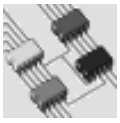


Proseminar
Mikroprozessoren

Uni Hamburg, FB Informatik, AB TECH
Technische Grundlagen der Informatik

Proseminar Mikroprozessoren

18.066

Seit der Einführung des ersten integrierten Mikroprozessors (Intel 4004, 4-bit, 1972) sind Mikroprozessoren in alle Bereiche des täglichen Lebens vorgedrungen - vom single-Chip Microcontroller in Chipkarten oder der Kaffeemaschine über Signalprozessoren im Mobiltelefon zum klassischen Mikroprozessor in Workstations.

Das Proseminar behandelt die Grundprinzipien von Aufbau, Organisation und Programmierung von Mikroprozessorsystemen:

- Aufbauprinzipien
- Vergleich und Bewertung der Standardprