung heraus hält, sagen wir:

Definition 2.8 (Bedienzeit b) *(service time): Verweilzeit eines Auftrages bei f=1*

In Rechnern ist es typisch, daß ein einzelner Auftrag ohne Verzug erledigt wird (vergleiche etwa: Auto an einer Verkehrsampel). Also: *Befehlsausfürungszeit, Additionszeit, Zugriffszeit (zu einem Speicher, der exklusiv höchstens einen bedient), Transportzeit über eine exklusiv benutze Leitung...*

Definition 2.9 (Kapazität) *(capacity) einer Funktionseinheit: (durch Art der Aufträge bestimmte) maximale Füllung der Funktionseinheit*

Ein Zentralprossor hat eine Kapazität von einem Rechenauftrag, möglicherweise aber von vielen Operationen. Ein Personal Computer unter MS-DOS hat eine Kapazität von einem Rechenauftrag, ein Speicher von 1 MByte kann höchstens eine Füllung von 2^{20} Speicheraufträgen zu je ein Byte haben, also ist seine Kapazität 1 MByte
Eine "Kanalkapazität" (Informationstheroie) ist keine Kapazität in unserem Sinne, sondern ein Grenzdurchsatz.
Belegt ist eine Funktionseinheit deren Füllung gleich ihrer Kapazität ist. Einfach (*exclusive*) heißt eine Funktionseinheit, deren Kapazität 1 ist.

> (Meist zergliedern wir Rechner bis auf eine Ebene hinunter, die bezüglich der betrachteten Aufträge aus "einfachen" Funktionseinheiten besteht.)

Definition 2.10 (Durchsatz d) *(throughput(rate)) einer Funktionseinheit: Zahl der von der Funktionseinheit in einem Zeitintervall beendeten Aufträge, geteilt durch die Dauer des Zeitintervalls.*

Der Durchsatz hängt von der Art der Aufträge, der Funktionseinheit und dem Auftragszugang ab!
Durchsatz des Rechensystems: Rechenaufträge/h; Durchsatz der Platte: Blocktransporte/s; Durchsatz des Rechnernetzes: Nachrichten/s ;...

Definition 2.11 (Grenzdurchsatz c) *(maximum throughput, bandwidth) Durch Art der Aufträge und der Funktionseinheit bestimmter maximaler Durchsatz.*

Es wird dabei vorausgesetzt, daß immer genügend viele Aufträge angeboten werden. Der Grenzdurchsatz ist ein wichtiges Leistungmaß für eine Funktionseinheit.
Grenzdurchsatz der Zentraleinheit: Millionen Operationen/s (Mips: million instructions per second), Grenzdurchsatz einer Leitung: Bits/s,...

Definition 2.12 (Auslastung g) *(utilisation): Verhältnis von Durchsatz zu Grenzdurchsatz. Bei einfachen (exklusiven) Funktionseinheiten: der relative Zeitanteil, in dem die Funktionseinheit belegt ist.*

2.1.1 Bemerkungen

Der Durchsatz ist also eine "Stromgröße", die Füllung (analog) eine "Ladungsgröße". Anders aber als bei der Elektrizität ist das Verhalten der wichtigen Systemkomponenten "Strom"- und "Ladungs"begrenzend (Grenzdurchsatz, Kapazität). Der Zusammenhang zwischen Auftragszufluß und Durchsatz ist nur bis zum Erreichen des Grenzdurchsatzes linear.

Zwischen dem Durchsatz d und den Mittelwerten von Verweilzeit $\overline{v}$ und der Füllung $\overline{f}$ gilt, wenn gleichviele Aufträge zugehen wie abgehen ("Flußgleichgewicht"):

$$\overline{f} = d * \overline{v} \quad (Littel'sche\ Formel)$$

Brauchen die Rechenaufträge im Mittel 80s und werden 100/h fertig, dann ist das Sytem im Mittel mit $\overline{f} = 100/h * 80s = 2,22$ *Aufträgen gefüllt.*

Aufträge bedingen sich häufig derart, daß erst die Fertigstellung von Auftrag A die Übergabe/Übernahme von Auftrag B erlaubt. Man sagt dann:

A ist präzedent (*precedent*) zu B.

Es darf nicht zugleich A präzedent zu B und B präzedent zu A sein; es liegt sonst eine Verklemmung vor.

Ist A präzedent zu B oder B präzedent zu A, dann heißen A und B sequentiell zueinander (*sequential*), sonst nebenläufig (*concurrent*). Achtung: andere Bezeichner bei Bode!

Gibt es für zwei Aufträge keinen Zeitpunkt, zu dem beide übergeben/übernommen aber nicht beendet sind (d.h. ihrer Verweilzeiten überlappen sich), so heißen die Aufträge seriell, bzw. die Bearbeitung der Aufträge heißt seriell (*serial*), sonst heißen sie kollateral (*parallel*).

mögliche Ausführungsfolge: sequentiell, nebenläufig
tatsächliche Ausführungsfolge : seriell, kollateral

- nur nebenläufige Aufträge können kollateral bearbeitet werden
- sequentielle und nebenläufige Aufträge können seriell bearbeitet werden.
  Maschinen für kollaterale Auftragsbearbeitung (Parallelrechner genannt) sind auf nebenläufige Systeme von Aufträgen begrenzt → Kapitel 7, vor allem aber: Vorlesung Parallelrechner im Sommersemester!

Schließlich noch ein paar Begriffe, um die Unvollkommenheiten technischer (realer) Funktionseinheiten zu beschreiben:

Einen Fehler hat eine Funktionseinheit, die nicht in der Lage ist, einen Auftrag nach ihrer bestimmungsgemäßen Funktion (ihrer Spezifikation) zu erledigen.

Eine unkorrekte Auftragsbearbeitung heißt Störung (*malfunction*). Fehler können vorübergehend (transient) oder beständig (permanent) sein.

- Ein Fehler kann ohne Auftreten einer Störung bestehen:
  - weil kein fehlerempfindlicher Auftrag bearbeitet wird
  - weil die fehlerhafte Funktionseinheit Teil eines fehlertolerierenden Systems ist, d.h. eines das störungsfreie Auftragsbearbeitung trotz Anwesenheit von Fehlern erlaubt. Fehlertoleranzmaßnahmen sind (in verschiedenem Ausmaß) in allen Rechnern notwendig!
- ein Fehler kann viele Störungen verursachen

Ausfall (*failure*) einer Funktionseinheit ist ein nicht tolerabler Fehler. Die mittleren zeitlichen Abstände von Fehlern und Ausfällen (*mean time between malfunction: MTBM, mean time between failure: MTBF*) sind wichtige Gütemaße einer Funktionseinheit. Vergleiche die folgende Grafik, wo auch andere kennzeichnende Zeiten für Funktionseinheiten angegeben sind.

2.2 von Neumann revisited

Vergleiche 1.Kap. TGI-Vorlesung, [GILOI W 91] S.33ff [JESSEN E 75] S.39ff [TANENBAUM A 90] S.31ff

Und allen Beteuerungen zum trotz sind fast alle unsere Computer weiterentwickelte von-Neumann Maschinen oder Kompositionen aus ihnen. Lesenswert: die Überlegungen der Erfinder

Zuse, K. : Der Computer - mein Lebenswerk, Verlag Moderne Industrie, 1970, Springer 1993

Burks, Goldstine, von Neumann : Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946 nachzulesen in

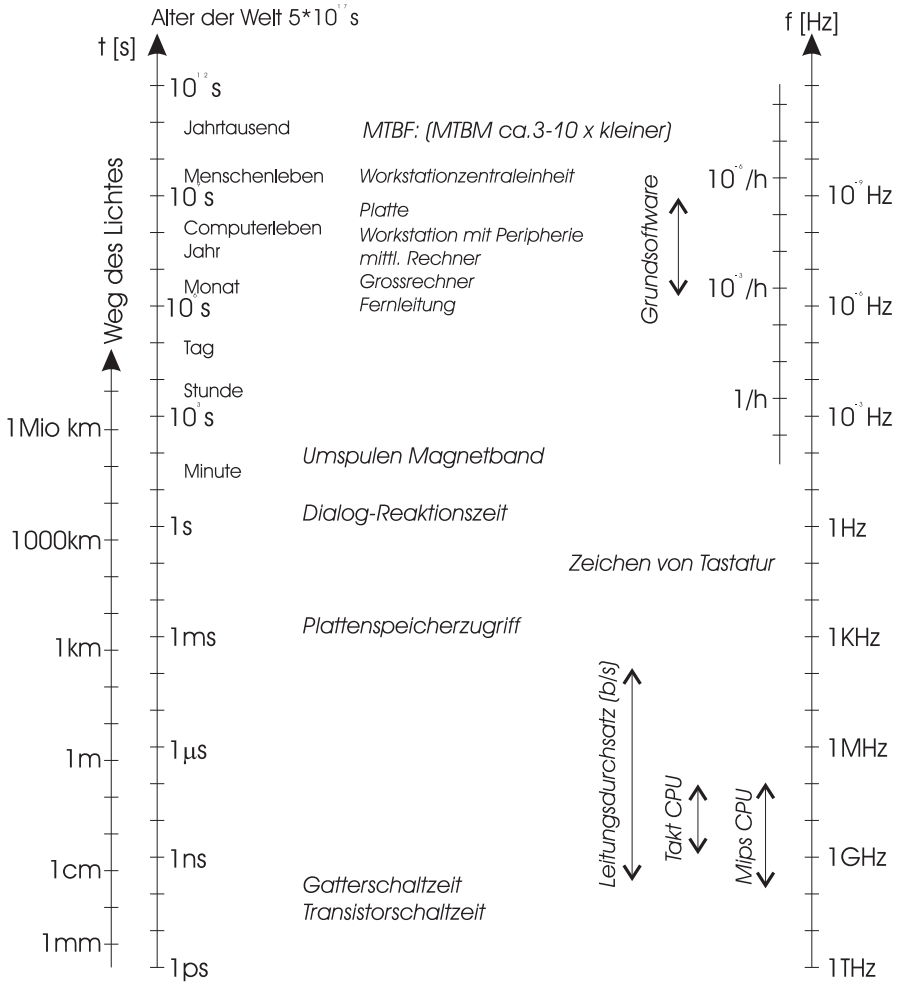


Abbildung 2.1: Überblick Zuverlässigkeit, Geschwindigkeit

- Taub : Collected Work of J.v.Neumann 1962
- Datamation Vol.8 No.10 pp 36-41, 1962
- Bell G. et.al.: Computer Stuctures - Readings and Examples, Mc Graw Hill, 1971

Von Neumann kannte Zuses Arbeiten 1946 nicht.

Das von-Neumann Konzept mit heutigen Augen:

1. Der Rechner besteht aus Speicher (*storage, memory*), Leitwerk (*control unit*), Rechenwerk (*atithmetic unit*), Ein/Ausgabegeräten (*I/O units*).
   Historie : Das war schon bei Zuse so.
   heutiger Standpunkt:
   - Aufspaltung des Speichers in Speicherhierarchie, oft vom Rechensystem verwaltet, aus technischen und wirtschaftlichen Gründen.
   - Leitwerk und Rechenwerk formen den Zentralprozessor (ZP, *central processing unit*, cpu), wichtigster Modul, erscheint in Mehrprozessoranlagen mehrfach.
   - Gelegentlich mehrere Rechenwerke im Zentralprozessor, zur Erhöhung des Grenzdurchsatzes (z.B. Feldrechner *array computer*)
   - Ein/Ausgabe Geräte über Gerätesteuerungen (*device controller*) indirekt angeschlossen
   - Verbindungsnetz (*interconnection net*) ist kritischer Teil

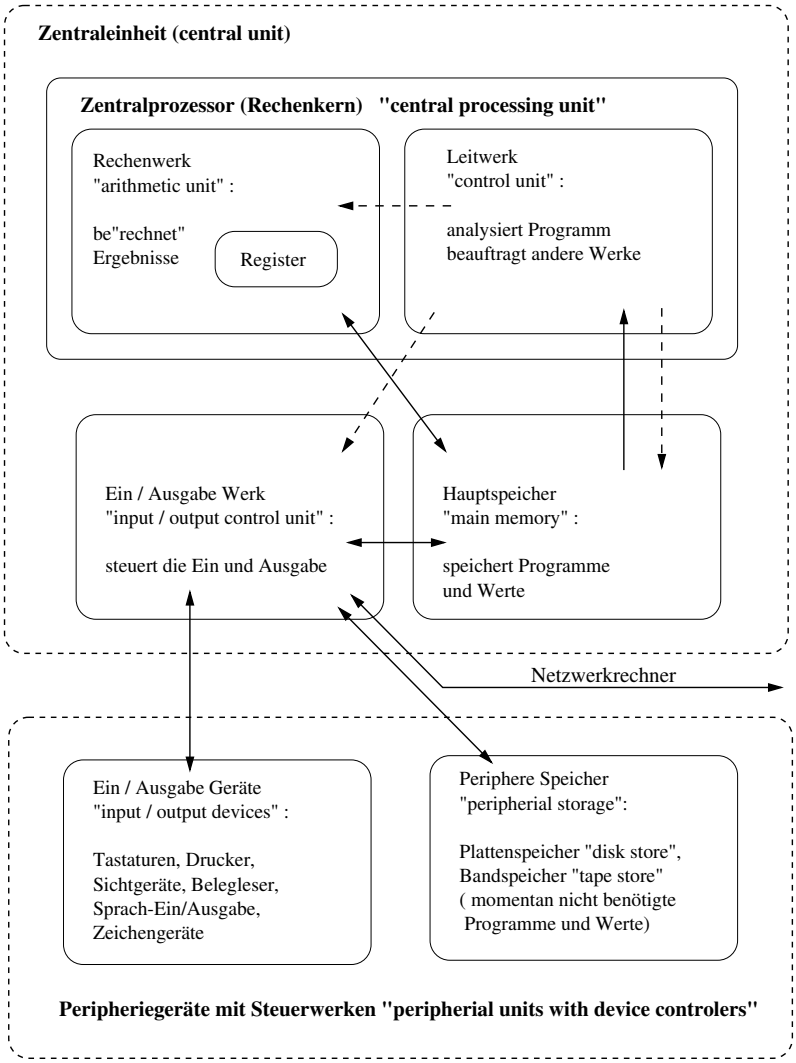


Abbildung 2.2: Übliche Gliederung eines Rechners

2. Die Struktur (Schaltbild) des Rechners ist unabhängig vom bearbeiteten Problem.
   Historie : Das war schon bei Zuse so.
   heutiger Standpunkt: praktisch unverändert!

3. Programm und Werte stehen in demselben Speicher und können beide durch die Maschine verändert werden.

   Historie: Neu, Zuse benutzte externe Programmsteuerung, das Programm war von der Maschine nicht änderbar.
   heutiger Standpunkt:
   Programm im internen Speicher: Ja ! kritische Voraussetzung für den Grenzdurchsatz der Zentraleinheit.
   In demselben Speicher: Manchmal bestehen getrennte Speicher für Werte und Programme (sog. Harvard-Architektur). Sinnvoll für Rechner, die unverändert für denselben Zweck eingesetzt werden (Steuerung, Taschenrechner), dann oft Programm in Festspeicher (ROM *read only memory*). Aber weit überwiegend werden Programme und Werte im selben Speicher untergebracht (Princeton Architektur) aber streng getrennt verwaltet als unveränderliche Teile (*pure segments*) und veränderliche Teile (*impure segments)*: wichtige Voraussetzung für effiziente Ausschnitte (Speicherhierarchie) und Rekursion, Mehrfachnutzung eines Programms.
   Veränderung des Programms: *shocking* ! Dieselbe Wirkung kann durch bedingte Sprünge und Adress-Modifikationen erreicht werden und ist sicherer und erlaubt Pogramme als *pure segments*

4. Der Speicher ist in Zellen (*cells*) gleicher Größe geteilt, die durch fortlaufende Nummern (Adressen, *addresses*) bezeichnet werden.

   Historie : Das war schon bei Zuse so.
   heutiger Standpunkt:
   Fast stets so gemacht; die Existenz von Zellen, die keine explizite Angabe des enthaltenen Typs besitzen, strukturierte Daten nicht unterstützen und erlauben, daß derselbe Bezeichener mehrfach einen neuen Wert zugeteilt erhält, ist trotzdem ein Hauptangriffspunkt der von Neumann Kritiker:

   - Datentypinformation soll zur Laufzeit des Programms von der Maschine mitgeführt werden, für
     - automatisch typangepasste Operationen (generische Operationen) oder
     - Überwachung der Konsistenz von Operand und Operation (Sicherheit)

     führt auf Typenkennungsmaschinen (*tagged architectures*) : Zellen enthalten zusätzlich Typenkennung
   - Datenstrukturen sollen nur indirekt über Leitdaten, nicht etwa durch direkten Zugriff zugänglich sein; zusätzlich soll evtl. die Berechtigung und korrekte Verwendung von der Maschine sichergestellt werden:
     Datenstruktur/Objektorientierte Rechner; das Prinzip kann auf Programmstrukturen ausgedehnt werden. ("Strukturierter Speicher")
   - Die Benutzung von Variablen im allgemeinen Sinn, die mehrfache Wertzuweisung erlauben, führt zu Programmen mit schwierig beweisbaren Eigenschaften; Kritiker verlangen *single assignment*-Variable. Damit wird die Variable auf die Rolle eines Zwischenergebnisses eingeengt (vgl. [BROY M 89] S.156]) So gebraucht in Datenflußmaschinen
   - noch weitergehend in Reduktionsmaschinen:
     Die Rechenvorschrift ist ein Ausdruck und identifiziert die Werte durch ihre Position im Ausdruck, nicht über Bezeichner/Adressen.

$Z := (A - B * C)/(D + B * C)$

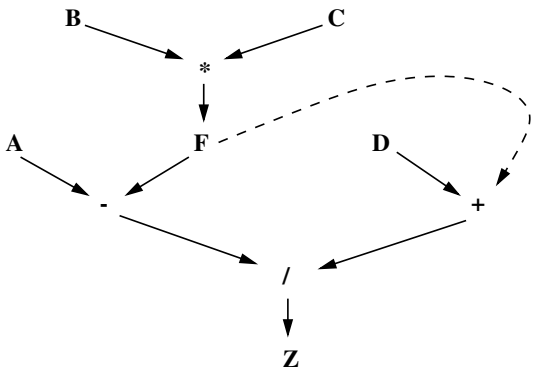


Abbildung 2.3: Datenflußmaschine

- A,B,C,D,F sind keine Variablen, sondern erhalten nur einmalige Wertzuweisungen
- Bereitstellung der Werte A,B,C,D,F steuert Ablauf der i.A. kollateralen Berechnung

Stringreduktion (grundsätzlich auch Möglichkeiten der kollateralen Berechnung) :

$(12 - 5 * 2)/(10 + 5 * 2)$
$(12 - 10)/(10 + 10)$
$(2)/(20)$
$(0,1)$

Datenflußmaschine benutzt bedingte Ausdrücke und Iteration, Reduktionsmaschine bedingte Ausdrücke und Rekursion: keine Einschränkung der Mächtigkeit gegenüber von Neumann !

5. Das Programm besteht aus einer Folge von Befehlen (*instructions*), die sequentielle Aufträge beschreiben und in der Aufzeichnungsreihenfolge auszuführen sind. Die Befehle enthalten i.A. nicht die Werte, mit denen zu rechnen ist, sondern die Adressen der sie beherbergenden Zellen.

   Historie : Schon bei Zuse so (streng sequentiell).
   heutiger Standpunkt:

   (a) Die Entwicklung schneller Rechner (d.h. hohen Grenzdurchsatzes, [Befehle/s] ist, soweit strukturelle Verbesserungen die Leistung erhöhen, ein einziger listenreicher Kampf gegen die Sequentialität. Aus Leistungsgründen sind u.U. viele Befehle desselben Programms in verschiedenen Stadien der Bearbeitung; trotzdem muß das Ergebnis dem bei serieller Verarbeitung erzielten entsprechen!

   Fließband (*pipelining*) Prozessoren; look-ahead-Mechanismen; mehrere Rechnenwerke, die gleichzeitig arbeiten

   (b) Man geht in Datenflußmaschinen von der sequentiellen Vorschrift ab und benutzt stattdessen einen maximal nebenläufigen Präzedenzgraphen als Programm; sie ähneln den Formularen und Kantorovic-Bäumen in [BROY M 92, I].

   (c) Der Umstand, daß die Operanden und Ergebnisse durch Adressen repräsentiert werden, ist direkte Folge der freien Parametrisierbarkeit, d.h. Flexibilität der Programme. Andererseits ergeben sich umständliche Abläufe: z.B. $A := B + C$

   Befehlsadresse an Hauptspeicher
   Befehl von Hauptspeicher
   Adresse von B an Hauptspeicher
   B von Hauptspeicher
   Adresse von C an Hauptspeicher
   C von Hauptspeicher
   Verknüpfung $B + C$
   Adresse von A an Hauptspeicher
   Ergebnis an Hauptspeicher

   Die Prozessor-Speicher-Schnittstelle wird von Kritikern oft als "von-Neumann Flaschenhals" (*von-Neuman bottleneck*) bezeichnet. In der Tat wird der Grenzdurchsatz der Zentraleinheit wesentlich durch die Zugriffe auf den Hauptspeicher begrenzt; dieser ist - bei akzeptabler Kapazität und Kosten - stets "zu langsam". Neben der Umgehung der Sequentialität ist daher die Verbesserung der Zugriffsverhältnisse zum Hauptspeicher das zweite entscheidende Mittel zur Erhöhung des Grenzdurchsatzes der Zentraleinheit!

   (d) (schon gehabt) Reduktionsmaschinen arbeiten ohne Befehle und ohne Adressen.

6. Von der Folge kann durch Sprungbefehle (*jump, branch instructions*) abgewichen werden, die die Programmfortsetzung an einer anderen Stelle bewirken. Die Ausführung eines Sprunges kann von gespeicherten, d.h. auch errechneten Werten abhängig gemacht werden.

   Historie : Neu ! nicht bei Zuse; entscheidende Bedeutung
   heutiger Standpunkt: unverändert; ähnlich wirken vom Rechner vorgenommene Modifikationen von Adressen; bei kollateraler Befehlsverarbeitung sind bedingte Operationen besser handhabbar als bedingte Sprünge.

7. Die Maschine benutzt Binärcodes, Zahlen werden dual dargestellt.

   Historie : schon bei Zuse
   heutiger Standpunkt: Neben Dualzahlen existieren auf vielen Maschinen auch binär codierte Dezimalzahlen.

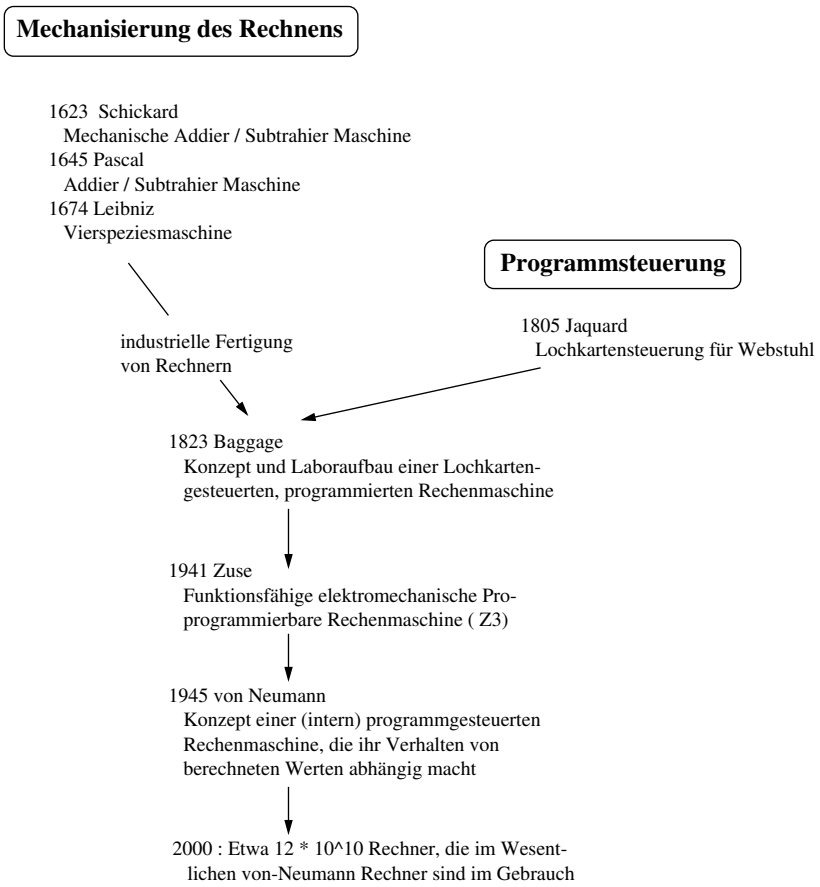


Abbildung 2.4: Ideengeschichte der programmgesteuerten Rechenmaschine

Erweiterungen des von-Neumann Konzeptes

- Adressmodifikationen: die Programmadresse im Befehlswort wird in zwei Stufen transformiert:
  - durch den Programmablauf in eine Prozessadresse (effektive Adresse)
  - durch das Betriebssystem und die Maschine in eine Maschinenadresse

  Dies erlaubt die Modifikation des Programmverhaltens ohne Modifikation des Programms !
- Gleichzeitige Bearbeitung in mehreren weitgehend unabhängigen Werken:
  Zentralprozessoren, Rechenwerken, Kanäle, Peripheriegeräte ...
- Hierarchie von Speichern, automatisch verwaltet; scheinbar der schnellste Speicher mit der Kapazität des größten vorhanden.
- Steuerung nicht durch ein Programm, sondern durch eine Hierachie von Steuervorgängen: (von Neumann) Programm, Mikroprogramm, weitere ("Nanoprogramm", "Picoprogramm")
- Unterbrechung der Programminterpretation im Prozessor durch interne und externe Signale.
- Parallelrechner und Rechnernetze und Erweiterung der dem Benutzer dargebotenen Maschine auf ein Hardware/Software-System Rechensystem mit höheren Sprachschnittstellen, langfristiger Datenerhaltung, komplizierter Betriebsführung

2.3 Interpretation

[JESSEN E 75] S.39ff, [TANAENBAUM A 90] S.2ff

Verallgemeinerung der von-Neumanschen Programmsteuerung:

Interpretation: Schrittweise Ableitung und Durchführung eines Auftragsprozesses nach komplexer Anweisung (Programm), bei der in jedem Ableitungsschritt der erreichte Zustand herangezogen werden kann.

Ein interpretierendes Werk heißt Prozessor, ein interpretierendes Programm heißt Interpreter (um dieses zu Interpretieren wird ein Prozessor gebraucht: Interpretationshierarchie !)

Übersetzung erzeugt ein semantisch äquivalentes Programm, kann Berechnungszustände nicht auswerten (da Übersetzung nicht ausführt). Übersetzung ist nicht durchweg möglich, wo Interpretation möglich ist (Änderung/Selbstreferenzierung des Programmtextes)

Gedankenmodell zur Interpretation:
Eine interpretierende Funktionseinheit F ist offenbar ein System aus zwei Kanälen und zwei Instanzen:

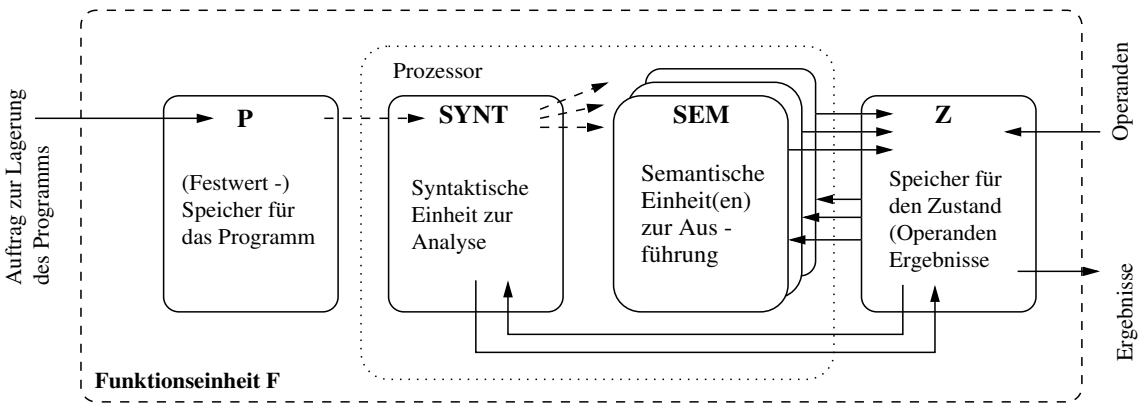


Abbildung 2.5: Interpretation

Kanäle:	P	(Festwert)Speicher für die komplexe Anweisung (Programm)
	Z	Speicher für "Zustand" der Berechnung (Operanden, Ergebnisse, aber auch Hilfsgrößen, z.B. Analysestatus von SYNT)
Instanzen:	SYNT	analysiert das Programm, benutzt dazu den Inhalt von Z, wenn erforderlich, beauftragt SEM "Semantische Einheit"
	SEM	setzt die Bedeutung des Programms in die Tat um, durch Transformation des Inhalts von Z " Semantische Einheit"

Typische Fragen:

- Besitzt F seine 4 Teile ausschließlich oder teilt es sie mit anderen?

  Teilung (*sharing*) z.B. P,Z in Mehrprozessoranlagen, P bei Mehrfachnutzung von Programmen, Z bei einfachen Formen der Interprozesskommunikation;

- Kann man die Arbeit von SYNT und SEM überlappen oder ist sie streng sequentiell?
- Soll man P und Z verschmelzen? (von Neumann)
- Welche Sprache in P?
  - Maschinensprache: dann ist SYNT sehr einfach
  - Höhere Sprache: dann wird SYNT so komplex, daß es seinerseits wieder nur durch Interpretation realisierbar wird. Interpretationshierarchie! *"High-Level-Language Machine"*
- Wie findet SYNT in P den (die) nächsten Schritt(e)?
  - explizit in P: schnell, aufwendig, lohnt, wenn viele Varianten (Mikroprogrammierung)
  - implizit: Befehlszählerprinzip; dann Sprünge für die seltenen Abweichungen von der Folge (von-Neumann Maschine, oft auch bei Mikroprogrammierung)

- Wie zieht SYNT den Zustand Z heran?
  - bei Ableitung des Folgeschritts
  - zusätzlich in SEM
- Wann lohnt es sich SYNT und SEM ihrerseits wieder durch Interpretation zu realisieren?

  Wenn diese sehr komplex sind. Wegen der Sequentialität (erst SYNT dann SEM) könnte diesselbe Funktionseinheit beide realisieren (Interpretationshierarchie!)

von Neumann Maschine:

P	Speicher mit Maschinenprogramm
SYNT	Leitwerk
SEM	Rechenwerk (E/A-Werk)
Z	(derselbe) Speicher mit Werten

Pascal-Interpreter:

P	Speicherbereich mit Pascalprogramm (Quelle)
SYNT	Programm, das ein Pascalprogramm syntaktisch analysiert
SEM	Prozeduren, von SYNT aufgerufen, die Pascalsemantik realisieren
Z	Strukturierter bezeichneter Speicher für Variable

Beide bilden eine Interpretationshierarchie !

2.4 Rechenanlagen

Definition 2.13 (Rechensystem) *(computing system) : Funktionseinheit zur Verarbeitung von Daten; besteht aus physischem Anteil: Rechenanlage (computer) und immateriellen Anteil: (Grund)- Programmausstattung (software).*

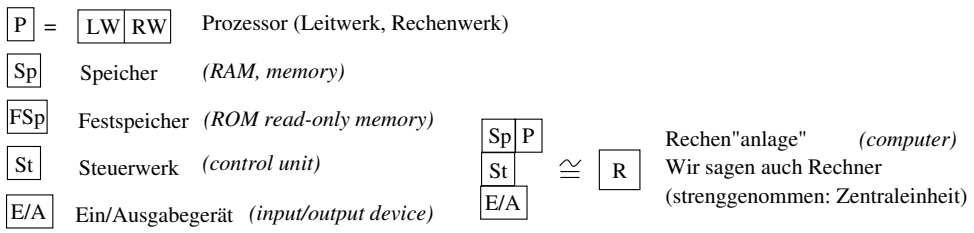


Abbildung 2.6: Komponenten eines Rechensystems

Es gibt Rechenanlagen von ca. 5 DM bis $5 * 10^7$ DM, von ca. 10 bis 10^{12} Rechenoperationen pro Sekunde, von ca. 10 bis 10^{12} Byte Speicherkapazität. Sie bestehen (nach Stone's Architekturbegriff) im wesentlichen aus denselben Moduln:

Beispiel: Taschenrechner (*calculator*)

Billigklasse, 10 DM physisch: Gehäuse, Tastatur, LCD-Anzeige, Batterie, ein (oder notfalls wenige) Chips; funktional: z.B. 10^4 Transistoren, 10mW, keine Reparatur (Wegwerfcomputer)
Der Festspeicher (FSp) enthält Programme; für jede Funktionstaste eines. Funktionstaste muß in Anfangsadresse verschlüsselt werden.
Ist der Rechner "programmierbar", dann brauchen wir noch einen Speicher für:

- Funktionstasten $\sim$ Anfangsadressen
- Ablaufbefehle (*goto, branch,...*)
- Konstante
- eventuell Variable

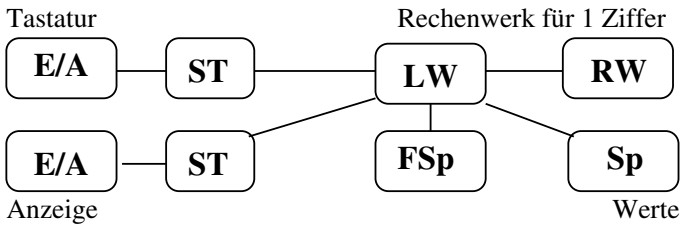


Abbildung 2.7: Aufbau eines Taschenrechners

Ist der Taschenrechner ein BASIC-pocket computer, dann heißt er schon *high-level-language-machine* , benutzt einen BASIC-Interpreter und einen richtigen von-Neumann Rechner (Mikroprozessor)

Beispiel: Kleinrechner, insbesondere

Arbeitsplatzrechner (*workstations)* ; Preis ca. 2 TDM bis 60 TDM und persönliche Rechner (*PC, personal computer*); Preis ca. 1 TDM bis 5 TDM. (oft im engeren Sinn 80x86 basierte Rechner mit MS-DOS, OS/2, Windows NT, Windows 9x, Windows 2000, Millenium).
Busorientierte Rechner, d.h. einige wenige gemeinsame Verkehrswege: E/A-Geräte sind bei-

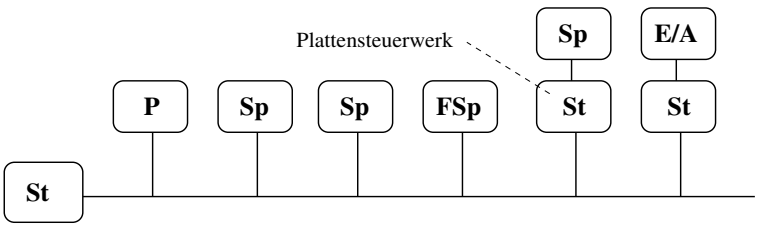


Abbildung 2.8: busorientierter Rechner

spielsweise Drucker oder Tastatur. Ein Vorteil ist die modulare Ausbaubarkeit (Steckplätze am Bus). Für höhere Leistung wird eine hierarchische Bus-Struktur verwendet.
Physischer Aufbau als eine Einheit (*Notebook, Laptop*) oder in zwei oder drei Einheiten (Tastatur mit/ohne Zentraleinheit, Platte, Floppy), Monitor; "Tower" oder unter-Monitor-Kasten für Zentraleinheit, Platte, Floppy; Maus, eventuell Drucker, CD-ROM Laufwerk.

Beispiel Workstation Hewlett-Packard / Apollo 715/64

Preis ca. 18 TDM, 40% Hochschulrabatt; Institut für Informatik hat etwa 1100 Workstations, davon waren 100-150 HP 715/64 (werden oder sind größtenteils ausgemustert) Betriebszeit etwa 6000 h/Jahr.

Tastatur: kapazitiv, getränkeempfindlich
Monitor : Farbdisplay mit Steuerung und Spannungsversorgung, ca. 25 kg
Maus neigt zu Tastenbruch und interner Verstaubung
Zentraleinheit mit E/A-Steuerungen, Platte, evt. Diskette, Band; ca. 8kg

Leistungsverbrauch	**eingeschaltet**	**ausgeschaltet**
Zentraleinheit	75 W	11,5 W
Display	147 W	31 W
gesamt	**222 W**	**42,5 W**

Das ergibt Stromkosten von 325 DM pro Jahr, falls ausgeschaltet außerhalb der Betriebszeit!

Die Leiterplatte: Raster 1,25mm und 0,625mm; Elemente teils flach aufgelötet, teils gesteckt und verlötet
Durchkontaktierung durch 0,2mm Bohrungen, acht Signalebenen, eine Masse-Ebene, eine 5V-Ebene
Hauptspeicher auf sechs Steckkarten, 8 große IC's, ca. 25 kleine IC's, Widerstandsbausteine als Busabschluß, einige hundert Siebkondensatoren zwischen Masse und 5V geschaltet. insgesamt etwa 1200 Bauelemente

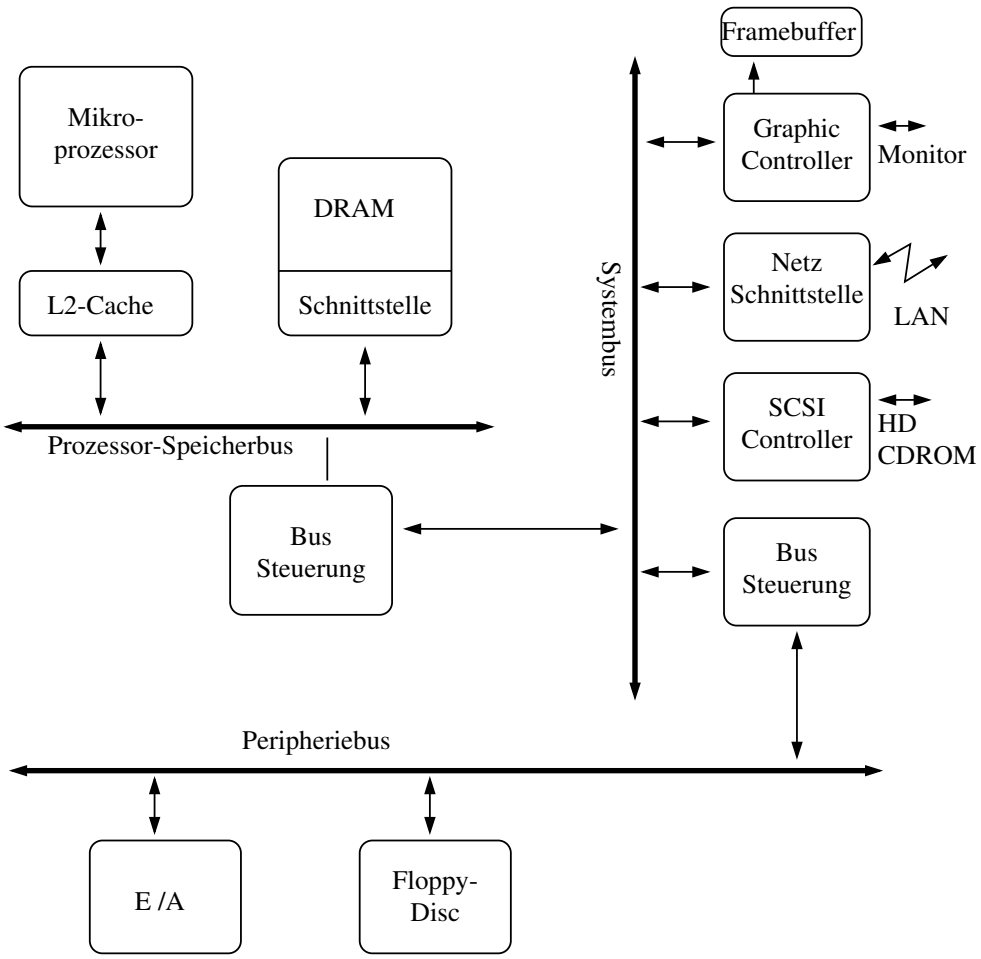


Abbildung 2.9: Aufbau von Arbeitsplatzrechnern, Bushierarchie

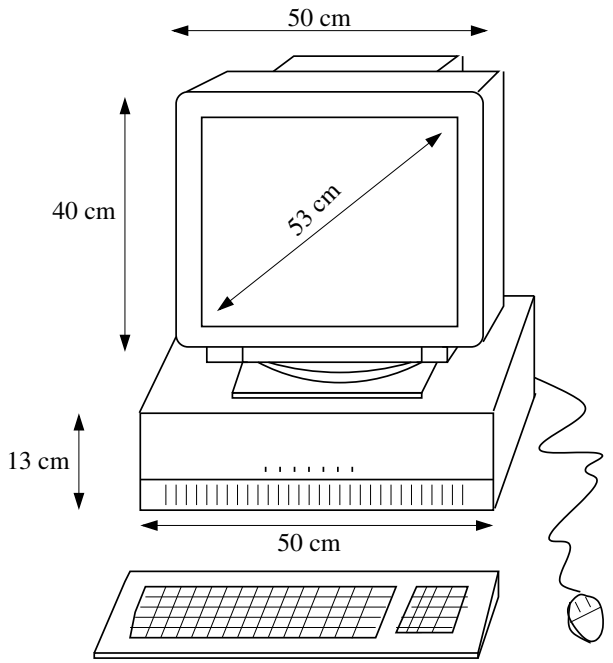


Abbildung 2.10: Hewlett Packard Apollo 715/64

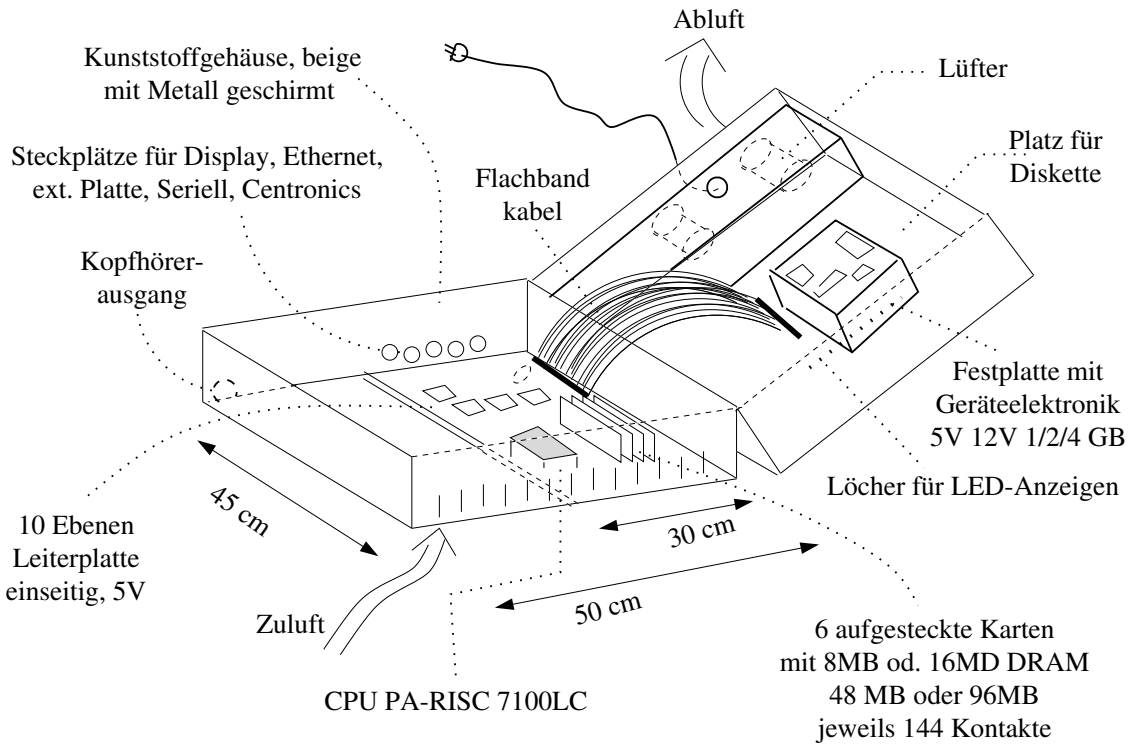


Abbildung 2.11: Zentraleinheit, geöffnet

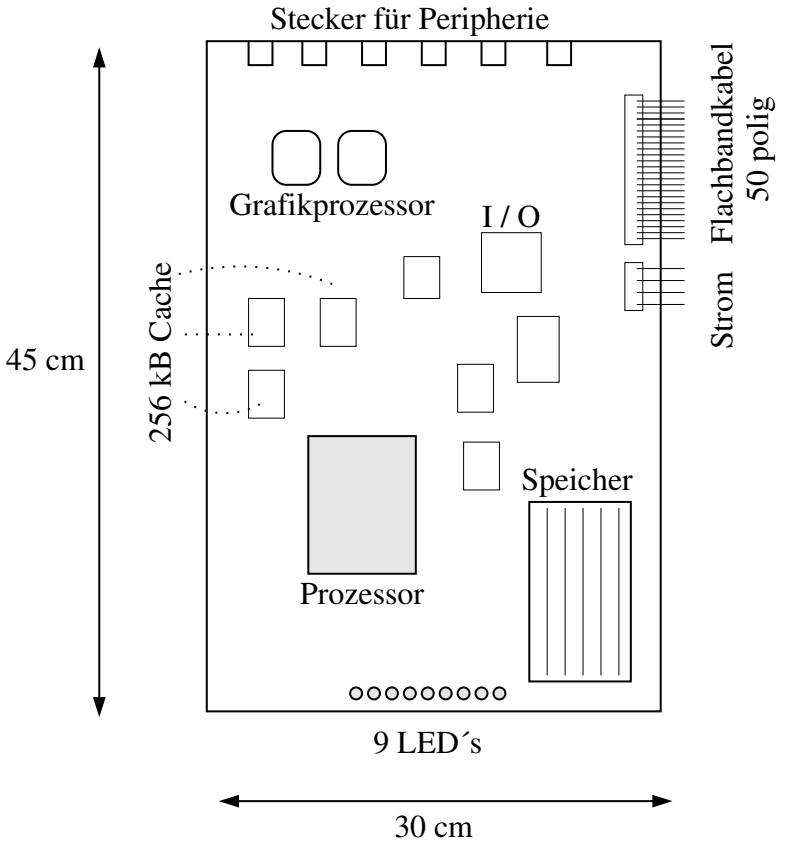


Abbildung 2.12: Leiterplatte der HP Apollo 715/64

CPU: PA 7100 LC, 10 W, 4x32 pins, Kühlkörper aufgeschraubt; Leistung 67 SpecInt 92 $\approx$ 45 MOp / s, 18 MFlOp / s, Taktfrequenz 64 Mhz (16ns)

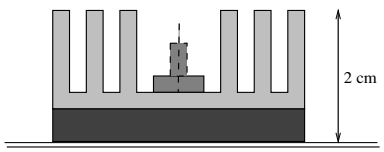


Abbildung 2.13: CPU PA 7100LC mit Kühlkörper

Display: Farbdisplay mit 1280 x 1024 Bildpunkten (keine Fernsehtechnik!) Eigene Stromversorgung, kein Lüfter

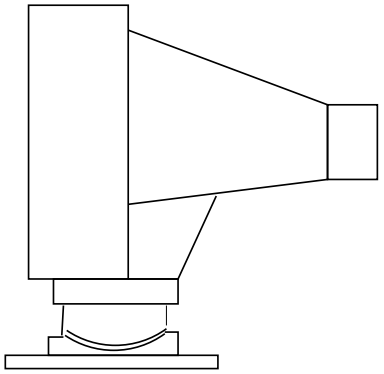


Abbildung 2.14: Display

Wartung:

Teile austauschen:
CPU, Speicherkarten, große Leiterplatte, Platte, Monitor; dazu Selbsttestprogramme mit *go/nogo* oder kryptischer Ausgabe. systematischer Austausch

vorbeugend:
Reinigung der Zentraleinheit von Staub (Wärmestau) und Maus; alle 6000 h

eigentliche Probleme:
Platten: Frühausfälle, sonst gut, Mäuse

Fünf Mitarbeiter für 1100 Rechner (etwa 700 DM / Rechnerjahr ! Soviel wie 2 Arbeitsstunden der HP-Wartung)

Beispiel: Laptop ("auf den Schenkeln"), Compaq LTE, 3 kg, 640 kB (Grundausstattung), 5,4 TDM bis 11,4 TDM, Netz/Akkubetrieb

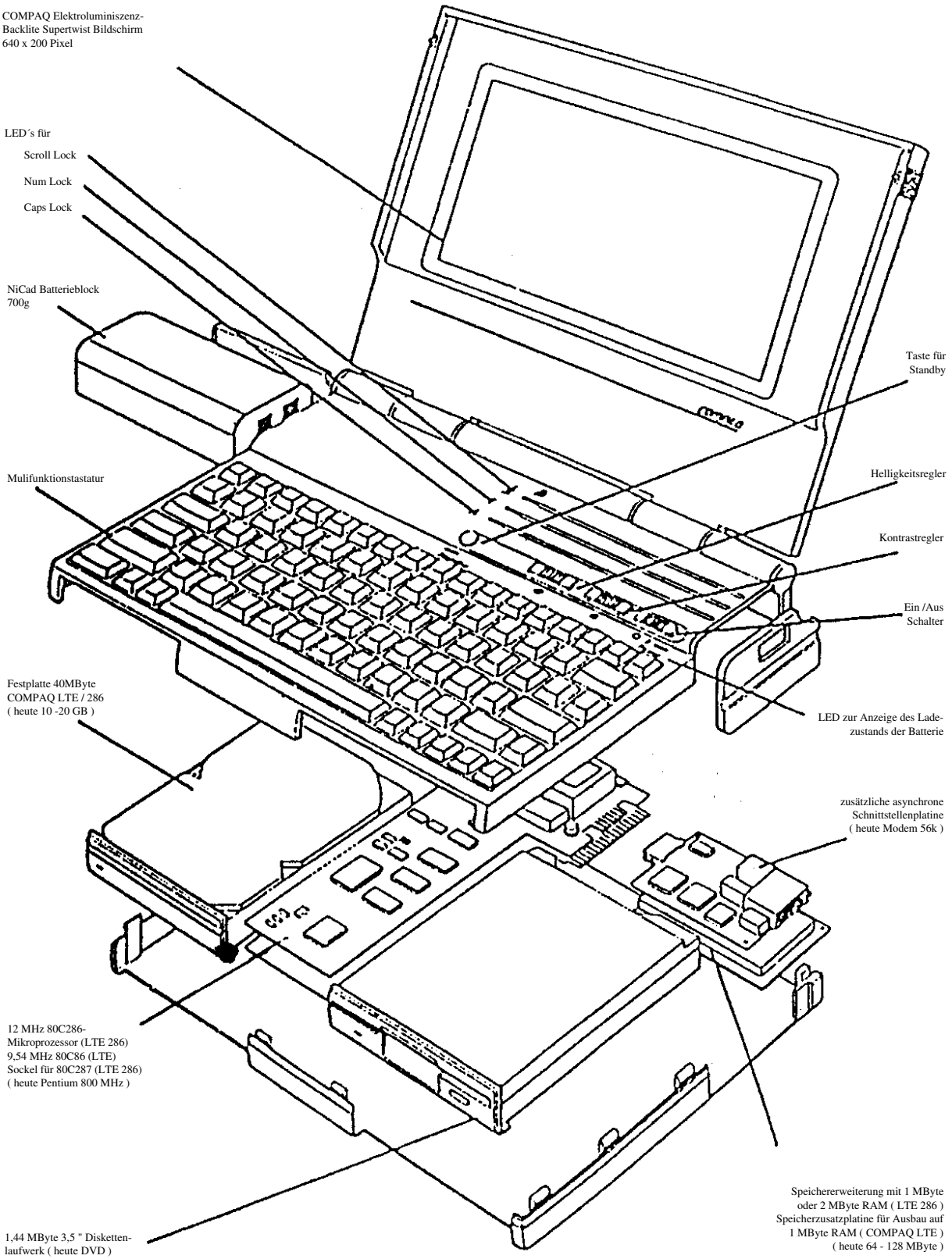


Abbildung 2.15: Laptop Compaq LTE

Beispiel für einen Höchstleistungsrechner: Hitachi SR 8000 F1 am LRZ

Das Leibniz Rechenzentrum der bayrischen Akademie der Wissenschaften installierte im 1. Quartal 2000 den Bundeshöchstleistungsrechner, der Benutzern in ganz Deutschland für sehr große Rechenaufgaben zur Verfügung stehen soll, Zugang erfolgt über Netz, die Kontingente werden von einem Beirat verteilt. Höchstleistungsrechner sind seit etwa 6 Jahren Parallelrechner.

Definition 2.14 (Parallelrechner) *Rechner, der Aufträge, die (in sich) nebenläufig sind, kollateral verarbeitet. [Gegenbeispiel: RISC-Prozessor, zwar kollaterale Verarbeitung, aber sequentielle Aufträge (Befehle des Programms)]*

Universelle Parallelrechner sind Multirechner. Andere Parallelrechner sind:

- Feldrechner (nicht universell)
- Mutiprozessoren (hier: gemeinsamer, zentraler Speicher (nicht so leistungsfähig)

Multirechner reichen bis etwa 9000 Rechner (schnellster Rechner Intel/Sandia ASCI Option Red, Grenzdurchsatz 2,9 TFlOp/s ($2{,}9*10^{12}$ Gleitpunktoperationen pro Sekunde; ASCI: *Advanced Strategie Computing Initative*) Multirechner enthalten konventionelle CPU's (z.B. Intel ASCI Option Red: Pentium II) und Hauptspeicher. Strukturell kennzeichnend ist das Verbindungswerk (*interconnection network*).

Verbindungswerk: Anforderungen

- hoher Durchsatz zwischen den Rechnern (mehr als in Fernrechnernetzen), z.B. 500..1000 MB/s
- kleine Transportzeit zwischen den Rechnern (viel weniger als in Fernrechnernetzen, da cm..m Distanzen), z.B. wenige μs; aber zusätzliche Softwarezeiten (Kommunikationssoftware) z.B. wenige 10 μs.
- möglichst alle Rechner untereinander gleich erreichbar (das ist schwierig bei großer Rechnerzahl! z.B Intel ASCI Option Red: dünner Quader), sonst muß das Verbindungswerk beim Programmentwurf sorgfältig berücksichtigt werden

Zu Lösungen und Bewertungen vergleiche 2.5 ! Das vorrangige Betriebsziel bei Parallelrechnern ist: Lösen einer sehr großen Aufgabe in beschränkter Zeit; wird erreicht durch extensive Parallelisierung z.B. Gittermethoden bei numerischer Lösung partieller Differentialgleichungen. Folge:

- Anlage ist im Einprogrammbetrieb
- für die Bearbeitung von Problemen mit kleinerer Parallelität muß das Betriebssystem die Anlage zerlegen können: Mehrprogrammbetrieb durch Partitionierung
- wie auf jedem anderen Rechner ist Mehrprogrammbetrieb durch Zeitmultiplex zusätzlich erforderlich (Auslastung steigern u.a.)

Ein Betriebssystem (Strategie, Benutzerverwaltung, externe Kommunikation / Peripherie mit "Mikrokernels" auf jedem Rechner, Prozessor/Speicher Kommunikationsmanagement)
Die wesentlichen Handicaps in der Parallelrechnernutzung sind:

- Entwicklung nebenläufiger Verfahren
- Programmier- und Testtechnik
- Abhängigkeit effizienter Problemlösungen von detaillierter Kenntnis der Verbindungs- und Kommunikationsfunktionalität: aufwendig, Portabilitätsprobleme.

Hitachi SR 8000 F1 LRZ

	1.Ausbaustufe	2.Ausbaustufe
Zahl der Knoten	112	168
Grenzdurchsatz	1,34 TFlOp/s	2,02 TFlOp/s
bei typ. LRZ-Last mind.	0,4 TFlOp/s	0,6 TFlOp/s
Hauptspeicherkapazität	0,93 TB	1,34 TB
Grenzdurchsatz		
interne Kommunikation	56 GB/s	75 GB/s
Plattenkapazität	2,2 TB	2,8 TB
Grenzdurchsatz Platten	2,3 GB/s	3,0 GB/s

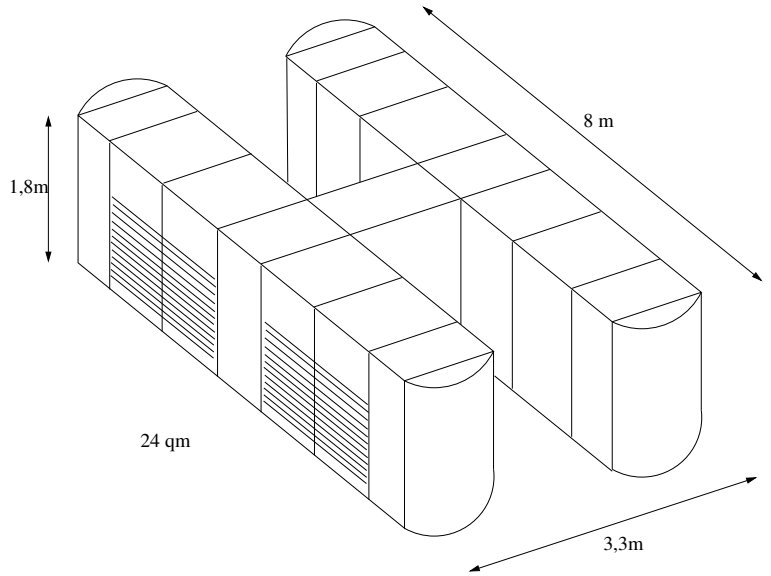


Abbildung 2.16: Hitachi SR 8000 F1 am LRZ

Die Maschine ist ausbaubar bis 512 Knoten (dann 6,1 TFlOp/s)

Verbindungswerk: "dreidimensionales Crossbar", [ein Crossbar verbindet konfliktfrei jeden der p Eingänge mit einem beliebigen, aber freien der q Ausgänge] 1GB/s von Knoten zu Knoten, 19μs Transportzeit für eine kurze Nachricht. Alle Knoten benachbart ?

Knoten der Hitachi SR 8000 F1

Die Knoten sind kleine Mehrprozessorsysteme aus 8+1 (Hilfsfunktionen) CPU's, die über ein Speicherzugangswerk (Crossbar mit aktiven Synchronisations- und Kommunikationsfunktionen zwischen den CPU's) auf den gemeinsamen modularen Arbeitsspeicher(8 oder 16 GB) zugreifen. Die CPU's sind eine Eigenentwicklung von Hitachi in Erweiterung der Funktionalität der IBM-Power-Prozessoren.

- 160 Gleitpunktregister (32 global, 128 Thread-spezifisch)
- 32 Festpunktregister
- 128 kB Daten-Cache
- 64 kB Befehls-Cache
- *prefetch* -Befehle laden Blöcke in Cache vor, *preload*-Befehle laden Register vor
- 64b-Adressen
- Grenzdurchsatz 1,5 GFlOp/s

Das Speicherzugangswerk enthält eine Kopie aller *tags* (Cache-Block-Kennzeichen) aller CPU's zur Konsistenzsicherung.
Die Knoten können als Vektorrechner betrieben werden (SPMD-Betrieb, (*single-program-multiple data*)d.h. alle Komponenten eines (möglichst langen!) Vektors werden in einer Operation verarbeitet; üblicherweise auf "pipeline". Hier werden 8 Pipelines der CPU parallelisiert. In dieser Betriebsart können alle 8 CPU's denselben Code annähernd synchron abarbeiten.(Hitachi: PVP, *pseudo vector processing* Sehr

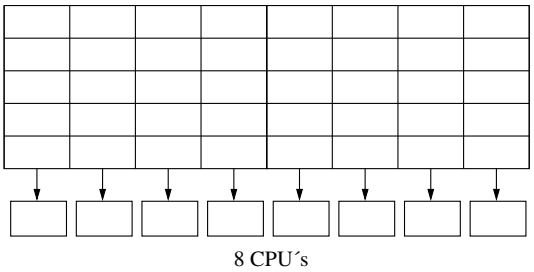


Abbildung 2.17: SR 8000 F1 als Vektorrechner

viele für Parallelrechner geeignete Aufgaben nutzen nebenläufige Operationen auf arrays (Vektoren)! Außerdem gewöhnlicher Multiprozessorbetrieb (verschiedene Programme)

2.5 Kommunikationsstrukturen

Strukturen und Verfahren zur Verbindung von Nachrichtenquellen und Nachrichtensenken: [HÄNDLER BODE 80] S.194ff, [KÄSTNER H 78] S.13ff

Zunehmende Bedeutung von Kommunikationsproblemen in Rechensystemen:

- asynchron arbeitende, lose gekoppelte, schematisch kommunizierende Einheiten sind wichtig wegen:
  - Modularisierungsvorteilen
  - Rechnernetzen
- VLSI (*very large scale integration*) macht Kommunikationswege kritisch:
  - Platz auf dem Chip
  - Kontakte am Chip
  - Verbindungen zwischen den Chips

Nachrichtenstationen (Quelle/Senke), z.B. Rechner, Speicher

- baut Datentransport auf/ab
- sendet/empfängt Nutzdaten
- nach Einsatz bei Auf/Abbau: aktiv (*master*) z.B. Steuerwerk, passiv (*slave*) z.B. Speichermodul

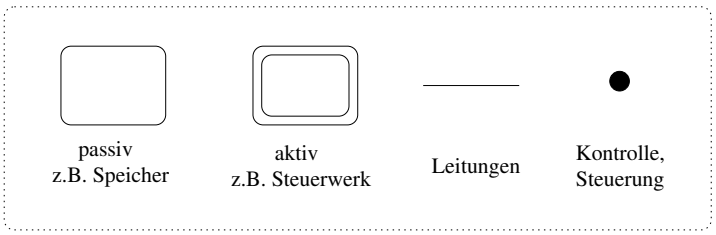


Abbildung 2.18: Symbole

Nachfolgend werden Leitungen i.a. nach Steuer- und Nutzdatentransport und nach Richtung und Gleichzeitigkeit des Nachrichtentransportes getrennt:

- (Voll) duplex "bidirektional": beide Richtungen gleichzeitig nutzbar (z.B. Telefon)
- Halbduplex : beide Richtungen im Wechsel nutzbar (z.B. Bus)
- Simplex : nur eine Richtung nutzbar

Nach Bitfüllung (im Leitungsquerschnitt):

- seriell
- parallel

Auf einer langen seriellen Leitung können viele Bits gleichzeitig unterwegs sein. ein Bit von 1 μs Länge mißt ca. 200m, von 1 ms Länge mißt ca. 200 km

"Leitungen" können sein (seriell, entfernungsabhängig):

- Drahtleitungen, verdrillt bis ca. 100 MBit/s
- Streifenleitungen (kurz) bis ca. 1GBit/s
- Koaxialkabel bis ca. 100 MBit/s
- Glasfaserkabel bis ca. 10 GBit/s
- Funkverbindungen bis ca. 100 MBit/s

Netze aus Nachrichtenstationen und Leitungen enthalten für die Nutzdaten:

- zugeordnete (eigene, dedizierte) Leitungen, diese verbinden genau zwei Nachrichtenstationen
- gemeinsame (omnibus!) Leitungen (bus) [außer als Ring!] stets halbduplex, Aufteilung unter den Kommunikationspartnern in:
  - Frequenzteilung (Multiplex), nicht in Rechnern
  - Zeitteilung (Multiplex)
    Synchronbus, starre Scheiben (fest zugeteilt oder nach Anforderung) oder Asynchronbus, variable Scheiben

Frequenzteilung und Zeitteilung mit fester Zuteilung starrer Scheiben sind einfach und ermöglichen eine gute Ausnutzung, erlauben aber keine Anpassung an wechselnde Nachfrage der Partner.

Steuerungen :
Bauen Zugang zu Leitungen / Verbindungen zum Partner auf und ab. Die Realisierung erfolgt:

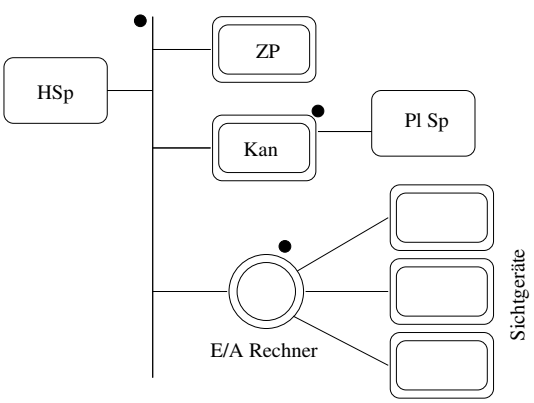


Abbildung 2.19: Beispiel Steuerung

- zentral: oft billig, gute Auslastung der Leitungen möglich, eindeutige Steuerung
- dezentral : leicht erweiterbar, fehlertolerant (Diagnose u.U. schwieriger) kein zusätzlicher Nachfrageverkehr

Nachrichtenvermittler:
wählt den weiteren Weg ankommender Nachrichten, kann oft auch Nachrichten speichern und umsetzen. Damit indirekte Übertragung

Vermittlungsprinzipien im Netz, wichtige Fälle:

Leitungsschaltung (*line switching*):
Vermittler bauen vor der Übertragung einen dedizierten Weg zwischen Quelle und Senke auf (typisch: Fernsprechen). Wesentliches Prinzip im Rechner: Zugang zu Speicher und Peripherie (wenn indirekt)

Teistreckenschaltung
Nachricht wandert nur zum jeweils nächsten Vermittler, von dort weitervermittelt

- als Ganzes, unter Zwischenspeicherung: Nachrichtenvermittlung (*message switching*)
- in Pakete zerlegt (*packet switching*)
  - jedes Paket wird zwischengespeichert, weiterer Weg unabhängig gewählt (*datagramm*)
  - jedes Paket folgt dem Leitpaket der Nachricht
    - nach Zwischenspeicherung (X.25, ATM)
    - ohne Zwischenspeicherung, Vermittlung *"on the fly"*, sehr kurze Pakete (*flits*).*"wormhole routing"* in einigen Parallelrechnern.

Beispiele zu Netzstrukturen

direkt (d.h. ohne Vermittlung), verteilte Steuerung

zugeordnete Wege : voller Verbund schlecht erweiterbar, $\binom{n}{2}$ Wege notwendig, durchsatzstark;

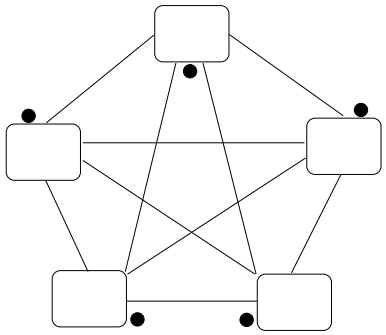


Abbildung 2.20: voller Verbund

aber in dieser Form nicht ausfallsicher

allgemeine Wege : Ethernet

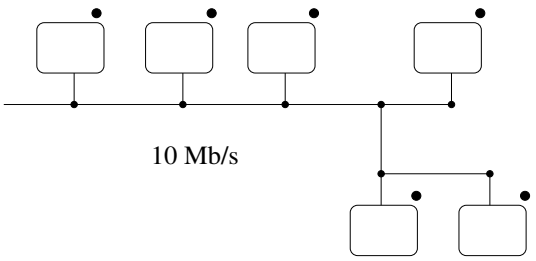


Abbildung 2.21: allgemeine Wege, Ethernet

gut erweiterbar, durchsatzschwach, dezentrales Zugangsverfahren ungeignet bei starkem Auftragszugang, nicht ausfallsicher, empfindlich gegen Subversion

direkt, zentrale Steuerung

zugeordnete Wege: Kleinstrechner mit Ein/Ausgabe über Zentralprozessor

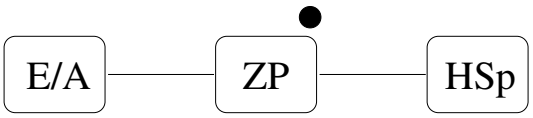


Abbildung 2.22: Ein/Ausgabe über Zentralprozessor

allgemeine Wege : zentraler Bus in Klein / Minirechnern (seit DEC PDP 11, 1970, "Unibus")

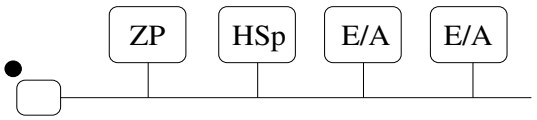


Abbildung 2.23: zentraler Bus

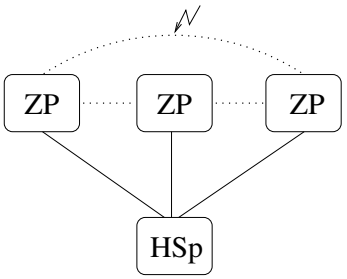


Abbildung 2.24: Kommunikation über zentralen Speicher

durchsatzschwach (aber stärker als bei verteilter Steuerung) noch weniger ausfallsicher

Ähnlich wirkt Kommunikation über zentralen Speicher, Mehrprozessoranlage: ohne gegenseitige "Alarme" zwischen den Zentralprozessoren nicht akzeptabel!

indirekt (über Vermittler), verteilte Steuerung

allgemeine Wege
Fernnetz (*wide area network*): typisch chaotische Struktur aus Leitungen und Vermittlern verschiedenen Grenzdurchsatzes. Teilstreckenvermittlung, gute Erweiterbarkeit, Fehlertoleranz möglich.

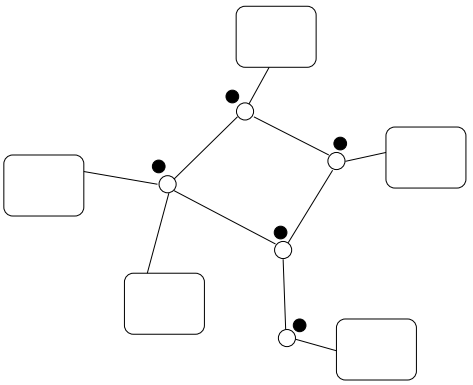


Abbildung 2.25: verteilte Steuerung, mit Vermittlern, chaotisch

In Parallelrechnern reguläre Strukturen, z.B. Gitter (Intel Paragon), oft Torus (Kosten $\sim$ n , Weglänge $\sim \sqrt{n}$)

Bessere Strukturen mit der Weglänge $\sim \log n$ sind Baum und Hypercube :

indirekt , zentrale Steuerung

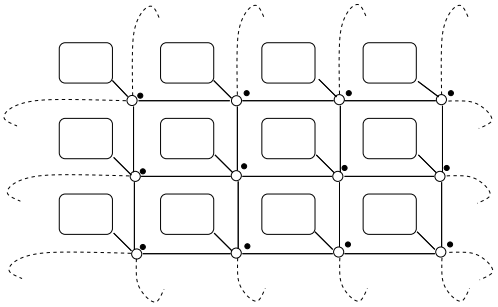


Abbildung 2.26: Gitter mit *wrap around* zum Torus

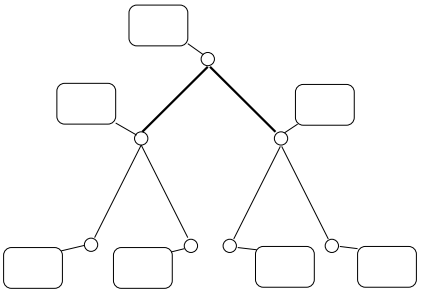


Abbildung 2.27: Baum, Kosten $\sim$ n

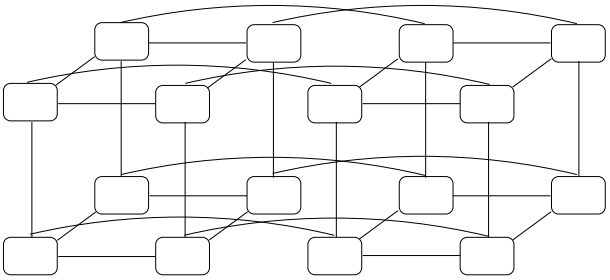


Abbildung 2.28: Hypercube, Kosten $\sim n \log n$

zugeordnete Wege

Crossbar (Kreuzschienenverteiler) verbindet jeden Zugreifer mit jedem freien Ausgang gleichzeitig; teuer, hohe Leistung. z.B. bei Parallelrechnern, Fernsprechen im Amtsbereich (kein Crossbar, Schalterkaskade)

allgemeine Wege

Übertragung zwischen E/A-Geräten, die an verschiedenen E/A-Bussen bzw. Unterkanälen hängen.

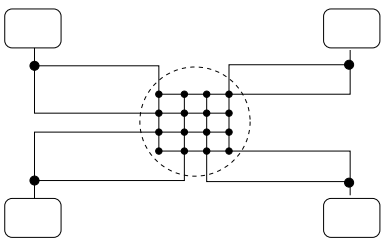


Abbildung 2.29: Crossbar mit vier Ein/Ausgängen

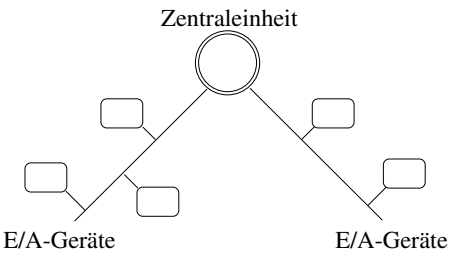


Abbildung 2.30: allgemeine Leitungen, zentral gesteuert

Weitere wichtige Begriffe zur Kommunikation

Schnittstelle (*interface*):

Menge der Festlegungen, die ordnungsgemäßes Verhalten eines Kommunikationspartners gegenüber einem/dem anderen beschreiben.

Physische Schnittstelle:

Leitungen, Stecker, Potentiale, Zeiten, Widerstände...

Kommunikationsprotokoll:

Syntax, Semantik, und zeitliche Ordnung des Nachrichtenaustausches zwischen zwei oder mehreren Partnern

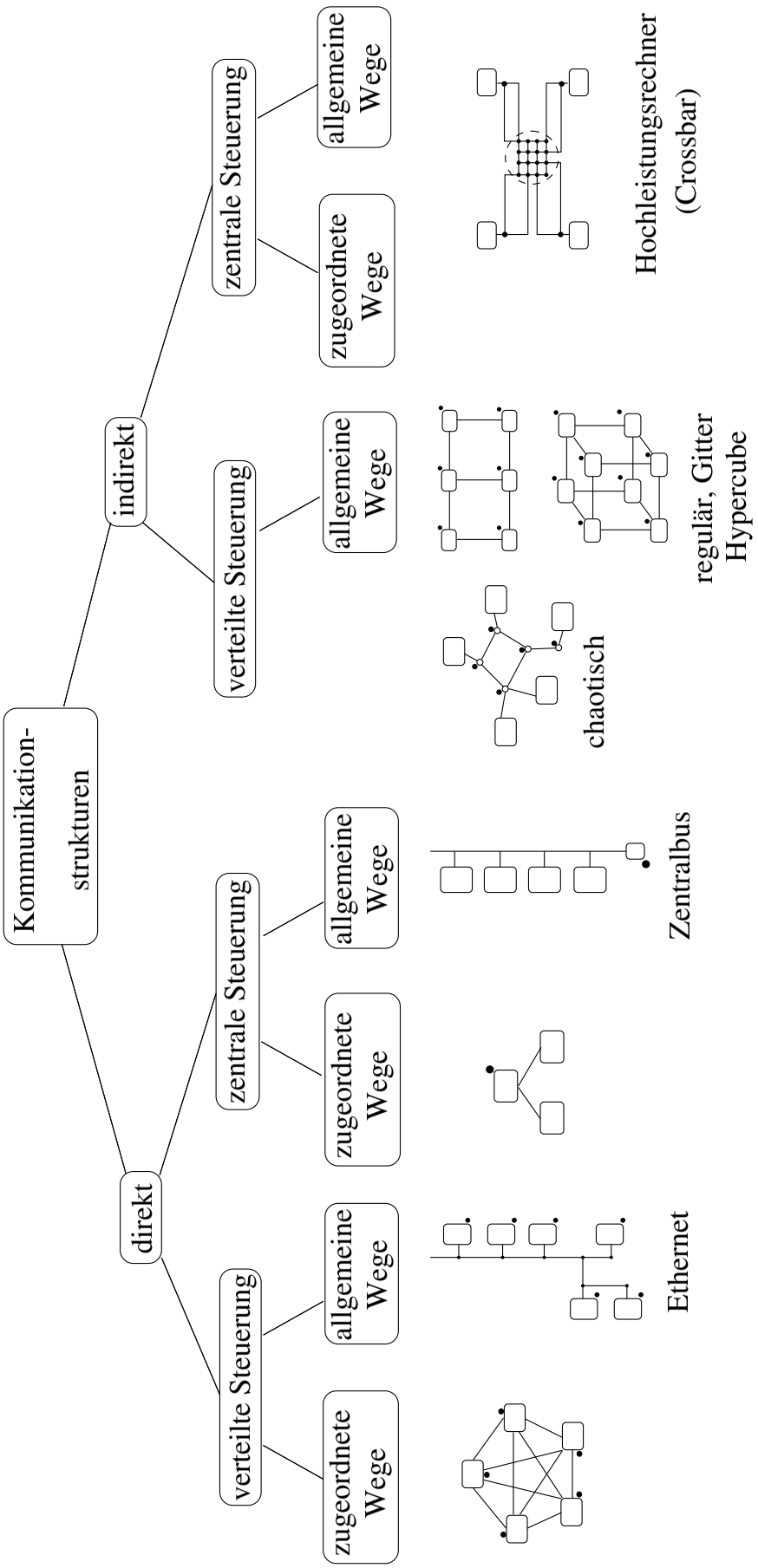


Abbildung 2.31: Übersicht Kommunikationsstrukturen

2.6 Interpretationshierarchie

[JESSEN E 75] S.65ff [GILOI W 80] S.11ff [TANENBAUM A 76] S.76ff

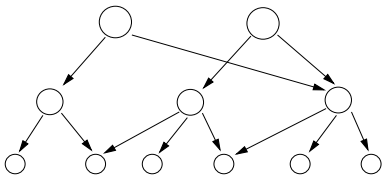


Abbildung 2.32: Skizzierung einer Hierarchie

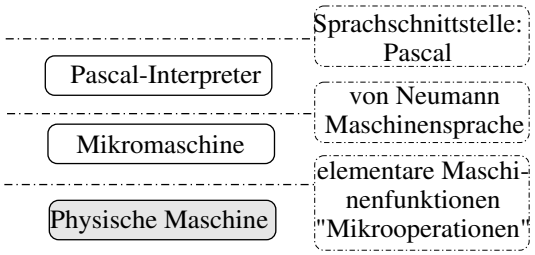


Abbildung 2.33: Darstellung als Schichtenbild

Hierarchie: Strukturierung einer Menge von Elementen als azyklischer Graph (oft: als Baum), wobei in verschiedenen Ebenen dieselbe Relation zwischen Vorgängern und Nachfolgern besteht, z.B.:

wird interpretiert von	Interpretationshierarchie
ruft auf	Aufrufhierarchie
wird realisiert durch	funktionale Hierarchie
besteht aus	konstruktive Hierarchie
setzt zum Test voraus	Testhierarchie
bildet Ausschnitt von	Ausschnittshierarchie
wird formuliert durch	Protokollhierarchie
wird berechnet als	Hierarchie in arithm. Ausdrücken
leitet	Management

Zweck: Übersichtlichkeit (Komplexitätsminderung), Anpassbarkeit, Sicherheit, Arbeitsteilung,...

Interpretationshierarchie

Für die Funktion von SYNT und/oder SEM wird eine Interpretation substituiert.

Beispiel dreistufige Hierarchie, Pascal-Interpreter auf "mikroprogrammierter" Maschine:

P	Hauptspeicher enthält Pascalprogramm
SYNT	syntaktische Analyse des Interpreters
SEM	semantischer (ausführender) Prozess des Interpreters
Z	Hauptspeicher enthält Variablen des Programms

SYNT und SEM durch von Neumannsche Maschine interpretiert, z.B.:

SYNT.P	Hauptspeicher enthält syntaktische Analyse
SYNT.SYNT	Leitwerk
SYNT.SEM	Rechenwerk
SYNT.Z	Hauptspeicher enthält Interpreterzustand

Entsprechendes gilt für SEM.
Leitwerk und Rechenwerk werden durch Mikromaschine interpretiert, z.B.: SYNT.SYNT (=Leitwerk) durch

Leitwerk.P	Mikroprogrammspeicher enthält Programm für Elementarzyklus des Leitwerks, formuliert in "lesen", "schreiben", "transportieren", "addieren",... Elementarfunktionen des Leitwerks und Speichers
Leitwerk.SYNT	Mikroleitwerk
Leitwerk.SEM	Elementarfunktionen des Leitwerks und Speichers
Leitwerk.Z	Register des Leitwerks und Register für Zustand der Leitwerksinterpretation

Wenn Leitwerk und Rechenwerk nicht gleichzeitig tätig sind, kann dieselbe Mikromaschine beide interpretieren!

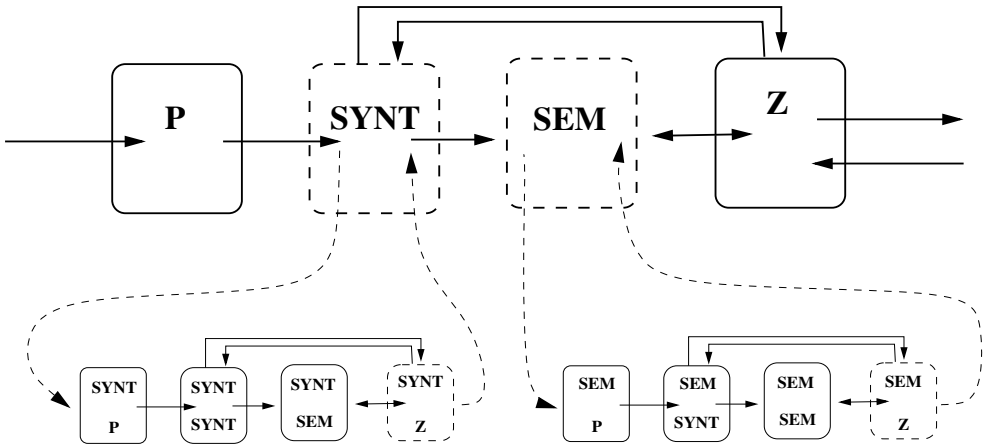


Abbildung 2.34: Interpretationshierarchie

Mikroprogrammierung : Interpretation unterhalb der von Neumann Ebene (hat nichts mit Mikroprozessoren zu tun).

Praktisch sind Rechensysteme noch höher geschichtet:

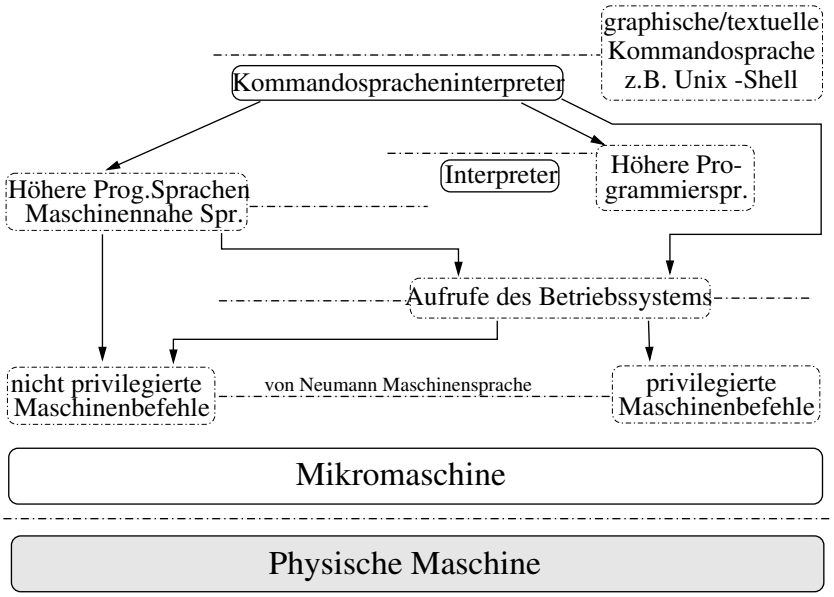


Abbildung 2.35: Beispiel: Schichtung eines Rechensystems

Die von Neumann Maschinensprache ist die Hardware-Software Schnittstelle.

2.7 Ausschnittshierarchie

Für den Interpretationsprozess müssen **P** und **Z** (aller Ebenen) in Speicher(n) realisiert werden.

Speicher sind: **schnell, aber teuer** je Bit oder **billig je Bit, aber langsam**
Speicher großer Kapazität sind stets langsam, aber im allgemeinen billig je Bit.

Interpretationen brauchen in kleinen Zeitintervallen nur kleine Untermengen des Inhalts von **P** und **Z**, diese ändern sich langsam und sind weitgehend kompakt im Adressraum. Die Interpretation verhält sich "lokal". Dies führt zur Idee des Auschschitts (erstmals F.R. Güntsch 1957, erste Maschine Ferranti Atlas

1963): **P** und **Z** in großem, billigen Speicher, den aktuellen Auschnitt in kleinem schnellen Speicher unterzubringen. " So billig wie der große, so schnell wie der kleine Speicher"

Beispiele (zusammen bilden sie eine Hierarchie):

	Speicher 1	Speicher 2	Speicher3
Typ	Cache	Hauptspeicher	Plattenspeicher
Zugriffszeit	10 ns	50 ns	10 ms
Kapazität	64 kB	64 MB	20 GB
Preis je Byte	0,1 Pf	0,01 Pf	0,0001 Pf

Der Austausch zwischen 1 und 2 erfolgt automatisch (Prozessorhardware),der Austausch zwischen 2 und 3 automatisch(Prozessorhardware, Betriebssystem) oder benutzerprogrammiert. Die Kapazität des Cache-Speichers ist oft durch die Unterbringung auf dem Chip bestimmt.

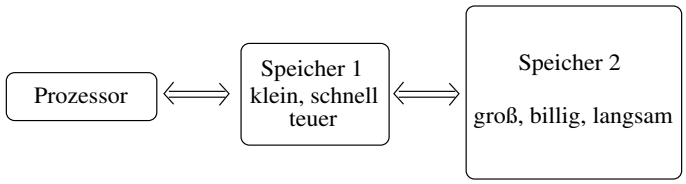


Abbildung 2.36: Prinzip des Ausschnitts

Grundprobleme des aktuellen Ausschnitts:

Die Trefferwahrscheinlichkeit h (*hit ratio*) hängt von k_A, LV, VV und der Prozesslokalität ab. Andere Zugreifer setzen Übereinstimmung zwischen Ausschnitt und Original voraus!

Strategischer Freiraum liegt im Austauschverfahren:

Ladestrategie *on demand* (z.B. Cache memory, Hauptspeicher) oder vorhersehend:

> Wie kann man trotz prinzipiell unbekanntem künftigem Interpretationsverlauf mit möglichst hoher Trefferwahrscheinlichkeit h das Gesuchte im Speicher bereitstellen?
> Beispielsweise durch zusätzliches Heranschaffen der adressräumlichen Umgebung zu jedem benötigten Wort.

Verdrängungsstrategie z.B. *least recently used* (das am längsten unbenötigte)(z.B. Cache memory)

> Die Vorhersage der Zukunft wird durch den Rückblick in die Vergangenheit ersetzt.

Kohärenzproblem Übereinstimmung zwischen Ausschnitt(Kopie) und Original erforderlich, wenn mehr

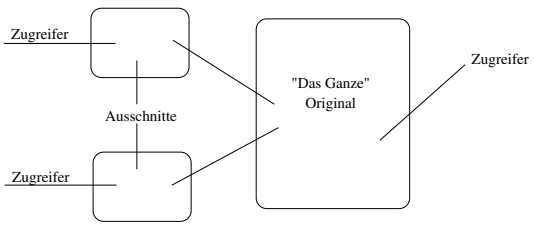


Abbildung 2.37: Kohärenzproblem

als ein Zugreifer existiert!

- bei Einprozessor z.B. CPU ⇔ Cache ⇔ Hauptspeicher ⇔ E/A

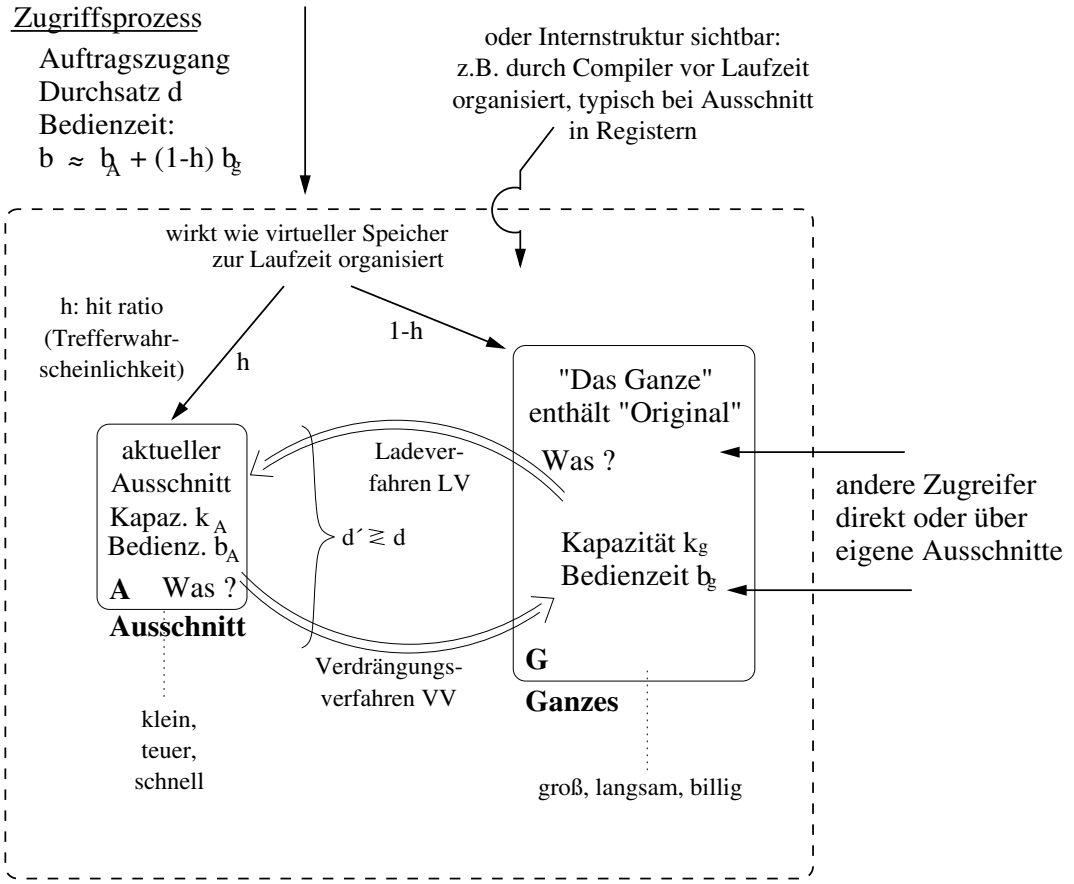


Abbildung 2.38:

- bei Mehrprozessor viele Cache's ein Hauptspeicher

- bei Parallelrechnern aus vielen Rechnern (d.h. CPU's mit eigenem Hauptspeicher) kann man alle Speicher allen Prozessoren zugänglich machen und bei nichtlokalen Zugriffen Kopien verschicken(*virtual global memory*

- bei verteilten Dateisystemen ebenso!

Übliche Lösung: *n-reader / 1 writer* Strategie, jeder Teilinhalt (Seite, Segment, Datei) hat jederzeit einen exklusiven Schreibeigentümer oder beliebig viele Lesereigentümer. Die Eigentümer erhalten lokale Kopien.

2.8 Durchsatzmasse

(stark gekürzte Fassung des 2.Kapitels in [BODE A 91/92])

Leistung: Funktionalität, zeitbewertet, (*performance*), also:

- Bedienzeiten ("Ausführungszeiten") von CPU Operationen, Programmen, Speicherzugriffszeiten,...
- Grenzdurchsätze (hier: Durchsatz bei hohem Auftragsangebot)
  - z.B. Operationsdurchsatz der CPU bei Abarbeitung eines/vieler Programme
  - z.B. (unrealistisch) *peak rate* Grenzdurchsatz bei laufender Beauftragung aller Werke eines komplexen Rechners (z.B. Hitachi SR8000F1) (*ein nicht überschreitbarer Durchsatz*)

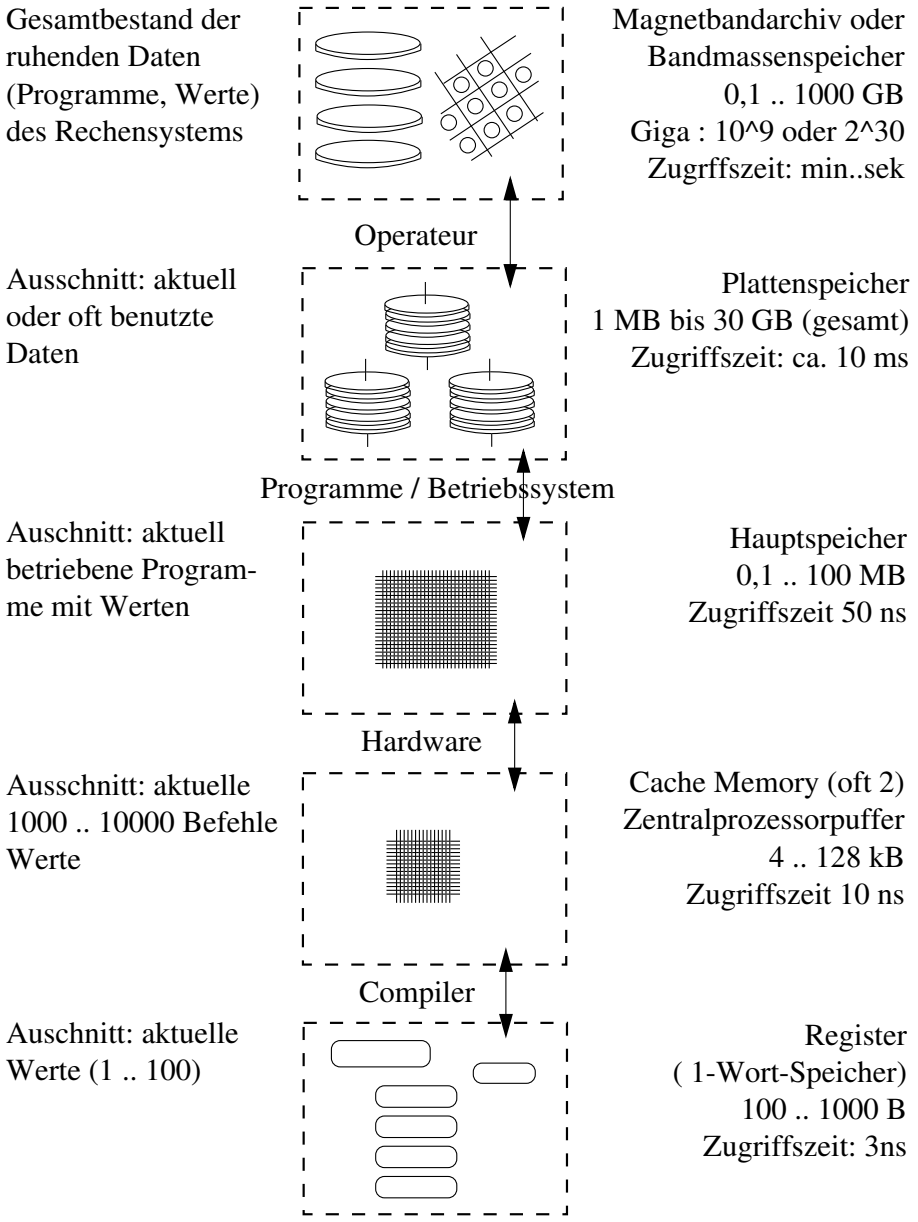


Abbildung 2.39: typische Ausschnittshierarchie in Rechnern/Rechensystemen

Zur Rechnerbewertung wesentlich:

- **Funktionalität**(Benutzerschnittstelle, Operationen, Datentypen, Konfigurierbarkeit, ...(nur kleiner Ausschnitt)
- **Leistung**
- **Zuverlässigkeit, Anschaffungs- und Betriebskosten,** ...

Auswertung statischer Hardware-Parameter

- Operations-Ausführungszeiten der CPU unbrauchbar bei Überlappung z.B. bei parallelen Werken.
- Speicherzugriffszeit (ebenso)
- Speichergrenzdurchsatz
- E/A Grenzdurchsatz

- gewichtete mittlere Operationsausführungzeit:

  $T := \sum_{i=1}^{n} h_i * t_i$ n Operationstypen

  h_i : relative Häufigkeit der Operation i
  t_i : Ausführungszeit der Operation i
  "Mix" (unbrauchbar bei Überlappung oder parallelen Werken)
  h_i sind stark abhängig von der CPU-Architektur und der Anwendung. Sie können in Kernprogrammen ermittelt werden.
  $\frac{1}{T}$ wird gelegentlich als Mips (*million instructions per second*) bezeichnet. Es gibt neben dieser (besonders schlechten) viele Mips-Definitionen.

Auswertung statischer Hardwareparameter	Laufzeitmessung an bestehenden Systemen	Messung des Betriebs bestehender Anlagen	Modellorientierte Verfahren
zur Auswahl	**zur Auswahl**	**zum Tuning**	**zum Enwurf**
Ausführungszeiten von CPU-Operationen Mixe Kernprogramme	Benchmarks synthetische Programme	Hardwaremonitore Softwaremonitore	analytische Modelle (Berechnungen) Simulation von Modellen
z.B. Add/Mult-Zeit, Speicherkapazität, Speichergrenzdurchsatz, Mips aus Mixen	Benchmark: typ. Messung der Gesamtausführungszeit von repräsentativen Programmen	Techniken: periodisch/ergebnisbezogen messen Benutzerprozess/ Betriebssystem i.A. aufwendige Nachbearbeitung, sorgfältige Experimentplanung	analytische Modelle stochastisch oder "operational" Simulation stochastisch oder tracegetrieben, Nachbearbeitung, Planung s. links

Beispiel: Gibson Mix (1970) bezogen auf IBM /360 - Rechner:

Operationstyp	h_i
laden, speichern	0,312
Festpunkt Add/Sub	0,061
Vergleich	0,038
Verzweigung	0,166
Gleitpunkt Add/Sub	0,069
Gleitpunkt Mult	0,036
Gleitpunkt Div	0,015
Festpunkt Mult	0,006
Festpunkt Div	0,002
Shift	0,044
Boolesche Operationen	0,016
registerfreie Operationen	0,053
Indizierung	0,180
	1,000

Laufzeitmessung an bestehenden oder künstlichen Programmen

Synthetische Benchmarks: Man benutzt Bausteine, die jeweils eine gewünschte Teilbelastung (z.B. Gleitpunkt, Adressrechnung, Prozeduraufrufe...) erzeugen und in Hinblick auf die gewünschte Gesamtlast zu einem Benchmark komponiert werden. Bausteine oft parametrisiert.
Eventuell Probleme mit Optimierung durch Compiler: er kann unter Umständen "sinnlose" Folgen erkennen und eliminieren. Also im Allgemeinen in Assembler.

(natürliche) Benchmarks (vgl. Kerbholz): Menge von als repräsentativ angesehenen Programme, deren Gesamtlaufzeit gemessen wird. Dabei wird i.A. mit erfasst:

- Compiler: Güte der Codeerzeugung, Optimierung
- Betriebssystem: geschickte Ablaufplanung (aber auch Eigenzeitbedarf)

Beispiele für Benchmarks:

Systembenchmarks (mit wesentlichem Betriebssystemanteil)

CPU-Benchmarks (numerikorientiert) z.B.:

Whetstone (*Wetzstein*):
Synthetisch, 9 Module (gewichtet), Fortran und mathematische Funktionen überbetont

Linpack:
Programmpakete für lineare Gleichungssysteme,Fortran

Lawrence Livermore Loops:
14 innere Schleifen von häufig benutzten Numerik-Programmen, Fortran. Sogenannter "Kernel" (besonders kennzeichnendes Teilprogramm)

CPU-Benchmarks (nicht numerikorientiert) z.B:

Dhrystone:
viele kleine Moduln, keine Gleitpunktoperationen, ADA, C, Pascal.

weitere: Quicksort, Sieb des Erastosthenes, Ackermannfunktion,...

Versuch der Objektivierung: Spec-Benchmark: Firmenkonsortium, verschiedene Realeases, natürliche Programme

Spec-Benchmarks

n reale Programme werden auf den zu untersuchenden Rechner x gegeben, die Laufzeit T_x gemessen und für jedes Programm i wird $SPEC - Ratio_i$ berechnet, als $SPEC - Mark_x$ wird das geometrische Mittel angegeben:

$$SPEC - Ratio_i := \frac{\frac{1}{T_x}}{\frac{1}{T_{SparcStation\,10/40}}} \qquad SPEC - Mark_x := \sqrt[n]{\prod_{i=1}^{n} SPEC - Ratio_i}$$

(Sparc-Station 10/40 etwa 30 VAX-Mips als Referenz)

Beispiel: *SPEC C Int 95* (*Component / Integer*): GO-Spiel, CPU-Simulator, C-Compiler,Dateikompression/ dekompression, LISP-Interpreter, Bildkompression/ dekompression, Perl-Interpreter, objektorientierte Datenbank

Beispiel: *SPEC C Fp 95* (*Component/floating-point*): partielle Differentialgleichungen, Simulationen

C-Benchmarks bewerten CPU-Cache-Compiler. Daneben gibt es auch Systembenchmarks. Benchmarks erlauben keine konsistenten Vergleiche zwischen Rechnern, da diese im Allgemeinen verschiedene Stärken haben (Abb.2.40)
CPU-Benchmarks erlauben wesentlich realistischere Grenzdurchsatzkennwerte:

Laufzeit = Operationsanzahl / CPU-Durchsatz
cpu-time = instruction count / instruction rate
Letzteres in Mips angegeben bemisst richtig die Ausnutzung von Nebenläufigkeit!
Bemerkung: oft wird $(instructionrate)^{-1}$ aufgeteilt in *clockcycles per instruction (CPI) * clockcycle time.*
instruction count: Befehlsliste / compilerabhängig
clockcycles per instruction: Befehlsliste / strukturabhängig
clockcycle time: technologieabhängig

verbleibende Probleme:

- verschiedene Benchmarks beanspruchen Funktionen verschieden stark (vgl. Abb. 2.40 VAX 11 / DEC 2060)
- auf einer Maschine kann die gleiche Funktion mehr Befehle benötigen, als auf einer anderen Maschine: A := B+C

Maschine A:	Maschine B
ADD Adr(B) Adr(C) Adr(A)	LADE Reg1 Adr(B) LADE Reg2 Adr(C) ADD Reg1 Reg2 SPEICHER Reg2 Adr(A)

Wenn Maschine B die Befehlsholphasen geschickt überlappt, braucht sie bei gleicher Technologie nicht wesentlich mehr Zeit als Maschine A. Für die Mips wird B aber 4mal höher bewertet ! Daher Spec-Benchmark: Bezug auf eine Vergleichsmaschine statt CPU- Durchsatz: Zeit^{-1} für äquivalente Funktionen

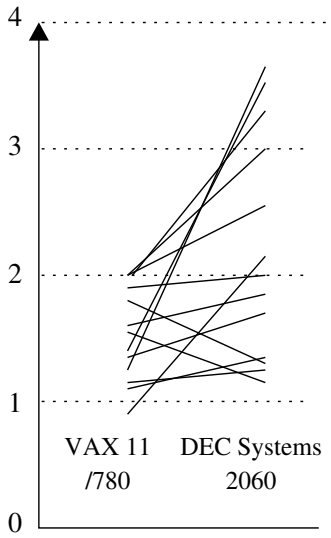


Abbildung 2.40: Vergleich

2.9 Kurze Geschichte von Konzepten und Produkten

Jahr	**Rechnerentwicklung [Technologie]**	**Restinformatik**
1936	[Fernsehen, Magnetband]	Church: λ -Kalkül Turing: Turing-Maschine Churchsche These
1941	Zuse Z3: programmgesteuerter, elektromechanischer Digitalrechenr, Gleitpunkt	
1946	v.Neumann u.a. Konzept für speicherprogrammierbare Rechner	Zuse: Plankalkül, höhere Programmiersprache
1947	[Bipolarer Transistor]	
1949	Wilkes: EDSAC v.Neumannrechner, Indexregister	
1951	Eckert, Mauchley UNIVAC 1, industrielle Rechnerproduktion Wilkes: Konzept der Mikroprogrammierung	
1954	[Kernspeicher]	Assembler, einfaches Betriebssystem
1956	UNIVAC: RR 1103: Unterbrechung	Backus: Fortran

Jahr	Rechnerentwicklung [Technologie]	Restinformatik
1957	MIT TX2: überlappte E/A, Güntsch: Konzept des virtuellen Speichers, Zuse: Konzept für Feldrechner. Sage-Rechnernetz	Mehrprogramm-Betriebssystem
1958	[Vorformen alphanumerischer Displays]	Algol
1960	RW 400 fehlertolerantes Multirechnersystem	Lisp
1961	IBM-Strech: Pipeline, [ECL]	Cobol, Quicksort
1962	IBM 709 verdeckte Basisregister, Schriftleser Wilkes Konzept Cache-Memory	Teilnehmer-Betriebssystem, Petri: Petrinetze
1963	Burroughs B5000: Highlevel-Language-Machine, Mehrprozessor, segmentweise Adressierung Ferranti Atlas: Seitenadressierung, Traps [integrierte Schaltungen in der Raumfahrt]	
1964	Control Data 6600: asymmetrischer Multirechner, Hochleistungscomputer, parallele spezielle Rechenwerke [Flüssigkeitskühlung] Operateur-Displays	BASIC
1965	IBM /360: Familie (1:300 Leistung) Großanwendung Mikroprogrammierung, DEC PDP8 "Minicomputer"	
1966	Texas Instruments: Taschenrechener [Feldeffekt-Transistor gewinnt Bedeutung; Prinzip: Liliental 1930]	Simula(1967), objektorientierte Programmiersprache
1969	ILLIAC IV Feldrechner [Halbleiterspeicher] IBM /360 Mod 85: Cache-Memory	relationale Datenbank
1971	[Telefax], Intel 4004 Mikroprozessor, IBM /370, DEC PDP11: Unibus	Pascal, Unix
1972	Diskette	C
1973	Xerox Alto: Workstation Syre: LAU- Datenflußrechner	Backus kritisiert v.Neumann-(Programmier)Stil
1974	[etwa 1974: Maus]	Prolog, Capabilities
1975	IBM: Laserdrucker, Ethernet	Expertensysteme
1977	DEC VAX-11, Commodore PET1: Personal Computer(595$)	Modula, TCP/IP
1978	Berkling: Reduktionsmaschine	Public-Key Kryptographie, Osi-Referenzmodell
1979		ADA
1980	Symbolics Lisp-Rechener	
1981	Apple Lisa, IBM PC (mit Intel 8088)	Fensterorientierte graphische Interaktion, Benutzeroberflächen (Xerox-Entwicklung) MS-DOS
1982	Patterson: RISC	
1983	Apple Macintosh (mit Motorola 68k)	C++
1984	Hillis: Connection-Machine: 64000 Rechenwerke, Sun-Workstations, Apollo-Workstations	
1985	Inmos Transputer	Verteilte Betriebssysteme
1986	virtuell globaler Speicher	Viren
1988	Laptop, Pasrsytec Supercluster (64 Rechenwerke)	OS/2
1989	Intel i860: Hochleistungs-Mikroprozessor mir RISC Basisprozessor	
1990	FDDI-1: 100Mbit/s Stadtnetz, ATM	
1991	Palmtop mit Schrifteingabe	Linux
1992	DEC Alpha: superskalarer Mikroprozessor 150 Mips	64bit
1993	Intel Pentium: superskalarer Mikroprozessor 60 Mips	WWW, Windows NT
1994		Java
1995		Windows 95
1996	Pentium MMX (Multimedia Zusatz)	
1997	Intel/Sandia ASCI Red überschreitet 1 TFlop/s (Multirechener)	
1999	10^9 Computer	
2000	Mobile Notebooks über Funk, Internet 10^6 Rechner	
2001	Intel Itanium 64 b	

2.10 Aussichten

Aus der Zukunftstagung des Münchner Kreises, 24.25.11.99 (etwas vereinfacht):

1. Fundamentale Technologietrends halten die nächsten 15 Jahre: Transistorfunktionen je Rechner: 1,4/a, Transistorfunktionen je Chip: 1,45/a, Speicher je Rechner: 1,44/a, Taktfrequenz: 1,37/a, CPU-Durchsatz: 1,53/Jahr, Kosten je Mips: $(1,45)^{-1}/a$, Lichtleiterdurchssatz: 2/a, größtes verlässliches Programm: 1,2/a (!)

2. Zahl der Computer (heute $14 * 10^9$): 1,48/a, Durchsatz des Internets (heute $10^{18} b/a$): 2,2/a

3. künftige Entwicklung nicht so sehr durch Technologie bestimmt, sondern durch

   - Mangel an Fachkräften
   - Mangel an Computer"bildung"
   - Mangel an Innovationskapital
   - Mangel an attraktiven Anwendungen
   - Komplexität (!)
   - Staatliche Rahmenbedingungen

4. Experten schätzen für etwa 2010 folgende Errungenschaften:

   - Multimedia in mobilen Geräten
   - 1000 Frequenzen (Tb/s) auf einem Lichtleiter
   - Internet überträgt in gesicherter Qualität (*"Quality of service"*)
   - Computerbedienung drastisch einfacher
   - 1 Gb Speicher-Chips
   - online-Sprachübersetzung (?)
   - Papier-Fachzeitschriften verschwunden
   - 50% der Europäer haben einen Personal Digital Assistent
   - 100 Computer je Haushalt in Europa
   - Programmierung in natürlicher Sprache

   und für 2015

   - papierloses Büro
   - Software-Agenten weit verbreitet

   und erst später

   - Quantencomputer
   - Computersteuerung durch Gestik, Blicke
   - vollautomatische Fabriken
   - Suchmaschinen mit qualitativer Selektion

2.11 Grundlegende strukturelle Freiheitsgrade von Rechnern, Klassifikationen

Ausführungsmodell: An der Hardware / Software- Schnittstelle sichtbare Datenstrukturen Operationen, Ablaufsteuerung, z.B. deklarativ, funktional, **prozedural**

hierzu noch: "Höhe" der Schnittstelle → *Bindungszeitpunkt*
universell ↔ speziell
Laufzeit / Platzbedarf

Ausschnitte
Nebenläufigkeit von Aufträgen nutzbar durch

- Zuweisung eines gesamten Auftrages an "seine" Funktionseinheit, wo er erledigt wird (Bode: "nebenläufige" Funktionseinheiten)
- Zerlegung (vorzugsweise seriell, *(pipelining)*) in Teilaufträge, die eigenen Funktionseinheiten zugewiesen werden

Beide Möglichkeiten sind kombinierbar !
Ebene der Nebenläufigkeit

- Programmläufe (*jobs*)
- Sub-Programmläufe (*tasks, threads*)
- von Neumann Operationen
- Operanden
- Mikrooperationen

Bindungszeitpunkt: Das Programm liegt primär in einer nicht-Maschinensprache (höherer Sprache) vor. Die semantische Lücke besteht in

Operationen
Datenstrukturen
Ablaufsteuerung
Ausschnittsbildung
Nebenläufigkeit ...

und kann überbrückt werden durch Abbildungen zu verschiedenen "Bindungszeitpunkten":

Programmierzeit
Übersetzungszeit (Präprozessor, Compiler)
Montage / Ladezeit
Laufzeit (Software"interpreter", Mikro"interpreter")

Meist werden alle Bindungszeitpunkte benutzt, teilweise die Lücke zu mehreren Zeitpunkten partiell überbrückt.

Bindung zur Laufzeit durch Mikrointerpreter → *Highlevel Language Machine* "Hohe" Schnittstelle.

Klassifikation

Strukturelle Klassifikation von Rechnern baut meist auf der Kollateralität auf.

Flynn (1972): Klassifizierte nach

- **SI,MI**: eine oder mehrere (durch die Rechnerhardware) gleichzeitige Programminterpretationen (*instruction streams*)
- **SD,MD**: eine oder mehrere gleichartige (durch Rechnerhardware) gleichzeitige Operationen auf Daten (*data streams*)

wichtig: **SISD** (Standard), **SIMD** (Vektor/ Feldrechner, **MIMD** (Multiprozessoren, Multirechner))

siehe Kap. 3.10: geeignet für (eine gewisse) Abgrenzung von weniger konventionellen Rechnerstrukturen.

Händler (1974) und Bode und Karl: ECS Erlangen Classification System.

Beschreibung von Rechnerstrukturen durch Tripel von Paaren: (k*k', d*d', w*w'):

Kollateralität von **Programminterpretationen**

- k Zahl der Leitwerke bzw. Prozessoren
- k' Zahl der (hardwareunterstützt) im Pipelining zusammen wirkenden Programminterpretationen (selten!)

Kollateralität von **Befehlsausfürungen** je Programminterpretation

- d Zahl der gleichen Rechenwerke je CPU
- d' Zahl der verschiedenen Rechenwerke je CPU

Kollateralität in Ausführung eines **Befehls**

- w Operandenwortlänge
- w' Zahl der je Befehl durchlaufenen Pipelinestufen

Beispiele:

Intel 8085 : (1,1,8*1) (vereinfachte Schreibweise: m statt m*1)
iPSC/2 Multirechner aus 128 Rechnern Intel80386: (128,1,32*1)
CM2 Feldrechner mit 64k Rechenwerken zu je 1 Bit: (1,65636,1)
Cray.1 Vektorrechner: (1,*12,64*14)
(vereinfacht geschrieben: *m statt 1*m, *14 erfasst nur ein Rechenwerk der Maschine (Gleitpunktpipeline); daneben u.a. Skalar"Prozessor".

Literatur zum heutigen Stand von ECS: [BODE A 90], [KARL W 93] Karl, W.: Parallele Prozessorarchitekturen, BI Wissenschaftsverlag, Mannheim 1993

2.12 Erscheinungsformen,Markt

Es kann hier nicht die ganze Entwicklung des Marktes und die ganze Breite der Erscheinungsformen von Computern dargestellt werden. Auf den folgenden Grafiken werden daher nur Beispiele aufgeführt:

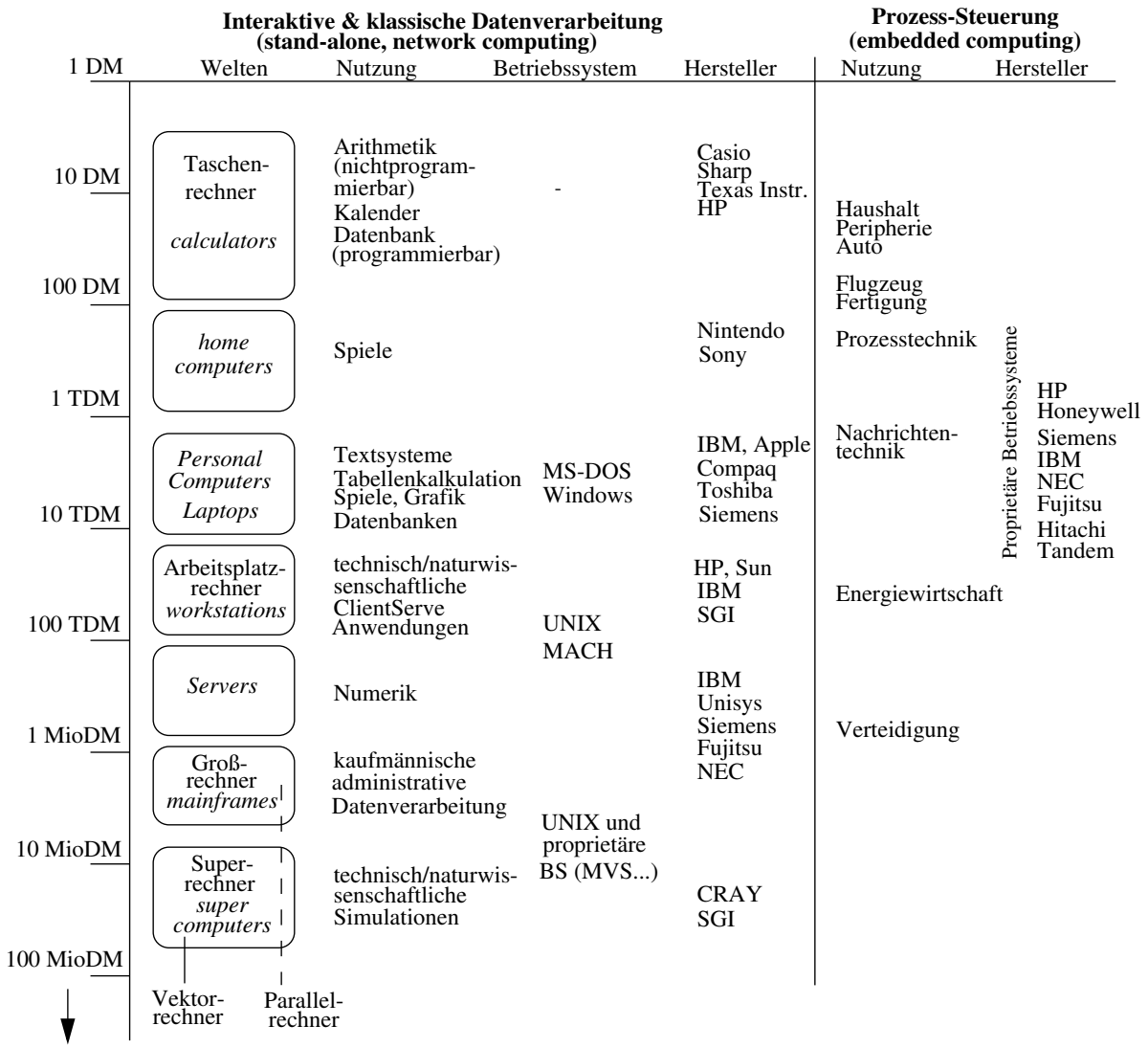


Abbildung 2.41: Erscheinungsformen

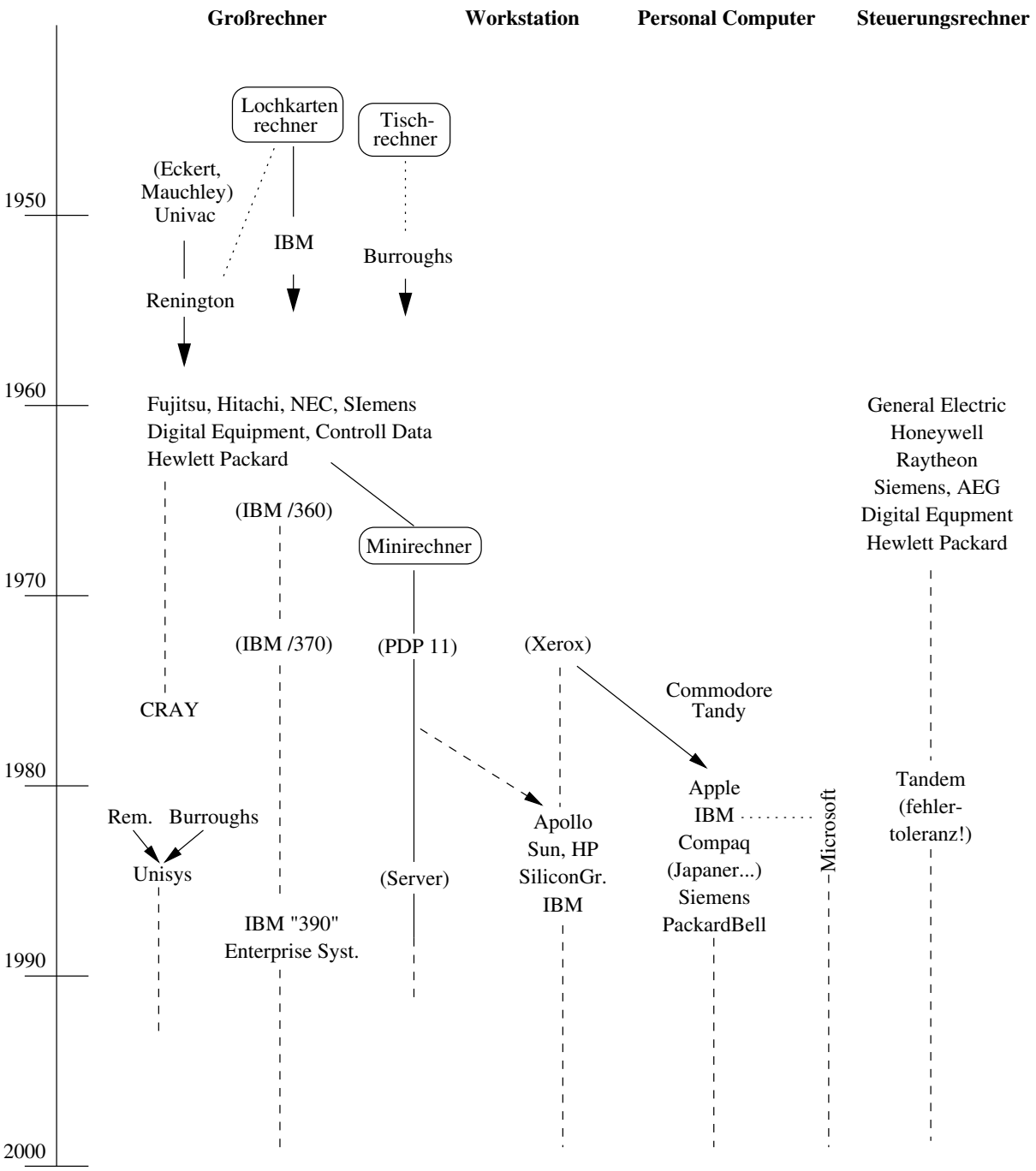


Abbildung 2.42: Markt

Kapitel 3

Zentralprozessoren

3.1 Beispiele: Mips R12000, Compaq Alpha, Intel Pentium, Itanium
3.2 Grundstrukturen
3.3 Befehle
3.4 Adressierung
3.5 Unterbrechung
3.6 Leitwerk
3.7 Rechenwerk
3.8 Operationensteuerung
3.9 Beschleunigung der Zentraleinheit: Cache-Memory, Pipelining
3.10 unkonventionelle Rechner

Literatur: Die ist das Lieblingsgebiet aller Autoren. Etwa (nach Alphabet):

[**BARBACCI SIEWIOREK 82**]: Barbacci, M, Siewiorek, D: The Design and Analysis of Instruction Set Processors, MC Graw 1982

[**BODE HÄNDLER 80,83**]

[**DALCIN M 96**]: Dalcin, M: Rechnerarchitektur Teubner1996 S.33-152

[**DASGUPTA S 89**] S. 101-322

[**GILOI W 91**]: S.39-341

[**HAYES J 88**]: S.176-475

[**HENNESSY PATTERSON 96**]: S.69-362

[**JESSEN E 75**] S.74-142

[**SIEWIOREK et al 82**]: S.62-92 und zahlreiche Beispiele

[**TANENBAUM A 90**]: S.31-75, 161-317, 431-486

[**VAN DE GOOR A 90**] S.41-219, 273-338, 391-472

[**UNGERER T 89**]: S.15-84

Für die behandelten Maschinen:

[**MIPS R 10000**]: MIPS R 10/12000 Microprocessor User's Manual, →www.mips.com

[**PENTIUM**]: Pentium Processor Family, Developer's Manual, Vol1: Pentium Processors, Intel, 1995ff

3.1 Beispiele

Die verschiedenen Optionen bei der Realisierung eines Prozessors sind voneinander abhängig. Es ist daher wichtig, diese an konkreten Beispielen zu sehen! Wir wählen dafür

- **MIPS R 12000** (Silicon Graphics): älteres RISC-Konzept (*reduced instruction set computer*); ähnlich wie SUN SPARC, Compaq Alpha, Hewlett Packard PA, IBM (e.a.) Power

- **Alpha 21264** (ursprünglich DEC (Digital Equipement Corporation), heute Compaq): jüngeres RISC-Konzept, Hochleistungsprozessor
- **Pentium III** (Intel): Beispiel eines CISC-Prozessors (*complex instruction set computer*), nach Verbreitung wichtigster Prozessor, belastet mit 30-jährigem Produkterfolg
- **Itanium** (Intel): kommendes erstes Beispiel einer vermutlich wichtigen Intel/HP Gemeinschaftsentwicklung (IA 64, *instruction set architechture 64*); compilerdefinierte Kollateralität (*VLIW: very long instruction word*)

3.1.1 Vorab: CISC und RISC

Die Hauptlinie der Rechnerfunktionalität von 1950 bis 1980 war gekennzeichnet durch

1. Aufstieg von wenigen zu vielen Datentypen, Datenformaten und Befehlen
2. Bildung von Komplexbefehlen für häufige Operationsfolgen, z.B. Schleifen, Prozeduraufrufe, Prozesswechsel
3. Annäherung an Semantik höherer Sprachen
4. Bildung von komplexen Privilegierungsstrukturen zu Gunsten des Betriebssystems
5. vielschichtige Ausschnittsbildung
6. bitsparende Befehlscodierung

Die Folgen der Punkte 1-6 waren

- vielfältig formatierte Befehle unterschiedlicher Ablaufstrukturen
- Compiler können die komplexe Funktionalität zur Codeerzeugung nicht wirkungsvoll einsetzen
- überkommene CPU's mit technologischem Stand von 1980 (40000 Transistoren je Chip) nicht integrierbar!

Wichtige Beispiele:

IBM /360 (1965) **/370** (1972) **ES9000** (1990): Familien großer Rechner, von 1965 bis 1985 marktbeherrschend im Datenverarbeitungs-Geschäft

DEC VAX (1978): Familie mittelgroßer Rechner, vermutlich bestentworfene Maschinenfunktionalität (TUM: Unterrichtsmaschine MI), strenge Orthogonalität der Teilfunktionen der Befehle; also z.B. Addition für alle Datentypen mit Register/ Speicherzellen/ Direktoperanden und raffinierten Adressmodifikationen auf allen Adresspositionen durch selbstbeschreibende Adressfelder.

Intel-Linie 8080 - Pentium: Durch Festhalten an schlechter Ausgangsposition und inkrementeller Erweiterung dasselbe äußere Bild einer komplexen Befehlsliste.

Ab 1978/83 Abkehr vom Trend Punkte 1-3 und 6. Neuer Ansatz: wenige, aber in Pipelining gut überlappbare, da im Ablauf gleich strukturierte Befehle:

RISC: reduced instruction set computer

in der einsetzenden akademischen Polemik nannte man die früheren Konzepte folglich

CISC: complex instruction set computer

Mit HP/Intel IA 64 (CPU Itanium von Intel) ab 2001 vom Compiler explizit festgelegte Parallelität mehrerer RISC-Pipelines, nebenläufige Befehle als Silben eines "langen" Befehlswortes.

RISC Entwicklung seit Ende der 70er Jahre aus sehr vielen, unterschiedlichen Motiven, unter Rückgriff auf Strukturmerkmale von Hochleistungsrechnern und(Rück-)Verlegung von Aufwand aus der Maschine in die Compilation. Prozessoren mittlerer bis hoher Leistung:
heute typisch für RISC:

- 80..180 Befehle desselben Formats und (weitgehend) derselben Ablaufstruktur
- daher einfach im Fließbandprinzip überlappbar (5..10 Stufen) [IBM Strech, 1961] möglichst je Takt eine Operation fertigstellen (d.h. $C_{CPU} \rightarrow f_{TAKT} * \#pipes$)
- Prozessor arbeitet effizient, da der Compiler dafür sorgt, daß die überlappten Funktionen hinreichend nebenläufig sind, durch Verschieben von nebenläufigen Befehlen im Code
- alle Operanden und Ergebnisse stehen in Registern (*load-store architecture*)
- keine Mikroprogrammierung
- große Anzahl von Registern (16..2048)
- damit sind Prozessoren ab 10..100 *10^3 Transistorfunktionen möglich, statt 100..2000 * 10^3 wie bei CISC-Prozessoren, "freie" Chipfläche kann für Ergänzungsfunktionen genutzt werden:
  - mehrfache Registersätze [RCA SPECTRA 1964] oder Register-*file* mit stackartiger Nutzung, um mit den Trägheitsnachteilen des Status fertig zu werden
  - Cache-Memory
  - Memory Management Unit
  - Zusatzrechenwerke (Gleitpunkt, Multimedia (Graphik)): komplexe Prozessoren mit RISC-Basisprozessor
  - mehrere Pipelines (*"superskalar"*)
  - vielstufige Pipelines (*"super-pipelined"*)

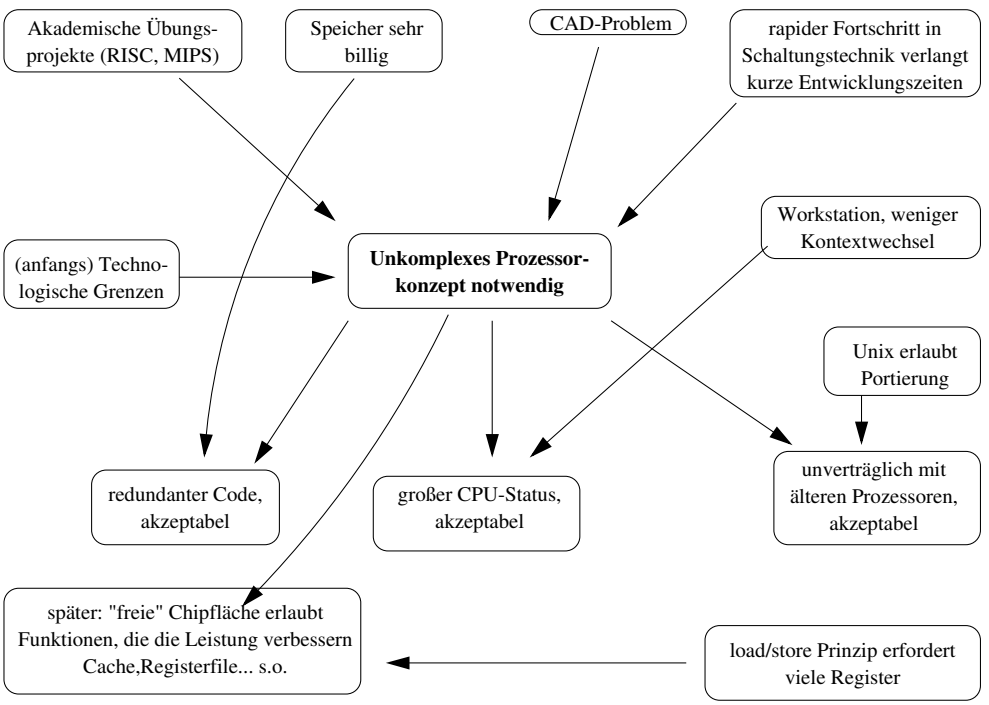


Abbildung 3.1: Motive für RISC

Rohe quantitative Bewertung (einer Pipeline)

Code-Redundanz:

- Befehle einheitlichen Formats brauchen mehr Platz; etwa um den Faktor 1,25
- RISC-Code enthält mehr Befehle für denselben Algorithmus (da unkomplex, da eingeschobene Null-Befehle): etwa um den Faktor 1,2

⇒ typisch 1,4..1,5 mal mehr Platzbedarf für Code (nicht Daten). Nachteilig für Speicher Kapazität/Kosten, Cache-Treffer-Verhältnis, Bus-Grenzdurchsatz...

Prozessor-Grenzdurchsatz RISC vs. CISC

RISC: etwa alle 1,2 Takte wird eine Operation fertig, d.h. :

$$C_{CPU} \text{ "peak rate"} = f_{TAKT} \qquad \text{"sustained rate"} = 1/1,2 * f_{TAKT}$$

Aber die Operationen sind (vgl. oben) etwa um den Faktor 1,2 unwirksamer als z.B. auf VAX oder Ähnlichen, also gemessen in VAX-MIPS:

$$C_{CPU-RISC} \approx \frac{1}{1,4} * f_{TAKT} = 0,7 * f_{TAKT}$$

CISC: Ein CISC-Prozessor (nicht überlappt) braucht je nach Mikroprogrammierungsschema im Mittel etwa 4..20 Takte, also

$$C_{CPU-CISC} \approx (\frac{1}{4}..\frac{1}{20}) * f_{TAKT}$$

Allerdings sind heute fast alle CISC-Prozessoren überlappt, holen allerdings (wegen der inhomogenen Befehle) weniger raus, etwa 1,5..3-fache Leistung. Beispiel bei Leistungsverdopplung durch Überlappung und durchschnittlichen 6 Takten pro Befehl:

$$C_{CPU-CISC} \approx 2 * \frac{1}{6} * f_{TAKT} = 0,3 * f_{TAKT}$$

Nachbemerkung: sehr roher Vergleich; unter Mängeln der Speicherhierarchie leidet RISC stärker; technologisch kann man im unkomplexen RISC höhere f_{TAKT} erreichen

heutiger Stand:

Alle neuen Maschinen sind als RISC-Prozessoren implementiert. Entweder

- ist ihre Funktionalität als RISC konzipiert (unser Beispiel MIPS R 10000 und fast alle seine Konkurrenten)
- oder ein RISC-Prozessor simuliert die CISC-Funktionalität (unser Beispiel Intel Pentium ist ein superskalarer RISC mit der Mikrooperations-Befehlsliste, der einen 80386 + MMX simuliert).

Bezüglich der Entwicklungslinien **Privilegierung** und **Ausschnittsbildung** hat die RISC-Entwicklung nicht mit dem Trend gebrochen!

neu: Steuerung mehrerer Pipelines mit Befehlssilben in vom Compiler festgelegter Kollateralität (→Itanium).

3.1.2 MIPS R 10000/12000

Stanford MIPS (1982): Akademisches Entwicklungsprojekt unter John Hennessy, 32b, 32(?) Befehle, 16 Register, 5-stufige Pipeline, 25000 Transistoren auf $50mm^2$ bei 2μ-Technik, 2Mhz Takt. Der Name kommt von "*Microprocessor with interlocked pipeline stages*": die Verriegelung wurde aber nicht vom Prozessor, sondern vom Compiler (durch Nulloperationen) sichergestellt.
Auf dieser Basis entstand MIPS-Computer Systems, eine Lizenzfirma, die an das MIPS-Konsortium Fertigungslizenzen abgab; Fertigung durch z.B. auch Siemens, NEC, Silicon Graphics. Nacheinander entstanden "Architekturen" (Prozessor-Funktionalitäten)

1986	MIPS I	R2000, R3000: 59 Befehle
1989	MIPS II	R6000
1992	MIPS III	R4400
1994	MIPS IV	R5000,R10000: 183 Befehle

inzwischen hat Silicon Graphics MIPS gekauft.

MIPS IV - Daten:

Byte/Halbwort/Wort/Doppelwort: 8/16/32/64b
Gleitpunkt einfach/doppelt: 32/64b (IEEE-Standard)

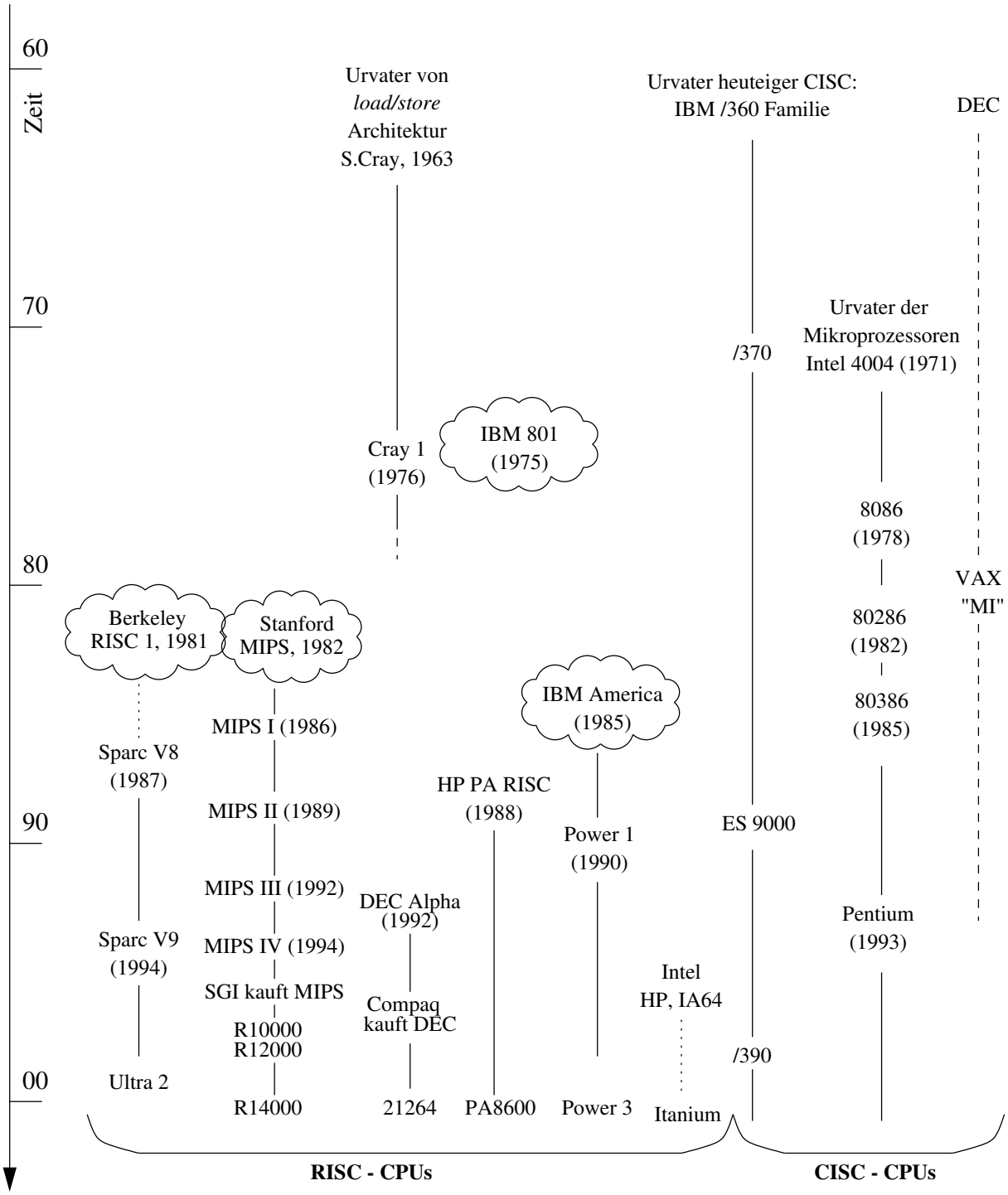


Abbildung 3.2: RISC-Szene mit CISC-Seitenblick

MIPS IV - Befehle:

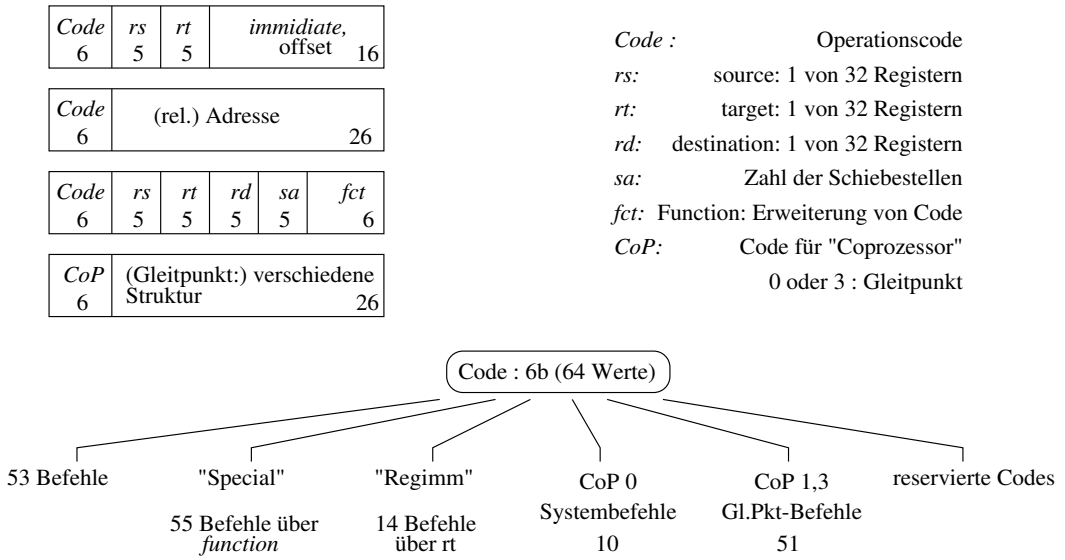


Abbildung 3.3: 183 Befehle des R10000

Pipelining Schema des R10000

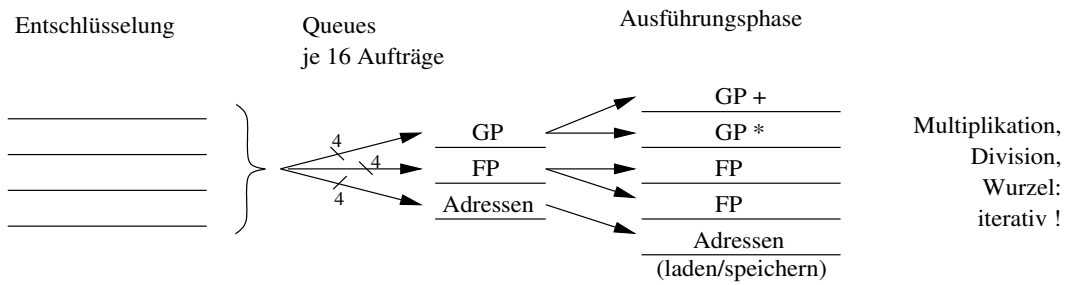


Abbildung 3.4: Pipelining-Schema des R10000

Entschlüsselung:	Befehl holen
4 pipes	Befehl entschlüsseln und Abhängigkeiten klären logische Register auf physische abbilden (anhand von *activ list*)
Queues:	Befehle können sich überholen !
FP-Ausführungsphase:	*pipe* 1: Add./Subtr., shift, Vergleich : 3 Stufen *pipe* 2: Add./Subtr., Multiplikation, Division: 3 Stufen
GP-Ausführungsphase:	*pipe* 1: Add., Subtr., anderes: 5 Stufen *pipe* 2: Multiplikation / Division / Wurzel : 5 Stufen
Adressausführung:	laden/speichern, Übergang von 44b virtuellen Adressen auf 40b Maschinenadressen: 5 Stufen

Weiterentwicklungen R12000 und R14000, (R16000, R18000) sind eher technologischer Art, bei mäßiger Erhöhung der Taktfrequenz:

R10000	300 MHz
R12000	390 MHz (Jahr 2000)
R14000	450 MHz geänderte Speicherschnittstelle (2000/2001)
R16000	600..800 MHz 2x64kB Cache (2001)

Mips bleibt gegen Entwicklung zurück.

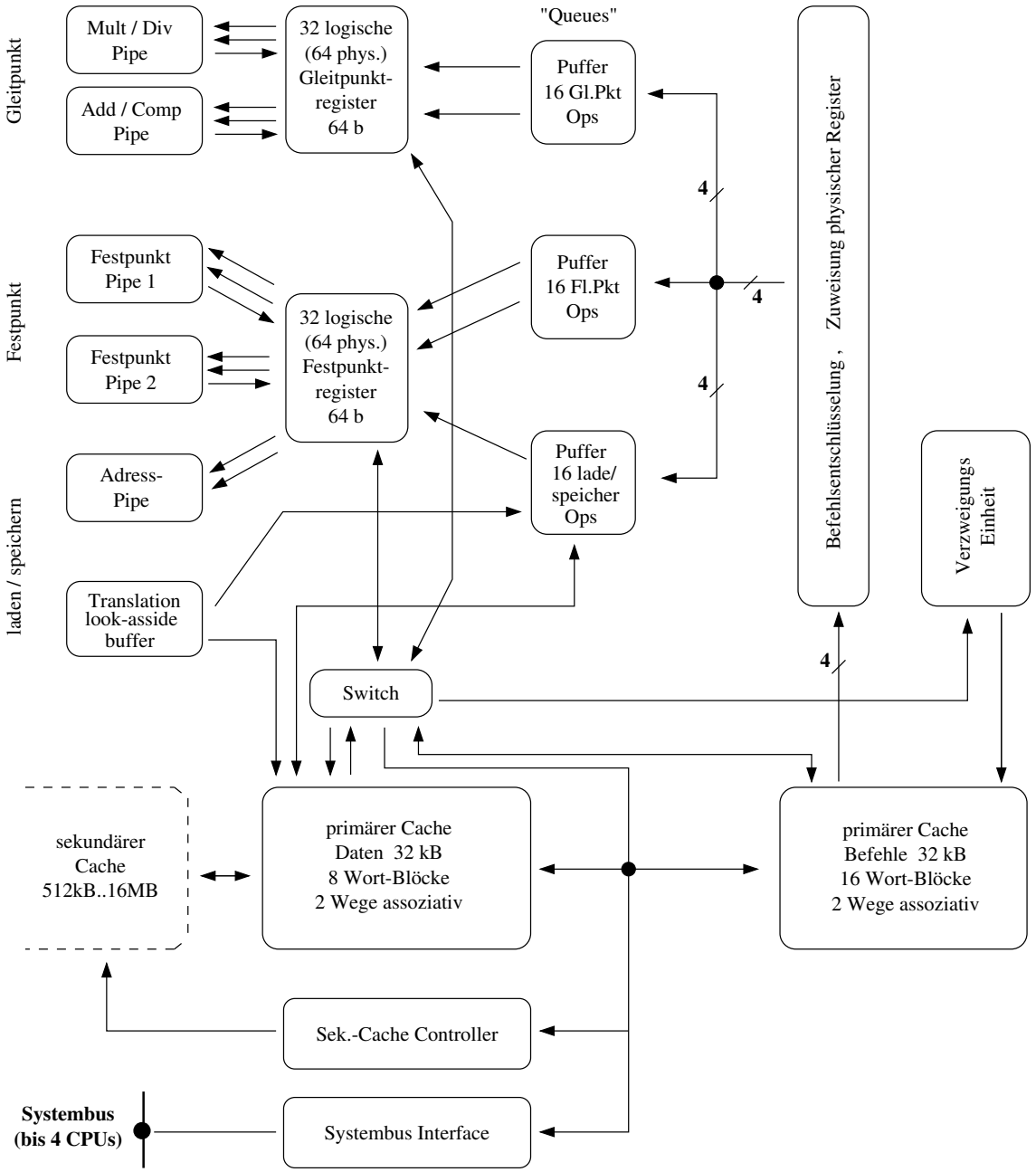


Abbildung 3.5: Aufbau des R10000

3.1.3 DEC / Compaq Alpha

DEC wurde inzwischen von Compaq erworben, Prozessoren werden von Samsung produziert und vertrieben; Fertigung auch bei Intel (?)

DEC Alpha sind interessant

- da meist schnellste Mikroprozessoren, dabei strukturell einfach
- da Architekturkonzept von 1992 (alle anderen sind an ältere Konzepte gebunden)

Gliederung des Beispiels:

- Alpha-Architektur
- Implementationen 21064, 21164, 21264, 21364, 21464 (5 Generationen)

- 21264 (seit 1998)

Größeres Beispiel: Digital Equipement Alpha-Architektur
Literatur: Digital Technical Journal Vol.4 No.4 1992, darin : R.L.Sites: Alpha AXP Architecture
laufende Berichte in der Zeitschrift Microprocessor Report

Ziele der Entwicklung für Alpha

1. Funktionalität (*architecture*) 25 Jahre brauchbar (IBM /360 ist 39 Jahre alt!)
2. Faktor 1000 an Leistung bis 2015, davon je ein Faktor 10 durch Taktfrequenz, mehrere Prozessoren und durch CPU-Durchsatz/Takt (*instructions per cycle*)
3. DEC OpenVMS, DEC OSF/1 Unix und Windows NT als Betriebssystem unterstützen (WindowsNT 1999 von Compaq aufgegeben)
4. unproblematische Portierung von VAX- und MIPS-Code

Funktionalität (*architecture*) der Alpha AXP

Großer Adressraum von 2^{64}B. Offene Basisadressen in Integer-Registern, relative Adressen bis 2^{16}B, aber Sprünge bis 2^{21}B
RISC-Ansatz: **konstantes Befehlsformat** 32b, ***load-store*-Prinzip**, einheitliche Ablaufstruktur

Unterstützung der Parallelarbeit in der CPU:

- fast kein vom Benutzerprogramm nutzbarer Prozessorstatus außer Unterbrechungspriotität, Rundungsmodus, Sperradresse (für exklusive Lese/Schreibfolge), Befehlszähler und den Registern: aber nur 32 Festpunkt/Adress- und 32 Gleitpunkregister: wenig bei "10 Befehlen zugleich" (Ziel des 21064)
  also auch keine Condition Codes (*flags*), kein Stringregister, Multiplikator-Quotientenregister o.Ä.
- fast kein Prozessorstatus, über dessen Benutzung schädliche Verkopplungen auftreten können, die (einfach überprüfbaren) Register ausgenommen.
- keine heimlichen Verkopplungen der Befehlsausführungen, wie
  - Skip-Befehle
  - delayed branch (müßte auch bei hoher Paralellität und im Verhältnis zur Taktzeit längerer Cache-Zugriffszeit viele Operationen betragen!)
  - keine zeitgenauen arithmetischen Alarme; stattdessen wird arithmetischer Alarm erst bei Operationsende bekannt; d.h. daß andere Operationen schon das falsche Ergebnis übernommen haben!
- einfache Alternativen sind ohne Sprungbefehl programmierbar durch bedingte Transportbefehle (z.B. MIN{a,b})
- bei bedingten Sprüngen wird vorausgesetzt, daß
  - Rückwärtssprünge meist vollzogen werden
  - Vorwärtssprünge meist nicht vollzogen werden

  Der Compiler/Programmierer kann bei modifizierten Sprungadressen den häufigsten Wert angeben.
- mit prefetch-Befehlen kann man vorabladen in das Cache-Memory erzwingen.

! Die späteren Implementationen 21164 ff. benutzen raffinierte automatische Sprungvorhersagetechniken

Datenformate im Grundsatz nur 64b Festpunkt und 64b Gleitpunkt (IEEE), aber auch VAX-G-Gleitpunktformat (→RA I). Daneben (übergangsweise) auch 32b-VAX-Formate. Bytes nur als 8B-64b-Wörter.

Befehle: 164, alle 32b-Format:

33	Laden/Speichern Festpunktregister, Bytemanipulation
8	Laden/Speichern Gleitpunktregister
2	effektive Adresse laden
40	Festpunktoperationen mit bedingten Transporten
14	Verzweigung mit Festpunktbedingung
6	Verzweigung mit Gleitpunktbedingung
47	Gleitpunktoperationen mit bedingten Transporten
14	Systembefehle (u.a. Einsprünge in "Pal Code")
x	seit 21164 zusätzlich wenige Befehle zur Video(de)kompression

Es gibt 4 verschiedene **Befehlsstrukturen**:

R-R-R Typ	für Dreiadressoperationen
R-R-D Typ	für laden/speichern (D: relative Adresse)
R-D Typ	für relativ-Verzweigungen
Call Pal Code	für privilegierte Funktionen

Privilegierte Funktionen

über "Pal Code" (Mikroprogramm/Hardware/Programm, je nach Implementation) definierbar. Liefert effizienten Unterbau für Open VMS, OSF/1 Unix und Windows NT.
Anpassbar an erwartete neuartige Betriebssystemstrukturen (Echtzeit, Sicherheit)
Seitenadressierung: vielfache Prozessadressräume von 2^{64}b, mit Adressraum-(Prozess-)nummern. Exakte Alarme, Seiten von 8, 16, 32 oder 64 kB Größe

Implementationen der Alpha-Architektur

Typ	21064	21164	21264	21364	Intel PIII
seit	92	2.Q.97	3.Q.99	01?	3.Q.98/00
Takt[Mhz]	200	600	700	1000?	1000
primär Cache [kB]	8D/8I	8/8	64/64	512/512	16/16
sek. Cache	offchip	onchip 96	offchip	onchip 1,5M	offchip
Issue	2	4	4		3*86 Befehle
Pipelinstufen	7/10	7/10	7/9		12/14
Operationen überhold.		6 laden	80		40 ROps "out of order"
Register renaming	-	-	48/41		40
Verzweige Tafel		2k*2b	4k*9b		512
Transl. Lookaside	12D/12I	48/48	128/128		32I/64D
C_{HSp} [MB/s]		400	2660	6000	1060
Gehäuse # pins	431	499	588	≈1000	370
IC-Prozess	0,75μ3M	0,35μ4M	0,25μ6M	0,18μ	0,18μ3M
Chip [mm^2]	234	209	205	350	106
# Transistoren	1,7M	9,3M	15,2M		24M
Herstellkosten		125$	160$	380$	40$
Leistung [W]	30	25	75		23
SPEC-Marks		19,2/26,6	35/55?		47/32

Besonderheiten des Alpha 21264:

- die 80 Festpunktregister sind doppelt vorhanden. Jedes "Cluster" arbeitet mit zwei Pipes zusammen; Abgleich der Registerinhalte braucht einen Takt(!). Operationen werden den Clustern dynamisch zugewiesen. Etwa 1% Leistungsverlust (in Takten), aber damit höherer Takt möglich!

- primäre Caches haben 2 Takte (Brutto-)Zugriffszeit, aber Grenzdurchsatz 128b (1Zugriff)/Takt; wegen der hohen Zahl von gleichzeitig möglichen Operationen verliert man dadurch trotzdem nur 4% Prozessordurchsatz

- Verzweigungsvorhersage nach zwei konkurrierenden Verfahren; ausgewählt wird das bisher bessere Verfahren. Mittlerer Verzug bei falscher Vorhersage 11 Takte! 6% aller Verzweigungen werden bei SPEC-Benchmark falsch vorhergesagt!

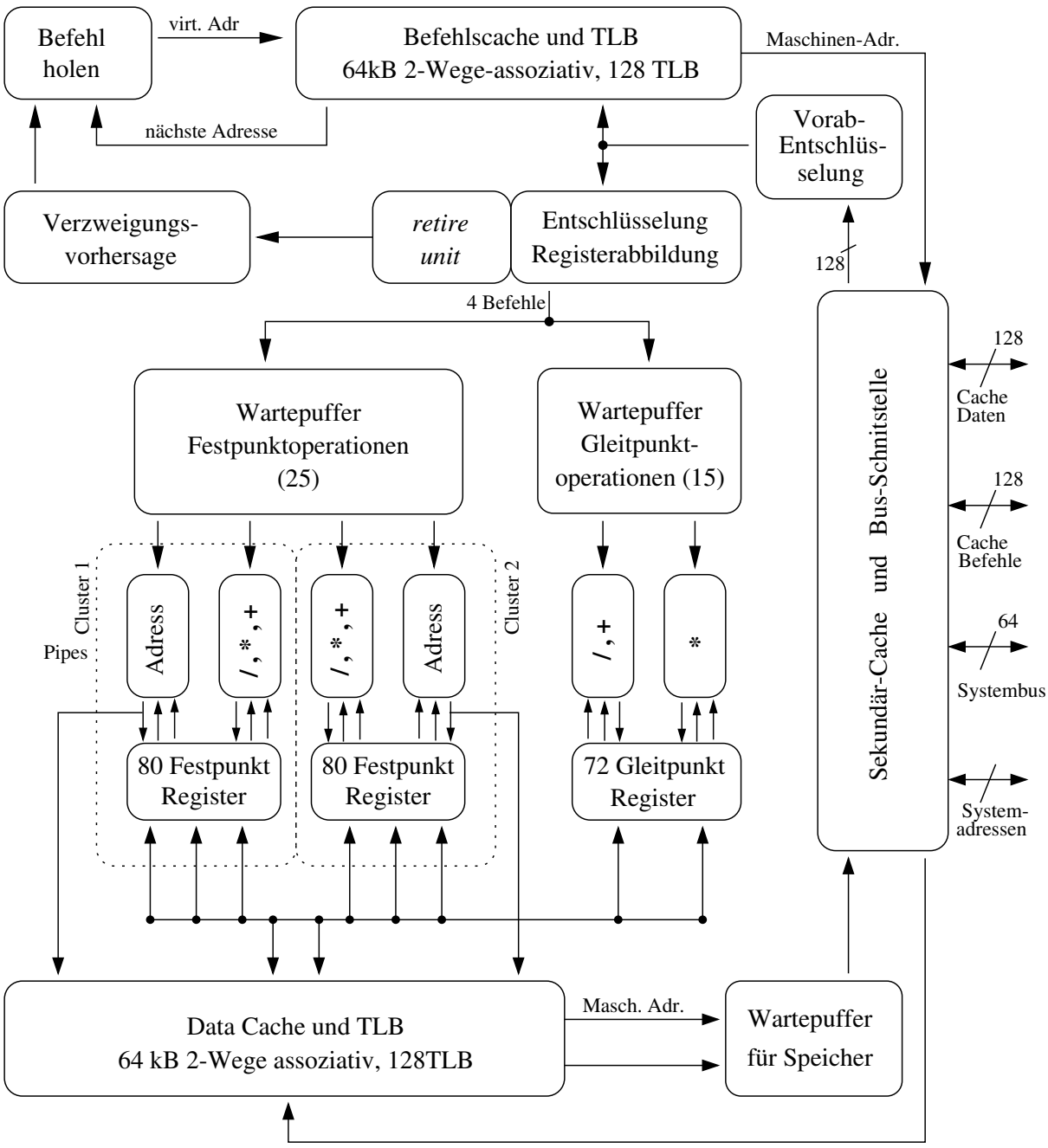


Abbildung 3.6: Aufbau des Alpha 21264

Name	Jahr	Technologie	Transistorfkt.	Takt [MHz]	Pins	Leistung [W]	Wortlänge	Datenbus	Register	Befehle	Durchsatz	Adressierung	Sonstiges
4004	71	p-MOS 10 µ	2000										Taschenrechnerchip
8008	72	p-MOS	3000	0,8	18	0,4	8	8	7x8b unsym.	66	Intel:"1"	16k	Display Chip (nicht 4004-kompatibel
8080	74	n-MOS *depletion load*	6000	3	40	1	8	8	7x8b unsym.	111	Intel: "10"	64k	Universalchip
8085	76	n-MOS *depletion load*	6500	5	40		8	8	7x8b unsym.	113		64k	
8086	78	*high density* n-MOS *dep.l.*	29T	8	40		16	16	8x16b unsym.	133	Intel: "100"	4x 64k in 1 MB	Adressmodifikation, Segmenierung nicht ganz 8080 kompatibel
8088	80						16	8	8x16b unsym.	133		4x 64k in 1 MB	IBM PC (1981)
80186	82						16	16					zusätzlich E/A Steuerung auf Chip
80286	82	n-MOS	130T	10	68	3,6	16	16		140	Intel: "300"	protected 1GB	Protected Mode mit Schutzmechanismen, Real Mode - PC AT
80386	85	CH-MOS III 1,2 µ	275T	16.. 33			32	32	8x32b sym.	151	3,6 Mips	real, prot. virt. 64TB	Seitenadressierung MMU auf Chip
80386sx	86	CH-MOS III 1,2 µ	275T	16.. 20			32	16	8x32b sym.	151		"	Austausch für 80286
80486	89	CH-MOS III1,2µ IV 0,8	1,2M	16.. 66	168		32	32	8x32b sym.	157	25 Mips (25MHz)	"	Gleitpunktrechenwerk, MMU 8kB Cache auf Chip
Pentium	93	BiCMOS 0,8 µ	3,1M	66	272	13	2 x 32	32	8x32b sym.	163	60	"	2 x 8kB Cache, 2 Festpunktpipes, 1 Gleitpunktpipe, Sprungvorhersage
Pentium	95	BiCMOS 0,35 µ		200	272		2 x 32	64	8x32b sym.	220?	120 ?	"	
P. MMX	96	BiCMOS 0,35 µ	4,5M	200	272	16	32, 64	64	+MMX	277	140 ?	"	MMX Multimediabefehle
Pentium	96	BiCMOS 0,28 µ	7,5M	200	528	30	32, 64	64	8x32b renam.	277	300 ?	"	15 Befehle out of order möglich 2 x 32 kB Cache
(pro) II	98	BiCMOS 0,28 µ		450	528		32, 64	64	8x32b renam.	277	500 ?	"	15 Befehle out of order möglich 2 x 32 kB Cache
P. III	99	BiCMOS 0,25 µ	8,2M	450	370	25	32, 64	64	8x32b renam.	347	550 ?	"	40 Befehle out of order möglich weitere MM-Befehle
P. III	00	BiCMOS 0,18 µ	24M	1000	370		32, 64	64	8x32b renam.	347	1100 ?	"	

Abbildung 3.7: Intel Prozessor-Familie

3.1.4 Intel Pentium

Intel wurde 1968 unter Beteiligung von IBM als Halbleiterhersteller gegründet, Unternehmensziel u.a. second source für IBM

1971: erster Mikroprozessor Intel 4004
1981: wählt IBM den Intel-Prozessor 8088 als Basis für seinen PC, begründet damit Intels überragende Stellung am Prozessormarkt

Intel ist Bauelementehersteller, nicht Sytemhersteller (Ausnahme: Parallelrechner)
Intel-Prozessoren sind seit 8008/8086 auf Eigentümlichkeiten festgelegt, die hohe Leistung nur unter hohem technologischen Aufwand erlauben. Das bezieht sich vor allem auf die nicht-orthogonale Verwendung der sehr wenigen (8) Festpunkt/Adressregister, die Befehlsstruktur und die Adressierung.

Datentypen:

Festpunktzahlen (8, 16, 32b)
Gleitpunktzahlen (32,64, 80b)
Dezimalstrings (1 oder 2 Ziffern je B)
Bitfelder (bis 32b) Bitstrings, Bytestrings (bis 2^{32} Elemente)
Adressen (32, 46b)

Register (virtuell! teilweise auf 40 physische abgebildet)

8 Allzweckregister zu 32b, auch als 8 Register zu 8, 16b verwendbar (aus Rücksicht auf ältere Prozessoren)

8 Gleitpunktregister zu 80b
MMX Register, weitere Register

Prozessormodi:

protected: Benutzermodus mit 16- oder 32 Bitadressen, prozessweise einstellbar, 8086-virtuelle Maschine

real address: Prozessor simuliert einen Intel 8086, 80286, 80386

Befehle: Wegen der vielen Varianten gibt es verschiedene Zählungen:

Systembefehle	32
Transporte	25
Sprünge, Aufrufe Programmunterbrechungen	9
Vergleiche, Tests, bed.Sprünge, bed. Transporte	18
Schleifen	4
Addressierung	5
andere	8
Arithmetik:	
Festpunkt	8
Dezimal	6
Gleitpunkt	97
Multimedia (57 MMX, 70 PIII)	127
Flags, Conditions	9
Konversion	5
"Logik" (und,oder...)	4
Shift, Rotation	16
Strings	7
Bits	6
gesamt	393

Die große Befehlsliste, die komplexe Struktur und Zugeständnisse an frühere Prozessoren belasten die Codeerzeugung, den Platzbedarf, die Laufzeit und die Herstellkosten des Prozessors.

	Intel (ohne mobile PII)				**HP PA**	**Compaq Alpha**
	32 b - Architektur			64 b - Architektur		
	(Pentium Pro) hohe Leistung Server, Workst.	(Pentium II) hohe Leistung PCs	geringe Leistung Pcs	IA 64 hohe Leistung		1997: 21164 600 MHz 0,35 µ
1998	Xeon 400MHz 0,25 µ D	P II 400 MHz 0,25 µ D	Celeron 300 MHz 0,25 µ C			
	Xeon 450 D	P II 450 D	Celeron 330 M			21264 575 MHz 0,35 µ
1999	Xeon 500 T	P III 500 K	Celeron 330 M	(Merced auf 2000 verschoben)	I: 8500 440 MHz 0,25 µ	
	Xeon 700 T 0,18 µ	P III 600 K 0,18 µ	Celeron 400 M			
		P III 1000 / 1150 0,18 µ CM	Celeron 700 M	(Itanium)	8600 560 MHz	21264 700 MHz
2001	zusätzliche Befehle	P III > 1000MHz P IV 1400 W	Celeron <800 M	Itanium McKinley 1 GHz 0,18 µ		21364 ?
2002				Madison 0,13 µ	2003: 8900 1,2GHz	

Intel Prozessor Layouts: T,K: 70 neue Befehle (FP-SIMD "MMX")

C: Covington D: Deschutes T: Tanner K: Katmai M: Mendocino CM: Coppermine W: Williamette

Abbildung 3.8: Intelprozessoren und einige Konkurrenten (*road maps*)

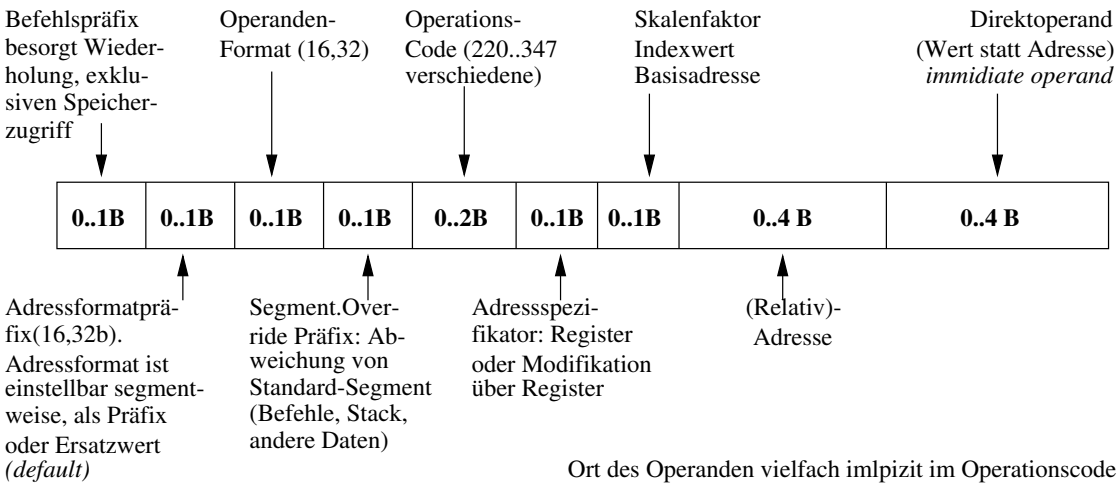


Abbildung 3.9: Aufbau Pentium II (ähnelt stark Pentium III)

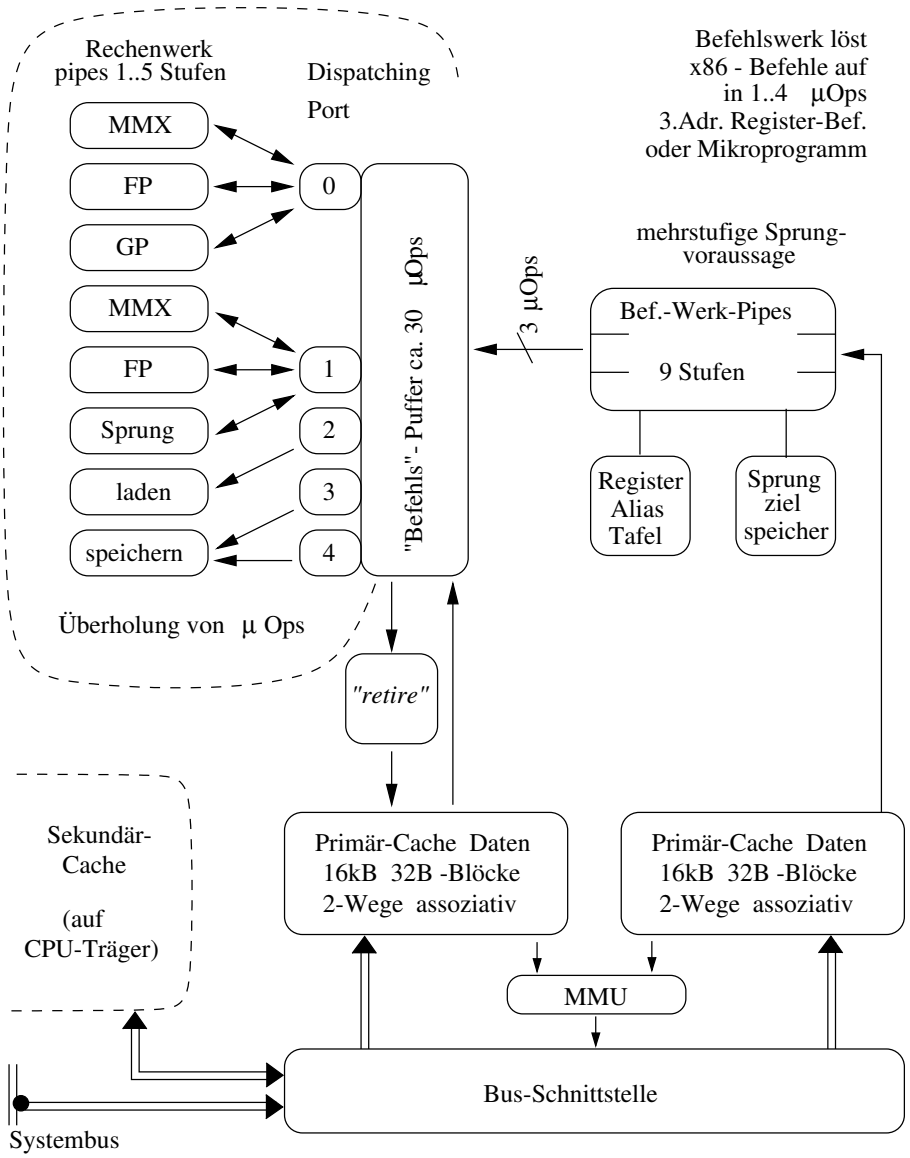


Abbildung 3.10: Aufbau Pentium II (ähnelt stark Pentium III)

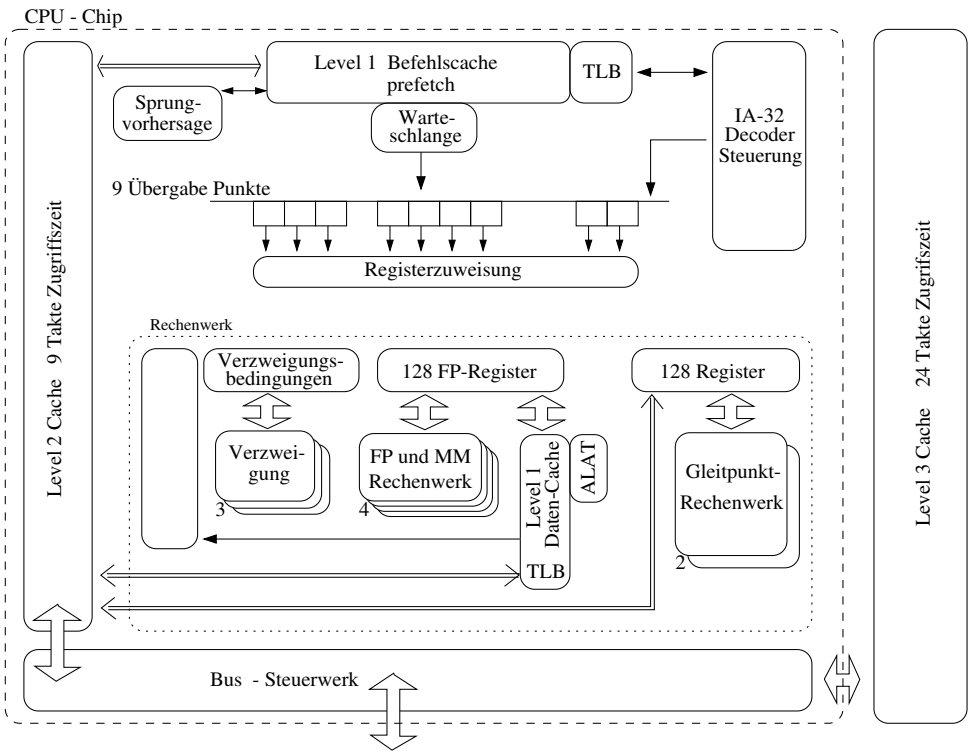


Abbildung 3.11: Intel Itanium CPU

3.1.5 Intel IA-64 (*instruction set architecture 64bits*) und Itanium-CPU

Intel hat mit HP seit einigen Jahren IA-64, die Funktionalität einer 64b-CPU-Generationsfolge veröffentlicht, um im 64b-CPU Markt (MIPS, Alpha, Power,...) claims abzustecken und Softwarefirmen Vorarbeiten zu ermöglichen. Die Pentium-Funktionalität (16b, 32b) ist in IA64 enthalten.
IA-64 enthält insbesondere Vorkehrungen mit denen der Compiler die Parallelität fördern, einschränken und optimistische Parallelitätsstrategien überprüfen kann:

- Prefetch-Befehle(*hint*)
- Sprunghinweise (Ziel, Kriterium)
- Überprüfung, daß vorgezogene Leseoperationen nicht durch Schreiben ungltig werden
- bedingte Ausführungen (statt Sprung, bzw. Verfolgung von Alternativen nach bedingten Sprüngen)
- Durchsetzung von Sequenz-konsistenten Lese/Schreiboperationen an Interaktionspunkten in Mehrprozessor-Programmläufen

IA-64 soll hohen CPU-Durchsatz auch ohne aufwendige Parallelitätssteuerung ermöglichen, d.h. Abkehr von den *"out of order"* Implementationen erlauben. Die Planung der Parallelität wird dem Compiler überlassen, der nebenläufige Befehle in ein Wort packt: *VLIW: very long instruction word* bzw. *EPIC: explicit parallel instruction computing*; Befehlswort 128b nimmt drei 41b-Befehle und ein Kennzeichen auf.

Adressen 64b ($16 * 10^{18}$ B), flacher Adressraum.
128 physische Register, davon

- 32 "global" für globale Daten
- 96 "rotating" für Aufrufsätze (zyklische Nutzung mit Überlauf in Speicher): Stack mit Basisregister

32 Register adressierbar (virtuelle Register)

Festpunkt 8/16/32/64b

Gleitpunkt 64/82b IEEE754/752
SIMD ("MMX")-Befehle
Unterstützung für überlappte Ausführung von Schleifen-Durchläufen (*"Software-pipelining"*)

erste Implementation von IA-64: Itanium CPU

Chip enthält CPU mit Cache1 und Cache2 (800Mhz, insgesamt 25 Mio. Transistor-Funktionen, bis 20 Operationen/Takt (Nutzung von SIMD), 0,18μ-Technologie) auf Chipträger (*cartridge*) mit Cache3 (4/8MB, 295Mio Transistorfunktionen).
Mehrprozessorkonfigurationen bis vier CPUs, 10stufige Pipeline (einschließlich Befehlswerk), 64 GB Hauptspeicher (max), über 4,2 GB/s Bus-Schnittstelle

Die Fertigstellung des Itaniums wurde mehrfach verzögert; Produktionsanlauf (Stand Sommer 2000): 1.Quartal 01. Nachfolger: Mc Kinley Herbst 2001 →schneller, Madison 2002 0,13μ, > 1GHz Taktfrequenz.

3.2 Grundstrukturen

Wir kennen schon:

- Interpretation, Prozessor ist Interpretationswerk
- von Neumannsche Interpretation
  - Leitwerk, Rechenwerk
  - Struktur und Ablaufsteuerung des Maschinenprogramms
  - Leitwerkszyklus, Befehls"zähler", Befehlsregister
- maschinennahe Programmierung der MI [BORGHOFF et al 86] der VAX verwandt

3.2.1 Gliederung des Prozessors auf verschiedenen Interpretationsebenen

3.2.2 Aktueller Ausschnitt im Prozessor

3.2.3 Privilegierung

3.2.4 Datentypen

3.2.1 Gliederung des Prozessors auf verschiedenen Ebenen

3.2.2 Aktueller Ausschnitt des Prozessors

soll der Prozessor alle erforderlichen Werte in jedem Schritt der Berechnung aus dem Speicher holen und jedes Ergebnis dort ablegen, dann

- Prozessor langsam, da abhängig vom (stets zu langsamen) Speicher
- lange Befehlsworte (→ 3.3)
- nach jedem Schritt kann auf ein anderes Programm übergegangen werden, ohne daß der aktuelle Auschnitt zurückgeschrieben wird !

oder soll der Prozessor einen aktuellen Ausschnitt aufbauen, bzw. soll der Compiler das veranlassen (z.B. in Registern), dann

- Prozessor schnell
- kürzere aber mehr Befehle (→ 3.3)
- träger Programmwechsel

Jede Operationsüberlappung (z.B. in RISC!) vergrößert den im Prozessor vorgehaltenen Ausschnitt: Prozessor schneller, aber träger bei Programmwechseln

Eine andere Bezeichnung für die Daten (Operanden, Ergebnisse, Befehle, interne Steuerdaten), die der Prozessor in sich aufbewahrt: Status
Im Laufe der Rechnerentwicklung eindeutiger Trend zu **mehr** Status, schlechte Reaktions- (Programmwechsel-) Zeit teilweise dadurch erträglich, daß Schaltzeiten, damit auch die Umschaltzeit kürzer werden.

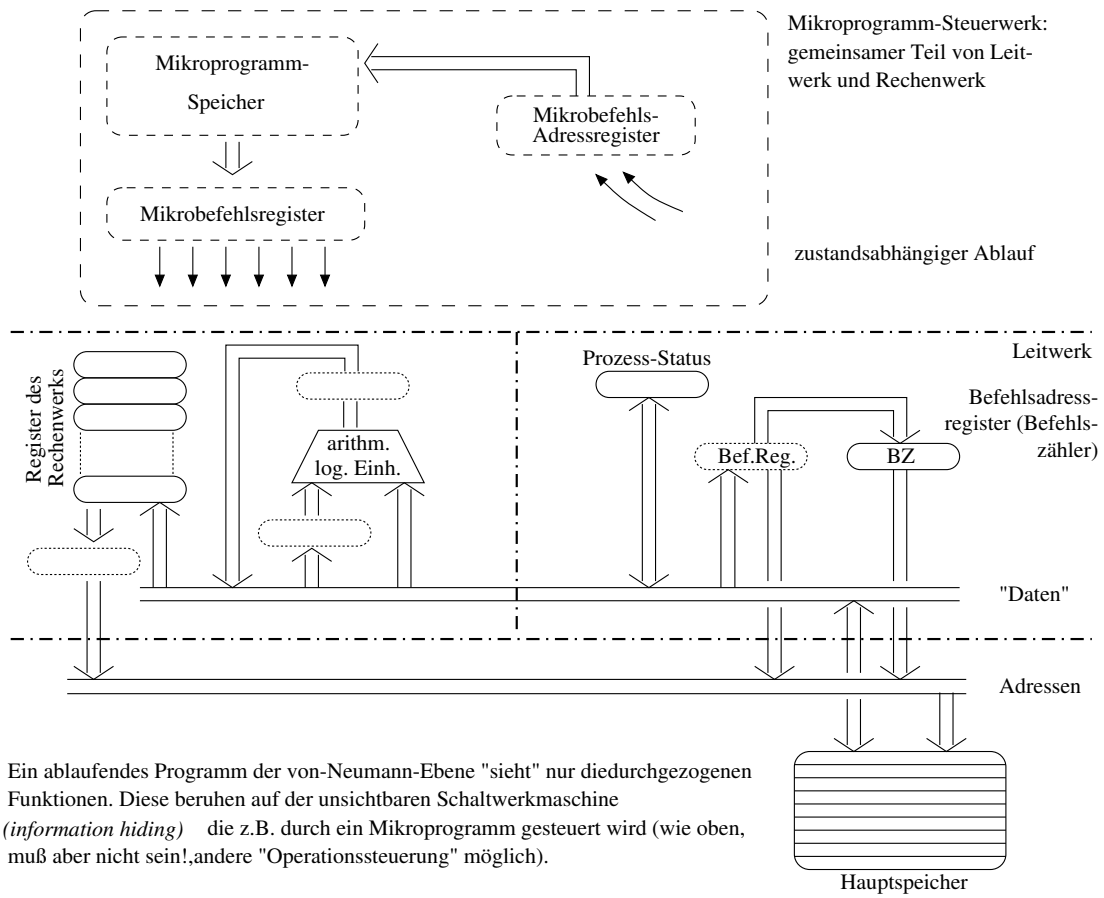


Abbildung 3.12: schlichtes Beispiel einer Prozessorgliederung

3.2.3 Privilegierung

Nur ein Benutzer, dem das Rechensystem (=Rechenanlage und Grundprogrammausstattung) und alle Daten vollständig und alleine gehören, darf auch alles anstellen, was programmierbar ist! Aber bereits wenn die Grundsoftware nach ihm auch anderen dienen soll, umso mehr, falls

- Daten fremder Benutzer im System (Programme, Werte) z.B. bei Mehrprogrammbetrieb, oder
- fremde (externe) Prozesse vom Rechner gesteuert werden können,

muß man **den Benutzer einschränken !**

Mehrprogrammbetrieb *multiprogramming*
ist Simultanbetrieb von Rechneraufträgen verschiedener Benutzer in der Zentraleinheit, d.h. kollaterale Ausführung. Bei Verdrängung aus dem Zentralprozessor kann ein Auftrag im Hauptspeicher bleiben (z.B. für 2..5 Aufträge) oder aus dem Hauptspeicher ebenfalls verdrängt werden (*swapping*: verdrängen/laden).

Gründe für Mehrprogrammbetrieb:

- zeitkritische Aufträge bevorzugen
- "gerechte" Wartezeiten (bei Zuteilung des Prozessors in Zeitscheiben an die wartenden Aufträge kann man Proportionalität von Warte- und Bedienzeit erreichen)
- Auslastung von Anlagenteilen aufrecht erhalten (Rechenphasen eines Ein/Ausgabe-intensiven Prozesses in die Rechnenzeit eines rechenintensiven Prozesses einschieben)

notwendige **Einschränkungen** des Benutzerprogrammlaufs:

- gewisse Unterbrechungswünsche sind vom Benutzer nicht unterdrückbar und führen zur Behandlung ins Betriebssystem; damit Prozessorentzug möglich bei Zeitüberschreitung oder Verstößen

Beispiel: DEC VAX: alle
Beispiel:R10000, Pentium, andere Rechner: alle bis auf z.B. arithmetische Alarme

- der Benutzerprogrammlauf kann nicht alle Befehle benutzen, bzw. die Befehle haben eingeschränkte Wirkung. Dadurch bleiben kritische Teile des Rechners unzugänglich (Ein/Ausgabe, Uhr/Wecker, Speicherbereiche); →Modi
- Die Maschine prüft die Rechte des Benutzerprogramms bei der Ausführung seiner Befehle. Die Rechte sind vom Benutzer nicht änderbar.
  Beispiel: Seitentafeln; aktuelle Priorität

Beispiel R 12000

- 32 verschiedene Unterbrechungswünsche, von denen das Benutzerprogramm nur 3 unterdrücken kann
- Adressumsetzung von Prozess- auf Maschinenadressen über Seiten/Kacheltafeln; damit auch E/A-Zugang einschränkbar!
- seitenweise Zugriffseinschränkung (lesen, schreiben, ausführen)
- 31 Sonderregister des Prozessors, dem Benutzerprogramm unzugänglich
- 30 Befehle sind dem Benutzerprogramm unzugänglich, Benutzung führt auf Prozessorentzug

Beispiel Pentium II

- raffinierte Adressraumstrukturierung für jedes Benutzerprogramm in "Segmente"
- mehrstufige Schutzfunktionen

Modi

Heutige Maschinen benutzen mindestens zwei Interpretationsmodi für Programme:

Benutzerprogramm:	Benutzermodus (*user mode*)	Das Benutzerprogramm wird nur im Benutzer-
Betriebssystem:	Systemmodus (*system mode*)	

modus interpretiert. Das Betriebssystem wird

Betriebssystemkern:	im Systemmodus interpretiert.
Systemprozesse	(verzögerbare Dienste): im Benutzermodus oder anderen Modi interpretiert.

Beispiel

R10000:	3 Modi:	Kernel (Betriebssystemkern) Supervisor (Restbetriebssystem) User (Benutzerprogramme), außerdem Error-Mode (Diagnostik)
Pentium, (Vax):	4 Modi	0:Betriebssystemkern 1: Betriebssystemprozesse 2: global benutzte Prozesse (z.B. Compiler) 3: Benutzerprogrammläufe

Der Wunsch nach mehr als zwei Modi rührt her von Strukturierungs- und Schutzwünschen bei sehr großen Betriebssystemen; diese sind wegen ihrer Größe nicht als korrekt und sicher anzusehen!

Fortsetzung in die Benutzerprozesse hinein: bei Gruppen kooperierender Benutzerprozesse Einführung von "Privilegierungs"niveaus bei Zugriffen auf Daten, Programme. Dann 16..64 Niveaus.

Allgemeine Sprechweise:

ein Modus ist "privilegierter" als ein anderer, gewisse Daten, Befehle sind "privilegiert" (d.h. nur in privilegiertem Modus zugreifbar/ interpretierbar)

Auswirkung des aktuellen Maschinenmodus auf Befehlsausführung:

Ausführung kann:

- möglich sein, unabhängig von Modus in gleicher Weise (z.B. Registerarithmetik)
- möglich, aber modusabhängig verändert sein (Bsp. Adressierung!)
- unmöglich sein; dann wird ein Unterbrechungswunsch erzeugt, über den der Prozessor die Bearbeitung des Betriebssystems an definierter Stelle aufnimmt, womit
  - das den Befehl benutzende Programm unterbrochen wird und evtl.
  - das Betriebssystem die mit dem Befehl gewünschte Wirkung sinngemäß herstellt, z.B. "Maschinenstop" → "Ende dieses Programmlaufs"

  (*instruction trap*)

Bedingungen für "virtuelle Maschinen"
[vgl.Siegert Betriebssysteme: Einführung, Oldenbourg]

Ein Basisbetriebssystem (*virtual machine monitor*) trägt mehrere Betriebssysteme, auf denen jeweils Anwenderprogramme laufen.

	Basisbetriebssystem:	im Systemmodus
Dabei laufen	Betriebssysteme:	im Benutzermodus
	Anwenderprogramme:	im Benutzermodus

Voraussetzungen an den Prozessor:

- Prozessor ist entziehbar (Uhr-Alarm, privilegiert)
- virtuelle Adressierung (zur privilegierten Hauptspeicherzuteilung, Schutzmechanismus)
- nicht-privilegierte Befehle im Benutzermodus und Systemmodus gleich wirkend
- alle Befehle, die privilegierte Wirkungen haben (Organisation des virtuellen Speichers, E/A, Unterbrechungsmasken, Uhr, Wecker,...) sind privilegiert und nur im Systemmodus ausführbar. Im Benutzermodus führt ihre Anwendung auf eine Unterbrechung: Befehlsalarm (*trap*)

Modus-Übergänge

in den privilegierten Modus:

- durch (gewisse oder alle) Unterbrechungen; solche können sich vor allem auch aus Verstößen ergeben (Befehle, Adressen z.B.). Kann sehr sinnvolle Konstruktion sein! → Privilegierung
- durch Befehl (Systemaufruf, Testpunkte)
  Beispiel: R10000: SYSCALL-Befehl (mit Parameter zur Spezifikation); BREAK-Befehl für Programmtest; *"trap"*-Befehle (12 Stück, testen Zusicherungen)
  Pentium: Systemaufruf durch *call* in Betriebssystemadressraum, wird über *call gate* geleitet; im Übrigen ähnlich

in den weniger privilegierten Modus:

- durch Befehl (im Allgemeinen privilegiert)
  Beispiel: R10000: ERET *exception return*, privilegiert
  Pentium: RET *return* (Gegenstück zu *call*), über *call gate*; IRET *return from interrupt*

3.2.4 Datentypen

Von großem Einfluß auf Struktur und Aufwand des Prozessors ist die Zahl und Art der Datentypen und die Zahl und Art der Operationen, die der Prozessor auf ihnen ausführen kann

Müssen die Datentypen und Operationen durch den Prozessor fixiert sein?

> Burroughs B1700 (1971): bitadressierbarer Speicher, beliebige Datenformate, Interpretation erfolgt durch (evtl. vom Benutzer selbst verfasste) Interpreter, i.A. je an einer Familie höherer Sprachen orientiert.

Aus Effizienz-Gründen und quasi-Standardisierung der 8/16/32/62bit Formate unüblich. Stattdessen prozessorfixierte Datentypen mit zugehörigen Operationen

> **Bits** (selten!)
> **Bitfeld** Einzelbitoperationen auf z.B. 16/32 Bits, $\vee$, $\wedge$, $=$, $\neq$: wichtig, allgemein üblich
> Bitstring direkt mit Anfangsposition, Länge oder indirekt über "Deskriptor" adressiert: gelegentlich z.B. VAX
> Dezimalzahl (Festpunkt): gelegentlich
> **Bytes** typisch 8 Bits (*EBCDIC extended binary coded decimals interchange code*) oder *ASCII (American standard code for information interchange)*, 7Bits. Weitverbreitet, vor allem bei Byteadressierformat.
> **Bytestring** wie Bitstring, aber weit verbreitet; manchmal auch Darstellung mit Endmarke. IBM ca.1955
> **Dezimalzifferstring** (*packed decimals*), verbreitet als zwei 4Bits-Ziffern je Byte. Zugang wie Bitstring, selten über Schlußzeichen (Vorzeichen)
> **Festpunktzahlen** (dual): auf allen Rechnern, oft mehrere Formate, z.B. 8/16/32/64 Bits. Operationen an Langworten oft als Unterprogramm, aber maschinenunterstützt (*carry bit* als Unterbrechungswunsch)
> **Gleitpunktzahlen** (dual): Auf vielen Rechnern, bei Mikroprozessoren oft über Coprozessor. Gelegentlich mehrere Formate; erstmals Zuse 1936
> Komplexe Zahlen: selten
> Felder: Auf Vektorrechnern, sonst selten; aber Hilfsoperationen (Indexmodifikation, Indexprüfung vgl. VAX)
> Listen, File, Queue: selten
> **Stack**: als Hardwarestruktur selten (English Electric KDF9, 1958; dann Burroughs-Maschinen). Für Ausdrucksberechnung und Laufzeitstack. Unterstützung durch Adressierung im Hauptspeicher häufig (*autoincrement, autodecrement*-Technik). Erstmals DEC PDP11, 1971 Beispiel: MI,VAX

Weitere Datentypen im Prozessor als Folge des von Neumann-Prinzips mit ebenfalls erheblichem Einfluß auf Prozessorleistung und Komplexität:

> (Befehle vgl. 3.3)
> **Adressen:** Heute stets mit einem Festpunktformat übereinstimmend, mit Festpunktoperationen manipulierbar; vgl. die sorgfältige Trennung von typgebundenen Pointern $\uparrow$ *sometype* und *integer* in Pascal.
> **Deskriptoren:** Größen, über die (oft obligatorisch) komplexe Datenstrukturen (und oft: Programme) zugegriffen werden. Sie enthalten Adresse, Typ, Zugriffsrechte, Strukturinformation, evt. Zugang zu Operationen auf der Datenstruktur.
> **Zugriffsausweise** (*capabilities*): Schlüssel, die auf (wenigen) Maschinen zugreifende Prozesse benötigen, um auf Daten bzw. Programme zuzugreifen.

Üblicherweise ist der Datentyp **nicht explizit** notiert, sondern durch die Interpretation implizit definiert. Anders in Typenkennungsmaschinen (*tagged architectures*):

typ	*datum*

Typ dient

- zur Operationsanpassung (generische Operationen), z.B. ein Additionsbefehl auf alle Gleit-/Festpunkt/ Byte/ dezimal Formate anwendbar; Algorithmusauswahl und Konversion automatisch.
- oder zur Erkennung fehlerhafter Zugriffe:
  - Addition eines Befehls
  - Änderung eines Zugriffsausweises oder Deskriptors
  - Ausführung eines Bitfeldes
  - etc.

Beispiel: Datenformate R10000:

Byte (8b): aber nur für *load / store*
Halbwort (16b): aber nur für *load / store*
Wort (32b): zahlreiche Operationen. Als Festpunktzahl: Vorzeichen und 31b Zahl im Zweierkomplement
Gleitpunktzahlen nach IEEE 754 - Standard
Einfache Genauigkeit *single precision* 32b: 1b Vorzeichen, 8b Charakteristik (d.h. Exponent $\in [-126..+127]$), 23b Mantisse
Doppelte Genauigkeit (*double precision* 64b : 1b Vorzeichen, 11b Charakteristik (d.h. Exponent $\in [-1022..+1023]$), 52b Mantisse
keine Bytestrings !

Beispiel: Datenformate Intel Pentium II:

Dezimalzahlen 4b, eine oder zwei Ziffern/Byte
Bytes 8b
Festpunktzahlen 8b / 16b / 32b, Zweierkomplement
Gleitpunktzahlen wie IEEE 754 : 32b / 64b / 80 ("*extended*")b
Bitfelder bis zu 32b
Bit/Bytestrings bis $2^{32}-1$ Elemente
Adressen 32b, 46b
Multimediale Daten: 64b (als 8B oder 4x16b oder 2x32b) SIMD-Befehle

Abbildung der Wortformate auf Byte-Raster

Jonathan Swift: Gullivers Reisen (1726):
Gulliver gelangt in das Land Liliput, dessen Einwohner nur 6 Zoll groß sind. Dort war es verboten, ein Ei an der stumpfen Seite zu öffnen, weil sich der Prinz des Landes dabei in den Finger geschnitten hatte. Ein Teil der Bevölkerung, die BIGENDIANS, wollte das nicht hinnehmen, es gab 6 Rebellionen, ein Kaiser starb, einer mußte gehen. Die BIGENDIANS wurden aus dem öffentlichen Dienst ausgeschlossen, 100 BIGENDIAN-Bücher verboten. Viele BIGENDIANS retteten sich in das benachbarte Blefuscu. Es kam zum Krieg, der 3 Jahre dauerte und Liliput 30.000 Soldaten, vierzig große und noch mehr kleine Schiffe kostete. Die Invasion von Liliput konnte nur dadurch verhindert werden, daß Gulliver ("der Menschenberg") die feindliche Flotte raubte.

Beispiel Abb.3.15: 32b vorzeichenlose Zahl, abgebildet auf Byteadressen-Raster. Big Endians beginnen Byteadressen auf der höchstwertigsten Position, Littel Endians auf der niederwertigsten.

Bei Datenaustausch zwischen Big Endians und Little Endians:

- Bytes wortweise: müssen die Wörter am Ziel zerlegt, die Byteanordnung gespiegelt werden
- Wörter byteweise: muß die Bytefolge in Vierergruppen am Ziel gespiegelt werden

Bytefelder und Zahlen müssen als solche bekannt (evt. auch bezeichnet) sein !

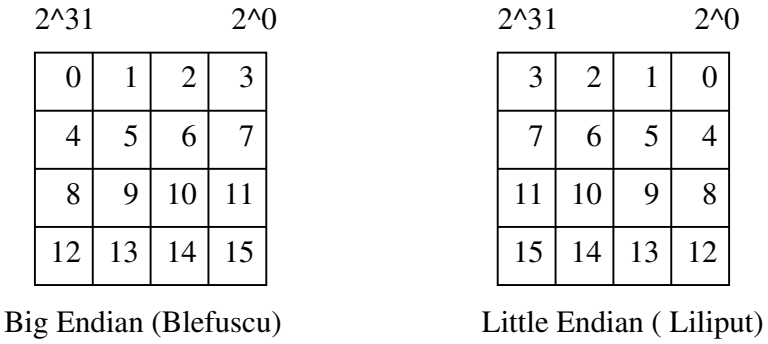


Abbildung 3.13: Big Endian, Little Endian

3.3 Befehle

3.3.1 Formate

Klassische Lösung (von Neumann, alle RISC-Prozessoren)
Einheitliches Format

Zwar einfaches Leitwerk, Befehlszähler generiert nächste Adresse, aber Redundanz, weil

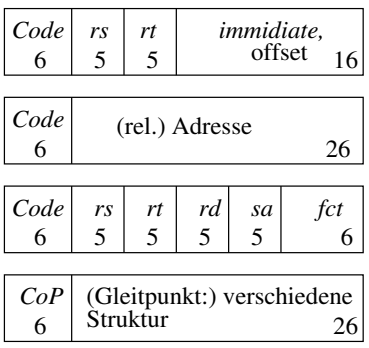
- Befehle verschieden häufig sind,
- Befehle nach Art verschieden viele und verschiedenformatige Adressen und Werte enthalten müssen.

Daher werden schon seit langem (Burroughs 1963) auch
verschiedene Formate benutzt

Meist beschreiben die ersten Bits, welches Format vorliegt.

Befehlszähler	→	Befehlsadressregister
instruction counter	→	*program pointer register*
		instruction address register

Beispiel R10000 Befehlsstruktur, ein Format



Code : Operationscode
rs: source: 1 von 32 Registern
rt: target: 1 von 32 Registern
rd: destination: 1 von 32 Registern
sa: Zahl der Schiebestellen
fct: Function: Erweiterung von Code
CoP: Code für "Coprozessor"
0 oder 3 : Gleitpunkt

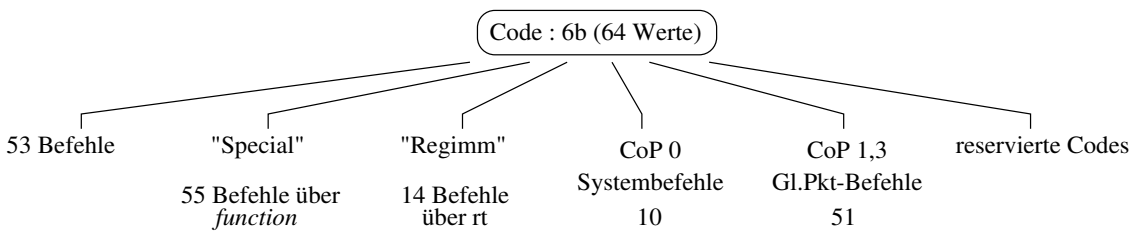


Abbildung 3.14: 183 Befehle

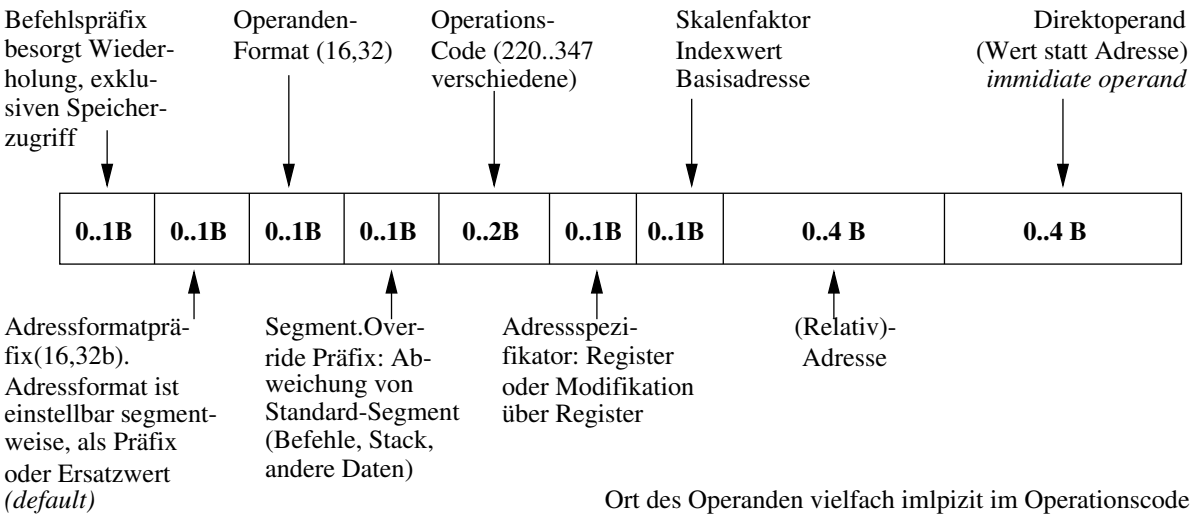


Abbildung 3.15: Befehlsformat des Pentium (ausgesprochen komplex)

Abbildung der Befehlsformate auf adressierbare Formate (Speicherzellen)

Einheitliches Format:

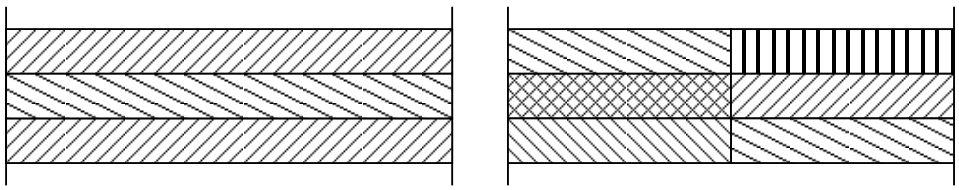


Abbildung 3.16:

Verschiedene Formate:

Maschinen mit kleinem adressierbaren Format, z.B. Byte-adressierende Maschinen wie IBM-Serien und viele Mikrorechner und VAX:

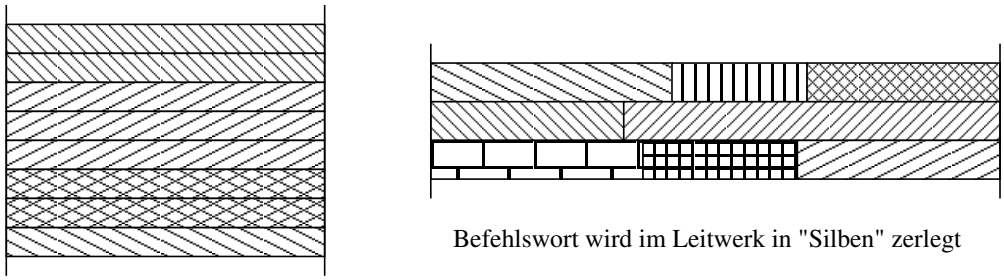


Abbildung 3.17:

Maschinen mit großen adressierbaren Einheiten (z.B. Control Data 6600 ff ab 1963)

3.3.2 Drei/zwei/eins/null-Adressprinzip

An dieser Stelle müßte nun erklärt werden, wie man zu einer Klasse von Algorithmen / zu einer Klasse von Programmiersprachen die optimale Maschine entwirft! Leider ist das jenseits der Fähigkeiten heutiger Informatiker. Der Stand der Methodik ist:

Halt dich an das Bewährte! Mehr noch: sei kompatibel!

Für einen geplanten Befehlssatz werden Probecodierungen und Simulationen gemacht → Platzbedarf,

Laufzeit; experimentelle Annäherung an das Optimum. Siehe Ansätze bei [PATTERSON HENNESSY 96]! Eine Ausnahme: solide Kenntisse bei arithmetischen Ausdrücken (aber das ist nur 5..10% eines Programmes!)

Zusammenhang zwischen Codeerzeugung, Befehlsformat, Platzbedarf, Prozessorstruktur und Programmlaufzeit bei arithmetischen Ausdrücken:
3-, 2-, 1-, 0-Adressprinzip

Behandlung am Beispiel: $Z := (A + B * C)/(D - E/F)$

Von uns (und dem Compiler) lieber so betrachtet:

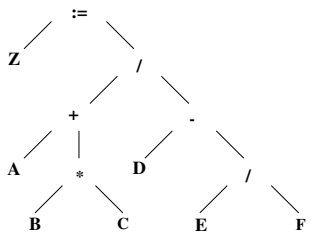


Abbildung 3.18: Beispielausdruck

Wir erfinden eine symbolische Schreibweise für Maschinenprogramme, z.B.:

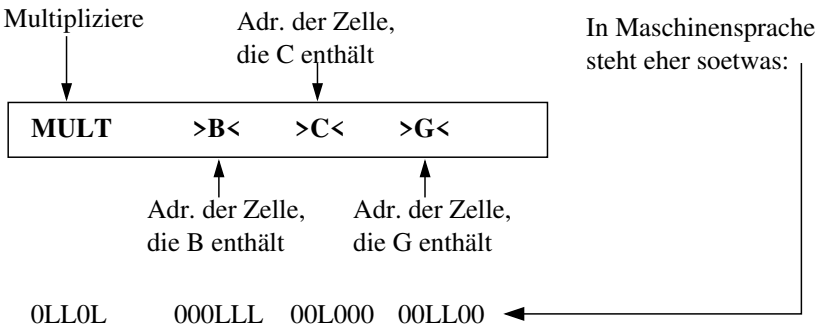


Abbildung 3.19: symbolische Schreibweise

Drei-Adressprinzip

Das folgende Programm tut's:

MULT	$>B<$	$>C<$	$>G<$
ADD	$>A<$	$>G<$	$>G<$
DIV	$>E<$	$>F<$	$>H<$
SUB	$>D<$	$>H<$	$>H<$
DIV	$>G<$	$>H<$	$>Z<$

G und H sind Hilfszellen für unsere Zwischenwerte. Noch einfache Codeerzeugung für Compiler

Platzbedarf: 5 Opcodes (MULT, ADD, DIV, SUB...), 15 Adressen
Operationen: 5

Verallgemeinert: Ausdruck mit v Variablen, $o = v - 1$ Operatoren, Adressformat a Bits, Opcodeformat c Bits:

Platzbedarf: (3-Adressprinzip) $o * (3a + c)$
Befehlszahl: o
Hauptspeicherzugriffe: $4o$
Zwischenergebniszellen in von Baumstruktur abhängiger Anzahl (Optimierungssache, Compiler!) Übereinstimmende Unterbäume einfach behandelbar; Zuweisung braucht keinen Befehl.

Drei-Adressprinzip

- mit vollen Speicheradressen unerträglich platzaufwendig; kann verbessert werden durch Basisadresse und Versatz (*displacement*), → 3.4
- mit Hauptspeicherzugriffen sehr anspruchsvoll bezüglich Hauptspeicherdurchsatz; lange Befehlsverweilzeiten (Ausführungszeiten), oder überlappung notwendig
- hat aber bei nur Hauptspeicherzugriffen den Vorteil, das nach jedem Befehl der Prozessor operandenfrei ist: kleiner Status, unterbrechungsgünstig

Wichtig sind Drei-Adress-Befehle bei Registermaschinen (und das ist ja heute der übliche Fall)

Registermaschinen

Zentralprozessor bewahrt aktuelle Werte (Operanden, Ergebnisse, Adressen) in adressierbaren Registern auf, mit

- kurzer Zugriffszeit
- kurzer Adresse

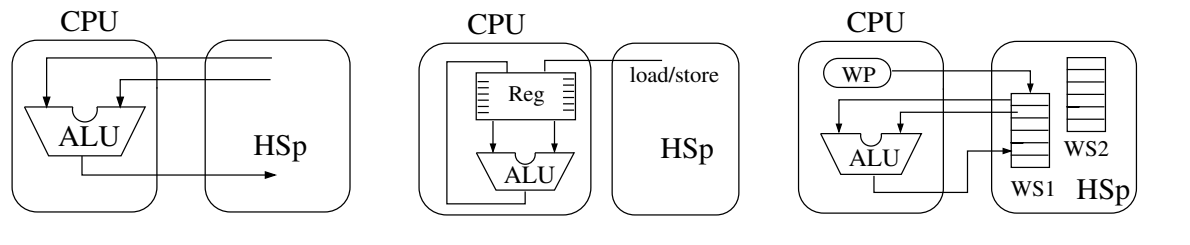
(Ferranti Pegasus, 1959; IBM /360, 1965; PDP11, VAX)

heutige Standard-Struktur: CISC: 8..32 Register, RISC: 32..2048 Register

Vorteil:	Zugriffszeit, Code-Platzbedarf, Hauptspeicher-Grenzdurchsatz kann kleiner sein
Nachteil:	Umspeicherungen bei Prozesswechsel (durch mehrfache Registersätze milderbar), mehr Befehlstypen, mehr Befehle im Programm

Registerzuteilung vom Compiler zur Übersetzungszeit festgelegt! Früh studiertes Optimierungsproblem.

Die Sache noch einmal grundsätzlich betrachtet, Exkurs Registermaschinen:
Drei wesentliche Varianten mit Mischformen:



Speicher-Speicher Maschine (SS-Typ)

gut: kleiner Status, einfache Befehlsliste
schlecht: Platzbedarf der Aderssen, langsam

Register-Register Maschine (RR-Typ)

gut: kompaktes Programm, kurze Adr., schnell
schlecht: großer Status, load/store-Befehle, s.RISC:

virtuelle Registermaschine
(workspaces WS1, WS2)

gut: kompaktes Programm, kleiner Status, einfache Befehlsliste
schlecht: u.U. langsam

Abbildung 3.20: Registermaschinen

virtuelle Registermaschine (*workspaces* WS_0, WS_1) selten! Günstig bei Prozessoren mit Hauptspeicher auf einem Chip (Texas Instruments TMS9900, IMMOS Transputer)

Beispiel: R10000

Allzweckregister:

64 Stück zu 64b, dabei zugleich 7 lesbar und 3 schreibbar (*pipelines*!)

Gleitpunktregister: 64 Stück zu 64b, dabei zugleich 5 lesbar und 3 schreibbar (*pipelines!*)

Der Prozessor kann (5b!) nur 32 Register von jeder Sorte adressieren. Er benutzt die physisch vorhandenen 64 Register als die funktional ("logisch") vorhandenen Register in aufeinanderfolgenden Rollen (*register-renaming*).

```
Beispiel   lade       R7,   m0
           add        R9,   R7,   R7
           speichere  R7,   m1          R7, R9, R11: funktionale ("logische") Register;
           lade       R7,   m2          m0..m4: Speicherzellen (Adressen)
           add        R11,  R7,   R7
           speichere  R7,   m4
```

Tatsächlich könnten die ersten und letzten drei Befehle parallel ausgeführt werden, wenn R7 durch zwei physische Register vertreten wird!

Beispiel: Pentium III (ein vernarbtes Antlitz)

Funktionale ("logische") Register

8 allgemeine Register

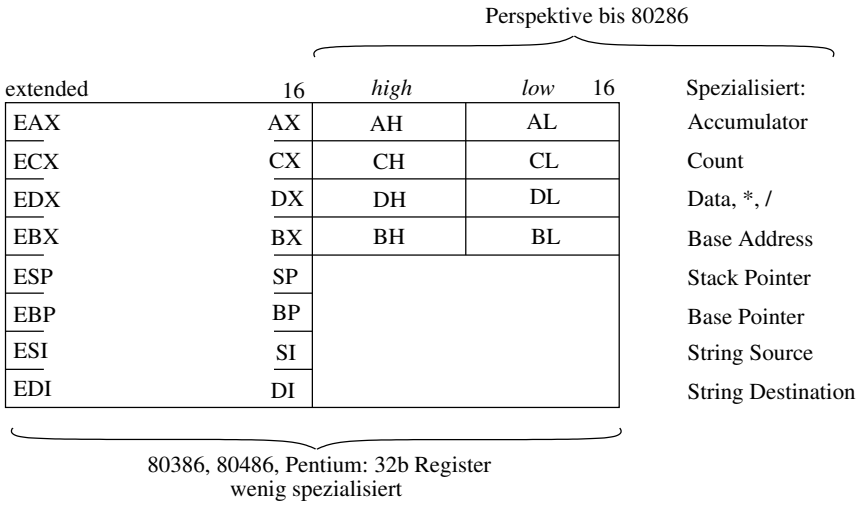


Abbildung 3.21: Register im Pentium

8 Gleitpunktregister 80b (*extended floating point*)

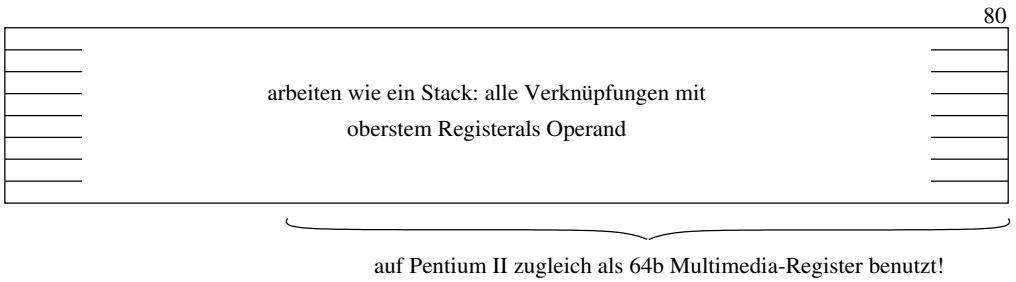


Abbildung 3.22: Gleitpunktregister 80b

Physische Register

40 Stück, die den 8+8 funktionalen Registern wechselnd zugeordnet werden (*renaming*). Auf dem Pentium viel nötiger als auf dem R10000!

Befehlsformate (leider brauchen wir mehr):

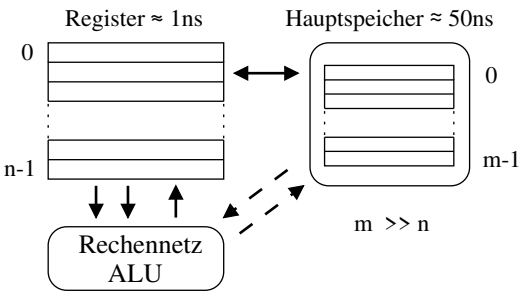


Abbildung 3.23: Registermaschine

1.Lösung: Alle verarbeitenden Operationen haben nur Registeradressen, daneben
lade (*load*) Befehle und
speichere (*store*) Befehle
einfache Programmstruktur, *load/store* Architektur; langsam ohne überlapptes Laden

2. Lösung: Verarbeitende Operationen erlauben eine oder zwei Speicheradressen, Rest Registeradressen; lade, speichere trotzdem notwendig! Varianten der Operationen, wenn z.B. Minuend, Subtrahend oder Differenz im Speicher steht! -schneller als 1.Lösung, komplexer Lösungen für das Problem, daß nach Wahl ein Operand (bei nicht kommutativen Operationen: welcher?) oder das Ergebnis im Speicher stehen soll:

   - verschiedene Varianten der Operationen (also 2^3 Subtraktionen?)
   - ergänzende lade/speicher-Befehle
   - ein Befehlstyp, aber selbstbeschreibende Adressen
     - Register als Adress-Unterraum (z.B. 0..31) dann kein Platz-Vorteil
     - besonderes Kennzeichen
       Beispiel : MI / VAX:

       Jedes Adressfeld (*"operand specification"*) enthält

       1..2 Bytes *operand specifier* Registeradresse und Deutungsvorschrift für das gesamte Adressfeld
       0..8 Bytes *specifier extension*

       Damit mit nur 8 Bits (!) 1 aus 16 Registern adressierbar, eine Wohltat für den Compiler!

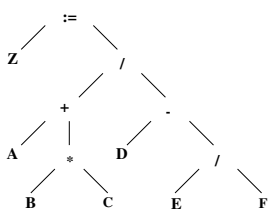
Übergang zu Befehlsformaten mit kleinerer Anzahl von Adressen je Operation durch zwei Prinzipien:

- Überdeckung: ein Operand wird durch das Ergebnis überschrieben (Achtung! Verlust, oder erst kopieren)
- Implizierung: eine Größe ist stets an einem stillschweigend bekannten Ort (z.B. "Akkumulator"-Register oder Stack-Spitze)

Beide können kombiniert werden.

Zwei-Adress-Prinzip
Ausnutzung der Überdeckung:

MULT	*>B<*	*>C<*	*B:=B*C*
ADD	*>B<*	*>A<*	*B:=B+A*
DIV	*>E<*	*>F<*	*E:=E/F*
SUB	*>D<*	*>E<*	*D:= D-E*
DIV	*>B<*	*>D<*	*B:=B/D*
TRANSP	*>Z<*	*>B<*	*Z:=B*



Platzbedarf: 6 Operationen, 12 Adressen
Operationen: 6

Wir haben die Argumente überschrieben! Zuweisung braucht nun einen Extra-Befehl.

Verallgemeinert:

o Zahl der Operatoren des Ausdrucks, a Adressformat, c Opcode-Format
Platzbedarf (Zwei-Adressprinzip): $o * (2a + c) + 1 * (2a + c)$ Zuweisung braucht Befehl; soll Minuend oder Subtrahend überschrieben werden? Zwei Befehle?
Befehlszahl: $o + 1$
Hauptspeicherzugriffe: $3(o + 1)$

Registermaschinen mit Zwei-Adressbefehlen:
Wichtig in Mischform mit Drei-Adress-Registertyp. Historisch wichtig (DEC10, IBM /360), ebenso Pentium:

"Anderthalb-Adressmaschine": Eine Registeradresse, eine Hauptspeicheradresse. Typisch 1 Operand aus Hauptspeicher.

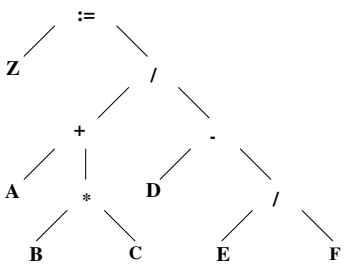
(Minuend, Subtrahend? Also zwei*zwei Subtraktionsbefehle oder erst transportieren?)

Ergebnis wird mit speichere-Befehl in den Hauptspeicher gebracht. Häufig in IBM /360 ff gebraucht.

Ein-Adress-Prinzip

Ausnutzung von Überdeckung und Implizierung: "Akkumulator" ist stets beteiligtes Register (AC)

HOLE	*>B<*	*AC := B*
MULT	*>C<*	*AC := AC*B*
ADD	*>A<*	*AC := AC+A*
SPEICH	*>G<*	*G := AC*
HOLE	*>E<*	*AC := E*
DIV	*>F<*	*AC := AC/F*
SPEICH	*>H<*	*H := AC*
HOLE	*>D<*	*AC := D*
SUB	*>H<*	*AC := AC-H*
SPEICH	*>H<*	*H := AC*
HOLE	*>G<*	*AC := G*
DIV	*>H<*	*AC := AC/H*
SPEICH	*>Z<*	*Z := AC*



Platzbedarf: 13 Operationscodes, 13 Adressen
Operationen: 13
Wenn man aber zunächst den Divisor berechnet und noch einen Befehl *SUB2 >X<* AC:=X-AC erfindet, kommt man auf 9(!) Operationen hinunter, falls der Compiler so clever sein darf.

Verallgemeinert:

Mit o Anzahl der Operatoren des Ausdrucks, a Anzahl Bits des Adressformates und c Anzahl Bits des Opcode-Formates

Platzbedarf (Ein-Adressprinzip): $o(a + c) + 2(a + c) + [laden, speichern]$
Anteil für laden, speichern hängt von Baumstruktur und Compiler ab!
Befehlszahl: $o + 2 + [laden, speichern]$
Hauptspeicherzugriffe: $2(o + 2 + [laden, speichern])$

Befehlsvarianten wie AC:=D-AC, und Zusatzbefehle wie AC:=-AC, AC:=1/AC sind sehr nützlich. Die Nutzung übereinstimmender Unterbäume braucht besondere Speicher- und Ladeoperation. Im wesentlichen werden keine Argumente überschrieben!

Null-Adressprinzip
(English Electric KDF9, 1959; Burroughs, 1962)

Arithmetischer Ausdruck wird in Postfix-Notation gebracht:

$$Z := (A + B * C)/(D - e/F) \quad \Rightarrow \quad >\text{Z}< ABC * +DEF/- :=$$

und das ist die Maschinensprache: also 7 Adressen, 6 Operationscodes

Die Maschine arbeitet mit einem Operationskeller. Abarbeitung von links: jeder Operand wir gekellert, ebenso die Adresse von Z, jede Operation wirkt auf die beiden oberen Kellereinträge und das Ergebnis ersetzt diese.

C B A >Z<	B*C A >Z<	 Zähler >Z<	F E D Zähler >Z<	E/F D Zähler >Z<	 Nenner Zähler >Z<	 Quotient >Z<
1	2	3	4	5	6	7

Abbildung 3.24: Nulladress-Maschine

Nulladress-Prinzip:

- Überdeckung (Operand auf Stack wird überschrieben) und Implizierung (beide Operanden und Ergebnis auf Stack) ⇒ Operation ohne Angabe von Adressen
- alle Variablen des Ausdrucks sind auf den Stack zu laden; aber dann kein laden/speichern, da Operanden stets korrekt bereitgestellt!
- Abspeicherung am Ende. Zusätzliches Abspeichern (*copy, nicht pop*)zur Behandlung übereinstimmender Unterbäume.

Verallgemeinert:

Mit o Anzahl der Operatoren des Ausdrucks, a Anzahl Bits des Adressformates und c Anzahl Bits des Opcode-Formates: **Platzbedarf** (Null-Adressprinzip): $(o+1)*a+o*c+(a+c) \quad = \quad o(a+c)+2a+c$
$(o+1)$ wegen Anzahl Variablen, $(a+c)$ wegen abspeichern
Befehlszahl: $o+1$ (wegen anfänglichen Ladens)
Hauptspeicherzugriffe $o+2$ (Befehle)

Bewertung der Ergebnisse

	Platzbedarf	Befehlszahl	Hauptspeicherzugriffe
3Adr.	$o*(3a+c)$	o	4o
2Adr.	$o*(2a+c)+(2a+c)+[l,s]$	$o+1+[laden, speichern]$	$3(o+1+[l,s])$
1Adr.	$o*(a+c)+2*(a+c)+[l,s]$	$o+2+[laden, speichern]$	$2(o+2+[l,s])$
0Adr.	$o*(a+c)+(2a+c)$	$o+1$	$o+2$

- [laden,speichern] ist Compiler-sensibel.
- 3Adr. und 0Adr. sind compilergünstig
- 3Adr. nutzt übereinstimmende Unterbäume ohne Mehraufwand
- 3Adr. nur bei unkomplexen Ausdrücken erträglich
- für komplexe Ausdrücke Nulladress-Prinzip günstig (diese sind selten!)

! tatsächlich weitgehende Verschiebung bei Benutzung von Registern bei 3Adr. und 2Adr.!

Zahlenbeispiel des von uns betrachteten Ausdrucks:

	#Opcodes (8b)	#Adr. 4b bwz.24b	Platzbedarf bei 4b - Adressen	Platzbedarf bei 24b - Adressen
3-Adr.	5	15	100	400
2-Adr.	6	12	96	336
1-Adr.	13	13	156	416
optimiert	9	9	108	288
0-Adr.	7	7	84	224

Heutige Rechner typisch Registermaschinen mit Drei- und Zwei-Adress-Befehlen, so auch MI/VAX. Auf RISCs aber keine gemischte Register/Speicher Adressen bei den verknüpfenden Operationen.

Bei Maschinen mit einheitlichem Befehlsformat muß das Format auch an den nicht-arithmetischen Befehlen ausgerichtet werden (am wichtigsten: Transporte und Sprünge)
Schade, daß das Teilgebiet, in dem man etwas über die Bewertung von Befehlsstrukturen weiss, eben nur so ein 5..10% Teilgebiet ist; größer ist der Arithmetikanteil nicht.

3.3.3 Befehlstypen

Die Semantik der Befehle ist bestimmt durch

- das gewählte Format (kleines Format zwingt zu Vorbefehlen, z.B. *compare-branch*; großes Format führt zu aufgeblähter Befehlswirkung mit Varianten; Vorteil von uneinheitlichen Formaten)
- Semantik der Benutzersprache
- Maschinenstruktur
- Ablauferfordernisse (z.B. Überlappbarkeit)

Befehle nach Häufigkeit in Programmen:

- Transporte (ca.50%)
- Sprünge (ca.25%)
- Arithmetik (ca.10%)
- Shifts (ca.3%)
- "logische" Befehle (ca.3%)
- Rest (ca.9%)

Die Angaben schwanken mit Maschinenarchitektur/ Sprache/ Algorithmus.

Mögliche Ordnung (im Folgenden benutzt) nach Ort der Ausführungsphase:

- Leitwerks- und Speicherbefehle
- Rechenwerksbefehle
- Ein/Ausgabe-Befehle

Natürlich durchlaufen alle Befehle die Abrufphase (*instruction fetch*) im Leitwerk.

Beispiele: Befehlslisten:

	Pentium II	R10000
Leitwerksbefehle:		
Systembefehle (mit E/A), meist privilegiert	32	10
Transporte	25	23
Sprünge, Aufrufe, programmierte Unterbrechungen	9	16
Vergleiche, Tests, bedingte Sprünge und Tranporte	18	18
Schleifen	4	-
Adressierung	5	-
Andere	8	3
Rechenwerksbefehle:		
Arithmetik Festpunkt	18	32
Arithmetik Gleitpunkt	97b	59
Arithmetik MMX	57	-
Flags, Conditions	9	-
Konversion	5	-
"Logik" (and, or...)	4	4
Shift, Rotation	16	15
Strings	7	-
Bits	6	-
	320	180

3.3.4 Leitwerksbefehle

Nullbefehl (für Leerplätze in Programmen: in einfach RISC-CPUs, um Folgeoperationen in Pipe zu verzögern) [Pentium: NOP (1Byte), R10000: nicht vorhanden]

STOP: beendet Interpretationsschleife; privilegiert; im Benutzermodus gelegentlich als Systemaufruf "Programmlauf-Ende Behandlung" ausgeführt.
[Pentium: HLT, privilegiert; R10000: ?]
zu den R10000-Beispielen: etwa schieben um 0 Stellen (aber Registerreservierung!), schreiben in Steuerregister (für Halt)

Transporte (besser kopieren): *move*

Dimensionen:	Ort von Quelle/Senke Transportformate Manipulationen (z.B. Komplement)
Register→Register:	Format z.B. Byte bis Wort (=Registerlänge)
Speicher→Register:	wie oben; wichtige Erweiterung: laden aller allgemeinen Register aus dem Speicher bei Unterprogramm-Rücksprung/Rückkehr aus System oder Unterbrechungsbehandlung, meist vom Stack.
Register→Speicher:	wie oben; "retten" aller allgemeinen Register in Speicher (meist Stack)

Häufig Quelle/Senke-abhängige Operatoren:

LADE	(Sp→Reg)	*load*
SPEICHERE	(Reg→Sp)	*store*

Beispiel: Transporte im Pentium II

Zunächst betrachten wir die Registerstruktur: Abb.3.23 Seite 68. Alle diese Register sind *"general purpose"*, können aber nicht von allen Befehlen gleich verwendet werden. Außerdem 8 80b-Gleitpunktregister, 6 Segmentregister

Format der zu transportierenden Einheit ist wählbar als Wort(16b!) und Doppelwort(32b), abhängig vom Ausführungsmodus des Prozessors:

Real mode: Der Prozessor simuliert einen 8086, binärkompatibel: keine privilegierten Funktionen. Die Formate sind 16b; die Adressen sind ebenfalls 16b, also 64kB adressierbar je Segment.

Protected mode: eigentlicher (*genuine*) Modus des Prozessors: 4 Privilegierungsstufen. Segmentdeskriptoren enthalten Angabe, ob Formate 16b oder 32b. Von dieser Vorgabe kann durch *override prefix* befehlsweise abgewichen werden. Ebenso: 16/32b Adressierung.

Virtual 8086 mode: Prozessor simuliert (mehrere) 8086 als virtuelle Maschinen, d.h. kritische Operationen rufen *traps* hervor, vgl 3.2.3 Seite 61

Gilt ebenso für andere Operationen !

Format: andere(!) Struktur für Operanden und Ergebnisse, die mit der *"floating point unit" (FPU)* verarbeitet werden.[Beim Pentium-Prozessor gibt es allerdings die *FPU* nicht mehr, sondern nur noch die mit "F" beginnenden Befehlsoperationscodes (für Festpunkt 32/64b, Gleitpunkt 32/64b), Ausführung in den "Gleitpunkt"-pipes mit Wortlänge 80b, davon 64b Mantisse, die auch für die Festpunktrechnung dient. Registerformat für 8 Gleitpunktregister ebenso.]
Hier wird das Format im Speicher durch den Operationscode der *load/store* Befehle festgelegt; in den Registern und in der Pipe Aufweitung auf das 80b Format.

Quelle und Senke (Register/Speicher) und Wert statt Adresse aus Operationscode ersichtlich. Also (r Register, m Speicherzelle):

MOV (20 Varianten): $r \to m$, $m \to r$, $r \to r$ (wichtig, da Register nicht gleichwertig), $Wert \to r$, $Wert \to m$ verkürzte Form arbeitet nur mit A-Register
MOVS (Stringbefehle, siehe Seite 76) transportiert $m \to m$
XCHG *exchange* $r \leftrightarrow r$, $m \leftrightarrow m$: letztere unter lock, d.h. die beiden Zellen werden für andere erst wieder zugänglich, wenn beide ihren endgültigen Wert haben (Synchronisationsmittel in Mehrprozessoranlagen)
Außerdem lade/speicher Befehle für die 80b Gleitpunktregister.

Beispiel: Transporte im R10000:

Zunächst wieder die Registerstruktur: Dazu ebenso viele 64b "Gleitpunkt"register, ebenfalls

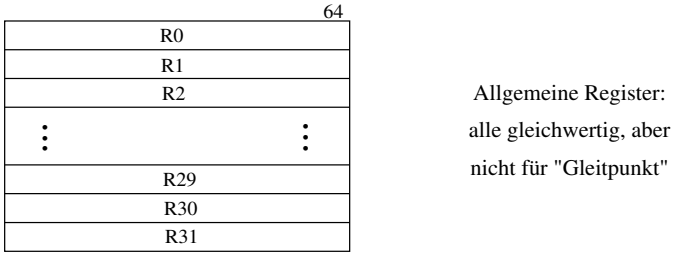


Abbildung 3.25: Registerstruktur R10000

untereinander gleichwertig, für "Gleitpunkt" lade/speicher Befehle.

Format, Quelle/Senke geht aus Operationscode hervor. Zu den allgemeinen Registern:

LB/SB	*load/store* Byte	(8b)
LH/SH	*load/store* Halfword	(16b)
LW/SW	*load/store* Word	(32b)
LD/SD	*load/store* Doubleword	(64b)

Keine Transporte zwischen den Registern (wegen Drei-Adress-Verknüpfungen unnötig!)

Daneben Hilfsbefehle, die aus/in Speicher nicht ausgerichtete Daten transportieren, d.h. Halbwörter beginnen nicht auf geraden (Byte-)Adressen, Wörter nicht auf durch 4 teilbaren Adressen:
LWL/LWR, SWL/SWR load word left/right, store word left/right
LDL/LDR, SDL/SDR load double left/right, store double left/right
Außerdem *load/store* Paare, die unter lock am Speicher arbeiten und das Abspeichern einer Bedingung unterwerfen (z.B. Semaphor-Operationen)

LL/SC load linked word / store conditional word.

Schließlich lade/speicher Operationen für "Gleitpunkt"- und Systemregister und Transporte zwischen allgemeinen Registern und "Gleitpunkt"- und Systemregistern.

Einschub: Delay-Slot Befehle bei Lade- und Sprungbefehlen im R10000

Sobald bekannt ist, daß ein Sprung auszuführen ist, sind die bereits vorsorglich geholten Nachfolgebefehle unbrauchbar. Der wirkliche Nachfolgebefehl steht erst verzögert zur Verfügung. Derselbe Effekt tritt bei Ladebefehlen auf: das zu ladende Datum steht verzögert zur Verfügung (Cache oder (schlimmer:)Hauptspeicher).
Abhilfe bei R10000: der auf den Sprung- oder Ladebefehl folgende Befehl muß nebenläufig zu diesem sein (d.h. dieser Befehl und Sprung/Ladebefehl sind unabhängig!). Der Befehl wird in jedem Fall ausgeführt. Er überbrückt also die Verzögerung. Daher *delay-slot* Befehl genannt.

Beispiel: Transporte in DEC VAX, 1978 auch bekannt als MI (Informatik III)

Orthogonales Konzept

Adressierung von Quelle/Senke durch Operandenspezifikation aus:

- *operand specifier*, 1..2 Bytes, enthält meist Registernummer und Deutungsvorschrift für
- *specifier extension*, 0..8 Bytes, enthält Adresse (4Bytes) oder Wert (bis 8Bytes) oder Basisadressen

Vorteil: einheitliche, nur Typ-abhängige Operatoren:

MOV B (Byte), **MOV H** (Halbwort, 2Byte) **MOV W** (Wort, 4Byte), **MOV L** (Langwort, 8Byte), **MOV Q** (Quadwort, 16Byte), **MOV F** (Gleitpunkt, 4Byte), **MOV D** (Gleitpunkt "doppelte" Genauigkeit, 8Byte).

Zwei Adressen (Operandenspezifikationen)

Wichtige Sonderfälle von Transporten:

- Registerinhalt(e) vom Stack (*pop*) oder auf den Stack (*push*). Stack üblicherweise zu kleineren Adressen wachsend.
  Beispiel Pentium II: Verwendung des Stackpointers SP/ESP:

POP	Einzelwort/Doppelwort
POP A	POP 8 allgemeine Register (16b)
POP AD	POP 8 allgemeine Register (32b)

  PUSH, PUSH A, PUSH AD: entsprechend.

  Daneben POP und PUSH für die "Gleitpunkt"register mit Statusinformation:

FSTORE	restore FPU state
FSAVE	store FPU state

  Diese beiden Operationen benutzen nicht den Stackpointer.

  Die Abspeicherung/Wiederherstellung der Registerinhalte ist Teil des/der Aufrufs/Rückkehr von Unterprogrammen, System (CALL oder Unterbrechung).

- Transporte unter Formatänderung, Negation, Komplementbildung. Beispiele bei Pentium II und R10000.

- Schreiben und lesen der privilegierten Register im Prozessor. Es überwiegen Codierungen der Register im Operationscode (statt Adressierung!). Mindestens die Schreiboperationen sind privilegiert.
  Beispiel Pentium II: unter den 32 "System"befehlen sind 7 Schreiboperationen und 5 Leseoperationen an privilegierten Registern.
  Beispiel R10000: unter den 10 "System"(CPO-) Befehlen sind 4 Schreiboperationen und 3 Leseoperationen an privilegierten Registern

Transporte von Zeichenketten

- Byte-Strings (auf Byte-adressierten Maschinen zugleich "Worttransport")
- Dezimalstrings
- Wortblöcke (wichtig für gewisse Speicherverwaltungstechniken, Puffer umkopieren).

Strings beschrieben durch Deskriptor

> Komponententyp
> Länge
> Anfangsadresse
> etc.

oder Strukturinformation im zugreifenden Befehl (Adressteil)

So VAX:	MOV P	*move packed decimals string*
	MOV C	*move character string*

Die Länge der Kette kann groß sein, z.B. 2^{16} auf VAX

Folge: Die Befehle müssen während (und nicht erst nach) der Ausführung unterbrechbar sein, d.h. der Intrabefehlsstatus muß rettbar sein:
- laufende Adresse
- Restlänge

Typisch durch Benutzung der allgemeinen Register (!).

Variante: Ketten mit expliziter Schlußmarke statt Längenangabe. Schlecht in überlappt arbeitenden Prozessoren, da nicht vorweg übersehbar, welche Zellen beschrieben werden.

Beispiel Zeichenkettenbefehle im Pentium II:

> Die Befehle werden mit Wiederholungspräfix *rep* realisiert (Ende bei Nulldurchgang im ECX Register) oder mit anderen Wiederholongspräfixen (z.B. Übereinstimmung mit Wort in Register). Mitwirkende Register:
>
> | | |
> |---|---|
> | ESI: | laufende Quelladresse |
> | EDI: | Laufende Zieladresse |
>
> | | |
> |---|---|
> | MOVS | transportiert String aus Bytes/Wörtern/Doppelwörtern im Speicher |
> | CMPS | vergleicht zwei Strings |
> | SCAS | Vergleicht einen String mit dem Inhalt von AL/AX/EAX, hält z.B. bei einem Treffer |
> | LODS | lädt die Elemente eines Strings in AL/AX/EAX (sinnvoll in Verarbeitungsschleife) |
> | STOS | Speichert den Inhalt von AL/AX/EAX als String (sinnvoll in Schleife) |

Letzter Fall gehört nur halb zu den Transporten: Beschaffung der effektiven "Prozess-" Adresse; wird oft gebraucht, um die vom Prozess erarbeitete Adresse (z.B. nach Basis/ indizierter/ indirekter Adressierung) festzuhalten oder weiter zu bearbeiten; liefert nicht die Maschinenadresse (d.h. z.B: nach Seitenadressierung).

Beispiel VAX/MI: MOVA B/W/L/F/Q/D *move address*

Beispiel Pentium II: LEA *load effective address*

Sprünge

Mit ca. 25% nicht etwa wegen des *goto*-Trauma schlimm: Sprünge gestalten die Befehlsadressfolge unvorhersehbar, ruinieren vorsorgliches Befehlsholen und Pipelining.

unbedingter Sprung: Ein-Adressbefehl; Modifikation der Adresse ist wichtig für Vielfach-Verzweigungen (*case, computed goto*), daher auch anderthalb-Adressformate häufig (Steuergröße im Register)

Beispiel Sprünge im R10000:

J *jump*:	Ein-Adressbefehl mit 26b Zieladresse; wird über Befehlszählerstand auf 44b virtuelle Adresse aufgeweitet.
JR *jump register*	Springt auf die in einem allgemeinen Register angegebene Adresse

Beispiel Sprünge im Pentium II:

JMP *jump*: schlichter Ein-Adressbefehl in 4 Hauptvarianten, weiteren durch Adressmodifikationen.
Der Pentium nutzt für jeden Prozess einen (virtuellen) Adressraum aus maximal 16k Segmenten zu je maximal 4GB Adressraum (also höchstens $2^{14} * 2^{32}$B adressierbar je Prozess). Als Sprungziel kommt in Frage:

- Eigenes Segment (32b Adresse oder 8b relativ; über Register oder Speicher ebenfalls möglich)
- Fremdes Segment über *call gate* und Berechtigungskontrolle; *call gate* ist Datenstruktur wie Segmentdeskriptor
- anderer Task über *task gate*.

Erst die gates enthalten Zieladressen (Zugriffsschutz!)

Bedingte Sprünge

- Einadress-Befehle, wenn Bedingung im Operationscode ("springe, wenn Inhalt des Akkumulators null" oder "springe, wenn Überlauf"). Besondere Rolle: *condition codes*, geben Aufschluß über Ausgang letztvollzogener Operationen (z.B. "null", "negativ", ...).
  Klassische Lösung! Aber unbrauchbar, wenn Reihenfolge der Befehle modifiziert wird (*condition code* schon bereit gestellt? Noch aufzuheben?)
- Zweiadress-Befehle, wenn ein beliebiges Register die Bedingung stellt (z.B. "R12 positiv")
- Dreiadress-Befehl, wenn zwei beliebige Register verglichen werden.

Die Lösungen über Register bleiben geeignet, wenn von der Ausführungsreihenfolge abgegangen wird! Bedingter Sprung muß Schreiben in "sein" Register abwarten, Register bleibt reserviert bis zu "seinem" bedingten Sprung.

Beispiel bedingte Sprünge im R10000:

kein Condition Code !
2 Vergleiche eines Registers: < 0, ≥ 0
4 Vergleiche zweier Register: $=, \neq, \leq, >$.
von allen Befehlen gibt es zwei Varianten: mit delay-slot-Befehl oder ohne (likely-Varianten).
Letztere wird benutzt, wenn Vollzug des Sprunges unwahrscheinlich ist!

Alle bedingten Sprünge haben Befehlszähler-relative Adressen!

Außerdem 2 bedingte Transporte. Damit einfachste bedingte Anweisung ohne verspätete Änderung der Befehlsadresse!

Unterprogrammsprünge/Rücksprünge

Aufgaben:

Sprung mit	• Sicherung von bisherigem Kontext, z.B. Registerinhalte. (nicht Prozess-Status im Betriebssystem-Sinn!) • Rücksprung vorbereiten • Zugang zu Parametern schaffen
Rücksprung mit	• Abbau lokaler Status (in Algol-Sprachen!) • Manchmal: Rückübertragung von Ergebnissen (*call by value*)

Verschiedene Techniken, unterscheidbar vor allem nach Art der Bereitstellung von Rücksprungadresse und Parametern:

Rücksprungadresse	Parameter	Folgen
Register	Register	Einfachster Parameterzugriff; eingeschränkte Registernutzung für andere Zwecke; Verschachtelung erschwert; kaum Anforderungen an Sprung/Rücksprung
Register	Speicher (Adresse in (Index) Register)	Schlechter Parameterzugriff; erschwerte Verschachtelung; kaum Anforderungen an Sprung/Rücksprung
Register	Speicher (auf Absprungort folgend)	Elegante Mehrfachnutzung der Rücksprunginformation (→ 3.4); Verschachtelung, aber keine Rekursion; kaum Anforderungen an Sprung/Rücksprung; kein *pure procedure*!
Speicher, Stack	Speicher, Stack	Aufwendiger Aufbau und Zugriff, macht Spezialbefehle und/oder gute Adressmodifikationen notwendig. Stack Pointer. Rekursion, *pure procedure*. Notwendig für Algol-Sprachen

Beispiel Pentium II

CALL:
- gewöhnlicher Unterprogrammsprung im eigenen Segment (absolut, relativ zu Befehlsadressregister oder indirekt über Register oder Speicherzelle)
- mit Segmentwechsel bei gleicher Privilegierung; dann Kellerung der bisherigen Segmentkennung (*segment selector*)
- mit Segmentwechsel zu größerer Privilegierung (geringeres *protection level*): nur indirekt über einen *call gate* genannten Deskriptor (Aufruffilter).

→ Genaueres im Adressierungskapitel; call benutzt ESP als Stackpointer, EBP als Basepointer (Frame Pointer, Basisadresse des Aufrufsatzes)

RET: Rücksprung mit Freigabe des Stacks (Stackpointer←Basepointer). Eventuell Segmentkennung wieder herstellen

Dazu 2 weitere Befehle für Auf/Abbau des Aufrufsatzes (*activation records*):

ENTER: 1.Parameter: Größe des Aufrufsatzes, 2.Parameter: Zahl der textuell umgebenden Blöcke (statische Schachtelungstiefe). Für *display (current environment vector,* Vektor von Base-Pointern der aktuell globalen Aufrufsätze)

LEAVE: Baut Aufrufsatz wieder ab

Also Folge:
CALL
ENTER
<Prozedurrumpf>
LEAVE
RET

Systemaufrufe

Sprünge mit Übergang in privilegierten Modus/Niveau, mit Angabe der gewünschten Systemdienstleistung, eventuell Pointer auf Versorgungsblock. Ablauf:

- Übergang auf Stack des Zielmodus
- Status des bisherigen Prozesses kellern (nicht nur Register, sondern auch Programmstatuswort, Betriebssystemdaten dieses Prozesses (Priorität etc. → Prozesskontrollblock).
- Übergang auf Zielmodus
- Verzweigung zum gewünschten Dienst, Prüfung der Berechtigung
- Prüfen der mitgebrachten Parameter (sind das vielleicht Adressen, zu denen der Aufrufer gar nicht zugreifen (lassen) darf?

Rückkehr aus dem System: setzt Prozess wieder ein (privilegiert!)

Beispiel Pentium II:

Kein Systemaufruf, keine Systemrückkehr! Konsequenter Weise durch CALL, RET erledigt, über *call gate* Deskriptor, vgl. 3.4 Adressierung. Dabei wird Funktion des Prozess-Rettens/ Restaurierens teils in Mikrocode, teils in angesprungenes Segment verlagert.

Ähnliche Struktur: Übergang in Unterbrechungsbehandlungen

Typisch durch Aufnehmen eines Unterbrechungs-Signals (*interrupt request*), aber auch durch Befehl (programmierte Unterbrechung). Rückkehr durch (i.A. privilegierten) Befehl.

Beispiel Pentium II:

INT, INTO, BOUND
IRET

Aufruf eines Mikroprogramms (letzter Leitwerksbefehl):

(z.B. VAX: XFC) Motive:

- Der Kunde hat sein eigenes Mikroprogramm für eine Operation(sfolge) geschrieben (*wehe ihm, er arbeitet ohne Netz im ungeschützten Prozessor, den er nicht versteht*)
- Der Hersteller will die Konkurenz abschütteln, indem er Schlüsselbefehlsfolgen ins dunkle innere des Prozessors verlagert
- zeitkritische Befehlsfolgen werden schneller (Abrufphasen entfallen)

Adresse im Befehl liefert Sprungziel im Mikroprogrammspeicher. Dort muß die korrekte Funktion und Rückkehr realisiert sein (Befehlsadresse setzen, Abrufphase (Mikroprogramm) aufrufen).

3.3.5 Rechenwerksbefehle

Einstellige Operationen Form: $y := f(x)$

z.B. Datentypkonversion

Kehrwert, Vorzeichenumkehr, Betragsbildung, Wurzel, Zählen

Normalisierung einer Gleitpunktzahl

Komplementierung (binär)

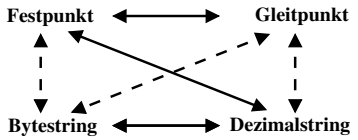


Abbildung 3.26: Datentyp-Konversion

Realisierbar als Zweiadress-Formate, Einadress-Formate (mit Überschreibung des Arguments x), Nulladress-Formate (mit zusätzlicher Implizierung)

Zweistellige Operationen Form $y := f(x_1, x_2)$

Vergleiche das zuvor über 3/2/1/0-Adressprinzip gesagte.

z.B. Grundrechenarten + - * / für

- kurze/lange Formate

- Festpunkt, Gleitpunkt, Bytestring, Dezimalstring

Boolesche Operationen: bitweise $\vee, \wedge, \oplus, \ldots$
Shifts

- mit/ohne Vorzeichenberücksichtigung (arithmetische, "logische" Shifts)
- zirkulär oder mit null/eins Vorzeichenauffüllung
- Stellenzahl

Beispiel zu Rechenwerksbefehlen R10000 ähnlich wie bei Pentium:

- Operationen auf allgemeinen Registern:
  Festpunkt, Boolesche Operationen, Shifts
  MIPS R10000: 32b, 64b befehlsgesteuert
  (Pentuium II: 16b, 32b modusgesteuert)
- Operationen auf Gleitpunktregistern (*"floating point unit"*):
  Festpunkt, Gleitpunkt, Konversion
  MIPS R10000: 32b, 64b
  (Pentium II: 32b, 64b; Gleitpunkt auch 80b; *packed decimals* 80b)

Floating point unit ist (Kompatibilitäts-) Erbe des früher aus technologischen Gründen nicht im Chip integrierten "arithmetischen Coprozessors" (kein Prozessor, sondern Rechenwerk!).
Heute: Gleitpunkt-Pipeline (andere Ablaufstruktur als "Festpunkt"-Pipeline)

R10000: Operationen auf allgemeinen Registern

Addition:	ADD, ADDI, ADDU, ADDIU	
Subtraktion:	SUB, SUBU	U:Arithmetik ohne Überlauf
Multiplikation	MULT, MULTU	I: *immediate operand*
Division:	DIV, DIVU	32b
Addition:	DADD, DADDI, DADDU, DADDIU	
Subtraktion:	DSUB, DSUBU	U:Arithmetik ohne Überlauf
Multiplikation	DMULT, DMULTU	I: *immediate operand*
Division:	DDIV, DDIVU	64b

Die Multiplikationen und Divisionen hinterlassen doppelt lange Ergebnisse in HI-/LO-Registern.
Die Befehle:

MFHI MTHI MFLO MTLO

holen/bringen Werte in diese Register.

Shifts: 15 Befehle aus folgender Menge:

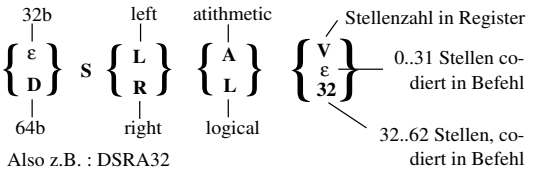


Abbildung 3.27: R10000: Shifts

R10000: Operationen auf den "floating point" Registern

Die Operationen sind Typ-gebunden:

S: 32b Gleitpunkt (23b Mantisse, 8b Charakteristik (Basis2))
D: 64b Gleitpunktzahl (52b Mantisse, 11b Charakteristik (Basis2))
W: 32b Festpunktzahl
L: 64b Festpunktzahl

Insgesamt 59 Befehle. Auswahl:

laden/speichern	32b Wort
	64b Doppelwort

laden/speichern mit Indizierung (32b, 64b)

Transporte zwischen allgemeinen Registern und "floating point" Registern bzw. Steuerregistern der "floating point" Einheit

ADD.*fmt*	C.*cond.fmt*
SUB.*fmt*	RECIP.*fmt* Näherung x^{-1}
MUL.*fmt*	RSQRT.*fmt* Näherung $\sqrt{x^{-1}}$
DIV.*fmt*	MADD.*fmt* Skalarprodukt
ABS.*fmt*	(Muliplikation, Summation)
NEG.*fmt*	SQRT.*fmt*

4 Konversionen (von beliebigem typmarkierten Wort nach S,D,W,L)

8 Rundungsoperationen

Einfache und bedingte Transporte zwischen den "floating point" Registern

Bedingte Transporte zwischen den allgemeinen Registern abhängig vom Ausgang von "floating point" Operationen.

Bedingte Sprünge

3.3.6 Operationen an höheren Datenstrukturen

Bitstrings: Vergleiche, Einbringen/Herausnehmen von Teilstrings, Suchen der ersten "Eins". [VAX: 5 Befehle]

Queues ((FIFO)-lieare Listen): Einsetzen/Herausnehmen eines Elementes [VAX: 2 unteilbare Befehle]

Semaphore: Prüfen und ändern einer Größe (p-Operation, unteilbar) [VAX: 3 Befehle]

Byte-String: Vergleiche, Zeichen(gruppe) finden, String byteweise an Tafel übersetzen. [VAX 6 Befehle, auch Pentium]

Felder : Berechnung der zu einem Index (n-Tupel) gehörenden Adresse; Prüfung auf Adress-Grenzen [VAX 1 Befehl]

3.3.7 Ein/Ausgabe-Befehle

Zwei grundverschiedene Ansätze:

1. *memory-mapped i/o*: Die E/A-Steuerwerke besitzen adressierbare Zellen, die vom Prozessor wie Hauptspeicherzellen lesbar/beschreibbar sind.

   ⇒ Es gibt keine besonderen E/A-Befehle, sondern nur Transporte.

   [R10000; bei Pentium II und vielen anderen Mikroprozessoren überwiegend gebraucht]

   Ein/Ausgabe über **blocktransportierende, gleichzeitig zum Prozessor arbeitende Werke** (Kanäle, E/A-Prozessoren):

   - Blocktransportbefehle
     ein/aus, Modus, Speicheradresse, Blocklänge, (Unter)Kanaladresse, Geräteadresse; faktisch Systemaufruf mit Versorgungsblock.

     Aus diesen baut u.U. das Betriebssystem eine Befehlskette für E/A.Prozessor (d.h. keine CPU-Befehle!)

2. *i/o-space-mapped i/o*: siehe Beispiel

Beispiel Pentium III:

memory-mapped i/o: Verwendung der MOVE-Befehle zwischen Zellen der Gerätecontroller (für Daten, Steuerwörter/Zustandswörter) und Registern oder Speicher (MOVS-Befehl).

i/o-space-mapped i/o: Verwendung eines getrennten *i/o-space* von 64kB, aber auf demselben Bus wie *address-space*, Befehle IN, OUT benutzen EAX-Register, Befehle INS, OUTS transportieren Bytestrings vom/zum Speicher

Für Minimal-Ein/Ausgabe (Kleinstkonfigurationen, Urladen, Testprogrammkern) gibt es vielfach Einfach-Ein/Ausgabe direkt von der CPU (nicht über Bus oder Kanäle)

3.4 Adressierung

3.4.1 Wert statt Adresse
3.4.2 Adressierbares Format
3.4.3 Programm-, Prozess-, Maschinenadressen
3.4.4 Transformation von Programmadressen in Prozessadressen
Substitution, Indizierung, Adressierung relativ zu Basisadressen, relativ zu Befehlszähler
3.4.5 Transformation von Prozessadressen in Maschinenadressen
"verdeckte" Basisadressen, Seitenadressierung
3.4.6 Unabhängige Adressräume: Segmente
3.4.7 Zugriffsschutz im Hauptspeicher

In der Transformation der in den Befehlen enthaltenen Adress-Angaben ("Programmadressen") liegen wesentliche Freiheitsgrade heutiger Programmier- und Betriebstechniken!

Literatur:
[TANENBAUM A 90] S.261-281, 321-352
[JESSEN E 75] S.88-104
[HENNESSY PATTERSON 96] S.94-103, 432-448

3.4.1 Wert statt Adresse

Oft ist es unnötig und unpraktisch, einen Wert zur Berechnung erst mittels einer im Befehl enthaltenen Adresse aus dem Speicher zu holen. Man kann dann den Wert statt der Adresse (*immediate operand*) im Befehl unterbringen. Zwei wichtige Einschränkungen:

- Wert muß in das Adressformat passen, oder wechselndes Format des Befehls
- Wert darf sich nicht ändern (sonst Änderung des Programms)
  - unsicher
  - Probleme bei Ausschnitt ($\rightarrow$ 2.7, sharing, Rekursion)
  - Probleme bei überlappt arbeitenden Prozessoren ($\rightarrow$ 3.9)

Also: Adressfeld im Befehl als (meist kurzer) konstanter Wert.

Sprechweise:
Wert (Operand): Objekt der Referenzstufe 0
Adresse: Objekt der Referenzstufe 1

Beispiele Pentium, R10000:

Einzelne Befehle führen auf einer Operandenposition einen Wert. Viel schlechter als VAX, wo in jedem Befehl auf Operandenposition ein Wert statt Adresse stehen kann. Hier gilt kein RISC Argument!

3.4.2 Adressierbares Format

Bis Ende der 60er Jahre

- Kurzwortmaschinen (12,16,24b) für kleine Leistung
- Langwortmaschinen (32,48,64b) für hohe Leistung; 2 Befehle je Wort

Byteadressierung (8b) seit IBM/360 (1965), heute ausschließliche Lösung: Bytes für Zeichenstrings, Befehle variabel oder 32b, Wortoperanden 8, 16, 32, 64b.

3.4.3 Programm-, Prozess-, Maschinenadressen

Programmadressen: in Befehlen und als Adresswerte im Programm vorliegende Adressen	Ebenso, aber Adressen in mehreren Adressräumen (→3.4.6)
Nach Vorgaben des Programms erzeugt der Prozessor aus Programmadressen Prozessadressen (→3.4.4), mittels: Indexmodifikation, Substitution, (Befehlszähler)relativer Adressierung, "offener" Basisadressierung. Hauptziel: Operanden/Ergebnis/Sprung-Adressen erst zur Laufzeit festlegen; Nebenziele: kurze Programmadressen, Erleichterung des Bindens	Ebenso, zusätzlich evtl. Abbildung vieler Programmadressräume in einen Prozessadressraum; mittels: "offener" Basisadressierung. falls nicht, dann:
Prozessadressen (effektive Adresse): im Prozessor benutzt	Ebenso, aber Adressen in mehreren Adressräumen (→3.4.6)
Nach Vorgaben des Betriebssystems erzeugt der Prozessor aus Prozessadressen Maschinenadressen (→3.4.5), mittels: "verdeckter" Basisadressierung, Seitenadressierung. Hauptziele: beliebige Lage des Programms und seiner Werte/ partielle Lagerung im Speicher	Ebenso, zusätzlich Abbildung vieler Prozessadressräume auf einen Maschinenadressraum; mittels "verdeckter" Basisadressierung, Seitenadressierung
Maschinenadressen: vom Prozessor gegenüber Hauptspeicher benutzt	

nun vorgehen:
Einführung der grundlegenden Adressierungstechniken als Lösungen zu programmiertechnischen, betrieblichen und anderen Forderungen.
Zunächst einzeln, mit Beispielen in Notation von *ad hoc* erfundenen, meist graphischen Notationen. Dann Übergang zur VAX, als Beispiel für ein System aufeinander abgestimmter Modifikationsformen.

Vorbemerkung zur Transformation von Programmadressen in Prozessadressen:

Die "nichtprivilegierten" Modifikationen treten situationsabhängig auf (z.B. Adressierung auf dem Stack anders als im übrigen Speicher). Wenn sie in den Befehlsablauf integriert werden (wie meist im Folgenden angenommen), ergeben sich daher stark schwankende Ablaufstrukturen und Zeitbedarfe (auch gleichartiger!) Befehle. Das führt zu schlechtem Pipelining, RISC-Prozessoren ersetzen daher diese Modifikationen durch Vorbefehle ("brauchen ja nur je einen Takt"), die die Adresse im Register aufbauen, über das dann indirekt auf den Speicher zugegriffen wird.

Die Funktion der Modifikation bleibt dieselbe! Sie wird nur auf mehrere Befehle verteilt (speicheraufwendig)!

3.4.4 Transformation von Programmadressen in Prozessadressen

Forderung I: Der Rechenprozess muß die Programmadressen für Operanden, Ergebnisse, Sprungziele zur Laufzeit ändern können

- Unterprogrammparameter/Rücksprünge
- Mehrfachinkarnationen (Rekursion, dynamische Datenstrukturen)
- Datenstrukturen unbekannten Platzbedarfs (dynamische Felder)
- Berechnete Indizies in Feldern

d.h. in Prozessadressen transformieren und zwar ohne Änderung der Befehle!

Lösung 1, Lösung 2:

3.4.4.1 Indexmodifikation (Indizierung) {Hier: fallweise angewendete additive Änderung der Adresse, vorgegeben durch Interpretationsprozess; anders: Basisadressierung (additive Korrektur, falls stets angewendet)}

3.4.4.2 Substitution (Ersetzung)

3.4.4.1 Indexmodifikation

Indexmodifikation ist eine additive Korrektur der Programmadresse um den Index, fallweise angewendet. Prozessadresse := Programmadresse+Index. Es gibt zwei Varianten:

1. **Index ist Relativgröße**, invariant gegen Lage im Programmadressraum (eigentlicher Index), Programmadresse im Befehlswort ist lageabhängig
   - Index in einem Feld (*array*); typabhängige Schrittweite (skalierter Index) günstig!
2. **Index ist lageabhängig**, im Befehlswort steht eine Relativgröße(!), (Index wie Basisadresse)
   - Parameterzugang, Rücksprungadressen
   - Zugang zu Variablen im Stack

Indexmodifikation verlangt: **im Befehlsformat**:

- Angabe, daß zu modifizieren ist: Extrabit, am besten der Adresse zuzuordnen, eventuell im Operator
- Angabe, welches (Index-)Register zu benutzen ist

im Prozessor

- (Spezielle Indexregister (veraltete Lösung, die im Allgemeinen die Einführung besonderer Indexbefehle, wie laden, speichern, Arithmetik, Test verlangt))
- allgemeine Register, die zugleich als Indexregister dienen.

Beispiel Index ist Relativgröße:

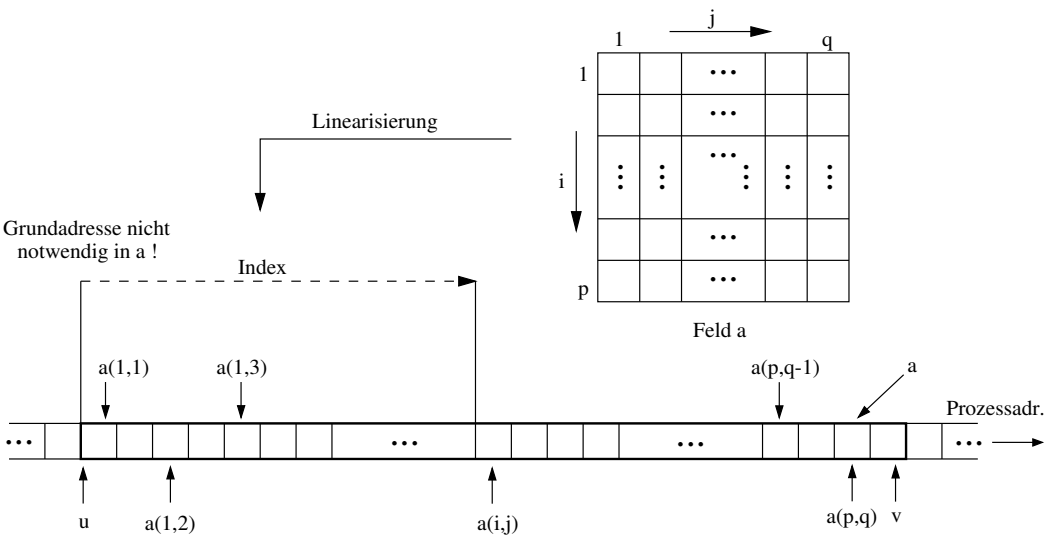


Abbildung 3.28: Index ist Relativgröße

Indizierte Variable a[i,j] adressieren zu
a: *array*[1..p,1..q] *of* t
t brauche $platzbedarf(t)$ Zellen
a) Index := $((i-1)*q+j-1)*platzbedarf(t)$ (rechnet i.A. das Programm)
b) Prozessadresse := Grundadresse + Index (rechnet der Prozessor)

- Vorsicht, daß $i \in [1..p] \wedge j \in [1..q]$!
- $platzbedarf$ erscheint bei b) im Falle des skalierten Index

Bemerkungen zu indizierten Variablen:

- skalierter Index: Adressierung in Vielfachen des Platzbedarfs ist günstig, vor allem bei Vielformat-Maschinen
- Prüfung auf Einhaltung der Adressgrenzen ist wichtig!
- arithmetische und Testoperationen am Index (=Festpunktgröße), z.B. auch falls Index zugleich Laufvariable in Schleife
- Information über Platzbedarf(t), Grundadresse, Stufigkeit, Indexgrenzen, Adressgrenzen (u,v) bildet Felddeskriptor, kann maschinengestützte Datenstruktur sein; dann Auswertung bei Zugriff durch Mikroprogramm.

Beispiel 1 Index ist lageabhängig:

```
type personalbogen = record name:
                         array[1..20] of
                         character;
                         ...
                         gehalt: real;
                         ...
                     end;
var personaldatei: array[1..1250] of personalbogen
```

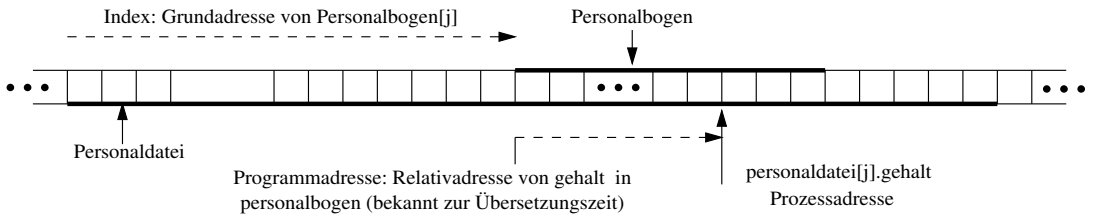


Abbildung 3.29: Index ist lageabhängig

Hier wäre auch ein doppelter Index denkbar:
Index1 := Grundadresse (*personaldatei*)
Index2 := j*Platzbedarf(*personalbogen*)
Prozessadresse := Programmadresse + Index1 + Index2

Beispiel 2 Index ist lageabhängig:

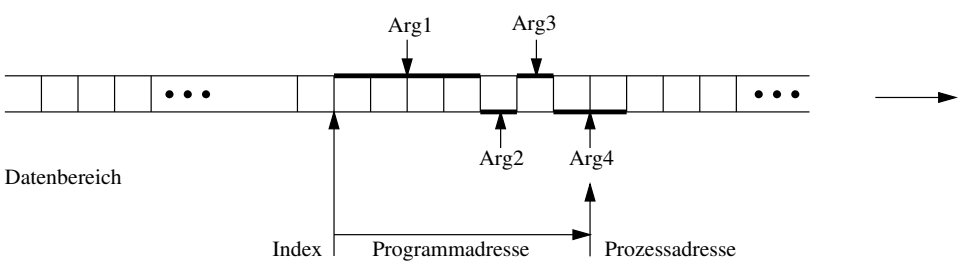


Abbildung 3.30: Beispiel 2: Index ist lageabhängig

Index als Grundadresse der Argumentliste zu einem Prozeduraufruf. Abb.3.32

Index: Grundadresse, ist aufrufabhängig, u.U. zur Programmier/Übersetzungszeit unbekannt
Programmadresse: Relativgröße, ist wegen spezifizierter Argumentliste/Typen zur Programmier/Übersetzungszeit bekannt: unveränderlicher Adressteil im Befehl!

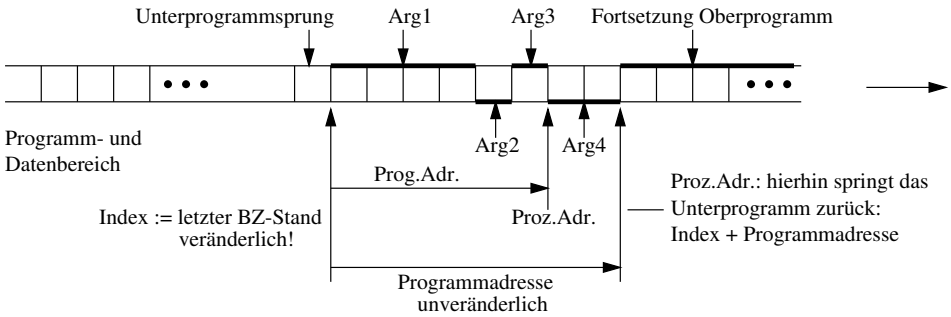


Abbildung 3.31: Beispiel 2: Kombination mit Rücksprungadresse

Beispiel 3 Index ist lageabhängig

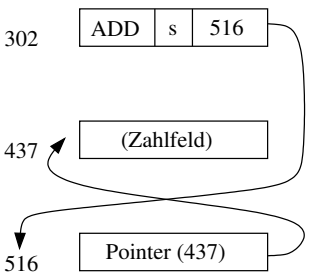
Laufzeitstapel (-Keller, *runtime stack*) für Programme mit (Block-) Prozeduraufrufschachtelung (inkl. Rekursion).
Wie aus Einführung in die Informatik bekannt, wird für jeden Block/Prozedur-eintritt/aufruf eine Datenstruktur Aufrufsatz instanziert

Aufrufsatz (AS, *activation record*), besteht aus:

Organisationsgrößen (Grundadresse des Vorgänger-AS und des umfassenden nach der Ordnung der textuellen Schachtelung (d.h. die globalen Größen definierenden)
Lokale Variablen (soweit Platzbedarf zur Übersetzungszeit bekannt))
Parameter (Versorgungsblock)
Registerinhalte einschließlich Befehlszähler (zu retten bei Folgeaufruf)
Platzbedarf für lokale Variable, sowei erst vor Eintritt bekannt
Weiteres (z.B. Ausdrucksberechnungskeller)

3.4.4.2 Substitution

(Adressersetzung, indirekte Adressierung; als Maschinenfunktion von H.Schechter (TUM) 1956 eingeführt)
Programmadresse bezeichnet nicht die Zelle des Wertes/Sprungziels, sondern die Zelle der Adresse des



ADD s 516 maschinentechnisches Abbild von (Pascal) q+p↑.zahl, wobei p in Zelle 516, Zahlenfeld in Zelle 437 (1.Feld im durch p↑ bezeichneten Record)

Abbildung 3.32: Beispiel: Substitution

Wertes/Sprungziels (dies ist also ein Objekt 2. Referenzstufe!) Substitution ist bei früheren Maschinen auch iterierbar gewesen (DEC 10/20).

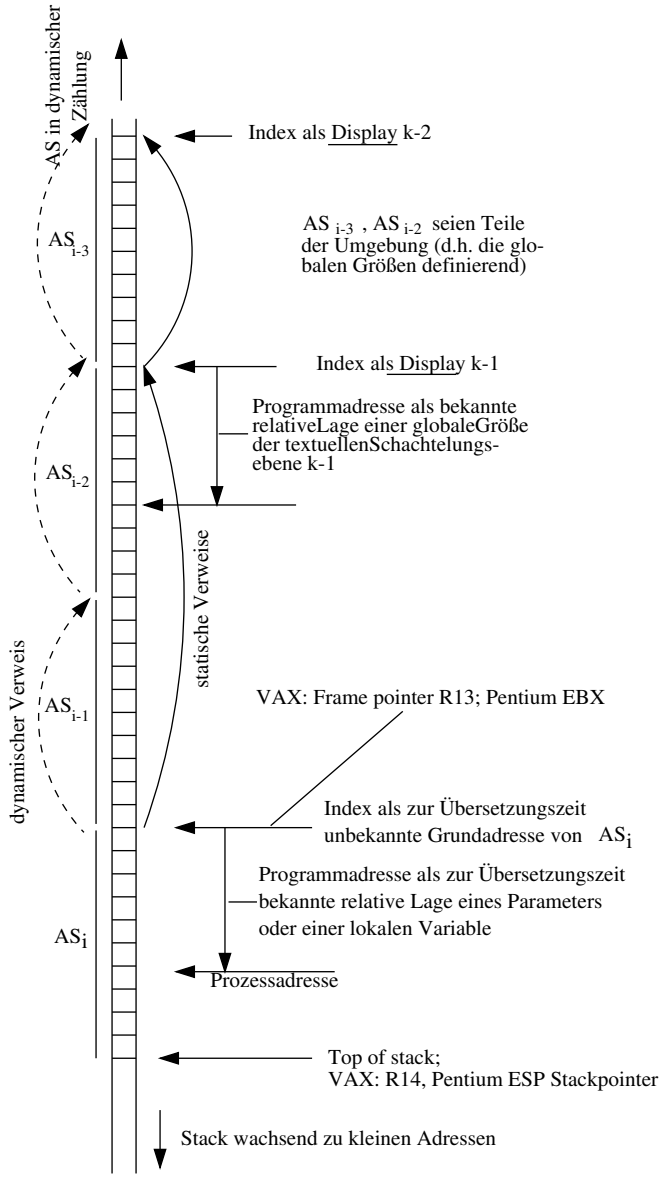


Abbildung 3.33: Beispiel 3: Laufzeitstapel aus Aufrufsätzen

Substitution über Register besonders wichitig:

- spart Bits im Befehl (wichtig bei langen Adressen)
- RISC-Prozessoren erlauben keine gelegentliche (Pipeline!) komplexe Adressmodifikation ⇒ Adresse durch Vorbefehlsfolge in Register aufbauen!

Beispiel 1 Konsekutive Anwendung von Indexmodifikation und Substitution:

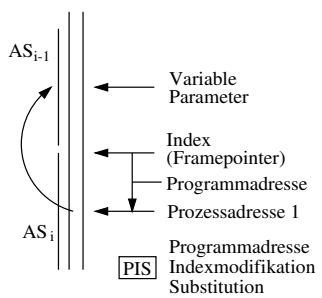
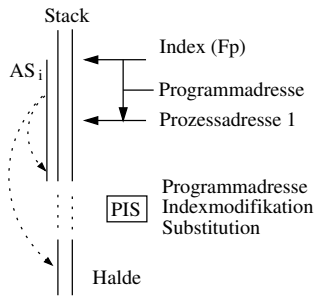


Abbildung 3.36

Zugriff auf Variablenparameter aus dem AS_{i-1}:

Programmadresse: Relativposition
Index: Grundadresse (AS_i)
Prozessadresse1: zeigt auf Pointer (Adresse), substituiert
Prozessadresse2: dort Variablenparameter

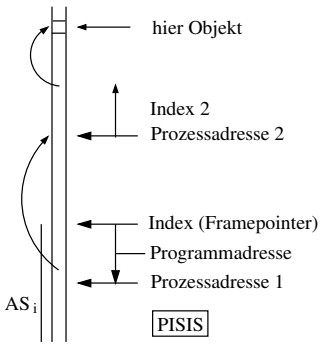
Beispiel 2 Konsekutive Anwendung von Indexmodifikation und Substitution:



Zugriff auf Variable unbekannten Platzbedarfs im Stack oder auf Halde

Abbildung 3.35: Beispiel 2

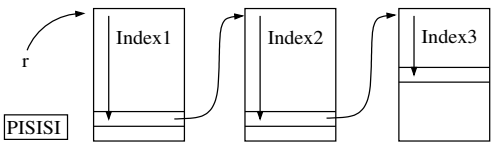
Beispiel 3 Konsekutive Anwendung von Indexmodifikation und Substitution:



Zugriff auf ein Objekt über indizierte Pointervariable in einem Array, das als Reference-Parameter übergeben wurde.

Abbildung 3.36: Beispiel 3

Beispiel 4 Konsekutive Anwendung von Indexmodifikation und Substitution:



Es gab Maschinen mit Index/Substitutionsfolge beliebiger Tiefe: DEC 10/20.

Abbildung 3.37: Beispiel 3

3.4.4.3 Kurzadressen

Forderung II: Kurzadressen innerhalb der aktuell zugreifenden Teilmenge der Hauptspeicherzellen (Nutzung der Lokalität)

(keine allgemeine Lösung): Substitution über Register. Wegen Ladens der Register und wegen kleiner Registerzahl nur für geringe Zahl von Adressen akzeptabel.

Lösung3: Befehlszähler-relative Adressierung. Alle (oder einige) Adressen sind als Relativangaben zum Befehlszählerstand zu verstehen.

- oft bei (kurzen) Sprüngen
- Auch für Variablen-Adressierung geeignet, aber dann Vermischung von *pure* und *impure part* schwer vermeidbar. Variablen-Adressierung für Compiler aufwendig!

Lösung 4: Jeder Zugriffstyp greift in einen Adress-Unterraum, z.B. für Programm, Stapel, Halde (z.B. PDP11, Intel 8086). Notbehelf. Vgl. auch →3.4.6, ("Segmente")

Lösung 5: Offene Basisadressen (relative Adressierung); zu jeder Wert- oder Sprungadresse wird eine Basisadresse angegeben, ihr Wert wird addiert. Eigentliche Adresse ist lageunabhängiger Relativwert, z.B. nur 12Bits bei 24Bit Basisadresse.

Klassisches Beispiel: IBM /360 und Nachfolgende. Eine Maschine mit verschiedenen Befehlsformaten, z.B. RX-Format:

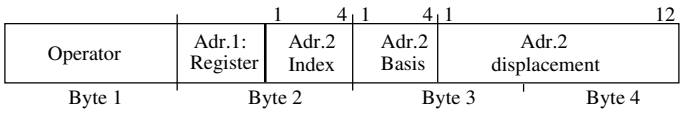


Abbildung 3.38: RX-Format

In 16 allgemeinen Registern stehen aktuelle Operanden, Ergebnisse, Indizies, Basisadressen
Prozessadresse2 := Adresse2 + <Adr.2Index> + <Adr.2Basis>

Da die Basisadresse 24 oder 32b lang ist, werden je Adressierungsangabe 8 oder 16b gespart, aber dafür Aufwand für Laden der Basisadresse, Belegung eines Registers!

3.4.4.4 Unterstützung des Bindens (*linking*)

Forderung 3: Die Adressierung unterstütze das Binden.

Binden: (*linking,* Montage): Aufbau eines zusammenhängenden linear adressierten Programms (mit Werten) aus Moduln, die je von null bis zu einer individuellen Obergrenze adressieren. Dazu sind:

- alle lageabhängigen Adressen additiv zu korrigieren nach gültiger Anfangsadresse des Moduls
- alle Querbezüge (aus dem Modul auf *externals*) endgültig zu adressieren.

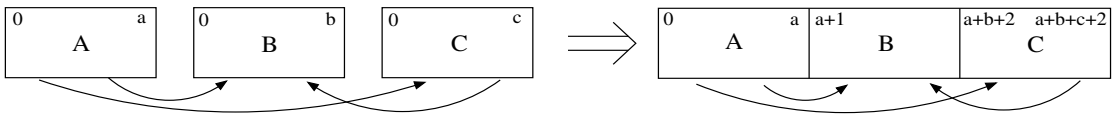


Abbildung 3.39: Binden

Binden findet vor dem Programmlauf statt, anhand von Compilerprotokollen über lageabhängige Größen und *externals*.

Lösungen für die Unterstützung des Bindens:

> Befehlszählerrelative Adressierung erübrigt die additive Korrektur der Adressen, nicht das Einsetzen der externen Referenzen; (*position independent code, PIC*)
>
> Adressierung über offene Basisadressen reduziert die additive Korrektur auf die Korrektur der Basisadressen.

Wiederaufnahme der Überlegungen:
Segmentweise Adressierung (Binden auf die Laufzeit verschieben) →3.4.6

ähnliches Problem: Laden (Fixieren).→3.4.5

3.4.4.5 Beispiele zur Bildung der Prozessadresse (effektiven Adresse):

Beispiel R10000:

> Die bedingten Sprünge sind Befehlszähler-relativ. Der unbedingte Sprung ist Befehlszähler-relativ oder ersetzt die hinteren 28b des aktuellen Befehlszählerstandes des Folgebefehls oder indirekt über Register.

Die Lade-/Speicherbefehle adressieren mit dem Inhalt eines (beliebigen) Registers + 16b Offset (im Befehl): also offene Basisadresse (aber Segment nur 64kB!). Lageabhängiger Index muß also im Register bereitgestellt werden!

Die Lade- und Speicherbefehle für die "Gleitpunkt"-Einheit erlauben auch Indizierung über ein zweites Register.

Beispiel Pentium II (hier 32b-protected mode):

Viele (nicht alle!) Befehle erlauben Werte/Sprungziele im Speicher, dann typisch anderthalb-Adressformat.
Steuerung der Modifikation über Mode-Byte im Befehl:

Mode: | MOD 2b | REG 3b | R/M 3b | Ein"Adressspezifikator"

R/M: gibt an, ob für die Adresse EAX, ECX, EDX, EBX, ESP, ESI, EDI benutzt wird oder ob ein SIB-Byte folgt

MOD: bestimmt mit R/M folgende Adressierungstypen:

- Register
- Register indirekt (aber nicht SP, BP!)
- Register mit Displacement 8b oder 32b (aber nicht SP!)
- absolute Adresse

SIB: Skalenfaktor, je ein Register als Basis und als Index: skalierter Index mit und ohne Displacement

Beispiel Digital Equipement VAX

Außerordentlich wirksame Transformationen von Programmadressen auf Prozessadressen: abgesehen von den Befehlszähler-relativen Adressen bei bedingten Sprüngen ist an jeder Adressposition eine von 12 Adressierungsformen möglich. An der Adressposition:

operand specifier: 1 oder 2B
operand specifier extension (0 bis 8B, für Adressen oder Werte)

Orthogonalitätsprinzip: jede sinnvolle Kombination von Operationscode und Adressierung anbieten! (vergleiche dagegen Pentium-Einschränkungen!)

Übersicht über die VAX-Adressierungsmodi:

register mode: Registerzugriff (über Kurzadresse)
register deferred mode: Substitution über Register
autoincrement mode: Substitution über Register, dann formatrichtiges Weiterschalten (ahmt häufigen Indexgebrauch nach; POP-Operation (Stack nach unten!)); mit Befehlszähler: Konstante statt Adresse
autoincrement deferred mode: Substitution über Register, dann über Zelle, dann weiterschalten um Adressformat (4);
mit Befehlszähler: unmodifizierte Adresse
literal mode: kurze Konstante
autodecrement mode: formatrichtiges Zurückschalten, dann Substitution über Register (Ahmt häufigen Indexgebrauch nach; PUSH)
displacement mode: Indizierung
mit Befehlszähler: Befehlszähler-relative Adressierung
displacement deferred mode: Indizierung, dann Substitution
index mode: skalierter Index

VAX-Modifikationen stellen extreme Position dar! Compilerfreundlich, kompakter Code aber Pipeline-feindlich!

3.4.5 Tranformation von Prozessadressen in Maschinenadressen

3.4.5.1 Keine Tranformation durch Prozessor

Im einfachsten Fall sind die Prozessadressen bereits die Maschinenadressen. Zwei unangenehme Folgen:

- Wenn das Programm lageabhängige Größen enthält (nicht zu modifizierende Adressen, offene Basisadressen, lageabhängige Indizies, Substitutionsadressen), dann ist das Programm auf den **Ausführungsort festgelegt**. Nicht akzeptabel bei Mehrprogrammbetrieb im Hauptspeicher! Abhilfe programmiert durch "Lader". (kein Problem bei Befehlszähler-relativer Adressierung!)
- **kein** (veränderlicher) **aktueller Ausschnitt** des Prozessadressraums im Hauptspeicher. Abhilfe programmiert durch "Overlay"(→ 3.4.5.4) (Kein Problem, wenn Hauptspeicher ausreichend groß!)

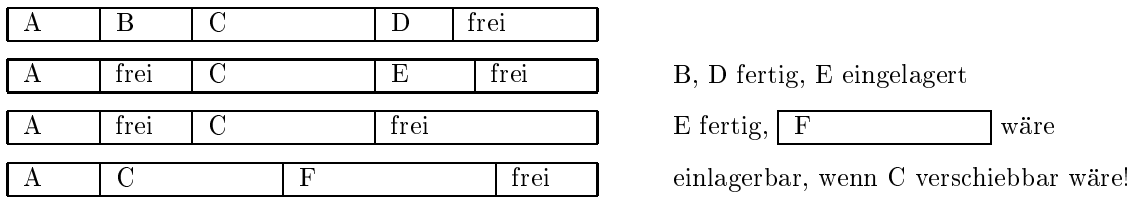
Lader: (Fixierer, *loader*) benutzt -wie Binder, mit dem er oft ein Programm ist, vom Assembler/Compiler hinterlassene Tabellen, die angeben wo im Programm und seinen Werten lageabhängige Größen (nicht zu modifizierende Adressen, offene Basisadressen oder(!) lageabhängige Indizies, Substitutionsadressen) stehen. Typisch erzeugen Assembler/Compiler Programme, die Prozessadressen von null aufsteigend generieren; dann genügt die Addition der Anfangsadresse des Programms (im Maschinenadressraum) auf alle lageabhängigen Größen.

Forderung III: Unterstützung des Laders.

Bereits bekannt: offene Basisadressierung (vgl. das zum Binden gesagte).

und übrigens: bei Befehlszähler-relativer Adressierung entfällt das Laden.

Noch schöner wäre es, wenn die Abblidung der Prozessadressen auf die Maschinenadressen veränderlich wäre: Hauptspeicher:



Das ist aber mit dem Lader nur möglich, wenn bekannt ist, wo C lageabhängige Größen führt. Da das Programm diese aber beliebig umspeichern darf, sind die vom Assembler/Compiler angelegte Tafeln wertlos, sobald das Programm gelaufen ist.

3.4.5.2 Verdeckte Basisadressen

Forderung IV: Lageinvariantes Programm

Schon bekannte Lösung: Befehlszähler-relative Adressierung. Andere Lösung: Verdeckte Basisadressierung (1962, IBM 7094)

Der Prozess benutzt einen Prozessadressraum von 0 bis n. Der Prozessor erhöht vor jedem Zugriff auf den Hauptspeicher die Prozessadresse um den Wert m der Anfangsadresse des zugeteilten Bereichs im Hauptspeicher aus einem "verdeckten" Basisregister. Dieses ist privilegiert, Inhalt nur im Systemmodus setzbar. Typisch kontrolliert der Prozessor an einem zweiten Register, ob die Prozessadresse höchstens n ist.

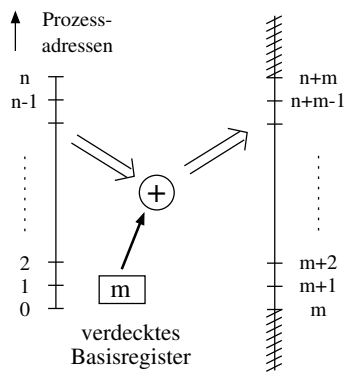


Abb. 3.42 verdecktes Basisregister

Verdeckte Basisadressierung:

- Verschiebung jetzt möglich: Transport des Programms mit seinen Werten, dann umsetzen der Basisadresse.
- Addition meist dadurch erleichtert, daß z.B. die letzten 5 oder 10 Bits der Basisadresse null sind, d.h. Programme beginnen auf einem 32/1k Adressraster. Folge: "Verschnitt" durch u.U. nicht nutzbare Hauptspeicherzonen (*"fragmentation"*).

Verschiebetechniken sind problematisch, weil

- Programme nicht immer verschiebbar sind (bei Ein/Ausgabe).
- Verschiebe-Zeitanteile lang(!) sein können.
  - schlecht: wenn Prozessor das Programm mit Blocktransport verschiebt.
  - besser: Verschiebung durch *swap* (Ausgabe/Eingabe von der Platte).
  - noch besser: Verschiebung durch blocktransportierende Steuerwerke oder Kanäle

Noch ein Nachteil der bisherigen Lösungen: Gemeinsame Speicherbereiche (für Kommunikation, für Mehrfachnutzung von Programmen) nicht für beliebige Prozesse realisierbar.

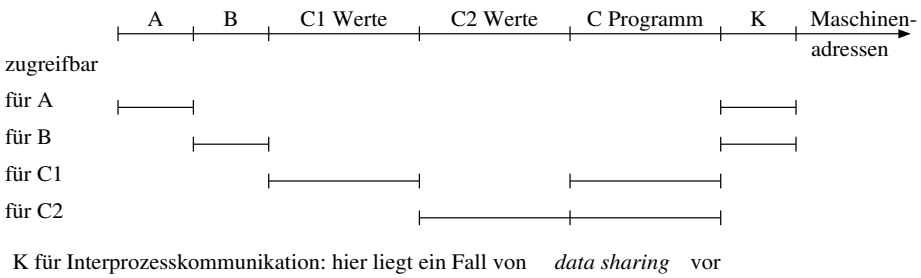


Abbildung 3.41: Zugreifbarkeit auf Hauptspeicher

Bei K liegt ein Fall von *data sharing* vor. Kommunikation wird sicherer, aber auch aufwendiger über Betriebssystem erledigt.

Bei C-Programm liegt ein Fall von *code sharing* vor. C-Programm hat in den Prozessadressräumen vom C1 und C2 dieselbe Lage! Notwendig bei direkten Selbstreferenzen in C-Programm (Sprünge).

3.4.5.3 Zerteilung: Lösung durch Seitenadressierung

Forderung V: (Zerteilte Unterbringung von Objekten im Maschinenadressraum (Hauptspeicher)! Gründe:

- Verschieben soll unnötig sein, vgl. S.91 weshalb nicht so:

A	frei	C	frei	
A	F	C	F	frei

F aufteilen

- Gemeinsame Nutzung von Speicherbereichen soll für beliebige Prozesse (nicht nur zwei im Speicher benachbarte) möglich sein.

Lösung: Seitenadressierung

(für den zweite Grund ist u.U. segmentweise Adressierung die bessere Lösung)
Der Maschinenadressraum wird in Abschnitte **fester** Größe geteilt: **Kacheln**, Seitenrahmen (*page frames, physical pages*). Typisch 2^{10}=1024 Wörter (4kB). Maschinenadresse zerfällt, z.B.

Kachel-Nr.	Zellen-Nr.

Kachel-Nr.: z.B. 1 aus $2^{14} = 16k$ Kacheln, d.h. 16M Hauptspeicherzellen
Zellen-Nr.: z.B. 1 aus 1024 Zellen (Bytes) der Kachel

Ebenso wird der Prozessadressraum in **Seiten** (*pages*) gleicher Größe geteilt. Prozessadresse zerfällt, z.B.:

Seiten-Nr.	Zellen-Nr.

Seiten-Nr.: z.B. 1 aus $2^{22} = 4M$ Seiten, d.h. 4G Hauptspeicherzellen
Zellen-Nr.: z.B. 1 aus 1024 Zellen (Bytes) der Seite

Abbildungen von Seitennummern auf Kachelnummern durch den Prozessor anhand von **privilegierten Tafeln**. Abbildung Seiten→Kacheln erlaubt die gewünschte Zerteilung im Maschinenadressraum:

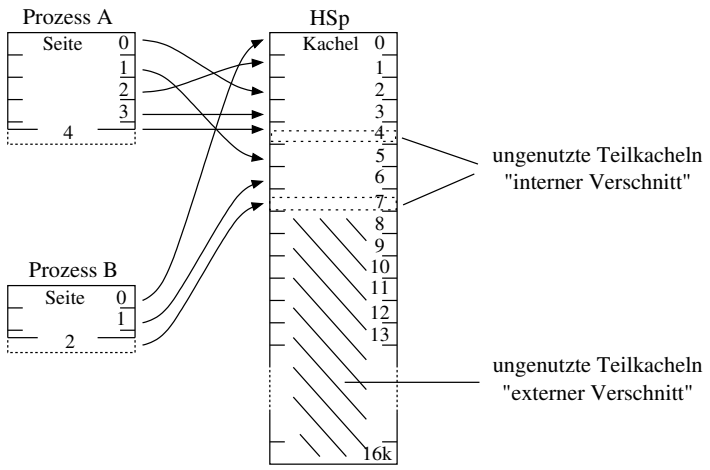


Abbildung 3.42: Seiten-Kachel Abblildung

Die Seiten/Kachel-Größe ist ein Kompromiss zwischen:

- Abwägen: kleine Seiten→Kachel-Tafeln (⇒lieber große Seiten) und geringer interner Verschnitt (⇒lieber kleine Seiten), spricht für ca. 64..256B
- Forderung, daß Seite auch Transportformat im Verkehr mit peripherem Speicher hat (lieber große Seiten)

Die Abbildung Seitennummer→Kachelnummer wird über eine dem Prozess eigene, aber ihm unzugreifbare Seiten-Kacheltafel realisiert, die vom Betriebstystem aufgestellt wird.

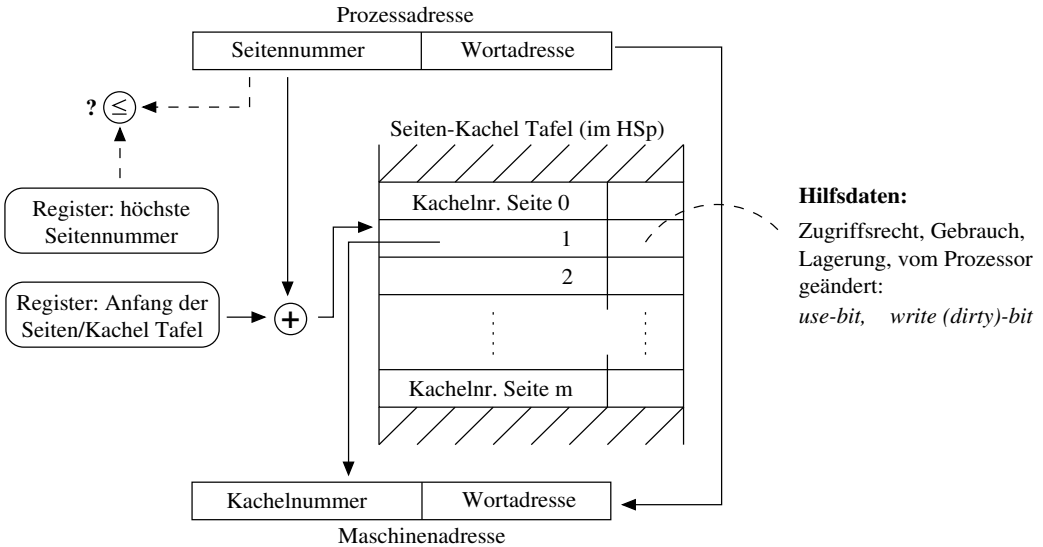


Abbildung 3.43: Funktion Seiten-Kachel Tabelle

Jeder Griff in den Prozessadressraum führt über diese Tafel (Verlangsamung!) oder (meist) 4..256 "assoziative" Register, mit den letztgebrauchten Seitennummer-Kachelnummer Paaren:

Seitennummer	Kachelnummer	Hilfsdaten
⋮	⋮	⋮

Aktueller Ausschnitt aus Seiten-Kacheltafel. (*"tanslation look-aside buffer", TLB*)

Das Register für die höchste Seitnnummer

- schützt den Prozess vor vermutlich schweren Fehlern (Zugriff auf nicht existente Objekte außerhalb des ihm zugeteilten Prozessadressraums
- schützt das System vor schwerwiegenden Fehlzugriffen in den Maschinenadressraum (indem beliebige Zelleninhalte als Kachelnummern aufgefaßt werden).

Seitenadressierung als Lösung des Zerteilungsproblems ist eine anerkannt schöne Sache, und sie löst auch noch die folgende Forderung V

3.4.5.4 Aktueller Ausschnitt im Hauptspeicher

Forderung V: Nur ein wechselnder aktueller Ausschnitt des Prozessadressenraums wird auf den Maschinenadressraum abgebildet. Dabei soll der Prozessadressenraum im Verhältnis zum Hauptspeicher

- größer (das ist die Regel)
- kleiner (bei Epigonen von Rechnern mit ungenügendem Adressraum) sein.

Lösung (keine!): Overlay

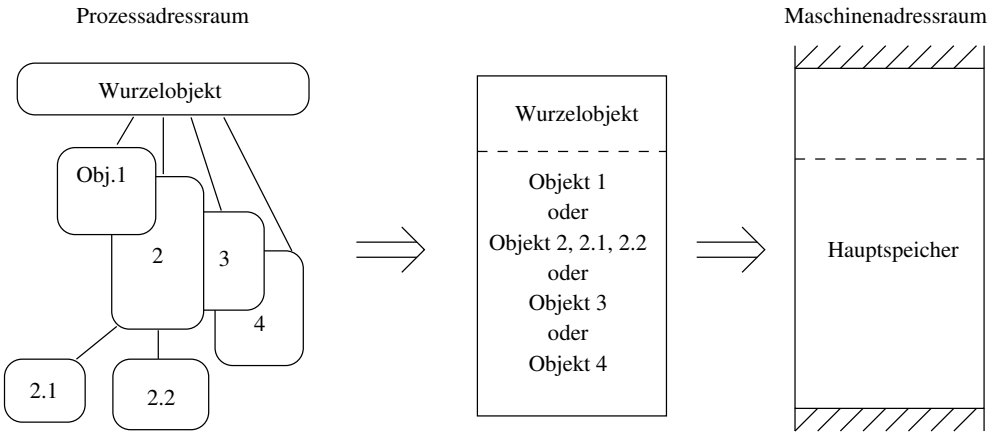


Abbildung 3.44: Overlay

Der Prozessraum wird durch Überlagerung mehrfach belegt: Eindeutigkeit nur für ein Objekt und seine Vorfahren im Baum. Kommunikation nur über gemeinsame Vorfahren. Programmaufbereitung durch Binder.

Heutige Lösung für den aktuellen Ausschnitt im Hauptspeicher:

Großer Adressraum 2^{32} .. 2^{64}B, gegliedert in

Seiten	Segmente
(Die Regel, starres Format)	(Programm/Datenstruktur bestimmt Format)
Güntsch 1957	Barton 1963
Killburn 1962 (Ferranti Atlas)	Burroughs B5000

Von diesen wird eine aktuelle Auswahl im Speicher gehalten, der als virtueller Speicher wirkt.

Lösung: Seitentausch: Seiten-Kachel-Tafel enthält Lagerklasse (Hauptspeicher, Platte). Der Versuch, eine im Hauptspeicher nicht vorhandene Seite zu erreichen, führt über einen Alarm in das Betriebssystem, das nachlädt (*demand fetching*). Eventuell Verdrängung einer Seite: dazu Gebrauchsinformation in Tafel, z.B. LRU (*least recently used*). Gleiches Verfahren bei (selten!) Segmenten möglich.

Beispiel R10000

Der Prozessor hat 3 Betriebsmodi (Privilegierungsstufen):

Kernel	(teils mit Seitenadressierung, teils direkt adressierend, Adressunterräume mit verschiedener Form der Cache-Nutzung)
Supervisor	(mit Seitenadressierung)
User	(mit Seitenadressierung)

Im 64b Adressmodus wird folgendes Format benutzt:

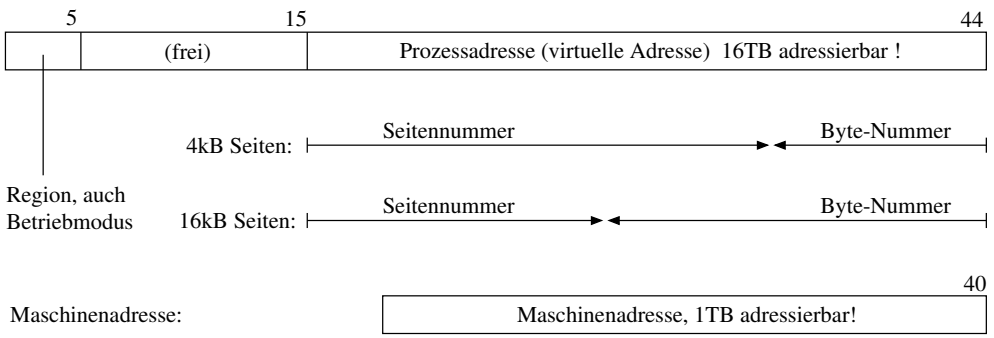


Abbildung 3.45: Adressformat R10000

Translation look-aside buffer mit 64 Eintragungen, mit Prozessnummern (erübrigt Löschung bei Kontextwechsel)

3.4.6 Unabhängige Teiladressräume

Forderung VI: unabhängige Teiladressräume

bereits kennengelernt: Binden. Die genannte Lösung durch Adresstransformation ist nur brauchbar, wenn

- vor Lauf alle Moduln nach Anzahl und Platzbedarf (bis auf einen, den letzten) bekannt sind
- die einzelnen Moduln nicht verschiedene Zugriffsart und Privilegierung untereinander haben.

Schön wäre es, wenn

- kein Binden erforderlich, Adressanpassung erst **zur Laufzeit** und
- Überwachung des Zugriffs oder Einsprungs auf fremde Segmente automatisch **zur Laufzeit** erfolgte.

D.h. der Prozessor soll Adressen verschiedener Prozessadressräume auf Maschinenadressen abbilden!

Lösung: segmentweise Adressierung: Der Prozessadressenraum besteht zur Laufzeit aus n unabhängigen Teilräumen, relativ zu null adressiert, entsprechend n Moduln mit kontrolliertem Übergang: Segmente. Größe, Zugriffsart/rechte, Gebrauchshäufigkeit i.A. segmentindividuell. Befehle enthalten nur segmentrelative Adressen, Übergang in anderes Segment im Allgemeinen explizit notwendig: Abbildungen auf Maschinenadressen:

- Segmenttafel, enthält zu jeder Segmentnummer (verdeckte) Basisadresse, Lagerklasse, Größe.
  oder

- Segmenttafel enthält statt Basisadresse Verweis auf segmenteigene Seiten-Kachel-Tafel

Erinnerung: aktueller Ausschnitt kann auch (und besser) Segmentuntermenge sein!

Gleichzeitig können mehrere Segmente implizit vereinbart sein (z.B. Code Segment, Stack Segment, Data Segment) der Wechsel solcher Segmente muß dann explizit genannt werden, z.B. mit Sonderbefehl, Nachladen Segmentregister (→Pentium)

Die Bezeichnung Segment ist geduldig und geliebt. Beispiele für anderen Gebrauch:

- die Objekte im Overlay-Baum
- "lineare" Segmentierung (IBM /360 usf...): Die ersten Adressbits sind Segmentnummer. Durch Zählen (Befehlsadresse), Modifikation kann das Segment unkontrolliert gewechselt werden.
- "Segmente" die auf offener Basisadressierung beruhen oder auf anderen Basisadressen, ohne daß nicht-Überdeckung der Segmente gesichert ist (z.B. Intel 8086)

Segmentweise Adressierung mit segmentindividueller Seitenadressierung:

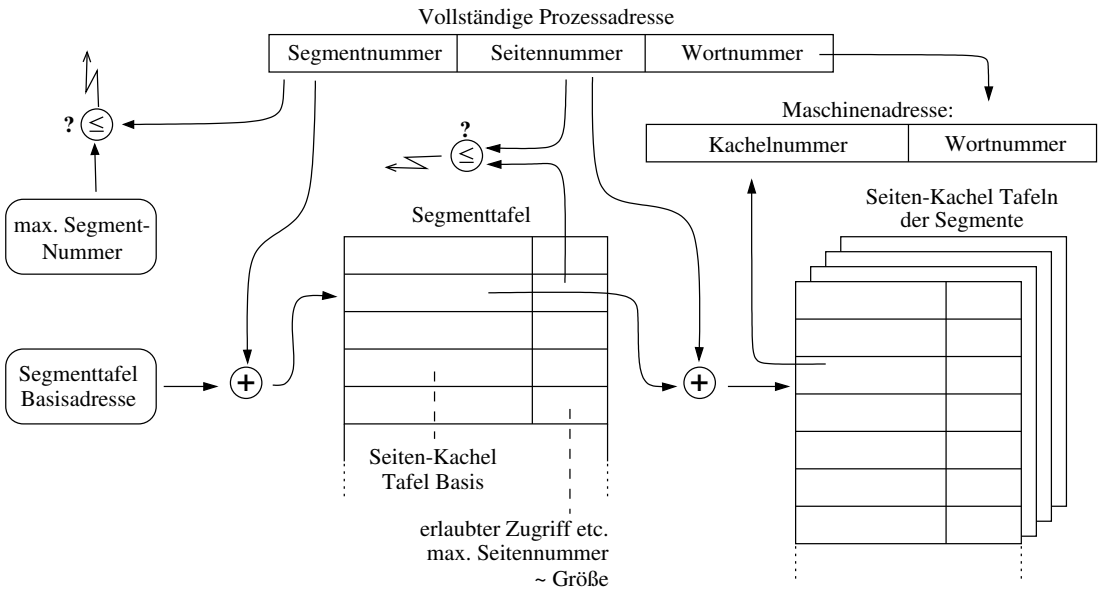


Abbildung 3.46: segmentindividuelle Seitenadressierung

Auch hierzu assoziative Register mit den aktuellen Segmentnummer-Seitennummer-Kachelnummer Entsprechungen; Tafeln u.U. teilweise im Hauptspeicher, teilweise auf Platte

Vorteil der Segment-Seiten Adressierung:

- gestreute Anordnung
- Ausschnitte aus großen Segmenten

Beispiel Segmentierung und Schutzmechanismen im Pentium III

> Vorgeschichte:
> 8080-CPU (1974) hatte 16b (64kB) Adressraum. Mit Übergang auf 8086 (1978) Erweiterung auf 20b (1MB) Maschinenadressraum, bei Aufrechterhaltung der 16b Programmadressen. Benutzung von 3 implizit bei Speicherzugriffen verwendeten Segmenten (Code, Stack, Data), deren Basisadressen vom Prozess frei wählbar in Segmentregistern speicherbar waren (damit Ausweitung auf 20b-Adresse). Dazu ein weiteres Segment (extra data), Segmentgrößen 64kB. aber keine Schutzmechanismen, kein *demand fetching*!
>
> Bei Übergang in 80286 (1982) Schutzmechanismen (etwa wie noch heute im Pentium III).

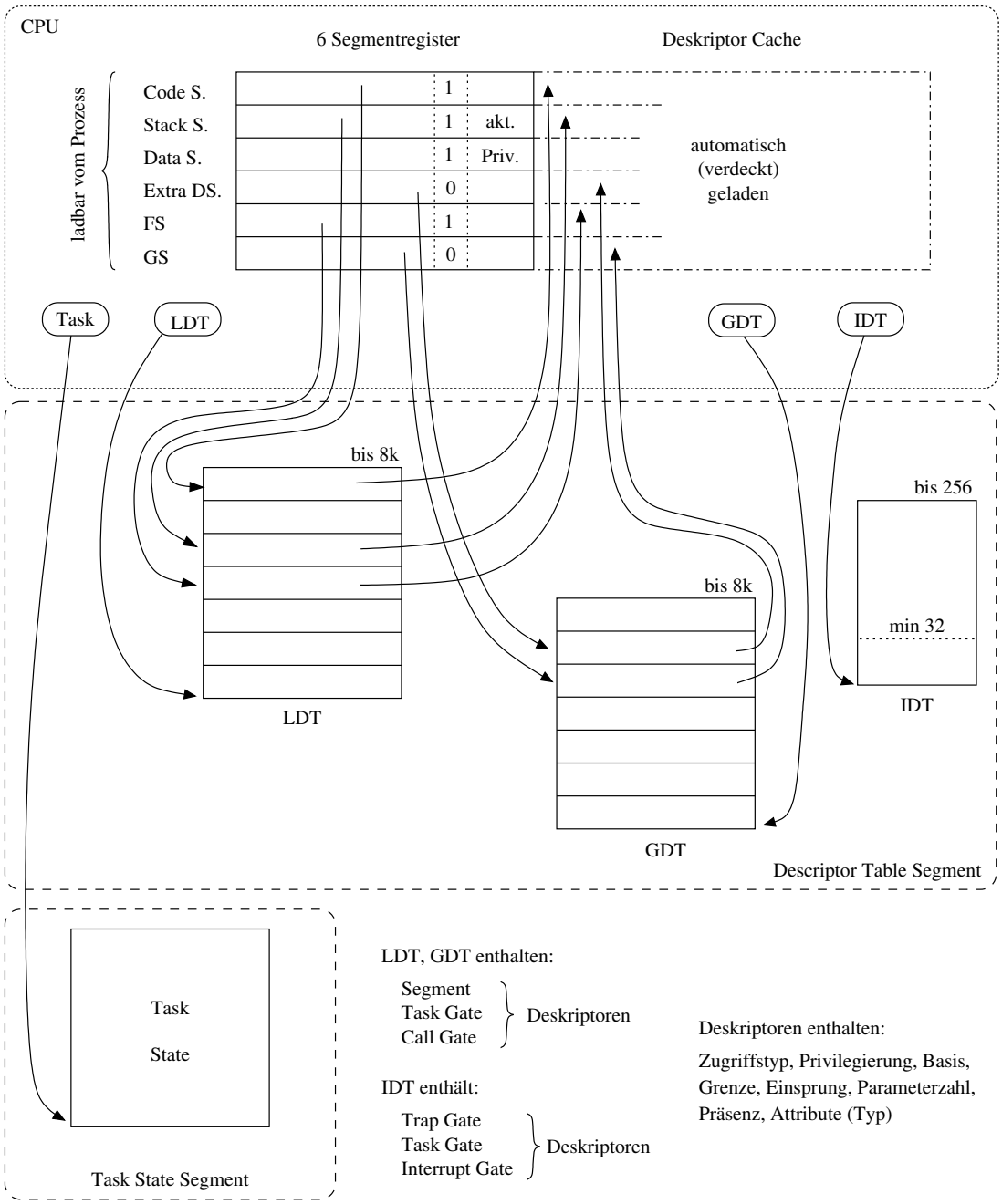


Abbildung 3.47: Segmentregister und Deskriptortabellen im Pentium II

Bei Übergang in 80386 (1985) zusätzlich Seitenadressierung, Segmentgröße bis 4GB. Ist im Wesentlichen so im Pentium erhalten.
Pentium III: Ausweitung auf 6 aktuell verfügbare Segmente (Segment-Register); zugehörige Segmentdeskriptoren aus

- *local descriptor table* ($\leq$ 8k Deskriptoren), privates Segmentverzeichnis des Prozesses
- *global descriptor table* ($\leq$ 8k Deskriptoren), öffentliches Segmentverzeichnis, hierin auch Betriebssystem!

Segmentdeskriptoren enthalten Attribute, Zugriffstyp, Privilegierung (Zugriff nur auf privilegirteren Code (ausgenommen: Programmbibliothek) oder minderprivilegierte Daten), Basis, Grenze, Präsenz im Speicher.

Daneben über dieselbe Tafel *task gate/call gate* Deskriptoren als Übergang in anderen Task bzw. in höher privilegierten Code; dabei Schutzmaßnahmen:

- Einsprung nur an erlaubter Stelle
- mitgebrachte Adressen werden überprüft, ob im Zugriffsbereich des Aufrufers liegend.

ähnlicher Mechanismus für Unterbrechungen

Zugang zu den Segment/task gate/call gate/trap gate/ interrupt gate Deskriptoren: Abb.3.49

Seitenadressierung

über die Basisadresse wird die Prozessadresse in eine 32b (wenig!) "lineare" Adresse umgesetzt. Diese wird zweistufig umgesetzt:

Index PM (10b)	Index ST (10b)	Byte der Seite (12b)

Index PM: Index im Page Directory, $\leq$1024 Verweise auf Seitentafeln
Index SM: Index in Seitentafel, ($\leq$1024 Kachelnummern)
Byte der Seite: Byte in der 4k-Seite

Damit soll (angesichts der Unternutzung des 2^{32}B Adressraums) verhindert werden, daß eine Tafel aus $2^{10+10} \approx 10^6$ Einträgen zu führen ist.
Durch mehrfache *page directories* sind größere Adressräume als 2^{32}B beherrschbar!

Vereinfachte Lösung: alle Segmente im selben linearen Adressraum, Segment hat eine 1024-Seitentafel.

Ende Beispiel Pentium II

3.4.7 Zugriffsschutz im Hauptspeicher ("Speicherschutz")

zu regeln:

1. **Wer** (welcher Prozess) darf
2. **worauf** (welches Segment bzw. welche Seite; nach Platzzuteilung: welche Kachel, welches Maschinenadressintervall)
3. **wie** (lesen, ausführen, schreiben)

zugreifen? Zugriffsrecht ist Funktion von Prozess und Objekt!

Lösungen:

- Falls keine privilegierten Adressabbildung: Einteilung des Speichers in ("Speicherschutz"-) Kacheln; jeder Kachel in Tafel ein(!) Zugreifer und Zugriffsart zugeschrieben, von allen zugreifbare Kacheln werden besonders gekennzeichnet. Kein Schutz von Kommunikationsgruppen. Z.B IBM /360 (bis zur /370, 1972)
- Bei verdeckter Basisadresse: Zugriffsschutz durch Vergleich mit zulässiger Höchstadresse ("wie?" nicht geregelt, vgl.3)
- Bei Seitenadressierung: " Wer worauf" geregelt durch Kachelnummern in Seiten-Kachel-Tafeln, "wie?" durch Hilfsdaten (vgl. Abb 3.45)
- Bei Segmentierung: Segment ist Schutzeinheit. "Wer worauf?" geregelt durch Segment-Tafel und Seiten-Kachel-Tafeln oder verdeckte Basisadresses/ Höchstadressen, "wie?" durch Hilfsdaten in Segment-Tafel(z.B. Abb. 3.48)

3.5 Unterbrechungen (*Interrupts*)

Aufgaben des Leitwerks:

- Funktion SYNT des von Neumanschen Interpretationsschemas

- Synchronisation des Interpretationsprozesses mit anderen Prozessen und Umschaltung des Interpretationsprozesses. Mechanismus: Unterbrechungen: (Univac 1958)
- weiteres (Operandenbeschaffung, Adresstransformation,...)

Grundsätzlicher Ablauf der Unterbrechung:
Unterbrechungswunsch → akzeptiert → Unterbrechung → Prozess-Status definiert retten (wegen Verschachtelung und last in-first out Folge) typisch unter Benutzung eines Stacks → Unterbrechungsbehandlung anspringen, abarbeiten → Wiedereinsetzen eines Prozess-Status

Beispiele für Prozesse, die asynchron zum Interpretationsprozess sind:

1. Prozess hardwaretechnischer Fehler (Klima, Spannung,...)
2. Uhrprozess
3. Interpretationsprozesse anderer Prozessoren
4. Prozesse in peripheren Geräten
5. Benutzer/Operateurverhalten
6. externe technische Prozesse

Ein Unterbrechungswunsch, der auf 1..6 zurückgeht, betrifft i.A. nicht den laufenden Prozess! Oft *interrupt request* im engeren Sinne genannt. Die Behandlung der durch diese ausgelösten Unterbrechungen kann nicht vom aktuellen Prozess, sondern nur vom Betriebssystem erfolgen, Nachricht an etwa betroffenen Prozess.
Daneben gibt es Wünsche, die auf die Erledigung des aktuellen Befehls zurückgehen und eine Ausnahmebehandlung erfordern, die dasselbe Verfahren benutzt (Status retten, Behandlung durch Programm...) Sie sind also nicht eigentlich asynchron; oft *exceptions* genannt:

7. Adressierungsfehler
8. Operandenfehler (z.B. Division durch null, Typfehler)
9. Operatorfehler (z.B. STOP verboten)
10. Operationsfehler (z.B. Überlauf)
11. Moduswechsel (hier liegt natürlich gar keine Ausnahme vor, da der Befehl den Wechsel verlangt! Aber Ablauf wie Unterbrechung gewünscht.)

Die Behandlung kann in einigen Fällen (8,10) vom Prozess selbst gemacht werden (z.B. Laufzeitsystem des Übersetzers). Schließlich gibt es noch weitere Unterbrechungseinleitungen, die vom aktuellen Programm ausgelöst werden:

12. Software-Interrupts (programmierte Unterbrechungen): das sind Befehle, die bestimmteUnterbrechungswünsche setzen können (z.B. bei ausgebliebenem "Hardware"-Unterbrechungswunsch).
13. *debugging traps* (Entwanzungs-Fallen): Ist für eine Befehlsfolge ein Merkbit im Prozessor gesetzt, dann wird nach jedem Befehl in eine Analyseroutine verzweigt: Überwachung ohne Änderung der Befehlsfolge. Sind nur Unterbrechungen an bestimmten Stellen erwünscht, kann man spezielle Unterbrechungen durch Befehl auslösen (*break*-Befehle, auch bedingbar, *debug mode*)

Varianten des Ablaufs

- Abbruch des laufenden Programms
  - nach einem Befehl (die Regel)
  - in einem Befehl (notwendig bei nicht-Beendbarkeit, z.B. Seitenalarm; empfehlenswert bei langlaufender Operation, z.B. Wortgruppentransport).Lösung:

    - Rücksetzung auf Anfang (kann man Status restaurieren? Wirkung zurücksetzen?)
    - Wierderaufnahme in der Operation (dann vermehrter (Mikro-) Status!)

- Verzweigung
  - auf eine einzige Zieladresse für alle Unterbrechungsursachen, dort folgt Analyse (billige Hardware, langsam)
  - Unterbrechungstyp bestimmt Zieladresse z.B. über Tafel im Speicher (*vector table*)
  - Unterbrechungswunsch bringt Zieladresse mit
  - Unterbrechungswunsch bringt einen auszuführenden Befehl mit

Wesentliche Komplizierung durch unabhängiges Auftreten der Wünsche 1..6 und (bei nicht-überlappender CPU nur einem) der Klasse 7..13: Es gibt i.A. viele anstehende Unterbrechungswünsche zu einem Zeitpunkt:

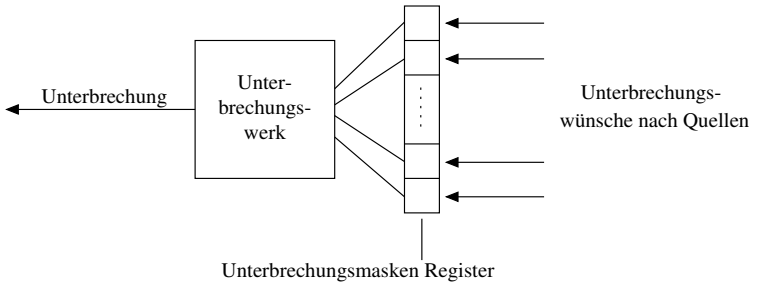


Abbildung 3.48: mehrere Unterbrechungswunsch-Quellen

Zwei Grundtechniken (oft kombiniert):

- Priorisierung der Unterbrechungswünsche, statisch nach Herkunft; im Prozessor ablaufender Prozess hat ebenfalls eine Prioritätsstufe. Der höchstpriorisierte Prozess bzw. Unterbrechungswunsch erhält den Prozessor.
- willkürliche "Maskierung" von Unterbrechungswünschen (global: Unterbrechungssperre)

Denkbare Prioritätsfolge für Unterbrechungswünsche:

A Ereignisse, die Abschluß der aktuellen CPU-Operation verbieten:
  - Befehl unlesbar
  - Adressierbefehl
  - Operatorfehler
  - Operationsfehler
  - Moduswechsel, programmierte Unterbrechung

B Ereignisse, die für einige Operationen rückstellbar sind:
  - Maschinenspannung
  - Uhr
  - schnelle externe Prozesse

C Ereignisse, deren Verschiebung prinzipiell längerfristig zulässig ist:
  - Meldung eines anderen Prozessors
  - Peripherie-Ereignisse
  - Benutzer, Operateur
  - langsame externe Prozesse

Klasse A stimmt weitgehend mit den *exceptions* überein. Taucht in der Betriebssystem Unterbrechungsbehandlung zu einem A-Ereignis dasselbe A-Ereignis auf, wird sofort oder nach endlicher Schachtelung der Prozessor gestoppt (*Lieber ein Ende mit Schrecken, als Schrecken ohne Ende*)

Die Prioritäten der Unterbrechungswünsche dürfen nicht verhindern, daß z.B. ein B-Ereignis gewisse

Unterbrechungsbehandlungen (z.B. Operationsfehler) eines unwichtigen Prozesses unterbricht.

Die ungünstige Einstellung der Peripherie ist nur akzeptabel, wenn die Bedienung im μs-Bereich stets über Kanal oder durch kurzfristige Zuteilung des Bus möglich ist. Auch dann bleiben Unterbrechungen katastrophal häufig, z.B. 100/s bei großem Rechner. Abhilfe durch Abfangen von Unterbrechungen:

- E/A-Prozessor (Abarbeitung langer E/A-Befehlsfolgen, mit Fehlerbehandlung, ohne Unterbrechung)
- Betriebssystem-Prozessor neben Nutzerprozessor

Beispiel Intel Pentium

Interrupt: extern und asynchron

- *maskable*: (pauschal) unterdrückbar durch Löschen des Interrupt enable Flipflops
- *non-maskable (NMI)*: wird unbedingt nach nächsten Operationsende aufgenommen; während seiner Behandlung schiebt der Prozessor weitere NMI auf

Exception: intern und synchron, Unterscheidung nach Wiederaufnahme:

- *fault*: auslösende Operation wird rückgesetzt (!), z.B. Seitenfehler
- *trap*: auslösende Operation wird vollendet, z.B. programmierte Unterbrechung
- *abort*: Programm wird abgesetzt, z.B. Doppel-*exception* (gewisse Fälle)

Übersicht der Unterbrechungswünsche:

Index	Anlass	Behandlung
0	Division	fault
1	Debug	fault oder trap
2	NMI	
3	Breakpoint	trap
4	overflow	trap
5	bound	fault
6	invalid opcode	fault
7	device not available	fault
8	double fault	abort
9	(reserved)	
10	invalid task state	fault
11	segment not present	fault
12	stack fault	fault
13	general protection	fault oder trap
14	page not present	fault
15	(reserved)	
16	floating point	fault
17	alignment	fault
18	machine check	
19-31	(reserved)	
maskable	*interrupts*	trap
32-255	(*maskable interrupts*)	
	(INT n Befehle)	

Priorität bei konkurrierenden Unterbrechungswünschen:

1: Traps (intern): Debug, breakpoint (des vorherigen Befehls)
2: externe: NMI, maskable Interrupts
3: faults bei Befehlszugriff (nächster Befehl)
4: faults bei Befehlszugriff/Dekodierung: Segmentlimit, page fault, illeagal opcode, Befehlsformat, (Coprozessor nicht vorhanden, bis80386)
5: faults bei Befehlsausführung: Floating point error, overflow, bound, task state, segment/page fault, stack, protection, alignment

Pentium unterbricht nicht im Befehl! D.h. Vollendung (*trap*) oder Rücksetzen (*fault*). Rückstellbarkeit der Unterbrechung erleichtert.

Interrupt deskription table IDT vgl. Abb. 3.49

Der Interrupt-Vektor (Index) (hier im allgemeinen Sinne: d.h. *interrupt* oder *exception*) bezeichnet einen 8b-Deskriptor in der IDT:

segment selector
offset
privilege level
segment present

Die IDT kann irgendwo im Speicher liegen (Basis-Adresse und Länge steht im IDTR-Register) vgl. einfachere Lösungen: die ersten 256 Zellen des Maschinenadressraums enthalten die Ansprungadressen der Unterbrechungsbehandlungen. [Beispiel: VAX/MI: System Control Block]

(Ende Beispiel Pentium)

3.6 Leitwerk

Leitwerkzyklus (*interpretive loop*) nun durch Adresstransformation und Unterbrechungsprüfung wesentlich komplizierter als in der TGI-Vorlesung; für eine schlichte serielle CPU:

```
transformiere Befehlsadresse;
hole Befehl aus Hauptspeicher;
entschlüssele;
solange nicht "STOP"
    [wenn nicht "Sprung"
        dann[transformiere Operandenadresse;
             hole Werte aus Hauptspeicher;
             beauftrage Rechenwerk und warte;
             transformiere Resultatsadresse;
             bringe Resultat in Hauptspeicher;
             erhöhe Befehlsadresse;]
        sonst wenn Sprung auszuführen
             dann Befehlsadresse := Zieladresse
             sonst erhöhe Befehlsadresse
    prüfe Unterberchungswünsche und unterbreche ggf. ;
    Co Unterbrechung im Befehl fehlt;
    transformiere Befehlsadresse;
    hole Befehl aus Hauptspeicher;
    entschlüssele;]
STOP
```

Das Leitwerk braucht entsprechend nicht nur

- Befehlszähler / Befehlsadressregister
- Befehlsregister
- Operationscode-Decoder

sondern auch

- Befehlswerk
  neben obigen z.B. weitere Register, falls Befehle über Wortgrenzen
- Adresswerk, etwa verdeckte Basis/Höchstadress-Register; assoziative Register (*translation lookaside buffer*) oder Segmenttafelbasis ("Deskriptor")-Register / Grenzregister
  Indexregister (falls hierfür nicht allgemeine Register)
- Unterbrechungswerk
- Uhr, Wecker
- Register für Programmstatus

Da das Leitwerk meist den Speicher-Prozessor Verkehr abwickelt, gehören zu ihm auch Einrichtungen zum Befehls/Operanden Vorgriff (*look ahead*) bzw. zur Befehls/Operanden Rücklage.

Beispiel Intel Pentium

Register des Leitwerks (außer Befehlsadressregister (EIP) und Befehlsregister) u.a :

- extended flag register
- Basisadressregister der Deskriptortafeln: GDTR, LDTR, IDTR, TR (task register) siehe Seite 97
- 4 Steuerregister CR0..CR4 (Paging, Cache, Konfiguration, Schutzmaßnahmen, 8086-Modus, machine checks)
- 4 debug Register
- Deskriptor Cache, Selektor Register
- Translation lookaside buffer
- Basisadresse/Länge Seitentafel usw.

Beispiel MIPS R10000:

30 Register für:

Steuerung TLB; Timer/Uhr; Prozessorstatus; Befehlszähler Unterbrechung; Prozessorversion; Multiprozessorkoordination; Diagnostik; Leistungsmessung; Cache

3.7 Rechenwerk und arithmetische Operationen

3.7.1 Festpunkt- und Gleitpunktzahlen
3.7.2 Komplementdarstellung und verallgemeinerte Addition von Festpunktzahlen
3.7.3 Multiplikation und Division von Festpunktzahlen
3.7.4 Gleitpunktoperationen
3.7.1 Aufbau von Rechenwerken

Literatur:

Goldberg, D : Computer Arithmetic in [HENNESSY PATTERSON 96]
[BODE HÄNDLER 80] vor allem Bitslice-Rechenwerke
[SPANIOL O 76] Arithmetik in Rechenanlagen, Teubner, Stuttgart 1976

Das Rechenwerk (*arithmetic unit, data path*) ist der Teil des Prozessors, der die durch das Programm explizit bestimmten Berechnungen ausführt (d.h. Adressrechnung finden oft nicht im Rechenwerk statt). Es besteht aus:

- Registern für Operanden, Ergebnisse (oft nach Festpunkt/Gleitpunkt getrennt)
- Rechenschaltnetz mit Hilfsregister (Shiftzähler, Multiplikator/Quotienten-Register, Addenden/ Multiplikanden Register..., arithmetisch-logische Einheit (ALU) kann als Pipeline und/oder funktionsspezifische Werke ausgebildet sein
- Einrichtungen für die Operationssteuerung

schon bekannt: Registerstruktur hängt von 3/2/1/0-Adressprinzip ab! Große Registerzahlen sind durchsatzgünstig, aber träge bei Prozesswechsel; erster Gesichtspunkt überwiegt, →3/2-Adressprinzip.

3.7.1 Festpunkt- und Gleitpunktzahlen

Radix-(polyadische) Darstellung: $a_m a_{m-1} ... a_i ... a_1$ ist eine Zahl mit Wert:

$$a = \sum_{i=1}^{m} a_i B^{i-1} * B^k$$

$a_i \in [0..B-1]$	Ziffer
B ganz, ≥ 2	Basis
B^k	Skalenfaktor, üblich
	k=-m B-AL-Punkt links neben a_m: a ist echter B-AL-Bruch
	k=0 B-AL-Punkt rechts neben a_1: a ist ganze Zahl

$a_m a_{m-1} ... a_i ... a_1$ mit festem k ist Festpunktzahl. In Rechnern typisch k=0 oder (seltener) k=-m. Punkt nicht notiert, für Addition und Subtraktion Punkt ohne Bedeutung.

Mängel der Festpunktdarstellung

- **Bereich** (Differenz zwischen betragsgrößter und betragskleinster (≈ 0, bei Festpunkt =0)) ist zu klein: $B^m - 1$ bei Festpunkt
- **Genauigkeit** (Zahl der korrekten Ziffern (einschließlich korrekt gerundeter) ohne führende oder nachfolgende Nullen) ist im Bereich kleiner Beträge für eine gute Annäherung an beliebige reelle Zahlen zu klein.

Daher Gleitpunkt (integrierter logarithmischer "Skalenfaktor")

Typische Gleitpunktdarstellung: $v\,c\,p$:

v: Vorzeichen: 1 Bit, übliche Konvention
c: Charakteristik: $C_s C_{s-1} ... C_1$ aus dieser ist der Exponent x berechenbar
m: Mantisse: $P_r P_{r-1} ... P_1$

und hat den Wert

$$a = \underbrace{\left(\sum_{i=1}^{r} P_i * B^{i-1} \right) * B^{-r}}_{Mantisse} * G^x$$

G ist Basis der Gleitpunktzahl. Vorteil der Gleitpunktdarstellung:

- Bereich kann wesentlich größer als $B^m - 1$ (hier m=r+s, Zahl der Ziffern) gemacht werden
- Genauigkeit im ganzen Bereich annähernd konstant

Abbild in hohen Sprachen *bin float, dec float* oder auch irreführend *real*.

In Rechnern aus technischen Gründen meist B=2, daneben B=10 (nur Festpunkt, dann a_i trotzdem binär codiert, G=2 oder G=16.

wichtiges Beispiel: IEEE 754 Gleitpunkt-Standard

(mit Definition von Rundung und *exceptions*)

einfaches Format (32b)

Vz	Char. 8b	Mantisse 23b

$$\text{Wert} = (-1)^{Vz} * 2^{\overbrace{Char-127}^{Exponent}} * \overbrace{(1.Mant)}^{d.h.implizite\,führende\,1}$$

dabei (Char=255) und $\begin{cases} Mant \neq 0 : \text{not a number} \\ Mant = 0 : \infty \end{cases}$

alle Stellen 0 : 0

doppeltes Format (64b)

Vz	Char. 11b	Mantisse 52b

3.7.2 Positive und negative Festpunktzahlen, verallgemeinerte Addition

Kurzfassung

Festpunktzahlen:
positiv: m-stellige Dualzahl mit Vorzeichen (0).
negativ: Vorzeichen (1) und Komplement zu
$2^m - 1$: Stellen-(Einer)Komplement, einfach bildbar, 2 Nullen, nach Addition Korrektur, falls Übertrag in Stelle m+2 ("Einerrücklauf")
2^m: Bereichs-(Zweier)Komplement, unsymmetrischer Darstellungsbereich $-2^m..2^m - 1$. heute üblich

Verallgemeinerte Addition wird korrekt durch Rechnung modulo 2^{m+1} ausgeführt, dabei

- Überlauf, falls $(++) \to -$ oder $(--) \to +$
- Einerrücklauf bei Stellenkomplement

Die bitparallele Addition verlangt Grundschritt und im Mittel $ld\,m$ Übertragsschritte.

3.7.3 Multiplikation und Division von Festpunktzahlen

Das Ergebnis bei ganzen Festpunktzahlen ist zwei Worte lang. Negative Zahlen werden (meist) erst in Vorzeichen-Betrags-Darstellung gebracht. Das Verfahren ist im Prinzip wie "von Hand":
Beispiel 14*13=182

```
   LLLO*LLOL
   LLLO
  0000
  OLLLO
 LLOO
LOOOLLO
LLLO
LOLLOLLO = 182
```

Beispiel für maschinelles Verfahren:

R1	enthält Muktiplikanden
R2	enthält anfangs Multiplikator, später hinteren Teil der Partialsumme
R3	enthält anfangs 0, später vorderen Teil der Partialsumme

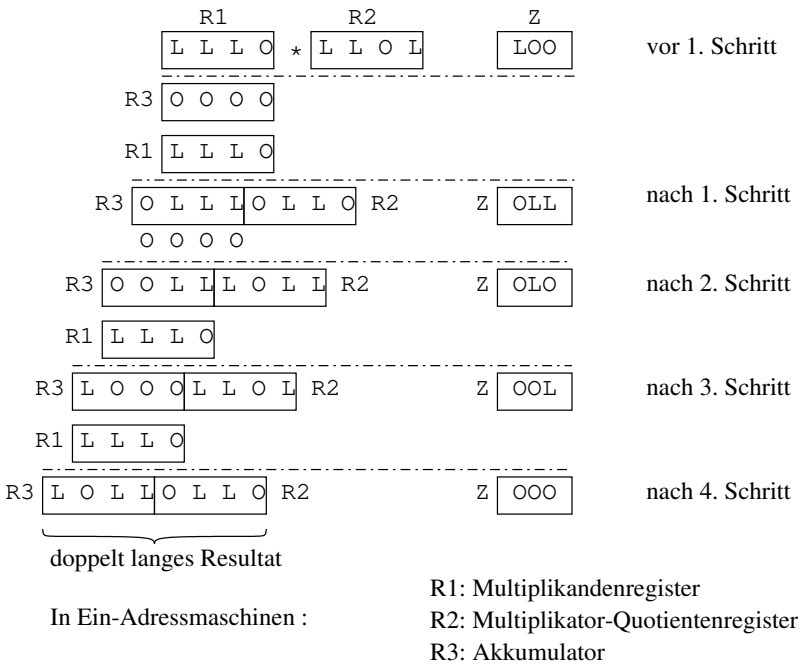


Abbildung 3.49: Maschinelles Multiplikationsverfahren

Division wird als Folge von (versuchsweisen) Subtraktionen und Shifts durchgeführt; ähnlich wie bei Multiplikation doppelt langes Ergebnis; typisch: $a\,div\,b$: ganzzahliger Quotient, $a\,mod\,b$: Rest.

Beispiel:

```
LLLOL : LOL = LOL Rest LOO
LOL
OLOOL
  LOL
  LOO
```

Nimmt mit dem maschinengeeigneten Algorithmus eine abschreckende Form an: siehe nächste Seite

3.7.4 Gleitpunktoperationen

Gleitpunktzahl besteht aus Vorzeichen , Charakteristik und Mantisse (siehe **3.7.1**). Um

- die Genauigkeit voll zu nutzen
- eine Eindeutige Darstellung zu sichern

werden Gleitpunktzahlen in der Maschine normalisiert geführt, d.h.

- bei Gleitpunktbasis 2 ohne führende Null in der Mantisse; d.h. das Bitfeld wird linksbündig verschoben; bei IEEE 754 ohne die implizite 1 vor dem Dualpunkt darzustellen.
- bei Geitpunktbasis 16 mit höchstens 3 führenden Nullen

Gleitpunktoperationen:

Addition/Subtraktion: sind die Charakteristiken ungleich, so wird die kleinere Charakteristik angepasst (führende Nullen in Mantisse), dann Addition/Subtraktion erforderlichenfalls Normalisierung
Multiplikation/Division: Die Charakteristiken werden Addiert/Subtrahiert und um den Betrag des kleinsten Exponenten korrigiert; Die Mantissen werden muliplizierten/dividiert, erforderlichenfalls Normalisierung.

Bei Gleitpunktoperationen kann das Assoziativ- und das Distributivgesetz verletzt werden!

Maschinengeeigneter Divisionsalgorithmus

	Z	R1	R2	R3
anfangs:	Stellenzahl	0	a	b
	L0L	000000	0LLL0L	000L0L
links		00000L	0LL0L0	
Subtr.		LLL0LL		
		L00000L		
Decrem.	L00			
links		0000LL	0L0L00	
Subtr.		LLL0LL		
		LLLLL0		
Add.		000L0L		
		L0000LL		
Decrem.	0LL			
links		000LLL	00L000	
Subtr.		LLL0LL		
		L0000L0		
Add.			00L00L	
Decrem.	0L0			
links		000L00	0L00L0	
Sub.		LLL0LL		
		LLLLLL		
Add.		000L0L		
		L000L00		
Decrem.	00L			
links		00L00L	000L00	
Subtr.		LLL0LL		
		L000L00		
Add.			000L0L	
Decrem.	000			
ENDE!		a mod b	a div b	

links (R1,R2) verschiebt ohne Vorzeichenstellen, füllt R1 mit R2, R2 rechts mit Null auf.

3.7.5 Aufbau von Rechenwerken

Einfache Formen

- allgemeine Register, evt. Gleitpunktregister
- arithmetisch-logische Einheit (ALU)
- Sonderregister (teilweise unsichtbar)
  - Shiftzähler
  - Stellenzähler
  - Mutiplikator/Quotientenregister
  - Akkumulator
  - Multiplikanden/Divisorregister

    können teilweise mit allgemeinen Registern übereinstimmen
- lokale Steuerungen

Co"prozessoren"

Sie sind keine Prozessoren, da sie nicht ein Programm abarbeiten; Befehlszählerfunktion liegt bei Zentralprozessor. Früher aus technologischen Gründen (Chipfläche) als Einzelbausteine bei Mikroprozessoren verbreitet. Anwendung: Arithmetik, Graphik.

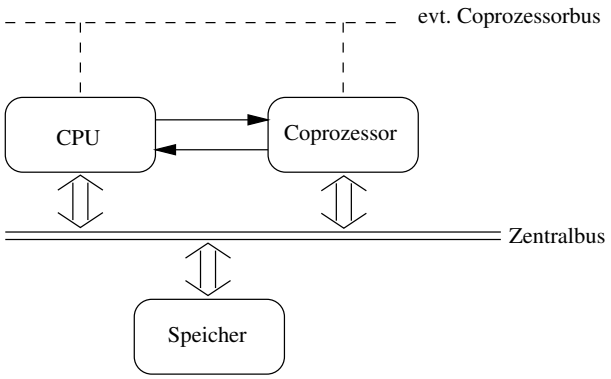


Abbildung 3.50: Anbindung eines Co"prozessor"

CPU	Coprozessor
gibt Befehlsadresse an Speicher	
erhält Befehl, entdeckt Coprozessorzugehörigkeit	(alternativ: Coprozessor entdeckt "seinen" Befehl)
verschickt Operator an Coprozessor	erhält Operator
modifiziert Adresse, gibt Adressen an Speicher	
(wartet oder führt einschiebbare Operationen aus)	erhält Operanden, direkt oder von CPU; eventuell auch aus lokalen Registern
	Operation
	Ergebnis an Speicher oder Register
	Condition Code an CPU
evt. Auswertung Condition Code	Fertigmeldung an CPU

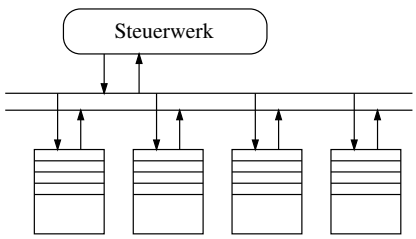
Coprozessoren sind auch auf dem Prozessorchip sinnvolle Einheiten! Sie ermöglichen die Erledigung langer, irregulärer struktureller arithmetischer Befehle neben dem Ablauf in einem Pipeline-Prozessor mit fester Taktzahl (→3.9) für den Rechenwerksschritt.

schnelle Rechenwerke

- Rechenwerk besteht aus kollateral arbeitsfähigen Spezialrechenwerken (z.B. Festpunktaddition, Festpunktmultiplikation, Boolesche Operationen, Gleitpunktoperationen); erstmals CDC 6600 (1963); seit Intel i860 auch *superscalar* genannt, weil mehr als eine Operation je Takt fertig werden kann.
- Rechenwerk ist ein Pipeline-Werk (z.B. Gleitpunkt: Charakteristik angleichen, Mantisse addieren, Mantisse normalisieren)
- Kombination beider Prinzipien: Pentium, R10000

Rechenwerke kundenspezifischer Prozessoren

Aufbau aus Bitscheiben (*bit slices*) von z.B. 8 oder 16b Breite, mit Definition der Befehle in mikroprogrammiertem Steuerwerk. Bis etwa 1985 Geschwindigkeitsvorteil z.B. AMD 2900/29300



3.8 Operationssteuerung

Die Befehlsausführung braucht eine Folge von Steuersignalen, hervorgebracht von einem Schaltwerk, der Operationssteuerung. Zwei Varianten:

- ad hoc-Steuerwerk: schnell, höherer Entwicklungs-/Wartungs-/Änderungsaufwand; hohe Herstellkosten, benutzt für häufige Operationen typisch RISC, Superrechner

- "Mikro"programmiertes Steuerwerk, komplementäre Eigenschaften und Anwendungen; (→ Interpretationshierarchie Kap.2.6)

Bei zentraler , synchroner Steuerung aller Teile der Zentraleinheit ist Operationssteuerung Teil des Leitwerks, sonst werkindividuelle Operationssteuerungen.

Mikroprogrammierung benutzt typische Verfahren der Codierung und der Ableitung des Folgeschritts ("SYNT", vgl. 2.3)

3.8.1 Ad-hoc-Steuerungen (*Hardwired Control*)

Grundschema: Steuerwerk (ein Schaltwerk!) und Operationswerk (oft ein Schaltwerk)

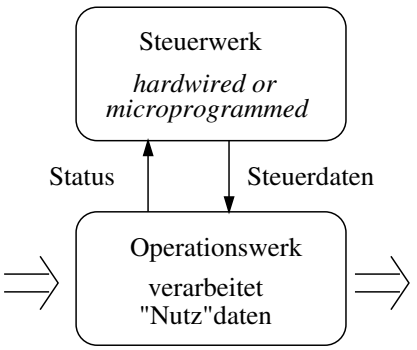


Abbildung 3.51: Steuerwerk-Operationwerk

Status und Steuerdaten seien Binärvektoren mit bitweiser Semantik (z.B. Multiplikator aufgezehrt bzw. invertiere Multiplikandenregisterinhalt)

Vorgegebene Funktion bei Rechnersteuerung aus Operationscode abgeleitet

Wichtigstes Entwurfsmuster für Steuerwerk in Ad-hoc Variante: Flußdiagramm; jede Aktion entspricht einem Internzustand des Steuerwerks. Übergänge vom Zustand des Operationswerks abhängig.

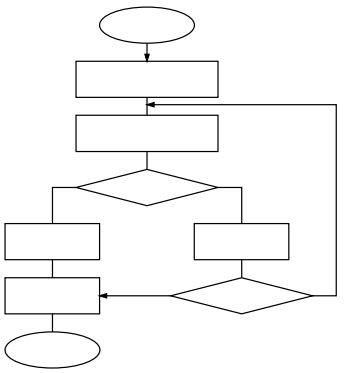


Abbildung 3.52: Flußdiagramm

Kanonische Realisierung des Flußdiagrammansatzes in Zeitschaltkette (D-Flipflops) und *Programmable Logical Area*) (PLA) möglich:

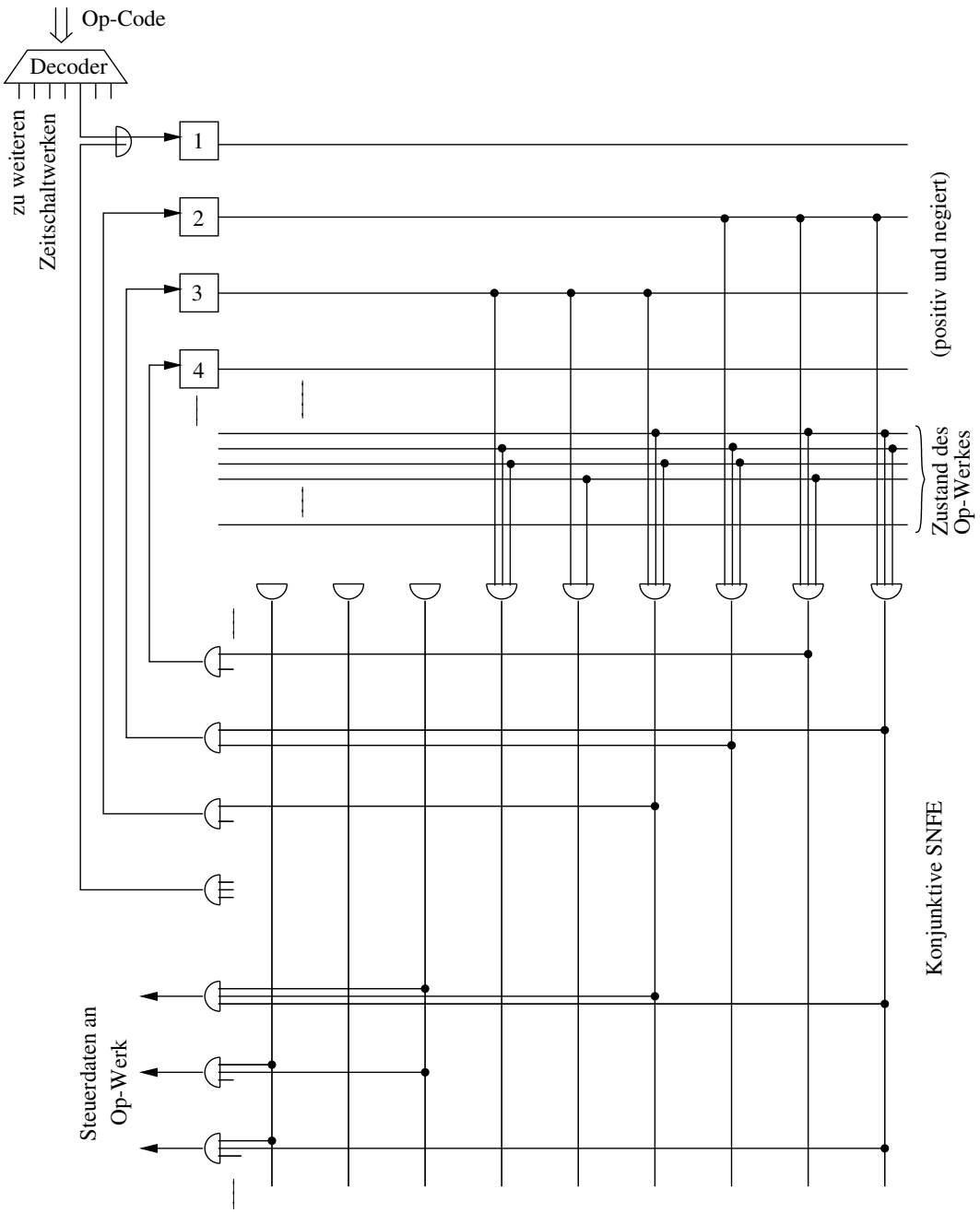


Abbildung 3.53: PLA

Das Grundschema der Zeitschaltketten-Steuerung ist verfeinerbar durch Übergangszeiten in den Folge-Internzustand, die vom Zeitbedarf im Operationswerk abhängen

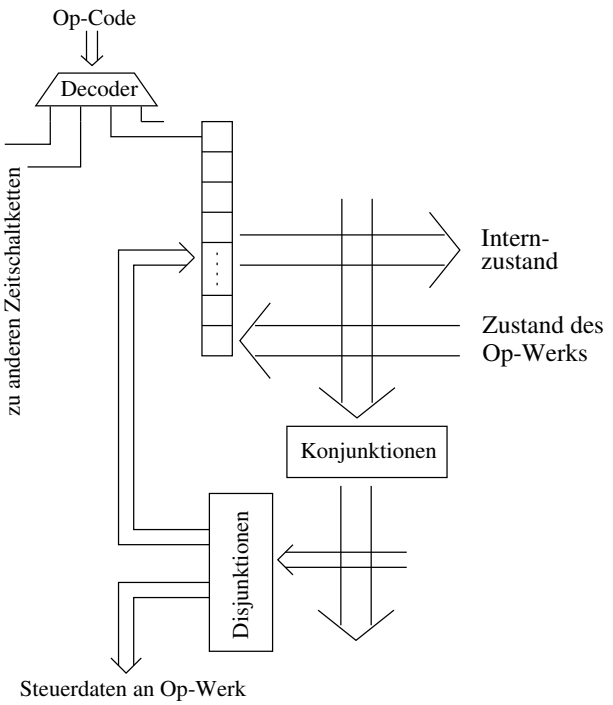


Abbildung 3.54: Zeitschaltkette, verfeinert

Statt 1ausZ-Code in Zeitschaltkette kann man den Internzustand mit $\lceil ld\, z \rceil$ codieren (aber komplexeres PLA!)

3.8.2 Mikroprogrammierte Operationensteuerung: Grundstruktur

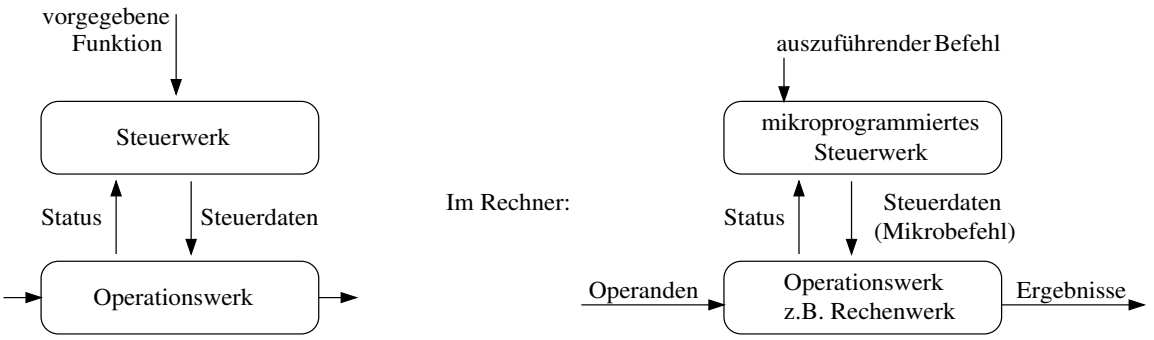


Abbildung 3.55: Steuerwerk-Operationswerk

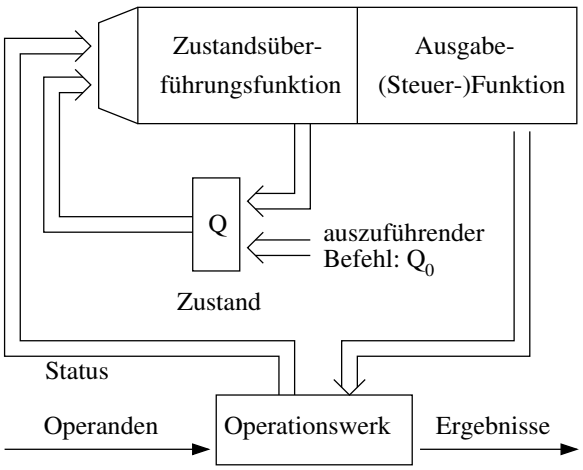
Grundstruktur vergrößert, Mealy Typ: siehe Abbildung 3.58

3.8.3 Erste Modifikation: Statusmultiplexer

n sei die Zahl der Eingangsbits am Schaltwerk (d.h. Status X und Steuerwerkszustand Q zusammen). Für die Auswahl des Folgeschritts sind nicht n Bits erforderlich, da vom Operationswerksstatus X jeweils nur wenige Bits von Bedeutung sind. Daher Vorschalten eines Multiplexers zur Statusselektion.

3.8.4 Zweite Modifikation: Mikro-Befehlszähler

Wird in jedem Schritt nur ein Bit aus X ausgewertet, dann kann man auch nur zwei Folgezustände unterscheiden. Dann kann man einen stets als Q+1 codieren und den anderen davon verschieden wählen: μBefehlszähler, Alternativfolgeadresse in besonderen bedingten Mikrosprungbefehlen oder im Mikrobefehl selbst: Abb. 3.60



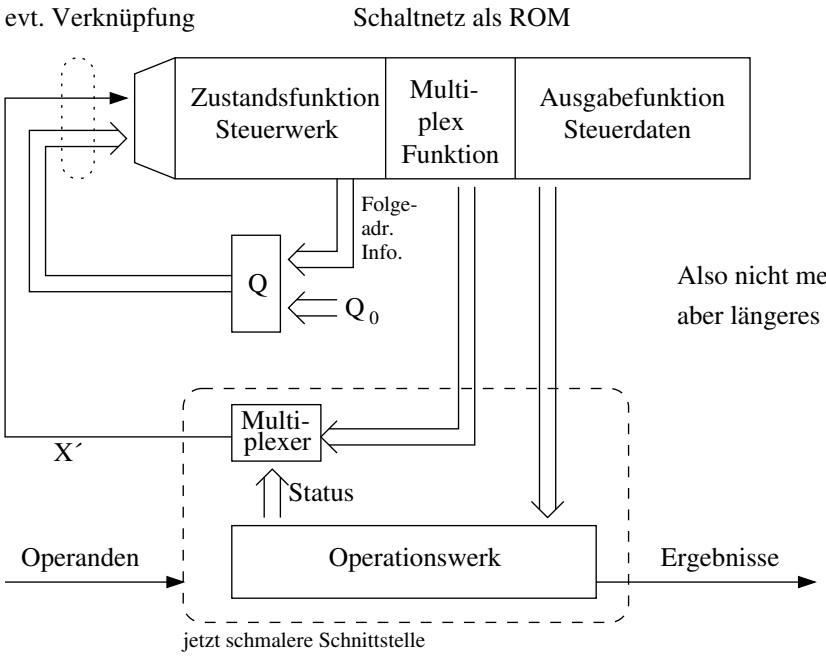
Suboptimale, billige Lösung:

Festspeicher (Schaltnetz)
Schreibspeichererlaubt Umdefinition

Q: Zustand des Steuerwerks

Q und X z.B. n=10..50b zusammen

Abbildung 3.56: Grundstruktur, vergrößert



Also nicht mehr 2^n Zellen,
aber längeres Wortformat

Abbildung 3.57: Statusmultiplexer

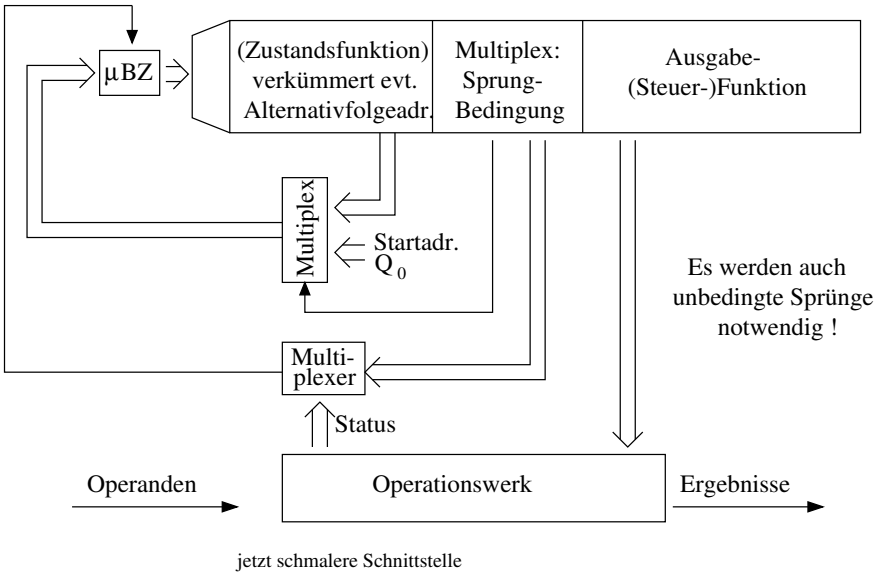


Abbildung 3.58: Mikrobefehlszähler

3.8.5 Mikroprogrammsteuerwerk

Beschleunigung durch Überlappung von μAbrufphase (μBefehl holen) und μAusführungsphase. Allerdings dann Status nach Vor-μBefehl noch nicht für Verzweigung nutzbar!

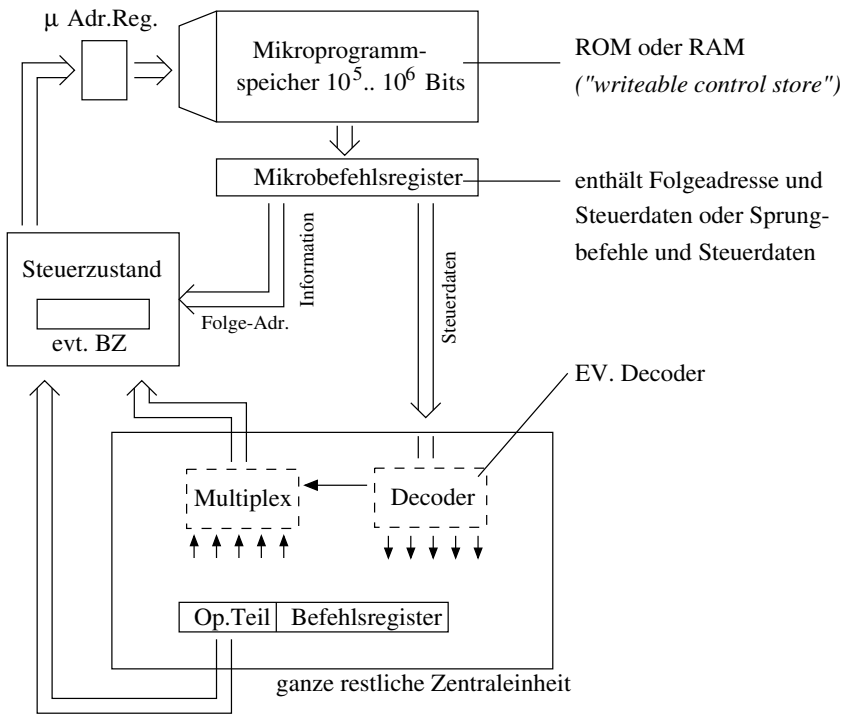


Abbildung 3.59: Mikroprogrammsteuerwerk

Sprünge sind in Mikroprogrammen häufiger als in von Neumann Programmen.

Übergang zum Folgemikrobefehl: Verfahren der **unbedingten Fortschaltung**:

- Ohne Aufbewahrung der letzten Befehlsadresse (also kommende Befehlsadresse im μProgramm mitführen: redundant)
  - 1 aus n Code (schnell) Zeitschaltkette
  - Volladresse (Entschlüsselung notwendig)
- mit Aufbewahrung der letzten Befehlsadresse
  - stets relative Adresse im μBefehl, besonders groß - Sprünge
  - stets hinterer Adressteil im μBefehl, besonders groß - "Seiten"sprünge
  - μBefehlszähler, inkrement oder unbedingter Sprung
    - unbedingter Sprung durch Relativ-Adresse im μBefehl, besonders groß - Sprünge
    - unbedingter Sprung durch hinteren Adressteil im μBefehl, besonders groß - "Seiten"sprünge
    - unbedingter Sprung mit Volladresse (von Neumann)

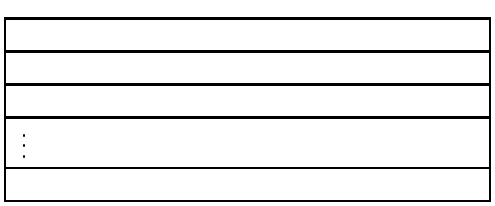
Trade off zwischen: μBefehlslänge, zusätzliche Befehle (statisch, dynamisch), schnelle Gewinnung der Folgeadresse, Mehrfachbedingungen, freie Anordnung der μBefehle

Übergang zum Folgemikrobefehl: Verfahren der **bedingten Fortschaltung**:

- ohne Aufbewahrung der letzten Befehlsadresse
  - bedingungsgesteuert. 1 aus n Code (schnell) Zeitschaltkette
  - Bedingungsbits als Relativadresse zu Folgeadresse
  - Bedingungsbits als hintere Adress-Stellen zu Folgeadresse
  - alternative Volladresse (binäre Bedingung)
- mit Aufbewahrung der letzten Befehlsadresse
  - zwei Relativadressen (binäre Bedingung)
  - zwei hintere Adress-Stellen (binäre Bedingung)
  - mehrere Relativadressen/hintere Adress-Stellen
  - μBefehlszähler: inkrement oder bedingter Sprung
    - bedingter Sprung als Relativadresse im μBefehl; auch Bedingungscode als Relativadresse möglich
    - bedingter Sprung durch hinteren Adressteil im μBefehl; auch Bedingungscode als hinterer Adressteil möglich
    - bedingte Sprünge mit Volladresse (von Neumann)

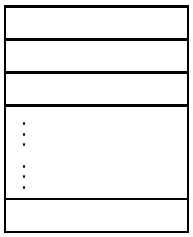
3.8.6 Horizontale Mikroprogrammierung

Man kann die Steuerinformation entweder in wenige lange



horizontal

oder viele kurze Mikrobefehle packen:



vertikal

Von **horizontaler Mikroprogrammierung** spricht man, wenn es eine feste Abbildung von Stellen (Gruppen) gibt. Funktionsbits (Gruppen) betätigen unmittelbar Tore, (Komplementierungen, Addition, Shifts...). Der Mikroprogrammierer setzt durch die Werte der Funktionsbits fest, welche Funktionen gleichzeitig ausgeführt werden. Einfachster Fall:

Jedes Bit im Mikrobefehl ist ein Funktionsbit. Dann spricht man von analytischem (Mikrobefehls)Code. Bei großen Rechnern kommt man spielend auf 150..200 Bits Mikrobefehlslänge (dazu ja Alternativ-Nachfolgeadresse,...). Äusserste Gestaltungsfreiheit zur Nutzung kollateraler (wohlgemerkt synchroner) Abläufe in der Maschine: für schnelle Maschinen.

Noch fast so schnell, ca. halbe Wortlänge: man fasst Funktionsbits zu Gruppen zusammen, die sich ausschließen (z.B. Shift und Komplementierung am selben Register) und codiert statt mit m Bits für m Funktionen mit $entier(ld\, m) + 1$ Bits: "quasihorizontal"

Beispiel:

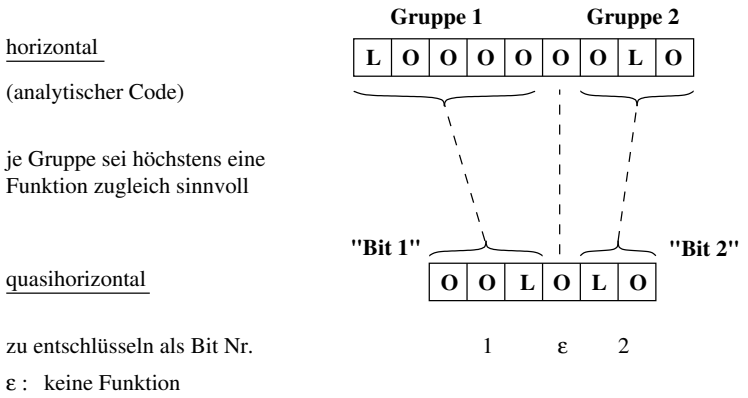


Abbildung 3.60: horizontale und quasihorizontale Mikroprogrammierung

Das Prinzip der Verschlüsselung von Gruppen einander ausschließender Elementarfunktionen spart Speicher, außerdem Leitungen (falls man die Entschlüsselung in das Operationswerk verlegt), kostet aber (wenig) Zeit für die Entschlüsselung.

3.8.7 Vertikale Mikroprogrammierung

Will man eine billige Maschine bauen, dann muß man auf die beliebigen Funktionsbit-Kombinationen der horizontalen Mikroprogrammierung verzichten. Man verliert an Leistung, erhält aber wesentlich kürzere Mikroprogrammworte (z.B. 16..40 Bits).
Typisch dann: gewöhnliche Sprungbefehle wie in der "Maschinen"Programmebene:
spart die Alternativ-Folgeadresse
bezahlt mit Zeit
(lohnt nur, wenn Sprünge nicht zu häufig!)
Mikrobefehle gleichen bei vertikaler Mikroprogrammierung stark den "Maschinen"befehlen (strukturell).

3.8.8 Rückblick

Mikroprogrammierung hat in den letzten 10 Jahren rückläufige Bedeutung:

> Mikroprogrammierung braucht 3..8 Mikroprogrammspeicher-Zyklen je Hauptspeicherzyklus.
> Das war früher leicht zu bieten, ist aber nicht realisierbar, wenn die Hauptspeicherfunkti-

on vom Cachememory wahrgenommen wird. Es gibt keine (Festspeicher) Technologie für 50..100kb Kapazität, die 3..8 mal schneller als der Cache ist!
Mikroprogrammierung hat hohe Grundkosten und mäßige Kosten je Befehlsfunktion. Also **gut für große, komplexe Befehlssätze** und zentral zusammengefasste, dabei **synchrone** Steuerung.
Heutige Rechner (RISC) haben aber mittlere unkomplexe Befehlssätze und asynchron arbeitende Einheiten; die Befehlsausführung wird dezentral gesteuert. Mikroprogrammierung ist dann zu teuer und zu inflexibel.

3.8.9 Ausführung fremder Befehlssätze

Auf einer Mikroprogrammierten Maschine wird der Befehlssatz einer fremden Maschine durch **Interpretation** ausgeführt, d.h. durch einen mikroprogrammierten Interpreter. Er wird typisch durch besondere Hardwarezusätze unterstützt (z.B. besondere Register, Schaltnetze). Das fremde Programm wird durch einen Spezialbefehl gestartet, der den simulierten fremden Befehlszähler und die "hiesige" Anfangsadresse setzt. Die Ausführungsphase dieses Befehls ist die Abrufphase der Fremdmaschine, mit deren Befehlscode-abhängigen Ausführungsphasen. Der Spezialbefehlbefehl endet, wenn alle Operationen des Fremdprogramms ausgeführt sind und kehrt zum "hiesigen" Befehlszählerstand zurück. Interpretation eines Fremdprogramms mit Hardwareunterstützung heißt **Emulation**.

Dasselbe Verfahren läßt sich auf ein Programm in höherer Sprache anwenden. Das Programm liegt im Speicher

- in Quellsprache (dann z.B. Quellnotationsbezüge möglich)
- in Zwischensprache (effizienter)

Auf RISC-Prozessoren (keine Mikroprogrammierung) vereinfachte Technik der Ausführung von CISC-Befehlen, zwecks Nutzung der Strukturvorteile (Durchsatz!): das Leitwerk transformiert jeden CISC-Befehl in 1..4 *risc operations* (ROPS), die auf eine (mehrere) Pipeline(s) gegeben werden. So Pentium, AMD K5, vgl. S.56 Abb. 3.12

Im weiteren Sinne sind auch Diagnostik-Mikroprogramme (wichtig! feingranulare Tests!) "fremde" Operationen

Vergleich von Mikroprogrammierung, Direktsteuerung und Übersetzung zur Laufzeit

CPU-Typ			
μprogrammierte CPU	typ. RISC (MIPS)	Pentium, Athlon	Metadata Crusoe
Befehlssatz der CPU			
beliebig, z.B. x86	pipelinegünstig, mittlere Größe	x86, groß!	x86, groß!
Maschinenprogramm wird zur Laufzeit			
interpretiert durch Interpreter im μProgrammspeicher	direkt ausgeführt durch verteilte adhoc-Steuerung	Befehlsweise übersetzt in 1..3 "R"Ops pro Befehl	Blockweise übersetzt und optimiert in "Atome" die in VLIW gepackt werden
mehrfach-Pipeline *(superscalar)*			
ungünstig, da asynchron und dezentral	gut beherrschbar	bei "ROps" gut beherrschbar	beherrschbar durch VLIW (1 Atom pro Pipeline)
Scheduling der CPU-Elementaroperationen			
starr	Dynamisch nach Präzedenz und Datenfluß. Überholungen! etwa 40 parallel	Dynamisch nach Präzedenz und Datenfluß. Überholungen! etwa 40 parallel	Bei Übersetzung unter Nutzung von Überholung festgelegt, danach keine Überholungen.

CPU-Typ			
μprogrammierte CPU	typ. RISC (MIPS)	Pentium, Athlon	Metadata Crusoe
spekulative Ausführungen			
selten	ja	ja	ja (Schattenregister)
besondere Eigenschaften			
		komplexe Speicher- und Schutzmechanismen	Durch einfache Steuerung ca. 50% weniger Transistoren; Takt- und Spannungsabsenkung bei Unterlast

3.9 Beschleunigung der ZE : Cache-Memory und Pipelining

Nochmal ganz grundsätzlich angesehen:

Im einfachsten Fall arbeitet ein Prozessor die Befehle seriell ab. $\overline{b}$ ist die mittlere Bearbeitungszeit je Befehl. Der Grenzdurchsatz der Zentraleinheit ist

$$c = \frac{1}{\overline{b}} \text{ Befehle/Zeiteinheit (z.B. "Mips")}$$

Das ist auch der Durchsatz (denn ist der Prozessor frei, arbeitet er den *idle loop* ab, schafft also trotzdem)

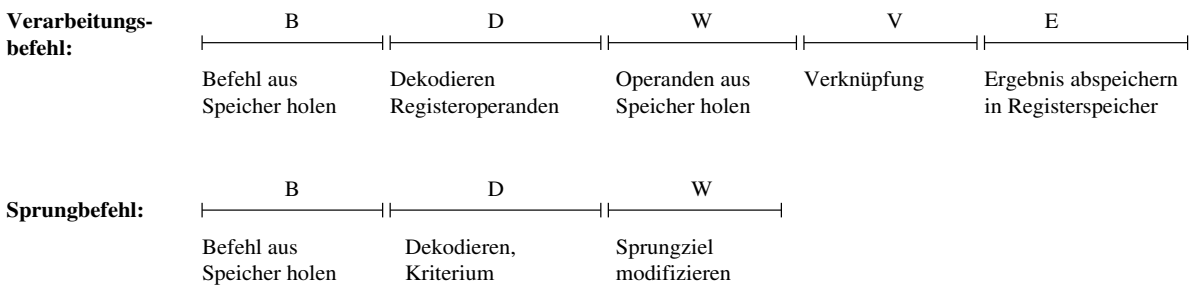


Abbildung 3.61: zeitliche Gliederung von Operationen

wesentliche Abhängigkeit der Bearbeitungszeit b von der Zugriffszeit zum Speicher. Zahl der Speicherzugriffe:

Phase B: Zahl hängt vom Verhältnis Befehlslänge:Wortlänge ab (im Speicher/Bus), z.B. $\frac{1}{2}..4$ (8b Wortlänge), außerdem von *"alignment"*, d.h. Ausrichtung der Befehlsanfänge auf Wortanfänge

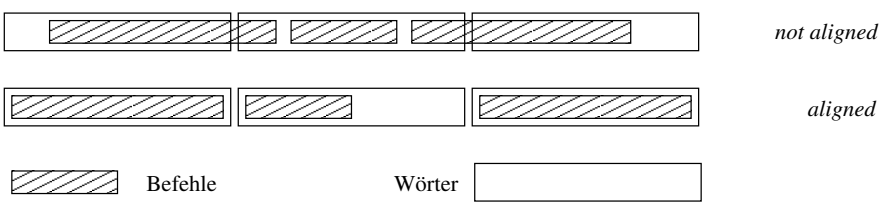


Abbildung 3.62: Ausrichtung der Befehlsanfänge

und noch davon, ob es für die Seitenadressierung zu einem extra Speicherzugriff kommt.

Phase W und E: Zahl hängt von Anzahl und Länge der Operanden ab, außerdem von Adressersetzung und den für Phase B genannten Einflußgrößen.

Typisch sind $\sigma \in \{1; 2\}$ Speicherzugriffe erforderlich, dank der Register. Sei σ z.B. 1,5 Speicherzugriffe

je Befehl, das heißt: machen wir den Prozessor intern auch noch so schnell, nie kommen wir über

$$c \leq \frac{1}{\sigma * T_{Zugriff\,HSp}} \quad , \quad c \leq \frac{1}{1,5 * 70ns} \approx 10\,Mips$$

Dabei ist $T_{Zugriff\,HSp} = T_{Bus} + T_{Hauptspeicher} \approx 20ns + 50ns$

Faktisch viel weniger, weil Phasen D und V eben auch Zeit brauchen!

Es gibt aber zwei Hilfsmittel: **Cache Memory** und **Pipelining**.
Beide sind kombinierbar, beide unterbrechungssensibel, beide verlangen meist, daß der Hauptspeicher viele Lese/Schreibvorgänge gleichzeitig abwickeln kann.

3.9.1 Cache Memory

Cache : das Versteck.
Cache memory ist (abgesehen von Registern) prozessornächstes Glied der Speicherhierarchie. Wird dem Prozessor zugeordnet, typisch als ein Teil von ihm angesehen.

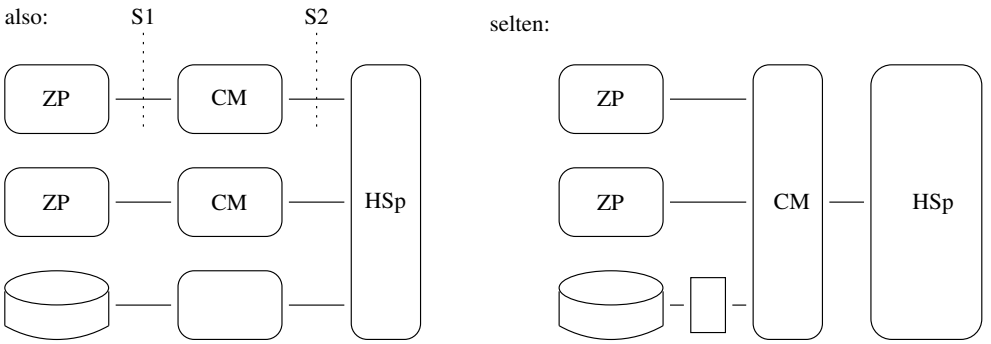


Abbildung 3.63: Cache Memory

Die Schnittstelle S_1 gleicht S_2, d.h. Cache memory ist "transparent" für Zentralprozessor. Meistens werden gewidmete CM für Befehle und Werte benutzt (*instruction/data cache*)

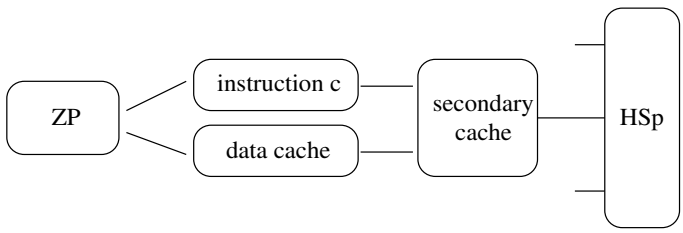


Abbildung 3.64: gewidmete Caches

Vgl. das in 2.7. über Ausschnitte gesagte: Cache memory ist prozessorprivater Speicher für aktuellen Ausshcnitt, 8kB bis 64kB, 5..10ns Zugriffszeit d.h. 5..20 mal schneller als Hauptspeicher und Bus. Trefferverhältnis (*hit ratio*) h=0,8..0,99.

Also bei $T_{Cache} = 0,2 * T_{Zugriff\,HSp}$ und $h = 0,95$:

$T_{''Speicher''} = 1 * T_{Cache} + 0,05 * T_{Zugriff\,HSp} = 0,25 * T_{Zugriff\,HSs}$

Das heißt unser Grenzdurchsatz c steigt von 10 Mips auf 40 Mips !

Cache memories sind auch deshalb gut, weil sie

- kaum Rückwirkungen auf die Gestaltung des Befehlssatzes haben
- den Durchsatz auf Bus und Hauptspeicher senken können

- (wenn klein) auf den CPU-Chip passen

und problematisch wegen der Konsistenz zum Hauptspeicher und untereinander.

Oft zusätzlich sekundäre Cache memories.

Die grundsätzlichen Probleme des "aktuellen Ausschnitts" wurden schon in 2.7 behandelt. Für das Cache memory gilt folgendes:

Kapazität: $K_{CM} = 8kB..64kB$, $K_{HSp} = 1MB..1GB$

Zugriffszeit: $b_{CM} = 2..4ns$, $b_{HSp} = 25..50ns$

> Maßgebend für den Hauptspeicher ist aber die Blockzugriffszeit, gesehen vom CM, $b_{Block\ HSp}$: Adresstransport, Zugriff, Datenblocktransport das sind 80..200ns (200HHz Takt: b_{CM}:1..2 Takte, $b_{Block\ HSp} = 16..40$ Takte.)

Block: ist Nachlade- und Plazierungseinheit, Blockgröße K_B 4..128B

Plazierungsstrategie: (wohin mit dem Block im CM?)

- vollassoziativ: auf einen beliebigen freien Platz (wie Seitenadressierung)
- direkt abbilden: auf genau einen Platz (möglicherweise belegt!)
- z-fach Gruppen-assoziativ (*set associative*): auf einen von z Plätzen (möglicherweise belegt!) es gibt Kapazität/z*Blockgröße viele Gruppen.

Auffinden eines Worts/Blocks im CM

Durch Prüfung der Blockkennung (*tag*) zu einem (direktabbildend), z (Gruppen-assoziativ) oder allen (voll assoziativ) Blöcken im Cache memory. Typische Struktur des CM:

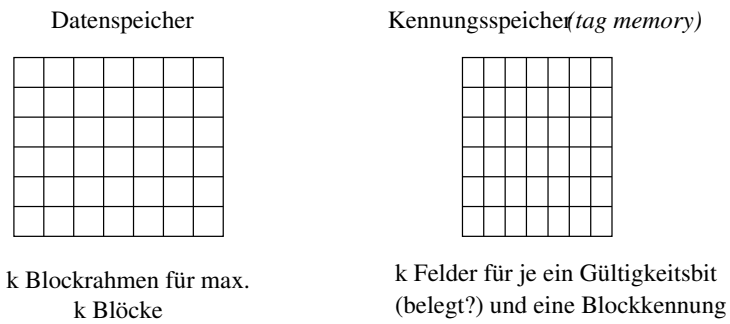


Abbildung 3.65: Struktur Cache memory

Ableitung der Blockkennung aus der Adresse:

Bk.	Nr.	Byte

Bk.: Blockkennung
Nr.: Gruppen-assoziativ: Nummer der Gruppe, in der der Block im CM liegen muß, z Blöcke/Gruppe
direkte Abbildung: Adresse des Blocks im CM
vollassoziativ: Feld nicht existent, Blockkennung entsprechend länger!
Byte: Byte im Block (z.B. 2^5)

Damit ergeben sich die Varianten in Abbildung 3.68

Das Auffinden eines Wortes bei den (z- oder voll-) assoziativen Techniken ist langsamer, obwohl alle Blockkennungen (der Gruppe) gleichzeitig geprüft werden. Direkt abbildende CM nutzen die Kapazität schlechter (unnötige Verdrängungen!).
Kennzeichnung der Leistungsfähigkeit eines CM durch **Trefferverhältnis h (*hit ratio*)**, Anteil der Zugriffe des Prozessors, die den gewünschten Block im CM finden. Komplement 1-h heißt *miss ratio*.
Typisches Beispiel:

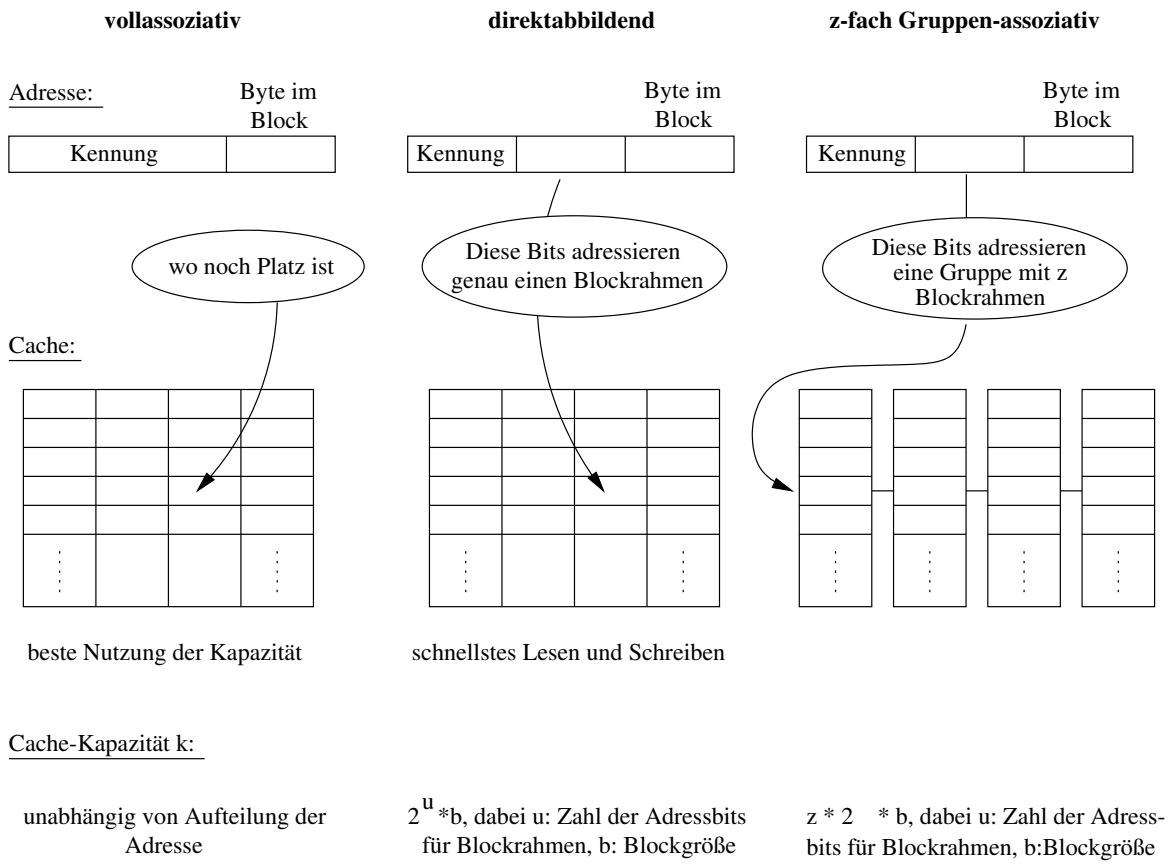


Abbildung 3.66: Varianten des Auffindens eines Worts/Blocks im CM

Miss ratio als Funktion der Kapazität des CM bei Blockgröße K_B=32B, Verdrängung LRU, VAX unter Ultrix, Mehrprogrammbetrieb, ca. 10^6 Zugriffe (Simulation!)

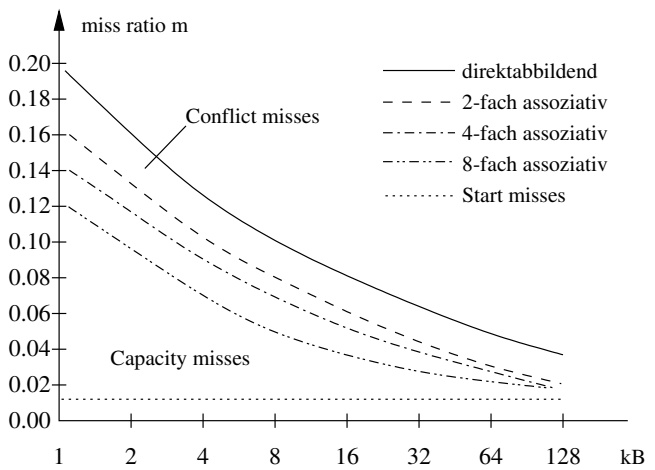


Abbildung 3.69

Die Fehlzugriffe (*misses*) sind zerlegbar:

- *Start (compulsory) Misses:* jeder zugegriffene Block muß einmal in das CM geladen werden und erzeugt davor einen *Miss*.
- *Capacity Misses*: ist die Kapazität des CM kleiner als die Menge der zugegriffenen Blöcke, dann kommt es (auch bei vollassoziativem Cache) zur Verdrängung und damit zu *miss* bei erneutem Zugriff.

- *Conflict Misses* sind diejenigen, die nicht in einem vollassoziativen, wohl aber in einem direkt abbildenden oder z-fach assoziativen Cache auftreten, da verdrängt wird, obwohl noch Blockrahmen frei sind.

Einfluß der Blockgröße K_B auf *miss ratio m*

Wenn die Blockgröße klein ist, fehlt der look-ahead Effekt. Ist sie aber nicht klein im Verhältnis zur Kapazität, führen breit gestreute Zugriffe zu Verdrängungen! Große Blöcke sind außerdem weniger schnell nachladbar, verlangen mehr Durchsatz an CM, Bus, Hauptspeicher!

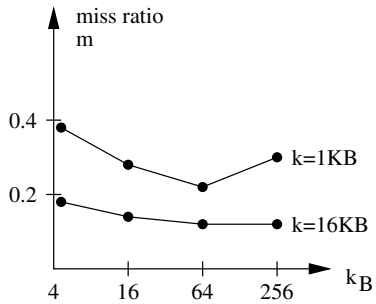


Abbildung 3.70

Also: Ladestrategie: blockweise on demand. Prefetching (vorausschauendes Laden) bisher nicht üblich, aber prefetch-Befehle!
Ersetzungsstrategie: LRU (*least recently used*) d.h. Verdrängung des am längsten nicht zugegriffenen Blocks oder Random (Zufall). Einfluß auf h ist **nicht** allzu groß. Bei direkt abbildenden Caches gibt es keine Ersetzungsstrategie (keine Wahl zu treffen)

Mittlere Zugriffszeit beim Lesen (90%)

$b_{Sp} = b_{CM} + (1-h) * b_{BlockHSp}$ z.B. $b_{Sp} = 10ns + (1-0,95) * 150ns = 18ns$

> Dabei b_{CM} Zeit für lesen der Gruppe und Prüfung auf Identität mit Blockkennung. In *hit*-Fall ist damit das Lesen abgeschlossen. Sonst (Häufigkeit 1-h) muß die Blockadresse an die Speichersteuerung geschickt werden, z.B. 4 Speichermoduln (adressverschränkt!) geben je 8B eines 32B-Blocks ab; muß an Cache geliefert werden. Gesamtzeitbadarf $b_{SpBlock}$ liegt beträchtlich über der Hauptspeicherzugriffszeit b_{HSp} (z.B. 30ns)

Der Zentralprozessor muß damit fertig werden, daß mit Häufigkeit m = 1-h der Speicherzugriff **viel** (bis zu 15x) länger braucht!
Man strebt $b_{Sp} < 2 * b_{cm}$ an, d.h. $1 - h = m < \frac{b_{CM}}{b_{BlockHSp}}$

Mittlere Zugriffszeit beim Schreiben (10%)

Der Zentralprozessor schreibt als Ergebnis typisch 1..8B, d.h. es kann mehr als ein Block betroffen sein.

Bei Treffer schreibt der Zentralprozessor blockweise in das CM: erst lesen, dann ändern, schreiben. Bei Nichttreffer (*miss*) zwei Möglichkeiten:

- zu beschreibender Block wird aus Hauptspeicher in CM geladen (*write allocate*). Dann wie oben.
- der Zentralprozessor schreibt nur in dern Hauptspeicher. Typisch wartet der ZP nicht das Ende des Schreibens ab, sondern schreibt lediglicj in einen Schreibpuffer, der ein oder mehrere Wörter faßt (*write around*).

Bei Treffer oder bei *write allocate* muß der Prozessor die Konsistenz wiederherstellen. Zwei Techniken:

- *write through (store through):* schreibt "sofort" auch in den Speicher
- *copy back*: schreibt bei Verdrängung des Blocks im Cache (und vor Verdrängung aus Hauptspeicher) zurück.

Durchsatzbedingungen

Der Prozessor laufe mit d_{ZP} Operationen/s. Je Operation macht er σ Zugriffe auf den "Speicher" (CM und HSp). Der Zugriffs-Durchsatz am CM ist $d_{CM} = \sigma * d_{ZP}$(in Wörtern/s).

Für schnelle ZP ist $d_{ZP} \approx \frac{1}{b_{CM}}$ (bei einer Pipe!) (1 Operation/Takt, $b_{CM} \approx 1$ Taktzyklus)

Mit $\sigma \approx 1,5$ kann das nur gehen, wenn

- der ZP mit jedem Cache(Block)zugriff mehrere Befehle erhält (*look-ahead*)
- zwei CM eingerichtet werden, eines für Befehle und eines für Werte!

Man bedeke, daß wir am CM auch noch den "Nachladedurchsatz" durchbringen müssen (Wortlänge w, Blockgröße K_B)
$d'_{CM} = \sigma * d_{ZP} * (1-h) * K_B/w$ (Wörter/s) z.B. $1,5 * \frac{1}{b_{CM}} * 0,05 * 32/4 = \frac{0,6}{b_{CM}}$!

Beispiel: Cache memory der VAX 11-780

k= 512 Gruppen à 2 Blocks à 8B = 8kB (Wortlänge 4B)
h z.B. 0,93; b_{CM} = 300ns; b_{Sp} = 300ns + (1-0,93)*1800ns = 426ns
$d_{ZP} \approx$ 0,7Mips; $d_{CM} = \sigma * d_{ZP}$(1+(1-h)*k_B/W + s) $\approx$ 2*0,7(1+(1-0,93)*8/4 + 0,1) = $1,736 * 10^x$
Dabei $c_{CM} = \frac{1}{b_{CM}} * k_0/W = 6,6 * 10^6$ Wörter/sek

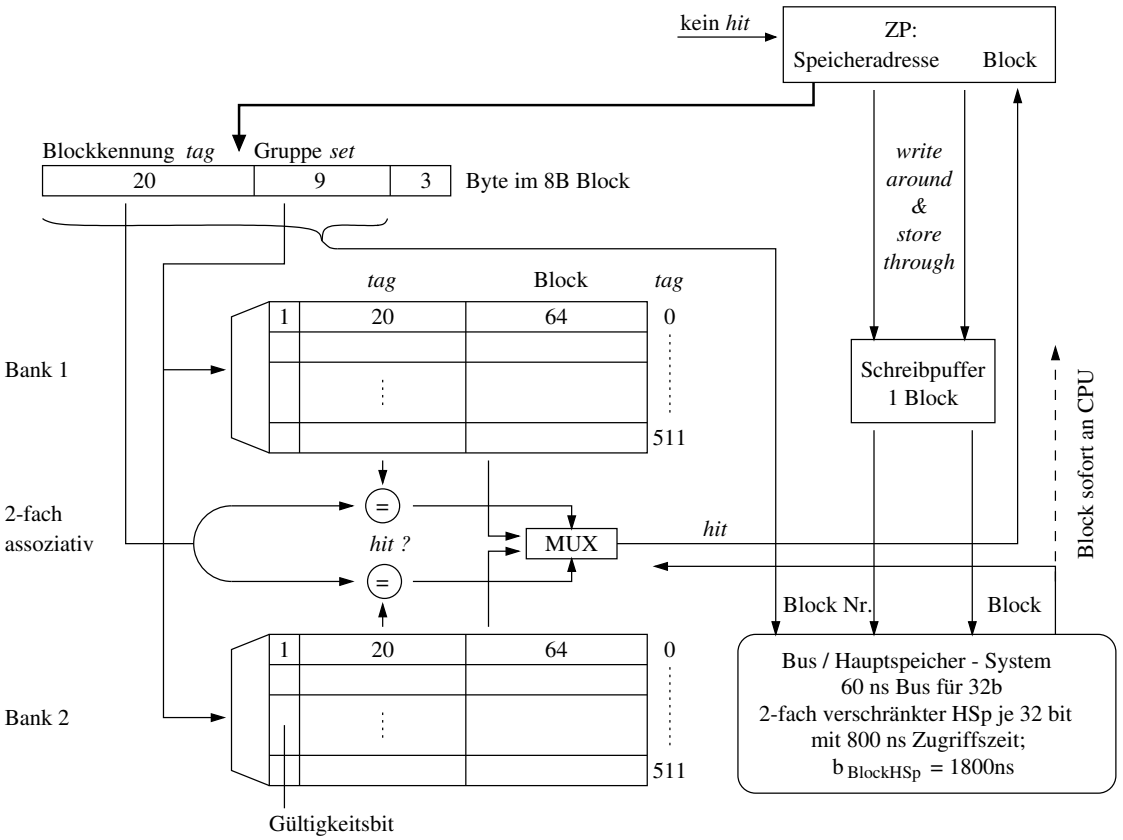


Abbildung 3.69: Beispiel: Cache memory der VAX 11-780

Hauptspeicher(Bus-)durchsatz

Der Hauptspeicher muß leisten:

- den Peripheriedurchsatz (typisch nur 10% des ZP-Zugriffsdurchsatzes)
- den Schreibdurchsatz (Annahme *write through*: $d_{ZP\,Schreib} = \sigma * d_{ZP} * s$ wobei s Schreibanteil, typisch 10%)
- den Nachladedurchsatz $d'_{CM} = \sigma * d_{ZP} * (1-h) * K_B/w$

Die beiden letzten Anteile sind überwiegend, also

$$d_{HSp} \approx d_{ZP\,Schreib} + d'_{CM} = \sigma * d_{ZP}\,(s + (1-h) * K_B/w)$$

bei typischen Werten: $\sigma = 1,5$ $d_{ZP} = 250\,Mips$ $s = 0,1$ $h = 0,95$ $K_B = 32B$ $w = 4B$ ist

$$d_{HSp} = 1,5 * 250 * 10^6/s * (0,1 + (1-0,95) * 32/4) = 187,5 * 10^6 \text{ Wortzugriffe/s}$$

Bei $b_{HSp} = 30ns$ 4 verschränkten Moduln (zwei Worte je Zugriff) leistet der Hauptspeicher:

$$c_{HSp} = \frac{4 * 2}{30ns} = 267 * 10^6 \text{ Wortzugriffe/s (knapp!)}$$

ohne Cache memory wäre $d_{HSp} = 1,5 * 250 * 10^6/s = 375 * 10^6/s$ erforderlich.

Das Beispiel lehrt:

> Die **Blockgröße ist krititisch** für die Durchsatzansprüche (und auch für die mittlere Zugriffszeit). Das CM ist geeignet, die Durchsatzansprüche an den Hauptspeicher und Bus herunterzusetzen. Wichtig bei **Mehrprozessoranlagen**!
>
> Wenn das Trefferverhältnis schlecht wird, ist das CM **schädlich**, weil die mittlere Zugriffszeit steigt und die Durchsatzanforderungen höher als bei Betrieb ohne CM sind!

Problem der Konsistenzverletzung

Dieses Problem besteht darin, daß verschiedene Werte in verschiedenen Zellen gleicher Maschinenadresse stehen. Voraussetzungen:

α:	mehr als eine Zelle gleicher Maschinenadresse vorhanden (Hauptspeicher, ein oder mehrere CM oder auch nur Befehls/Operandenvorgriffe)
$\wedge\beta$:	es wird auch geschrieben
$\wedge\gamma$:	der Schreiber erfaßt nicht alle Zellen derselben Adresse.

Gegenmaßnahmen:

$\overline{\alpha}$:	ein Speicher (CM) für alle Zugreifer (zu wenig Durchsatz, zu klein)
$\overline{\beta}$:	Bei Schreiben werden alle Zellen derselben Adresse "gleichzeitig" beschrieben (Zeitverlust, Durchsatz!) $\rightarrow$*store through*.
$\overline{\gamma}$:	Bei Schreiben werden außer dem Geschriebenen alle weiteren Inhalte von Zellen dieser Adresse für ungültig erklärt, bei Bedarf später berichtigt

Die früher beschriebenen Maßnahmen ZP schreibt in CM und "sofort": *store through/write through* oder später: *copy back/write allocate* in den Hauptspeicher bringen nur den Hauptspeicher auf den Stand des CM ! Um z.B. auch die Eingabe von Außen im CM zugänglich zu machen, muß der Kanal o.Ä. alle CM-Blöcke gleicher Adresse ungültig machen!

Allgemeines Problem: p Prozessoren mit eigenem Cache memory und daneben noch Ein/Ausgabe auf Hauptspeicher: Abbildung 3.72

Eine Lösung für Problem der Konsistenz in einem Multispeichersystem, in dem Kopien bestehen:

n Readers / one Writer-Protokoll Abbildung 3.73

Jedes Objekt (Seite, Datei) hat jederzeit

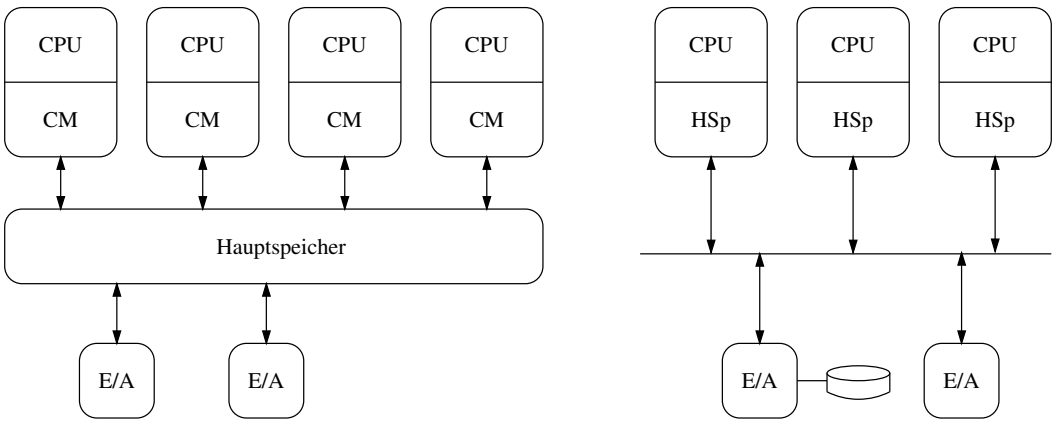
- entweder einen Schreibeigentümer, und dann sind alle Zugriffe außer dessen untersagt
- oder n (beliebig) Lesezugreifer

Adressierung des CM in Maschinenadressen hat folgende Vorteile gegenüber der Adressierung in Prozessadressen:

- gemeinsame Zellen mehrerer Prozesse sind im CM eindeutig bestimmbar: Konsistenz
- bei Prozesswechsel muß Cache memory nicht gelöscht werden (vorteilhaft, falls Prozess bald wieder aufgenommen wird, z.B. bei Unterbrechungsbehandlung). Nach Löschung arbeitet CM zunächst mit schlechtem *hit ration*! 1000 Unterbrechungen/sek $\Rightarrow$100.000 Operationen Abstand, d.h. CM-Inhalt bei großem Cache gerade aufgebaut bis zur nächsten Unterbrechung

Adressierung in Prozessadressen spart Adressumsetzung, aber

- diese ist mit Gruppenselektion parallelisierbar
- Konsistenz verlangt Umsetzung Prozessadresse(P1) → Maschinenadresse → Prozessadresse(P2) zur Erkennung von "Alias"-Zellen



Mehrprozessorsystem mit konsistentem Speicherbild in CM und HSp

Verteiltes Dateisystem oder Rechnersystem mit virtuell globalem (konsitenten!) Speicher, der auf verteilten Hauptspeichern realisiert ist, zwischen denen Kopien wandern.

Abbildung 3.70: mehrere Prozessoren und E/A

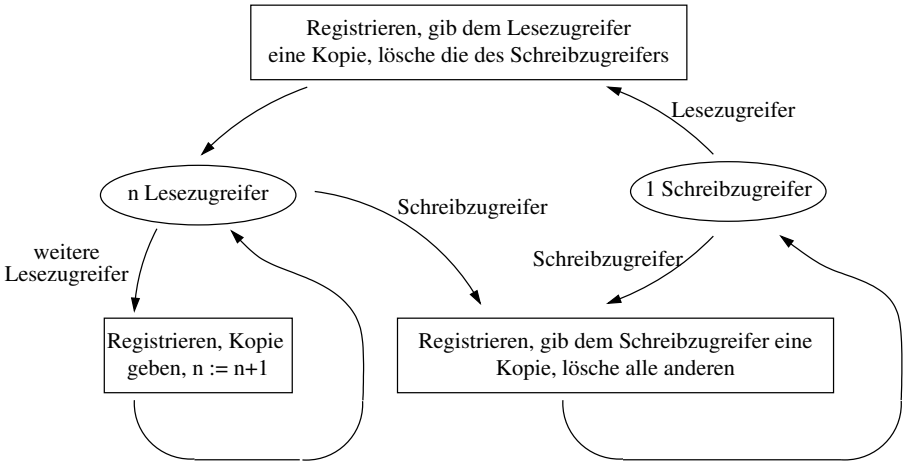


Abbildung 3.71: n reader, one writer

Beispiel: R10000

Cache	primär Befehle I-Cache	primär Daten D-Cache	sekundär S-Cache
Kapazität	32kB	32kB	0,5..16MB
Blockgröße	64B	32B 64 oder 128B	
Gruppenassoziativ	2-fach	2-fach	2-fach
Zugriffszeit	1 Takt	1 Takt	6 Takte
Adressierung	virtuell für Ort im Set, reel für Tag		
Grenzdurchsatz	128B/Takt	128B/Takt	128B/Takt
blockierend (sperrt mei miss)	nein	nein	
anderes	Befehle vorentschlüsselt	write allocate write back	write back

Konsistenzverfahren je Seite im Seitendeskriptor festgelegt (nur D-Cache): uncached, cacheable (non coherent, coherent exclusive, coherent exclusive on Write)

besondere Befehle: *PREFetch*: lädt ein Wort in das D-Cache

zum { lesen / schreiben (exclusiv!) } { in angegebenen Set / Set nach LRU }

Cache: viele Varianten

Cache { I / D / S } { Block ungültig zurückschreiben / *tag* lesen/schreiben / Synchronisation }

Pipeline

Die Bearbeitungszeit wird in k unabhängig ausführbare Abschnitte aufgeteilt, also z.b. in die genannten Phasen. Für jeden Abschnitt wird ein unabhängig arbeitsfähiges "Werk" bereitgestellt, das die Operationen nacheinander duchlaufen:

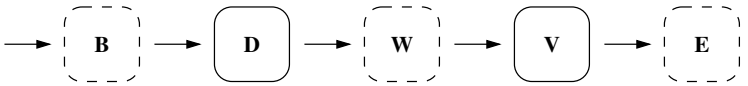


Abbildung 3.74 Phasen eines Befehls

B, W, (E) sind (fast) kein Teil der CPU, sondern werden durch (hoffentlich verschiedene) Moduln des Hauptspeichers (Cache) realisiert. (E: möglichst in Register schreiben!)Zeitschema der Operationsüberlappung:

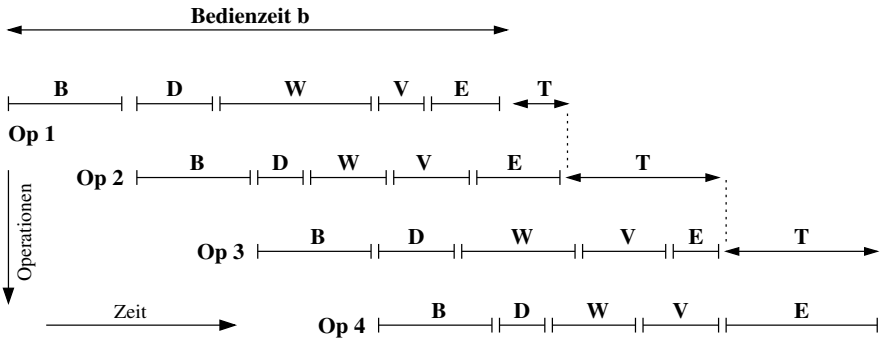


Abbildung 3.75 Zeitschema Operationsüberlappung

Nun ist $c = \frac{1}{T}$ unabhängig von b (Bearbeitungszeit)! $\overline{T}$ Abstand aufeinanderfolgender Fertigstellungen.

Ideal

Wir machen alle Abschnitte gleich lang; lange (B,W,E) teilen wir zuvor auf. z.B.

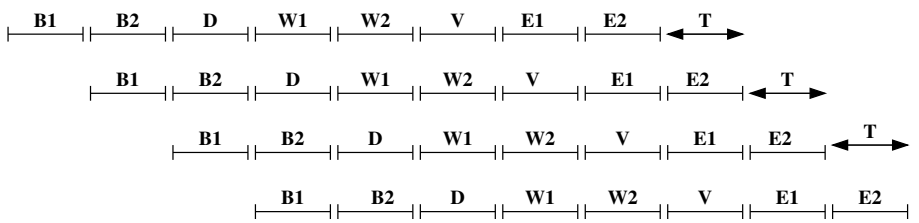


Abbildung 3.76 gleichlange Abschnitte

Es ist $T = \frac{b}{k}$; $c = \frac{k}{b}$

Wenn wir nur k genügend groß machen, darf b sehr lang (aber konstant gegen k) sein:

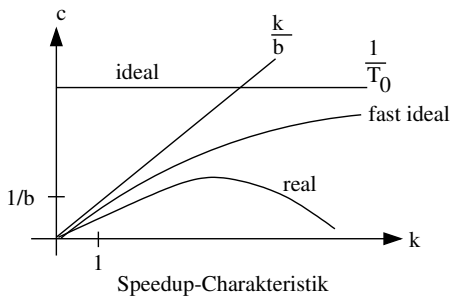


Abbildung 3.75: Speedup-Characteristik

Tatsächlich brauchen wir aber mindestens T_0 je Abschnitt (Synchronisation, Übergabe):

fast ideal: $T = \frac{b}{k} + T_0 \quad c = \frac{1}{T} = \frac{1}{\frac{b}{k}+T_0} = \frac{k}{b+kT_0} = \frac{1}{T_0}$

Real sinkt c mit wachsendem k wieder, was ein ein paar Problemen liegt. Auch schon für das Ideal haben wir einige Zwänge eingeführt:

- gleichartige Operationsgliederung
- Angleichung der Länge der Abschnitte, also wir streichen
  - exzessive Zahl von Operanden
  - schwankende Zeit für Adressmodifikationen
  - Operationen mit komplexen Verknüpfungen wie Division oder Gleipunktoperationen
    - erledigen wir durch Unterprogramm (meist unakzeptabel)
    - kommen auf einen externen Koprozessor (technologisch überholt)
    - auf eine eigene Pipeline (vgl. Pentium, R10000).

Problem 1

Trotzdem entkommt man Schwankungen der Zeitabschnitte nicht:

1. Bus/Speicher belegt
2. Cache memory hat nichts

→ Stockung des Flusses durch die Pipeline

Problem 2

Kein Wert kann gelesen werden, bevor er produziert und geschrieben ist: Operanden, Modifikationsgrößen, Befehlsadresse: *Read after Write (RAW)*-Konflikt.

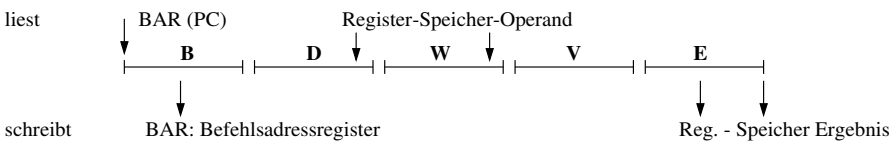


Abbildung 3.76: RAW

Abbildung 3.79 :offenbar besonders ungeschickt!
Die Größe sollte nicht in den Speicher, sondern in ein Register. Besser noch "Register-Bypass": das Rechenwerk-Ergebnis wird unmittelbar Operand der Folgeoperation: Abbildung 3.80

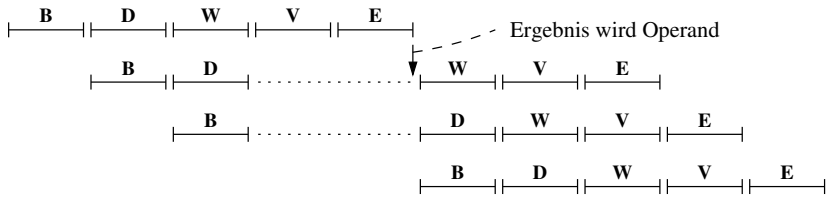


Abbildung 3.77: typischer RAW Konflikt

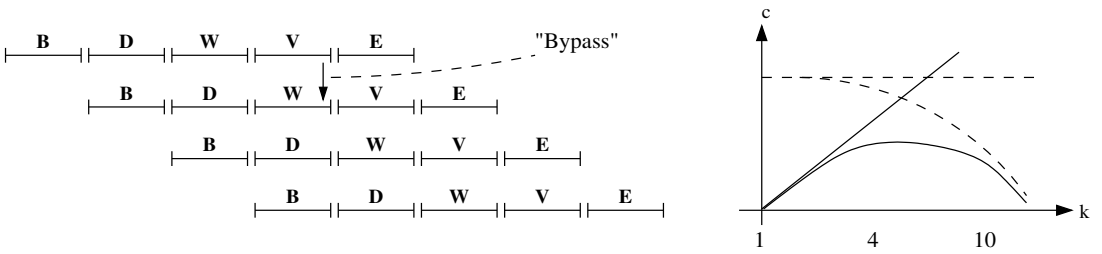


Abbildung 3.78: Register-Bypass

Problem 3

Dasselbe gilt für die kommende Befehlsadresse: sie steht erst fest

- bei einem Sprungbefehl: nach Dekodierung und eventuell notwendiger Prüfung des Kriteriums und zusätzlicher eventuell erforderlicher Adressmodifikation
- sonst: bei fester Befehlslänge sofort, bei wechselnder Befehlslänge nach Dekodierung; bei VAX aller Operand Specifier!

Also bei fester Befehlslänge:

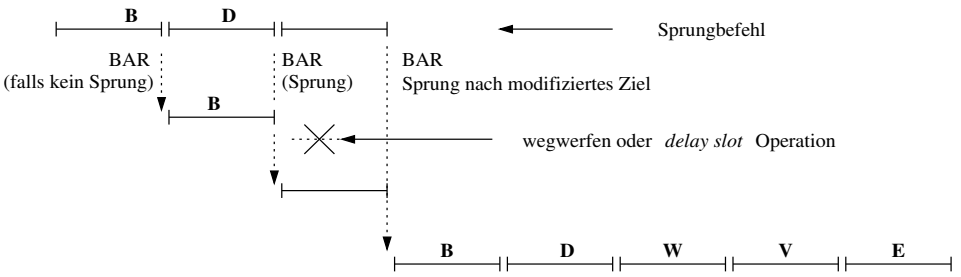


Abbildung 3.79: Sprungbefehl

Ein Befehl ist umsonst geholt worden! Unschädlich, wenn bald abgebrochen. Kann vom Compiler genutzt werden, um einen Befehl einzusetzen, der unabhängig vom Sprung auszuführen ist (*delay-branch* Technik/ *delay slot*-Operation).

Tatsächlich kann es weitere Konflikte geben, wenn sich Operationen beim Schreiben von Ergebnissen überholen: der zuletzt schreibenden bestimmt den endgültigen Wert (WAW-Konflikt, *Write after Write*) Das tritt ein, wenn Operationen nicht lediglich einmal und in derselben Phase schreiben: Operationen verschiedener Dauer.
Schließlich gibt es noch WAR-Konflikte: wenn fälschlich vor dem Lesen geschrieben wurde, wird nicht der "aktuelle", sondern der "zukünftige" Wert gelesen.

Insbesondere die Probleme 2 und 3 zeigen, daß die beliebige Vergrößerung von k keinen Vorteil bringt, z.B. Problem 3: Abbildung 3.82

Je größer wir k machen, umso mehr Operationen sind im Voraus begonnen, die im Falle eines Sprunges revidiert werden müssen.

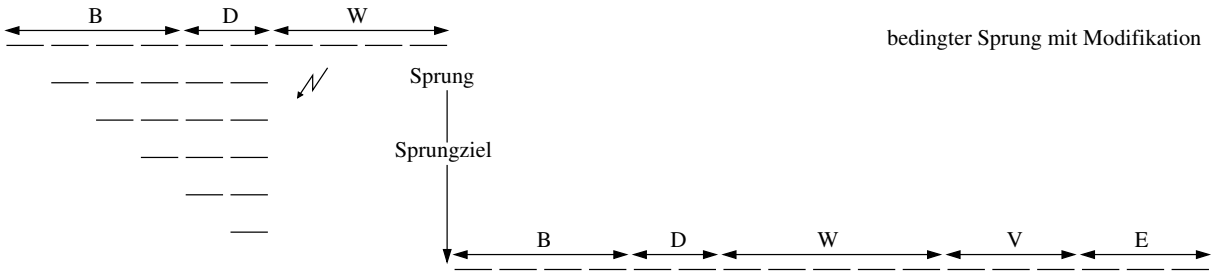


Abbildung 3.80: großes k

Also ist es zwar schön, $T_0 = \frac{b}{k}$ klein zu machen, da $c = \frac{1}{T_0}$, aber am besten indem wir b klein und k nicht allzu groß (z.B. 6) machen! Dazu:

- Hauptspeicherzugriffe durch Cache-Zugriffe ersetzen
- Operanden/Ergebnisse in Register; besondere Register↔Speicher-Befehle (*load store* Architektur).

Exkurs: Sprünge in überlappt arbeitenden Maschinen, z.B. Pipeline

Problem 4: Unterbrechungen

Bei den asynchronen Unterbrechungswünschen (z.B. Periperie, Uhr, Netzspannung) besteht das Problem lediglich darin, den Sprung/Moduswechsel in die Pipeline einzuschieben; die begonnenen Operationen werden ferttiggestellt: kein Problem!
Bei den synchronen Unterbrechungswünschen (*exceptions, faults,* z.B. Arithmetik, Seite-fehlt Alarm) ist in der Pipline nur die Phase und der Alarmtyp, aber nicht die Adresse des Befehls bekannt! Also: Behandlung/Wiederaufnahme verlangt zu den laufenden Operationen die Befehlsadresse aufzubewahren! sonst *imprecise interrupt.*
Die Unterbrechungen sollen in der Reihenfolge auftreten, die sie bei serieller Ausführung haben! Im Allgemeinen soll die Berechnung aus der Unterbrechungsstelle wiederaufnehmbar sein, also

- unvollendete Vorbefehle fertigstellen (Unterbrechungen möglich)
- begonnene Folgebefehle müssen wirkungslos bleiben!

Etwas genauer: Lösung von Problem 4 synchrone Unterbrechungen:

Bei Auftreten eines Unterbrechungswunsches (bei mehreren: den bei serieller Ausführung frühesten) Nachtyp, Phase (B,D,W,V,E...) und betroffenen Befehl merken.

Allen Folgebefehlen das Schreiben verbieten, sobald wie möglich stillsetzen/rücksetzen.

Die Vorbefehle vollenden, Sprünge setzen, vollziehen! Bei Auftreten von Unterbrechungen nach derselben Regel vorgehen.

Früheste Unterbrechung ausführen; unterbrochenen Befehl ggf. wieder aufnehmen. Folgebefehle neu starten.

Pipelines sind gut,

- weil sie beträchtliche Leistungssteigerung bringen (z.B. 5 Stufen; $c = \frac{1}{1,2*T_0}$; d.h. Speedup von $\frac{5}{1,2} = 4,17$

und problematisch

- weil ihre Effizienz stark vom Befehlssatz abhängt.

Mehrfache Pipelines (*superscalar processors*) wirken (m-Pipelines) in erster Näherung wie eine Pipeline der Stufigkeit m*k (nämlich unserer Probleme 1..4). Sie erfordern aber für die n-fache Leistungssteigerung nicht den n-fachen Takt. Sie sind dann auf Operationstypen spezialisierbar; aber den Durchsatz bringen sie nur, wenn die Operationen in der passenden Mischung auftreten! Prozessoren mit hoher Stufigkeit heißen *super pipelined*

Beispiele

	Pentium III	R12000
Befehlswerk-Pipes	3 (9 Stufen)	4 (3 Stufen)
je Takt maximal gestartet	3	4
FP-Pipes	2 (12 Stufen)	2 (6 Stufen)
GP-Pipes	1	2
Sprungpipe	1	x
laden,speichern	2 (14 Stufen)	1 (6Stufen)
Multimedia-Pipe	2	x
Anzahl Pipes	8	5
Zahl der Operationen, deren Ausführung sich überholen kann	40 Risc-Ops ca.15 Intel-Befehle	48

3.10 Unkonventionelle Rechner

Die folgenden Neuerungen gegenüber von Neumanns Konzept sind so weit verbreitet, daß sie als "konventionell" angesehen werden:

- Unterbrechungen
- Adressmodifikationen
- kollaterale Operationen bei scheinbarer Aufrechterhaltung der Sequenz
- Ausschnitt(shierarchie)
- Interpretationshierarchie

Die ersten beiden sind Weiterentwicklung der Architektur (Blaauwsche Architektur), die letzten drei sind (schwerwiegende) Verbesserung der Realisierung.

Die unkonventionellen Maschinen folgen den folgenden Koordinaten:

- Kollateralität an der Hardware/software Schnittstelle, d.h. vom Programmierer gestaltbar
- Abkehr vom von Neumannschen Programm- und Variablenkonzept

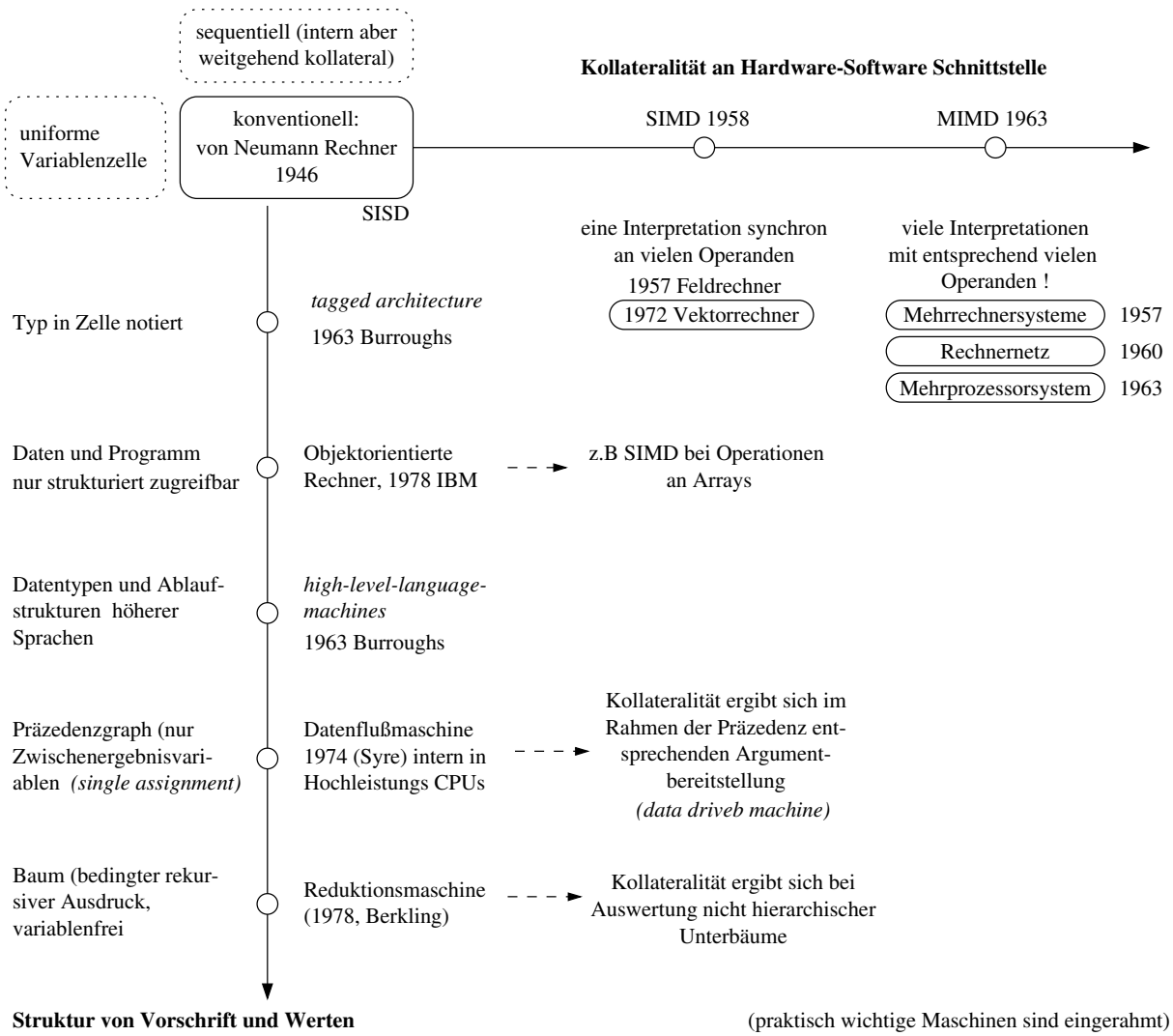


Abbildung 3.81: Überblick unkonventionelle Rechner

SIMD (*sigle instuction stream, multiple data stream*): eine Programminterpretation, die in (fast) jedem Schritt viele Werte verarbeitet (→ 2.11 Klassifikation):

Feldrechner: ein gewöhnlicher Programmspeicher, ein gewöhnliches Leitwerk, aber k Rechenwerke in regulärer Anordnung, die gleichzeitig die gleiche oder keine Operation ausführen. Technisch weniger wichtig als der

Vektorrechner: ein gewöhnlicher Programmspeicher, ein gewöhnliches Leitwerk, ein m-stufiges Rechenwerk nach Pipelineprinzip, in dem m Komponenten von Vektoren in verschiedenen Phasen der gleichen Operation sind.
Technisch bis 1995 wichtig als Hochleistungsrechner z.B.: Cray, Fujitsu VPP700:

- mehrere Pipelines je Prozessor, Verkopplung von Pipelines (*chaining*)
- mehrere Prozessoren
- wirksame "vektorisierende" und parallelisierende Compiler (insbesondere Fortran)

assoziativer Rechner: ein gewöhnlicher Programmspeicher, ein gewöhnliches Leitwerk, alle Daten und Operationen in p (groß!) Zellen eines assoziativen Speichers: Auswahl der Zellen für lesen, schreiben und verarbeiten nur aufgrund inhaltlicher Kriterien, nicht mit Adressen. selten

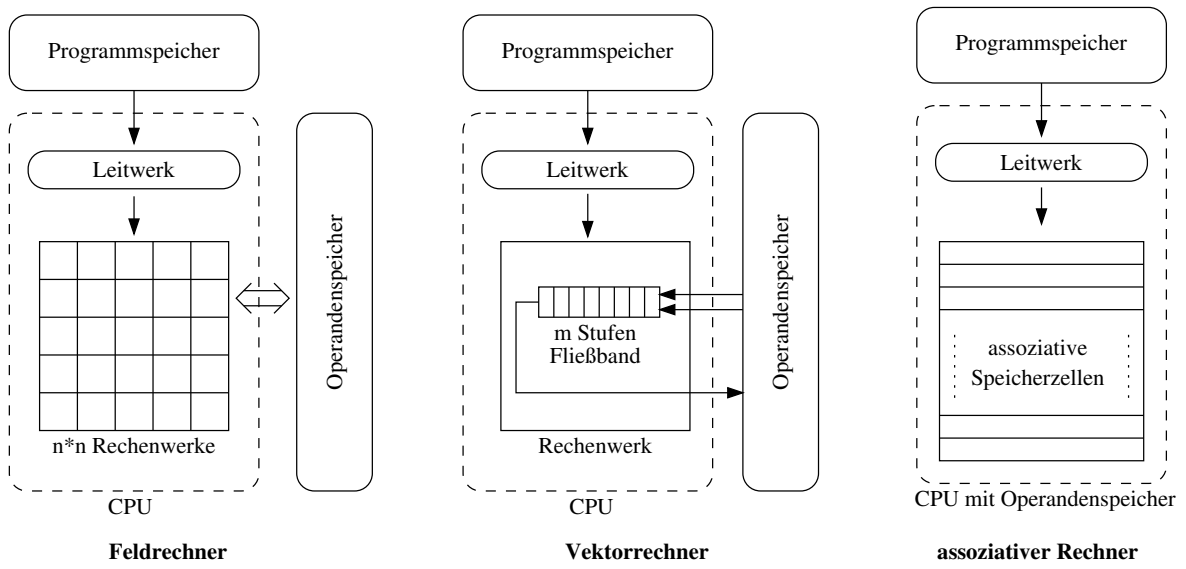


Abbildung 3.82: SIMD

MIMD: Maschinen mit mehreren Programminterpretationen, die je einen Wertestrom verarbeiten (*multiple instruction streams, multiple data streams*) (→ 2.11 Klassifikationen):

Mehrprozessoranlagen: rechnen mit mehreren Zentralprozessoren, die gemeinsamen Hauptspeicher benutzen, gemeinsames Betriebssystem. Häufig an der Leistungsobergrenze, wo genügend starker Einzelprozessor nicht verfügbar. →Kap.7
Mehrrechner-Systeme: Hyperstruktur durch Verbindung von (z.B. von Neumann) Rechnern. → Parallelrechner, → Rechnernetze (eigene Vorlesungen)

Nun die zweite Richtung unkonventioneller Rechener: Abkehr vom von Neumannschen Konzept des sequentiellen Programms und der uniformen wiederverwendbaren Zellen:

Tagged architectures: Typenkennung in Speicherzellen. Kleiner Schritt, nutzbar für

- Prüfung der korrekten Verwendung von Typen: z.B. keine Gleitpunktzahl als Sprungziel oder
- Anpassung der Operationen (generische Operationen) an den Typ, dann z.B. ein Additionsbefehl für Gleit- und Festpunktzahlen.

Seit 1963 (Burroughs B5000) bekannt

Objektorientierte Rechner: auf konventionellen Maschinen werden Zugriffe in Daten- oder Programmstrukturen durch Berechnung von Adressen durch den Compiler bzw. sein Laufzeitsystem (i.A. zusätzlich unterstützt durch die maschinelle Transformation von Programmadressen auf Prozessadressen) ermöglicht. Wird durch Softwarefehler oder Programmiererlist eine Zugriffsadresse anders gewonnen, wehrt sich der

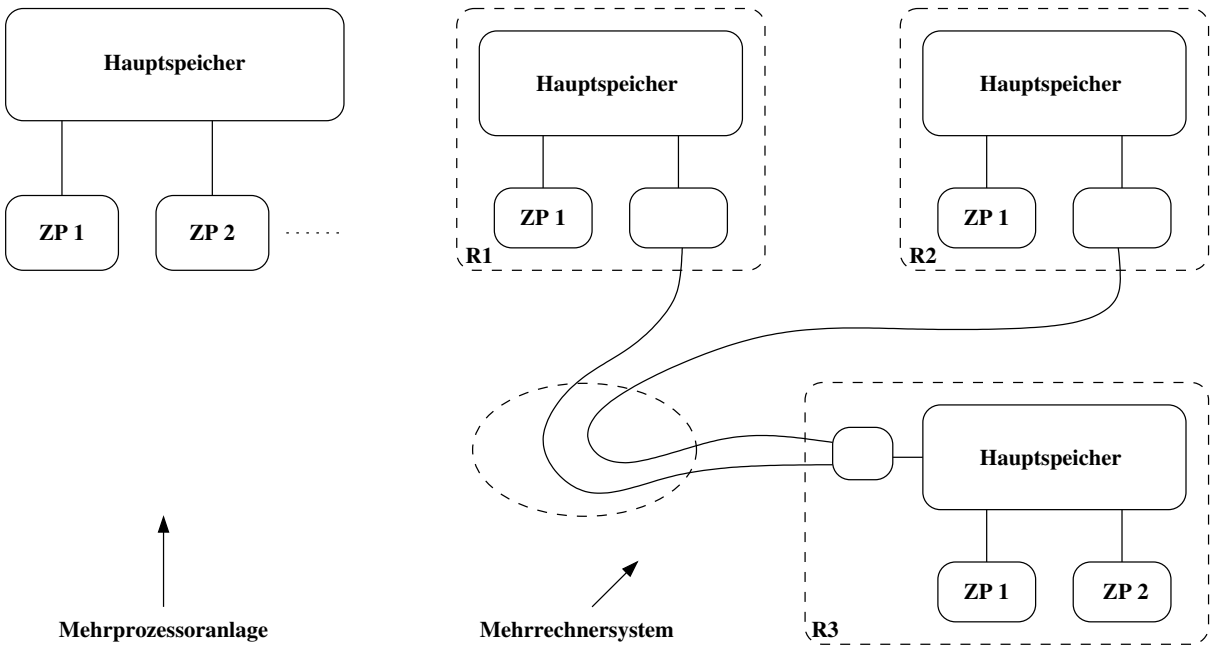


Abbildung 3.83: MIMD

Rechner nicht (solange nicht fremde Adressen nachgefragt werden).
Objektorientierte Maschinen setzen Adressen für die Programmläufe außer Kraft und lassen Daten- und Programmstrukturen ("Objekte") symbolisch bezeichnen und erlauben Innenzugriffe nur über das Symbol; die Speicherabbildung bleibt geheim. Bei Zugriff wird Berechtigung geprüft. "Verkapselung" (*encapsulation*).
Objektorientierte Rechner bieten im Allgemeinen zugleich die Funktionen der tagged architectures. Zugriffe werden indirekt (typisch durch mikroprogrammierte Prozeduren) ausgeführt. Sicherheitsgewinn gegen ziemlich hohen Leistungsverlust.

Datenflußmaschinen ersetzen das sequentielle Programm durch einen Präzedenzgraphen, z.B.

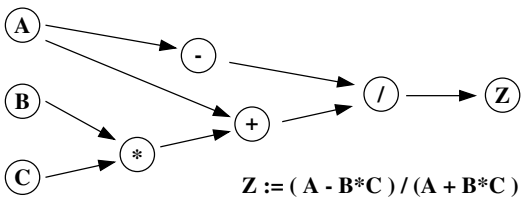


Abbildung 3.84: Präzedenzgraph

Freigegeben wird jeweils der Rechenschritt, dessen Argumente bereitstehen (*data driven*-Prinzip). Zur Synchronisation müssen im Allgemeinen Ergebnisse festgehalten werden (H1..H4); diese Variable enthalten nur eine Wertzuweisung (*single assignment,* Bauers Zwischenergebnisvariable).
Datenflußmaschinen brauchen einen Pool von beauftragbaren Werken (sonst ist die Nebenläufigkeit nicht nutzbar). Erweiterung um bedingte Weitergabe von Werten und um Iteration.
Trotz der kollateralen Berechnungen ist das Gesamtergebnis funktional, da die Argumentweitergabe gerade die notwendige Synchronisation ergibt. →Vorlesung "Parallelrechner"

High-Level-Language Maschinen spezialisiert auf eine oder wenige, verwandte höhere Sprachen; führen (meist vorübersetztes) Quellprogramm durch Interpretation (z.B. mikroprogrammierter Interpreter) aus. Erreichbar sind Laufzeit- und Speicherplatzvorteile von 2..5 gegenüber konventioneller Maschine ähnlichen Aufwands.

Reduktionsmaschine: die Rechenvorschrift ist ein variablenfreier Ausdruck; mit bedingten und rekursiven Ausdrücken ist die Maschine nicht minder mächtig (Vorbilder ohne Speicherbegrenzung: λ-Kalkül,

Turingmaschune). Die Berechnung erfolgt, indem jeweils berechenbare Teilausdrücke berechnet und durch ihren Wert ersetzt ("reduziert") werden, bis der gesamte Ausdruck zu einem Wert reduziert ist.
Die Werte sind durch ihre Position im Ausdruck, nicht durch Bezeichner realisierbar. Es gibt keinen anderen Zustand der Berechnung als den aktuellen Ausdruck: einfache Semantik, einfacher Nachweis von Eigenschaften der Berechnung [Klaus Berkling 1978].

Kapitel 4

Hauptspeicher und Verkehrswege

4.1 Speichertechnik [HAHN BAUER 75] vgl. TGI-Vorlesung

Strukturspeicher: Speicherung durch Änderung der Struktur eines festen Körpers. Das Lesen und Schreiben, nicht die Aufrechterhaltung des Gespeicherten kostet Energie:

- magnetisierbare Schichten (Platte, Band)
- optisch Speicher (CD-ROM)
- Lochkarten/Streifen

Speicher mit stabiler Strom/Ladungsverteilung: Schreiben und Aufrechterhaltung kostet Energie (flüchtige Speicherung!), Lesen kostet wenig Energie:

- Flipflops (statische Halbleiterspeicher)
- Ladungsspeicher (dynamische Halbleiterspeicher)

Aufbau als Matrixspeicher, *random access memory*, keine Bewegung.

Kapazität, Zugriffszeit, Zykluszeit

Typ	Zugriffszeit	Durchsatz
intern, flüchtig	30ns	2GB/s
peripher, permanent	3 ms	20MB/s
Verhältnis flüchtig/permanent	$1{:}10^5$	1:100

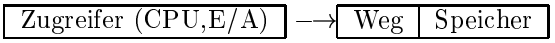
Die Verhältnisse führen zusammen mit der Lokalität zu Blocktransporten und zu Mehrprogrammbetrieb (neben anderen Gründen)

Die hohe Mindestzugriffszeit (Kopfeinstellung und Restumdrehung) und die oben genannten Lokalitätseigenschaften führen zu **Blockzugriffsverfahren** mit großen Blöcken (256..4096B) am Hintergrundspeicher; daher lohnt sich die Einführung von Werken, die unabhängig vom Prozessor Blocktransporte ausführen:

- Kanäle
- E/A-Prozessoren
- *Direct memory access*-Steuerwerke

4.2 Zugang der CPU zum Hauptspeicher

Für den Zugreifer erscheint der Hauptspeicher als Teil S eines Systems WS



mit dem Weg zum Hauptspeicher (Transportaufträge) und dem Speicher (Lese/Schreibaufträge). Zeitliche Abwicklung:

Leseauftrag (90% aller Zugriffe)

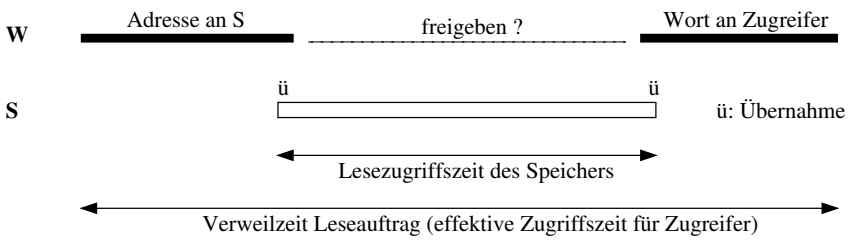


Abbildung 4.1: Lesezugriff

Schreibauftrag

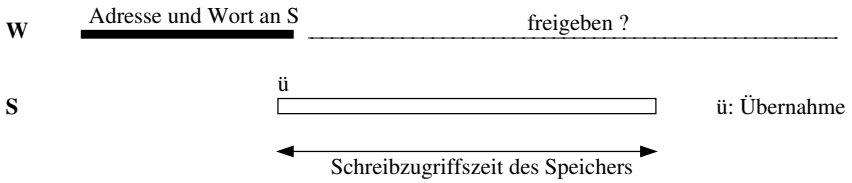


Abbildung 4.2: Schreibzugriff

In Rechnern gibt es im Allgemeinen viele koexistente Zugreifer:

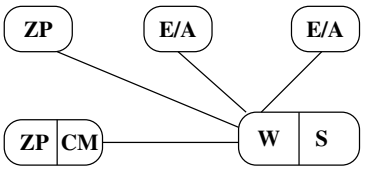


Abbildung 4.3: mehrere Zugreifer

- ZP Zentralprozessoren
- E/A Ein/Ausgabeeinheiten
  - Steuerwerke
  - Kanäle...

Zentralprozessoren hoher Leistung haben i.A. zu einem bestimmten Zeitpunktmehr als einen offenen Auftrag an WS, wegen Fließbandtechnik, Cache memory nachladen in Blöcken. Nun wird nach folgendem Schema vorgegangen:

A nur ein Zugreifer vorhanden

A1 höchstens ein offener Auftrag

A2 mehr als ein offener Auftrag (nicht relevant)

B nur ein Zugreifer aktiv

B1 höchstens ein offener Auftrag

B2 mehr als ein offener Auftrag (nicht relevant)

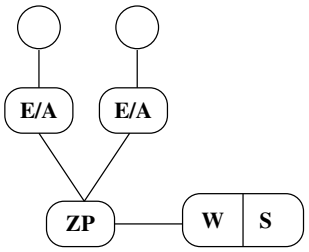
C mehrere Zugreifer aktiv

C1 WS einfache Funktionseinheit

C2 WS keine einfache Funktionseinheit:

* W getrennt zuteilen
* S aufspalten (Verschränkung)
* gewidmete Wege

A. Nur ein Zugreifer vorhanden



Das ist trotz E/A möglich, wenn der (einzige!) Zentralprozessor die WS-Aufträge für alle abwickelt! Dann wird nicht gerechnet, solange ZP für E/A transportiert. Wohl kann es Betrieb der Geräte G gleichzeitig zum Rechnen geben, wenn nämlich E/A genügend große Puffer hat. Ist ZP ohnehin nicht ausgelastet, dann kann das eine gute Organisation sein.

Abbildung 4.4: 1 Zugreifer

A.1 ZP habe jederzeit höchstens einen offenen Auftrag an WS.

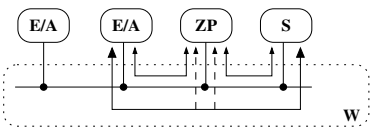


Abbildung 4.5

Dann W trivial. Der Weg W kann auch noch Zugang zu E/A bieten (Bus) Transport über ZP-Register oder mit E/A Befehlen oder mit Transportbefehlen (dann E/A adressierbar) scheinbar direkt, tatsächlich über ZP.

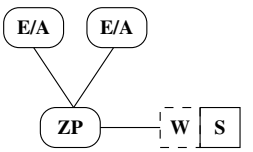


Abbildung 4.6

A.2 Mehr als ein unerledigter ("offener") Auftrag. Nicht üblich bei einfach-ZP; und komplexe ZP haben E/A mit direktem Zugriff auf WS

B Höchstens ein Zugreifer ist aktiv,
obwohl mehrere vorhanden sind; Gegenseitige Synchronisation.

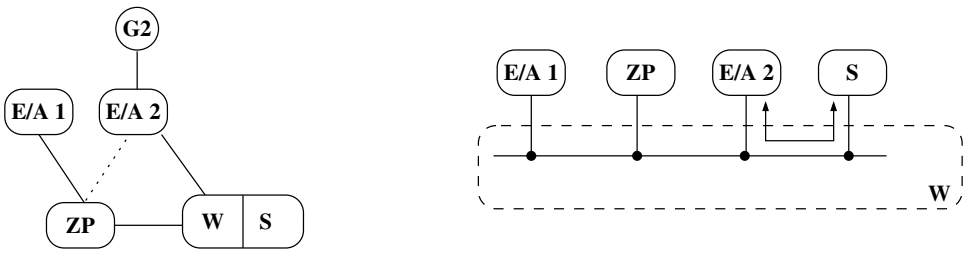


Abbildung 4.7: höchstens ein Zugreifer aktiv

E/A 2 z.B. habe eigenen Speicherzugang, z.B. Steuerwerk mit direktem Speicherzugang (DMA, *direct memory access controller*). Auch nun gibt es kein Rechnen, solange E/A 2 zugreift; Kann ZP aber den für G2 notwendigen Durchsatz nicht aufbringen, oder kürzt auch nur E/A 2 durch Direktzugang die Transportzeit wesentlich, wird diese Struktur notwendig/nützlich sein.

Wieder ist der Fall "mehr als ein Auftrag" nicht relevant.

C Mehrere Zugreifer können gleichzeitig aktiv sein

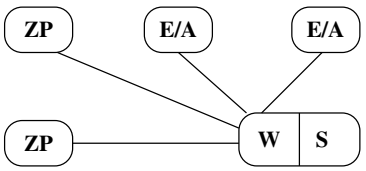


Abbildung 4.8: Schreibzugriff

C.1 Zunächst sei WS weiter eine "einfache" Funktionseinheit (→Def.2.9). Falls nun bisher ZP oder E/A WS nicht weitgehend mit ihren Zugriffen ausgelastet hätten, dann ergibt sich eine Leistungssteigerung

durch den Multiplexbetrieb aller Zugreifer, obwohl Grenzdurchsatz von WS unverändert. Nun wird ein *arbiter* A (Vorrangwerk) notwendig, das Zugriffskonflikte entscheidet; typisch Vorrang für E/A mit zeitkritischer Peripherie, z.B. Plattenspeicher.
Der Multiplexbetrieb auf Ebene einzelner Zugriffe heißt auch *cycle stealing*. Jetzt sind gleichzeitig Rechnen und E/A-Transporte möglich. Puffer höchstens zum Ausgleich kurzzeitigen Wartens erforderlich.

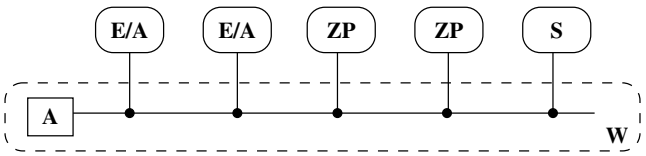


Abbildung 4.9: WS als einfache Funktionseinheit (hier Bus)

Für größere Ansprüche ist nun WS als einfache Funktionseinheit nicht mehr akzeptabel →

C.2 WS keine einfache Funktionseinheit (d.h. mehr als ein Auftrag zugleich). Erster Schritt W freigeben, sobald möglich:

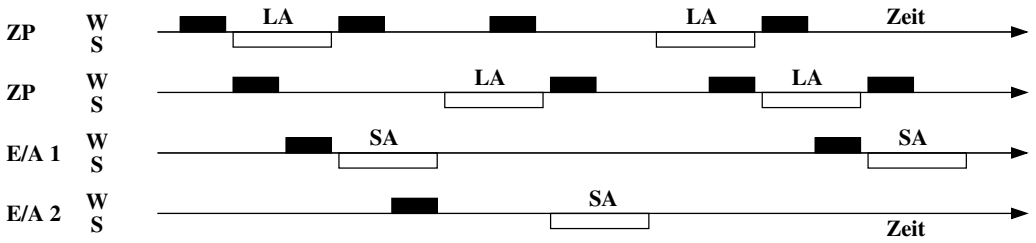


Abbildung 4.10: W freigeben sobald möglich

Nun Arbitrierung auch an S notwendig; Zwischenspeicherung von Zugriffsaufträgen vor S. Besser: zusätzlich Grenzdurchsatz von S verdoppeln, durch Aufteilung in zwei einfache Funktionseinheiten (1,2):

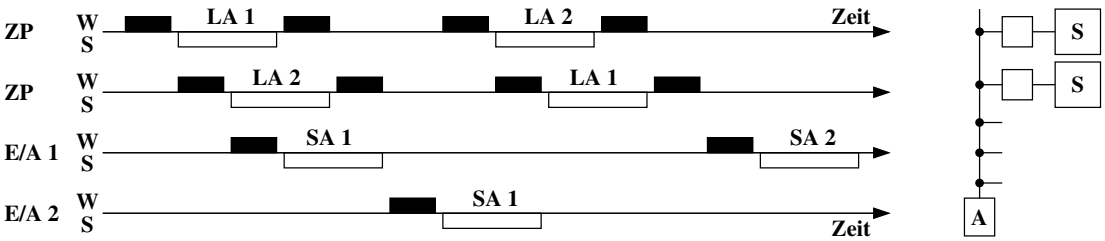


Abbildung 4.11: Aufteilung in zwei einfache Funktionseinheiten

Offenbar ist ausgewogene Verteilung der Zugriffe auf S1 und S2 Voraussetzung.

C.2 WS keine einfache Funktionseinheit; Erhöhung des Grenzdurchsatzes durch Aufspaltung. Man stellt die annähernd gleiche Verteilung der Zugriffe sicher durch Adressverschränkung (*interleaving*): der Speicher wird in 2^k unabhängig arbeitsfähige Moduln von 2^m Zellen geteilt. Die letzten k Adressbits werden als Moduladresse benutzt.

Adr. im Modul	Adr. des Mod.

Auf diese Art und Weise werden kosekutive Adressen im Zyklus modulo 2^k auf die Moduln verteilt. Konsekutive Zugriffe durchlaufen also zyklisch alle Moduln

Bemerkungen: einfach, wirksam aber

- ungeeignet bei Ausfall eines Moduls
- Verschränkung hängt von Zahl der Moduln ab (kauft Kunde nicht 2^k Moduln der Größe 2^m; dann
  - 2^{k-1}, 2^{k-2} Verschränkung oder
  - verschiedene Verschränkung im Adressraum
- Probleme mit Moduln verschiedener Größe

Weitere Fortschritte sind nur noch möglich, wenn W in gewidmete Wege aufgelöst wird. Damit ist die Freigabe von W nicht mehr nötig (wenigstens solange nicht mehr als ein offener Auftrag je Zugreifer).

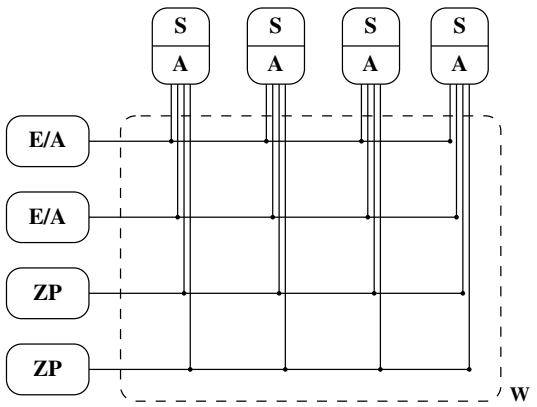


Abbildung 4.12: Gewidmete Wege statt Bus engen die Konfigurationsmöglichkeiten ein!

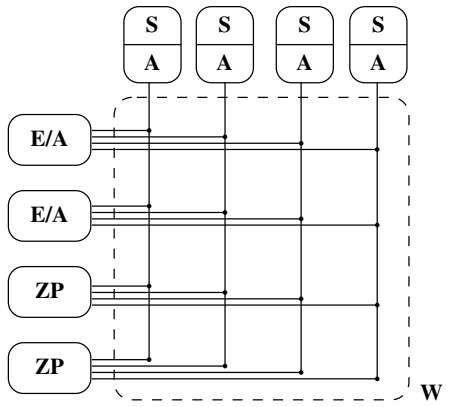


Abbildung 4.13: Duale Form ist dann günstiger, wenn die Zugreifer mehr als einen nicht erledigten Zugriffswunsch haben dürfen: Pipeline-Prozessoren, Cache memory

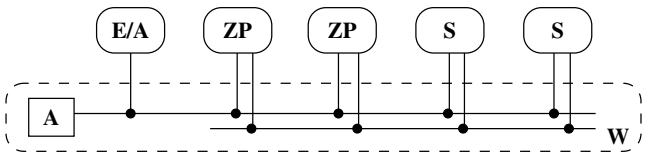


Abbildung 4.14: Besonderer Speicherbus zwischen ZP und S, neben dem Zentralbus

Weniger aufwendig, aber durchsatzschwächer sind Lösungen mit zentraler Arbitrierung:

Zentaler Arbiter kennt Zugreiferprioritäten und Belegung des S-Moduln. Darf Zugang nicht erschweren!

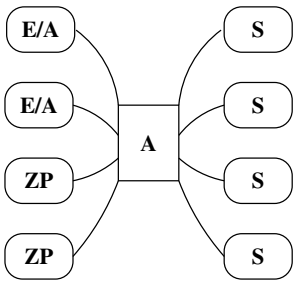


Abbildung 4.15: zentrale Arbitrierung

Zusätzliche Wünsche machen ihn zum Speicherzugangswerk:

- Generierung von Adressfolgen z.B. für Cache nachladen
- Pufferung von Aufträgen
- Fehlersicherung

Bei großer Zahl von Zugreifern sind zentrale Lösungen nicht mehr akzeptabel und auch nicht notwendig. Bsp. Parallelrechner: siehe 2.5

Rückblick auf **Zugreifer, die mehr als einen Zugriffsauftrag haben, der unerledigt ist**:

Bei einfachen gewidmeten Wegen (Abb. 4.12 und 4.15) ist ein Freigeben des Weges vor Beendigung notwendig! Bei mehrfachen gewidmeten Wegen (Abb. 4.14) ist dies nicht notwendig, falls hinreichend viele S zugänglich.

Achtung! Leseaufträge können sich durch verschiedene Wartezeiten überholen! Eintreffendes Wort muß durch Adresse identifiziert werden, die mit ihm zurückläuft.

4.3 Hauptspeicherzugang über Bus

[HENNESSY PATTERSON 96], 6.3 Bus: vgl. 2.5. Seite 26, Bushierarchie in Workstation/PC: S. 17

Einsatz/Bauformen:

(reiner) Prozessor-Speicherbus: verbindet nur Prozessor(en) und Speichermoduln miteinander, schnell, "breit", rechnerspezifisch.
Prozessorspezifischer System-(Zentral)Bus: verbindet Prozessor-Speicher Bus (Busübergang!) und E/A-Einheiten und eventuell weitere Speichermoduln
Standardisierter System-(Zentral-)Bus: nach herstellerunabhängigem oder *de facto* Standard; daher breites Angebot von passenden Baugruppen und Programmbausteinen verschiedener Hersteller.
Peripherie-(I/O) Bus: ebenso, aber auf Vielzahl von Geräten kleinen und mittleren Durchsatzes orientiert.

Beispiele zu Prozessor-Speicher Bussen:

	HP Summit	SGI Challenge	SUN XD Bus
Breite (Daten)	128b	256b	144b
Takt	60MHz	48MHz	66MHz
Bus-Master	mehrfach	mehrfach	mehrfach
Grenzdurchsatz	960 MB/s	1200MB/s	1056MB/s
Standard	nein	nein	nein

alle diese Busse geben den Bus innerhalb einer Transaktion frei (vgl. Seite 135)

"split transaction" , " connect/disconnect"
"pipelined bus" , "packed switched bus"

und erlauben Mehrworttransporte *"burst mode"* (verringert Adress-overhead)

Diese Busse arbeiten nach starrem Zeitschema ("Synchronbus"), spart Synchronisationsoverhead, ungeeignet für lange Busse mit Einheiten unterschiedlichen Zeitverhaltens.

Zentralbus

vgl. 2.5 zentale Steuerung ZS, Buszuteilung,Takt, anderes.

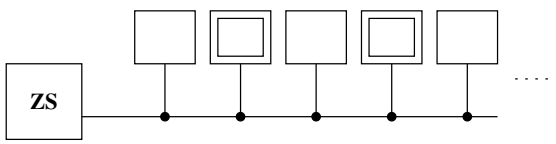


Abbildung 4.16: Zentralbus

Eine Einheit (aktive Einheit, Skizze:Doppelkasten), die gerade den Ablauf über den Bus vorgibt heißt *master*. Eine andere Einheit (aktiv oder passiv), die gerade passiver Partner des *master* ist heißt *slave*. Im einfachsten Fall ist nur der ZP *master*; bei mehreren ZP oder bei DMA-E/A-Einheiten muß der Bus multi-master-fähig sein. ZS muß dann Zuteilungskonflikte auflösen (*arbiter*-Funktion).

Der Bus umfaßt Datenleitungen, (meist) Adressleitungen, Steuerleitungen, Versorgungsleitungen.

Teilnehmer am Bus

Nach Anzahl begrenzt durch

- elektrische Grenzen, zulässige Buslänge
- Adressierbarkeit
- (manchmal: durch Verfügbarkeit von teilnehmereigenen Leitungen im Bus)

Teilnehmerfunktionen:

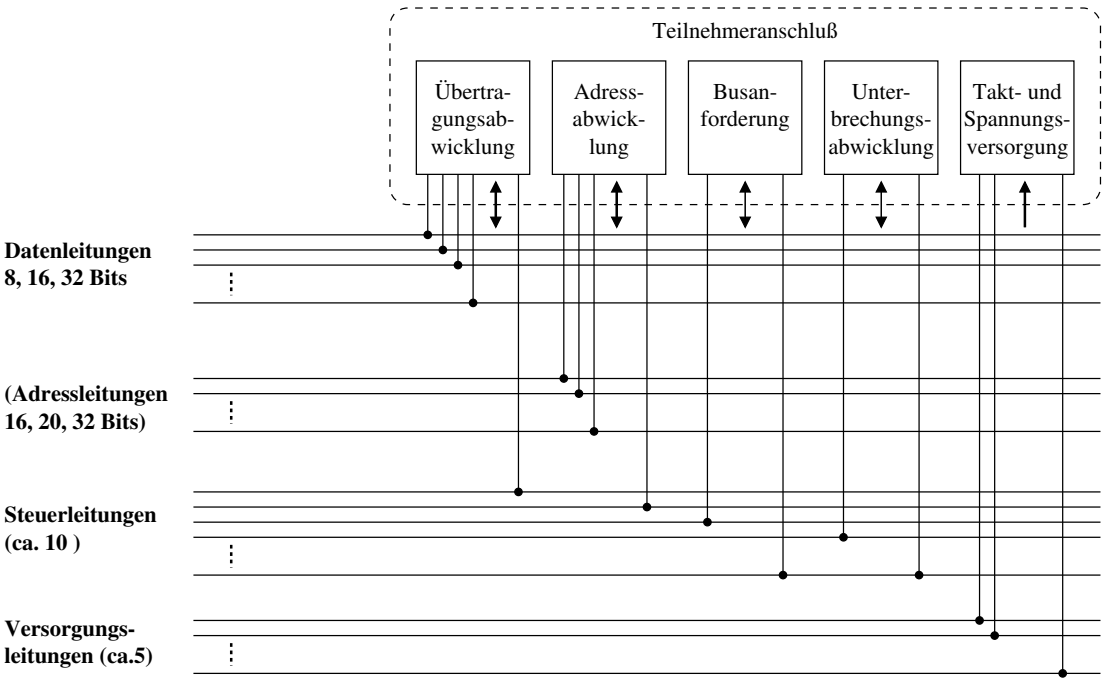


Abbildung 4.17: Anschluss an einen Bus

Zwei Grundfunktionen: Datenübertragung und Unterbrechungswunschübertragung

Datenübertragung

Busanforderung: der Teilnehmer meldet sich bei der Steuerung an:

a. über eine Sammelleitung oder
b. über eine ihm zugehörige Leitung (begrenzte Zahl!) oder
c. über ein anderes Verfahren

Identifikation:

b. trivial
a. durch Abfrage seitens der Steuerung (*polling*)
   oder auch ohne Identifikation (durch *daisy chaining*, siehe unten)

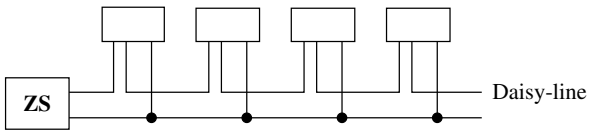


Abb. 4.18 daisy chaining

Auswahl eines von mehreren Anforderern:

- feste/veränderliche Prioritäten
- oder nach Lage am Bus: *daisy chaining* Abb. 4.18 Das Auswahlsignal geht von ZS aus, wird vom nächsten Anforderer aufgenommen und nicht weitergegeben.

Sobald der Bus frei ist, wird der Ausgewählte *master* und beginnt mit **Übertragung** in Schreib- oder Lesezyklen:

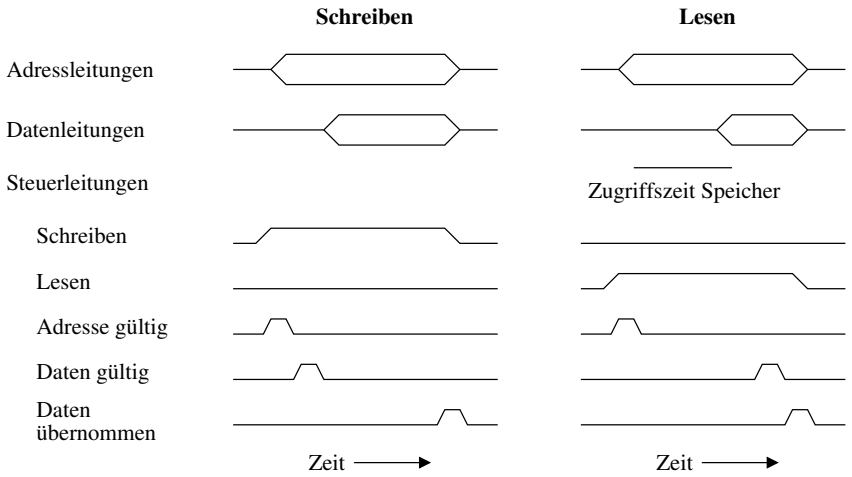


Abbildung 4.18: asynchrones Quittungsprotokoll einfachster Art (*hand shake*)

Sind keine Adressleitungen vorhanden, so werden die Datenleitungen hierfür eingesetzt (Streckung des Schreibzyklus).

Neben der Datenübertragung muß der Bus noch den Transport von **Unterbrechungswünschen** leisten. Im Wesentlichen wie Anforderung-Identifikation:

a. Sammelleitung oder

b. Sammelleitung der Gruppe, der der Teilnehmer zugeordnet ist, z.B. 8 Unterbrechungsebenen oder

c. über eine ihm zugehörige Leitung

Identifikation: wie bei Busanforderung

Kapitel 5

Periphere Speicher

Es gibt kein gutes, aktuelles Buch über Rechnerperipherie, aber Systematik und Physik und Technologie älteren Standes in [PROEBSTER W 87] Peripherie von Informationssystemen, Springer 1987

5.1 Bauformen und Technologie der peripheren Speicher mit bewegtem Medium

5.2 Magnetische Plattenspeicher

5.3 Optische periphere Speicher

Abgrenzung der peripheren Speicher (*storage*) von den zentralen (Hauptspeicher, (*memory*):

- Zugriffszeitlücke: 8ms:20ns ($4 * 10^5$) zwischen Hauptspeicher und Plattenspeicher
- hoher Sockelbetrag der Zugriffszeit bei gutem Grenzdurchsatz (z.B. 10ms zu 20MB/s, d.h. 10ms + 0,2ms für 4KB Seite) Folge:
  - Blockzugriffe statt Wortzugriffe
  - Anschluß über blockübertragende Steuerwerke/Kanäle, Mehrprogrammbetrieb

Ersatz der langsamen peripheren Speicher durch Speicher mit Mikro-Sekunden Zugriffszeit oft versucht (Halbleiter, Optik, Magnetik), liefert keine preiswürdigen Produkte!

5.1 Bauformen und Technologie der peripheren Speicher mit bewegtem Medium

Flexibler Träger (Band, Diskette): Träger ruht, außer für positionieren, lesen, schreiben. Also Beschleunigung (leichte Träger!) wichtig. Schreib/Lesekopf liegt auf dem Medium. Aufzeichnung in Bewegungsrichtung (üblich) oder quer (VHS-Video, DAT-Steamer).

starrer Träger (Platte, CD): Träger rotiert dauernd, muß nicht leicht sein. Magnetischer Schreib/Lesekopf schwimmt auf einem Luftpolster ($0{,}05\mu$). Bei optischen Schreib/Leseköpfen ist der Abstand größer (unempfindlicher!).
Speichermedien zum Datenversand (Diskette, CD-ROM) werden wahrscheinlich durch Rechnernetze verdrängt.

magnetisch: l:lesen s: schreiben

s	Durchflutung hartmagnetischer Werkstoffe, Polarisierung tangential (in Bewegungsrichtung oder quer) oder vertikal (höhere Dichte, neue Platten)	wichtig: Band (Kassette) 50GB, Platte 150GB 8ms
l	Induktion magnetoresistiver Effekt (bei vertikaler Polarisierung)	wichtig: Band (Kassette) 50GB, Platte 150GB

magnetisch-thermisch: schreiben, **magnetisch-optisch:** lesen

s	vormagnetisierte "harte" Schicht wird von Laserstrahl bitabhängig erhitzt (Curie-Temperatur) und gegenmagnetisiert	Magneto-Optische Platte 5,2GB im Prinzip bessere Zugriffszeit
l	reflektierter Laserstrahl ist magnetisierungsabhängig polarisiert	Magneto-optische Platte

thermisch: schreiben, **optisch:** lesen

s	Laserstrahl brennt Loch oder Vertiefung in Schicht Laserstrahl bewirkt Phasenübergang kristallin nach amorph in Schicht	wichtig: CDROM 0,7GB, DVD *digital versatile disk* (Audio, Daten) 4,7GB, CDRW: CD-kompatibel
l	Absorbtion bzw. Reflektion Holographie: Interferenzbild von Datenwelle und kohärenter Hilfswelle (schreiben), Beleuchtung durch Hilfswelle (lesen)	CDROM, CDRW,DVD holographischer Speicher z.B. in Lithium-Niobat-Kristall (Labor)

5.2 Magnetischer Plattenspeicher

Bauformen:

Flexible Platte (Diskette, *floppy disk*): 8" , 5.25" , 3.5" , 3" Zoll; 1,44..100MB, schlechte Zugriffszeit und Übertragungsrate, geringe Lebensdauer, kein eigentlicher peripherer Speicher, sondern (veraltetes) Austauschmedium.

Steife Platte (eine/mehrere je Baueinheit), *hard disk* : 14", 8", 5.25", 3.5", 2.5", 1.8".

- unbewegliche Köpfe (*fixed head disk*), gute Zugriffszeit bei kurzen Blöcken, da keine Kopfeinstellzeit, aber teuer und kleine Kapazität, rückläufige Bedeutung.
- bewegliche Köpfe
  - Platte(nstapel) nicht austauschbar (*fixed disk*) wichtigster Typ: "Festplatte".
  - Platte(nstapel) austauschbar (*removable disk*). Rückläufige Bedeutung.

Stand der Technologie (*fixed disk*)

Durchmesser: 3.5", 2.5", 1.8"
Zahl der Spuren (in Plattenstapeln Zylinder): bis 10000
Umdrehungsfrequenz: 3600.10000 /min (60..167/s)
Bitdichte in Spur: bis 1000/mm (es erscheinen 20000/mm erreichbar!)
Spurdichte: 100/mm (es erscheinen 5000/mm erreichbar) resultierende Bitflächendichte: $10^5/mm$ ($10^8/mm$)
Zugriffszeit: Kopfeinstellzeit: 0..20ms (Lokalität!)
Restumdrehungszeit: 0..16ms
Übertragungszeit durch Bytegrenzdurchsatz gegeben (bis 50MB/s)
Kapazität: 1..150GB, Schichtstärke: 0,1μ, Kopfhöhe über Schicht: 0,05μ

Seit 1970 hat sich die Zugriffszeit^{-1} etwa um den Faktor 20, die Kapazität aber um den Faktor 10000 erhöht (1,1/a bzw. 1,35/a)

Erhöhte Leistung durch Systeme aus parallelen gleichen Platten:

RAID: (*redundant array of inexpensive/independent disks)*

Ziele:

- Fehlertoleranz
- Durchsatz
- Zugriffszeit bei sehr großen Blöcken

Klassifikation als RAID 0 .. RAID 6 (nicht inklusiv!) Davon sind folgende wichtig:

RAID 0 (*striping*): große Blöcke werden in *stripes* zerlegt und auf mehrere Platten verteilt. Bessere Zugriffszeiten. Für unzerlegte (kleine) Blöcke besserer Grenzdurchsatz. Verfügbarkeit schlechter!

RAID 1 (*mirroring*: die Inhalte aller Platten werden auf je eineer weiteren Platte gespeichert (100% Redundanz). Bessere Verfügbarkeit, doppelter Grenzdurchsatz.

RAID 5 Kombination von *striping* und einem Fehlerkorrekturverfahren, das die Korrektur eines verfälschten Blocks erlaubt (binäre Summe aller Bits gleicher Position aller Platten wird berechnet und zusätzlich (auf die Platten verteilt) gespeichert: Redundanz 1/n (n Platten).) knapp n-facher Grenzdurchsatz (Schreiben Korrekturinformationen)

5.3 Optische periphere Speicher

Bauformen wie (Einscheiben-) Platten, Medium austauschbar.

CD-ROMs (*compact disk - read only memory*: Spiralaufschreibung (aber spurenweise Adressierung), eingeprägtes Binärmuster unter transparenter Deckschicht; höhere Speicherdichte, bei Massenherstellung geringerer Preis als bei Plattenspeichern; standardisierter Träger (80/120mm Durchmesser, etwa 700MB, 1..4MB/s, 175ms(!) Zugriffszeit).

WORMs (*write once read multiple*): laserbeschriftete Platte; Tellurschicht unter transparenter Deckschicht; brauchbar zur Archivierung, evt. Daten/Programmaustausch; standardisierter Träger (5.25"), billig, aber teures Laufwerk. 2,6GB, 35ms.

Magneto-optische Speicher: magnetische Speicherschicht, Laser-thermisches Schreiben (reversibel!), optisches Lesen (Drehung der Polarisationsebene); spielen seit langer Zeit eine Randrolle. 3.5"; 1,3GB.

Kapitel 6

Ein/Ausgabe-Werk

6.1 Ein/Ausgabe über Zentralprozessor

6.2 Ein/Ausgabe über System/Peripherie-Bus

6.3 Ein/Ausgabe über eigene Speicherzugänge

[BODE HÄNDLER 80], [FÄRBER G 84] (Bus)
[HAAKE PESCHKE 74] in [STEINBUCH K 74]: Taschenbuch der Informatik, Springer, Berlin 1974
[JESSEN E 75], [KÄSTNER H 78], [SIWIOREK et al 82] (Beispiele!)
[PATTERSON HENNESSY 96] Kap.6.3

Eher ein Stiefkind der Rechnerarchitekten, bescheidene Ansätze zur Standardisierung.

6.1 Ein/Ausgabe über Zentralprozessor

Der E/A-Strom fließt zwischen Speicher und peripheren Geräten. Älteste, heute (bis auf Sonderfunktionen) veraltete Lösung:

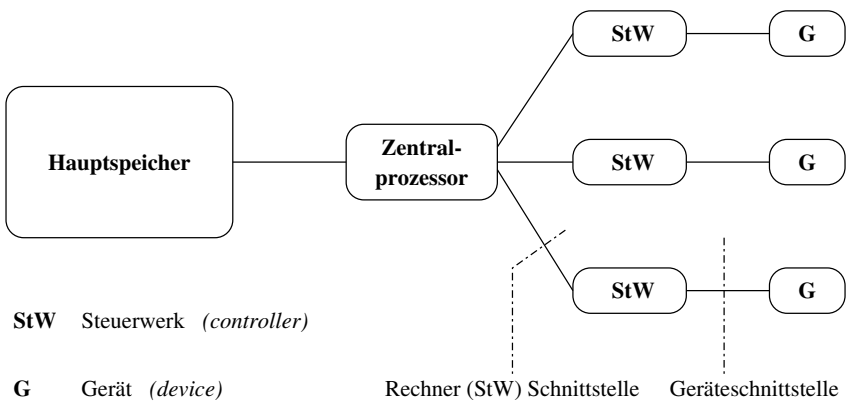


Abbildung 6.1: veraltete Lösung

Folgen dieser bis etwa 1959 ausschließlichen Lösung:

- kein Rechnen während Transport von/zum Speicher, aber Überlappung Rechnen/Gerätefunktionen (im Prinzip) möglich
- Zugriffskonflikte am Zentralprozessor meist durch ihn implizit geregelt (nur ein G aktiv)
- geeignet bei kleinster E/A-Leistung
- Pufferung im Steuerwerk, Unterbrechung günstig

E/A-Befehle sind Transporte von/zu Registern und E/A-Schnittstellen. Transportiert werden Anweisungen, Zustände, Nutzdaten.
geeignet

- bei kleinen E/A-Mengen
- wenn Zentralprozessor billiges programmiertes Steuerwerk ist

besonders vorteilhaft

- für Zustandsanzeigen des ZP (z.B. Wartungstableau)
- für Urladen (wenn nicht von ROM)
- für Test-Ein/Ausgabe

In den beiden letzten Fällen sollen nur minimale Maschinenfunktionen vorausgesetzt werden!

6.2 Ein/Ausgabe über System-/Peripherie-Bus

Gute Ausbaubarkeit, Zugriffskontrolle nun am Bussteuerwerk.

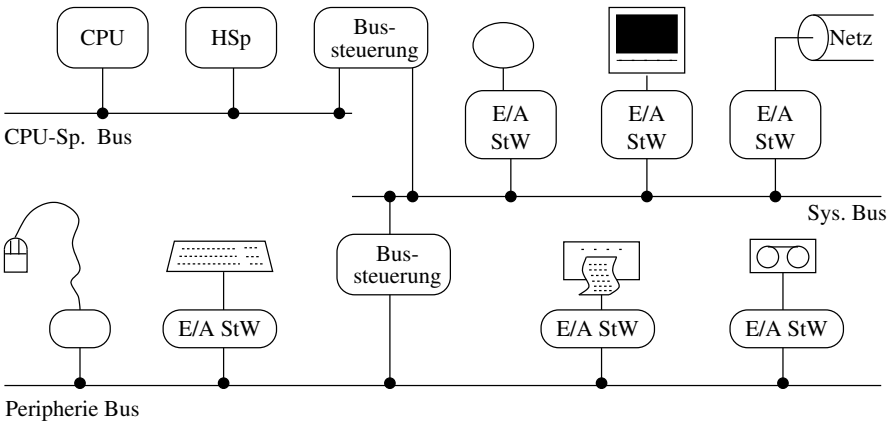


Abbildung 6.2: Bushierarchie, vgl Abb.2.9

Falls E/A wortweise durch Transportoperationen der CPU (z.B. aus Zelle des Hauptspeichers in Zelle eines E/A-Steuerwerks) erfolgt, besteht der Fortschritt nur in der Vereinfachung der CPU-Schnittstelle! besser: autonomer Blocktransport durch das Steuerwerk (*direct memory access, DMA*). Dann schneller Transport wechselnd mit oder (besser) während des Rechnens. Es bleiben aber Interferenzen am System- und Speicherbus; i.A wird die CPU zurückgestellt! vgl. S.136,137.

Beispiele für Peripheriebusse ([HENNESSY PATTERSON 96])

	S-Bus (DEC)	Microchannel	PCI	IPI	SCSI 2
Breite (Daten)	32b	32b	32/64b	16b	8/16b
Taktfrequenz	16..25MHz	asynchron	33MHz	asynchron	10MHz oder asynchron
Master	mehrere	mehrere	mehrere	einer	mehrere
Grenzdurchsatz 32b lesen	33 MB/s	20 MB/s	33 MB/s	25 MB/s	20 oder 6 MB/s
Grenzdurchsatz maximal	89 MB/s	75 MB/s	111 MB/s	25 MB/s	20 oder 6 MB/s
Standard				ANSI X3.129	ANSI X3.131

IPI: Intelligent periphery Interface ; Microchannel kein Kanal im nachfolgenden Sinn!

6.3 Ein/Ausgänge über eigene Speicherzugänge

(typisch für Großrechner)

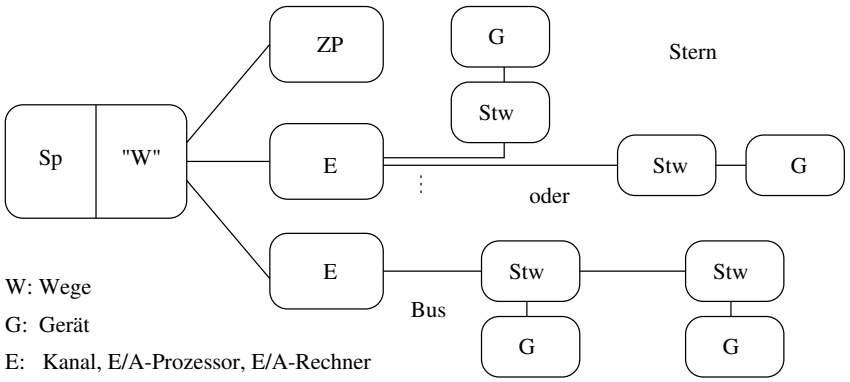


Abbildung 6.3: eigene Speicherzugänge

Zuggriffskonflikte nun an "W" (Wegenetz, vgl. 3.2alt) oder an den Speichermoduln.

E-Komponenten führen vorzugsweise Blocktransporte durch (Datensatz, Druckzeile, Bildschirminhalt). E-Komponente heißt:

E/A-Rechner, falls Programm im eigenen Speicher
E/A-Prozessor, falls Programm im Hauptspeicher
E/A-Kanal, falls nicht programmgesteuert

genauer:

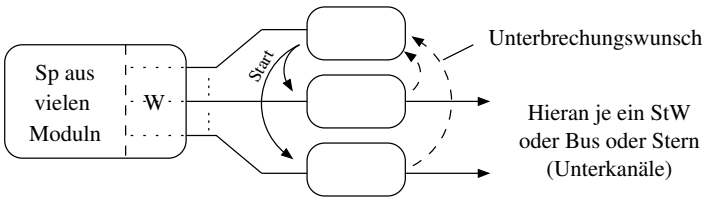


Abbildung 6.4: eigene Zugänge und Verbindungen untereinander

Typische Lösung für Großrechner. Meist Steuerwerke über mehrere E zugänglich (Fehlertoleranz, bessere Auslastung); ähnlich u.U. Geräte von mehreren Steuerwerken zugänglich:

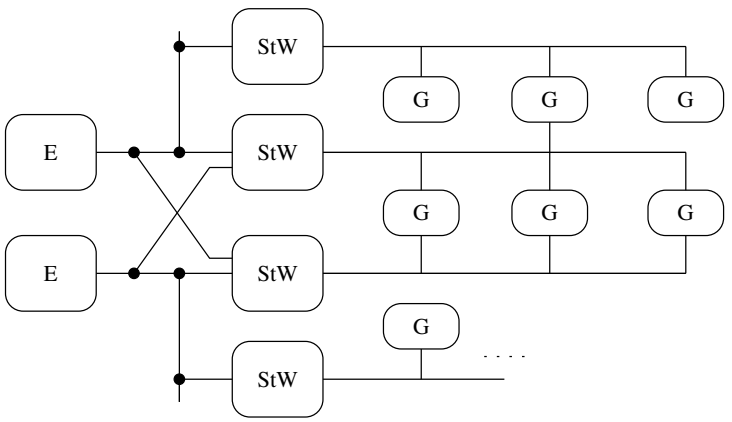


Abbildung 6.5: Steuerwerke an mehreren E-Komponenten

Zentralprozessor beauftragt E über ein Startsignal (Inter-Prozessor-Alarm); Fertigmeldung von E an Z ist ein Unterbrechungswunsch. Erweiterung durch

- Startinformation
- Fehlerprotokoll

Kanal (E/A-Prozessor, E/A-Rechner) sichert die Weiterarbeit des Zentralprozessors nach Start des E/A-Vorgangs. Erfindung (1958) fällt mit dem Konzept des Mehrprogrammbetriebs zusammen, da sonst die Weiterarbeit kaum nutzbar wäre! (gleiche Wirkung: DMA-Steuerwerk)

Heute typisch (bei Großrechnern):
E/A-Befehl im Benutzerprogramm → Einsprung in das Betriebssystem → E/A-Steuerwortfolge, E/A-Auftrag in (Kanal-/Geräte-)Warteschlange → (endlich) Auftrag in feste Kanalzelle im Hauptspeicher (mit Verweis auf Ort des Datenblocks im Hauptspeicher, Art der Operation, Gerät) → Start des Kanals, Transport vom/zum Gerät → Fertigmeldung durch Kanal → Unterbrechung → Betriebssystem kann (!) Benutzerprogramm wieder aufnehmen.

Kanäle/EA-Prozessoren bedienen oftmals im Zeitmultiplex mehrere Steuerwerke

- byteweise: Byte-Multiplexkanal
- blockweise: Block-Multiplexkanal

Zur Vermeidung unnötiger Unterbrechungswünsche an den Zenralprozessor werden die E/A-Kanäle mit Steuerwort**folgen** beauftragt, die

- verschiedene Blocktransporte mit demselben Steuerwerk an verschiedenen Speicherorten enthalten (gestreutes Schreiben/Lesen) oder
- auch andere Steuerwortfolgen zu einen Gesamtauftrag montieren

Sogenanntes *data/command chaining* durch Verweise in Steuerwortfolgen.

Kapitel 7

Mehrprozessoranlagen

multiprocessors (mit Ausblick auf Parallelrechner)

Eine Mehrprozessoranlage ist ein Rechner mit mehr als einem Zentralprozessor gleicher Befehlsliste; wenigstens ein Teil des Hauptspeichers liegt im Maschinenadressraum aller Zentralprozessoren.

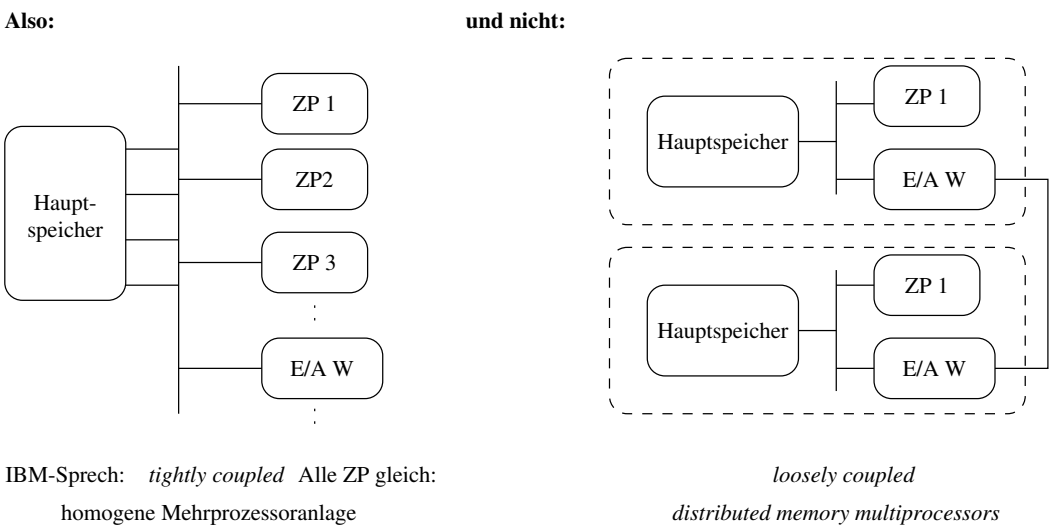


Abbildung 7.1: Mehrprozessoranlagen

Letzteres wird Mehrrechneranlage, Rechnerverbund, Rechnerensemble oder (bei weitgehend funktioneller Autonomie) Rechnernetz genannt (im amerikanischen oft auch *multiprocessor* genannt).

Mehrprozessoranlagen: erforderliche Zusatzfunktionen

- (wechselseitiger Ausschluß der Hauptspeicherzugriffe, Bus/Speicher Vorrangsteuerung, Konsistenzsicherung von Cachememories: alles bereits bei Kanälen notwendig!); Cachekonsistenz untereinander!
- garantierte Lese/Schreibfolgen bei p-Operationen (nicht mehr über Unterbrechungssperre durchsetzbar!); Alarme zwischen den Prozessoren; eindeutige Zuweisung externer Unterbrechungswünsche
- (Anpassung von Speicher, Verkehrswegen, und Peripherie an die höhere Leistung)
- korrekte Operationsfolgen im Speicher

Übrigens erfordert die Mehrprozessoranlage bei Belastung der Prozessoren mit verschiedenen Prozessen auch keine durchgreifenden Änderungen in prozessorientierten Betriebssystemen. Typisch: **symmetrischer** Betrieb (jeder ZP kann jede Funktion ausführen).

Die Nützlichkeit von Mehrprozessoranlagen ist verschieden erfaßbar:

1. Sind Mehrprozessoren strukturell überlegen (d.h. gegeben sei eine bestimmte Grenzdurchsatz/ Zuverlässigkeitsforderung, ist es besser, die Leistung durch einen Prozessor oder durch n Prozessoren der Leistung 1/n zu erbringen)?
2. Lohnt der Ausbau der Einprozessoranlage zum Mehrprozessor, wenn der Austausch gegen einen stärkeren Prozessor unmöglich ist?

Antwort zu 1. Aufteilung ist ungünstig vom Standpunkt der Leistung:

- einzelnes Programm läuft zu langsam (s. unten)
- Durchsatz ist niedriger (Behinderung; mehr als ein Programmlauf erforderlich) (s. unten),

Eine Aufteilung ist günstig vom Standpunkt der Verfügbarkeit (s. unten)

Antwort zu 2. Der Ausbau lohnt, wenn es keinen stärkeren Prozessor gibt (S.152)

Also sind Mehrprozessoranlagen empfehlenswert

- bei hohen Verfügbarkeitsanforderungen an die Prozessorfunktion, 2..4 Prozessoren
- am oberen Ende einer Rechnerfamilie 2..16 Prozessoren

Beispiel Zweiprozessoranlage

v_2 sei die **Verfügbarkeit** der einzelnen Prozessoren der Zweiprozessoranlage
P[wenigstens ein Prozessor verfügbar] $= 1 - (1 - v_2)^2 = v_2(2 - v_2) = v_2 + v_2(1 - v_2) > v_2$!
Der doppelt schnelle Prozessor (aus der Einprozessoranlage) habe die Verfügbarkeit v_1. Leider lehrt die Erfahrung $v_1 < v_2$ (je schneller, desto unzuverlässiger). Also:

P[wenigstens ein Prozessor verfügbar] $= v_2 + v_2(1 - v_2) > v_2 > v_1$

Also positiv für den Zweiprozessor, falls die Komplexität der Software nicht den Erfolg zunichte macht (Beispiele vorhanden)

Verweilzeit, Grenzdurchsatz

Die Zweiprozessoranlage erledigt jede geschlossene ZP-Rechenphase zweimal langsamer als die Einprozessoranlage doppelter Leistung:

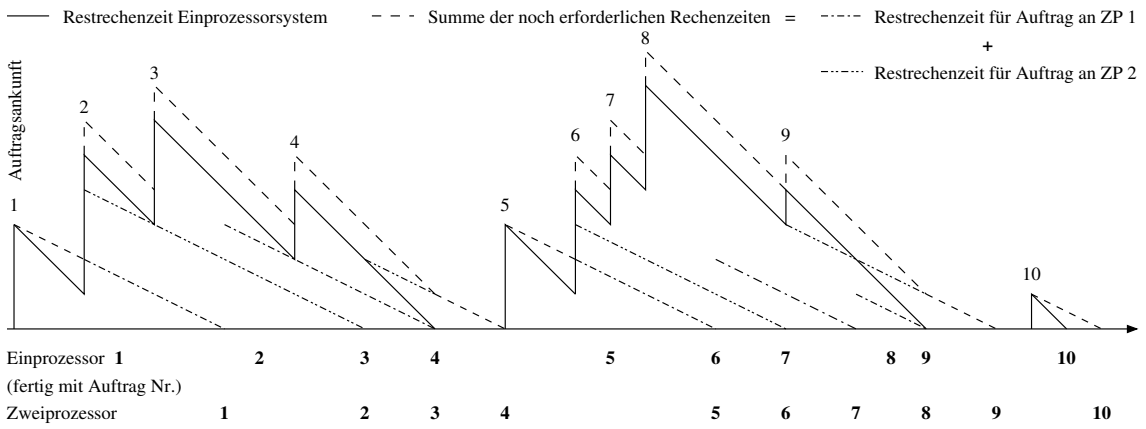


Abbildung 7.2: Vergleich: Einprozessoranlage vs. Zweiprozessoranlage

Schlechte Auslastung bei Füllung 1 !
(oben zugrunde gelegt: first come - first served)

Der Grenzdurchsatz wird bei der Zweiprozessoranlage gemindert durch

- wechselseitigen Ausschluß an Verkehrswegen und Speichermoduln (*hardware lockout*)
- (schlimmer:) wechselseitiger Ausschluß an den koordinierenden Routinen/Datenstrukturen (*software lockout*)

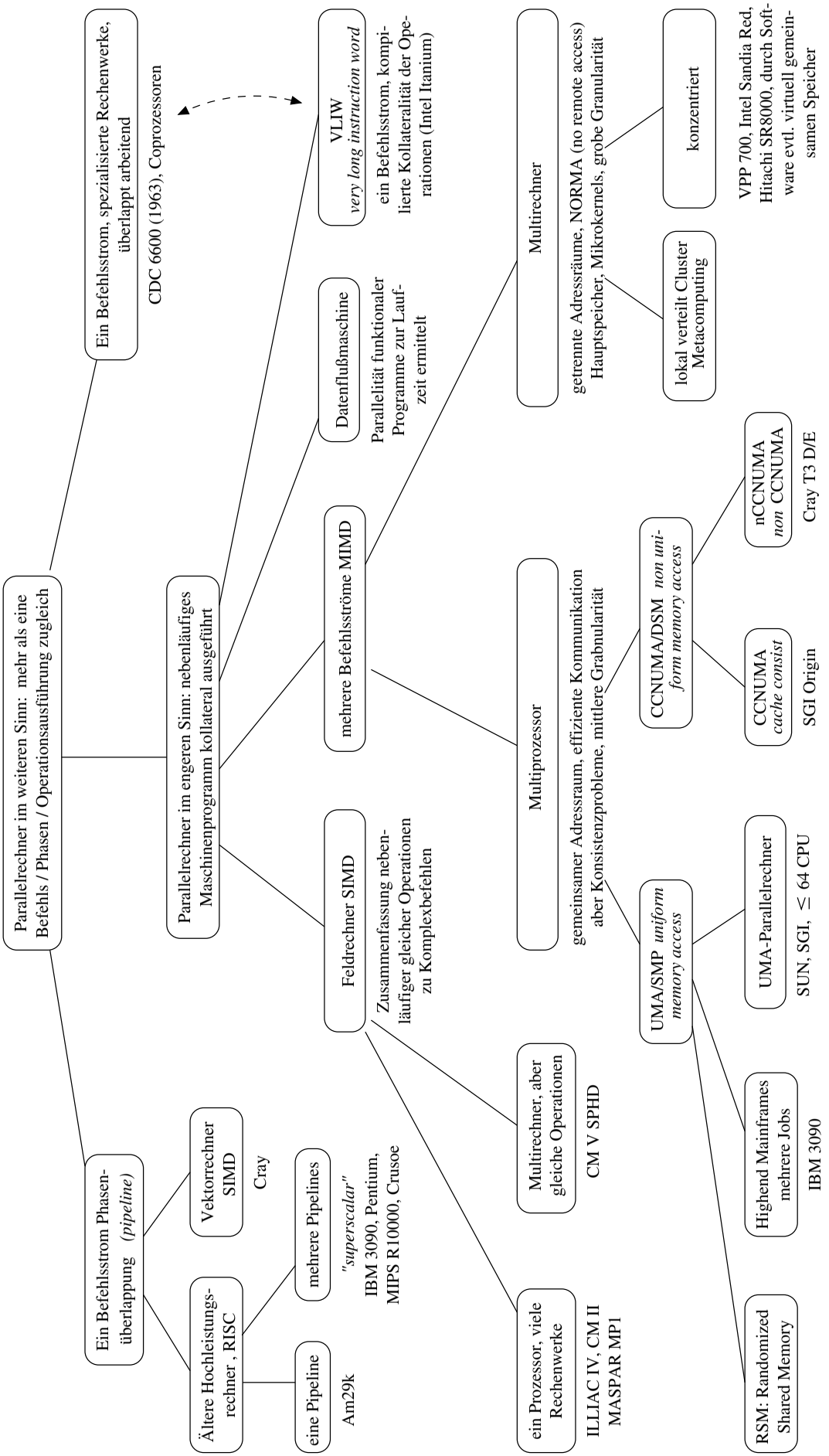


Abbildung 7.3: Erscheinungformen Parallelrechner

Nochmals zu 2.: Lohnt der Ausbau auf Mehrprozessoren zur Grenzdurchsatzsteigerung, wenn der Übergang auf schnellere Prozessoren nicht möglich ist?

> Ja! Allerdings tritt wegen der lockout-Effekte an Bus(sen), Speicher(moduln) keine lineare Erhöhung des Prozessorgrenzdurchsatzes ein. Bei heutiger Technologie höchstens 32 Prozessoren, Cache-gepuffert auf einem Hauptspeicher.

Der Betrieb erfolgt unter einem Betriebssystem mit zentraler Auftragsverwaltung. Ein frei werdender Prozessor bekommt den "nächsten" Auftrag.

Ausblick: Parallelrechner (Vorlesung Sommersemester)

Im weiteren Sinn: Parallelrechner sind Rechner mit mehr als einer Operations-/Befehlsphasenausführung zu einem Zeitpunkt. Allerdings sind die weitaus meisten an der Hardware-Software-Schnittstelle trotzdem serielle Rechner (z.B. RISCs)

Im engeren Sinn: Parallelrechner sind Rechner für **nebenläufige Programme**, die kollateral ausgeführt werden. Wichtigste Vertreter (→2.11 Klassifikation)

SIMD	Vektorrechner (VPP7000 ist ein Multivektorrechner)
	Feldrechner (aktuell bedeutungslos)
MIMD	Multiprozessorsysteme (aber alle Prozessoren arbeitsteilig an einer Aufgabe!)
	Multirechnersysteme (wir sahen die Hitachi SR8000)
Datenflußrecher	(aktuell bedeutungslos)

Motiv der Parallelrechnerentwicklung ist **Kürzung der Bedienzeit großer Aufträge** (z.B. Wettervorhersage) und nicht Durchsatz von kleinen und mittleren Aufträgen oder Verfügbarkeit (wie bei Mehrprozessorsystemen unter einem Betriebssystem).

Literaturverzeichnis

[BODE A 90] Bode A. (Herausgeber): RISC-Architektur, Wissenschaftsverlag, Mannheim 1990 (2.Auflage)

" Systeme, optimierende Compiler, Anwendeungen, Leistungsmessung, Parallelismus ", Systematische Teile, Produktbeschreibung (von Firmen)

[BODE HÄNDLER 80] Bode A. Händler W. : Rechnerarichtektur, 1. Band Grundlagen und Verfahren, Springer, Berlin 1980

"Rechnerarchitektur", Leiszungs- und Zuverlässigkeitsanalyse, Technologie (1980!), Automaten, Petrinetze, CHDL's, Busse, Mikroprogrammierung

[BODE HÄNDLER 83] Bode A. Händler W. : Rechnerarichtektur, 2. Band Strukturen, Springer, Berlin 1983

Rechnerklassifikationen, Parallelismus, Virtualität, Pipelining: Beispiele CDC Cyber, IBM /370 Burrongles 6700, STAR, Cray 1, ILLIAC IV, ECPA und weitere

[BODE A 95/96] Bode A. Folienkopien Rechnerarchitektur WS 95/96 Fachschaft MPI TUM 1996

Herr Bode und ich halten zu 90% die gleichen Vorlesungen, die wir gemeinsam weiterentwickeln

[DASGUPTA S 90] Dasgupta S. : Computer Architecture: A Modern Synthesis, Wiley, New York, 1989

Band 1: Foundations *"Architektur", Technologie, Rechnerentwurf, Registermaschinen, Mikroprogrammierung, Stacks, RISC, Speicher, viel unkonventionelles Querschnittswissen*
Band 2: Advanced Topics

[GILOI W 91] Giloi W. Rechnerarchitektur, Springer, Heidelberg 1991

Das Buch tritt engagiert für innovative Rechnerarchitekturen ein, insbesondere Datentyparchitekturen, Datenflußarchitekturen, Parallelrechner

[HAYES J 88] Hayes J.P. : Computer Architecture and Organisation, Mc Grawn Hill, New York 1988 ca. 45,-

sehr viel technisches Detailwissen, Beispiele IBM /379, Motorola

[HENNESSY PATTERSON 96] Hennessy, J., Patterson, D.: Computer Architecture - A Quantitative Approach, Kaufmann, San Mateo, 1996 ca. 75,-

Schwerpunkt ist der Prozessorentwurf mit vielen Messungen und Analysen zur Hardware-Software Schnittstelle. Verfasser haben entscheidene Verdienste um die RISC Entwicklung: Hennessy MIPS, Patterson RISC (→ SPARC). Schwachstellen: Unkonventionelle Rechner, E/A , didaktisch gut

[HWANG BRIGGS 86] Hwang k., Briggs F.: Computer Architecture and Parallel Processing, Mc Grawn Hill, New York 1986

wegen der raschen Parallelrechnerentwicklung etwas veraltet(!) aber sehr viel strukturelles Wissen

[JESSEN E 75] Architektur digitaler Rechenanlagen, Springer, Heidelberg 1975 ca. 25,-

im Ansatz besonders gutes Buch, aber sehr alt, hier aus Sympathie mit dem Verfasser genannt

[HWANG K 98] Hwang K.: Advanced Computer Architecture, Mc Grawn Hill New York 1998

ein gutes Parallelrechnerbuch mit Seitenblicken auf Einprozessorsysteme

[MÜLLER-SCHLOER SCHMITTER 91] Müller-Schloer, C. , Schmitter, E (ed): RISC-Workstation-Architekturen, Springer, Berlin 1991

umfassende, neue Übersicht über Workstation-Technologie/ Strukturen/ Produkte

[SIEVIOREK et al 81] Sieviorek, D. Bell, G., Newell,A : Computer Structures, Principles and Examples, Mc Grawn Hill New York 1981 ca. 70,-

Bibliothek zur Rechnerorganisation in einem Band, systematische Einfürung und Beschreibung von Rechnern, meist aus Sicht ihrer Entwerfer. Sehr gutes, veraltetes Buch Historisches Material !

[TANENBAUM A 90] Tanenbaum, A. : Structured Computer Organisation, Prentice Hall, Englewoods Cliffs 1990 ca. 80,-

Rechner als Hierachie von Sprachschichten, Beispiele Intel80386, Motorola68030, IBM PC, RISC

[UNGERER T 90] Ungerer, Th.: Innovative Rechnerarchitekturen - Bestandsaufnahme, Trend Möglichkeiten, Mc Grawn Hill , Hamburg 1989

konventionelle Rechner, aber etwa zu 75% eine moderne, bewertende Darstellung unkonventioneller Rechner

[MÄRTIN C 94] Märtin, Ch.: Rechnerarchitektur - Struktur, Organisation, Implementierungstechnik, Hanser, Mnchen 1994

[DALCIN M 96] Dalcin, M.: Rechnerarchitektur, Teubner, Stuttgart 1996, Anhang: Modellierung und Bewertung, Mikroprogrammmierung

[UNGERER T 95] Ungerer, Th. Mikroprozessortechnik - Architektur und Funktionsweise Superskalarer Mikroprozessoren, International Thomson, 1995

[UNGERER T 99] Ungerer Th.: Parallelrechnerund parallele Programmierung, Spektrum, Heidelberg 1997

sehr gutes Parallelrechnerbuch

[VAN DE GOOR A 89] Van de Goor, A.J.: Computer Architecture and Design, Edison Wesley Wokinham 1989

Einführung auch in Betriebssysteme, Beispiel Motorola 68020

[STONE H 75] Stone, H. (ed) : Introduction to Computer architecture, Sience Research Ney York 1975

Zeitschriften zum Thema der Vorlesung

Computer (IEEE): *breit: Syteme, Programmierung, Anwendung* *

IEEE Transactions on Computers: *eher theoretisch*

BYTE *kleinrechnerorientiert*

Computer Design: *Rechnertechnologie*

ACM Transaction on Computersystems: *Detail- und Spezialprobleme*

IEEE Mirco : *Systeme, TEchnologie, Produkte*

Microprocessor Report: *Produkte, kritisch, viel Insiderwissen* *

Microprocessors and Microsystems: *Anwendunden*

ACM Computer Architecture News : *aktuell, gründlich* *

* empfehlenswert

	Bode A 90	Bode Händler 80 83	Bode A 91 92	Dasgupta S 89	Hayes J 88	Henn. Patterson 90	Hwang 98	Müller Schloer 91	Siwiorek 81	Tanenbaum 90	Ungerer T 89	van de Goor 89	Giloi W 91	Märtin C 94
Rechnerarchitektur		+	+	+		o					+	o	+	+
Prinzipien	o	+	+	o	o	+	+	o	+	+	o		+	o
Zentralprozessoren	+	+	+	+	+	+	o	o	+	+	o	+	+	+
Hauptspeicher		+	+	+	+	+	+	+	+	o		o	+	o
periphere Speicher				o		+		+						
E/A Werke				o	+	o		+	+	o		+		
Mehrprozesssor Parallelrechner		+		+	+	+	+		+		+	+	+	+
Rechnernetze		o			+		o	o	o	o				
E/A Geräte				o				o						
Leistung v. Rechnern	+	+	+			+			o					+
Unkonventionelle R.		+	o			o			+	o	+		+	
Historie		+			+	o	o		+	+				
Beispiele	**RISC Systeme**	**viele ältere Rechner**	**VAX, AMD 29k**	**RISC , /370 VAX**	**/370 68k**	**VAX ,Cray, /370, x86, RISC**	**Parallelrechner & Software**	**RISC-Workstation**	**Sehr viele ältere Rechner**	**x86, 68k**	**viele unkonventionelle R.**	**VAX, 68k**		**IBM Intel CDC Alpha**
Besonders	**Software, Anwendung**	**Methoden**	**Fast diese Vorlesung**	**Technologie**		**Entwurfsfragen**	**Algorithmen, Betr.Sys.**	**Software, UNIX**	**Originalarbeiten**	**Schichtenkonzept**		**Betriebssysteme**		

Abbildung 7.4: Abdeckung des Vorlesungsstoffes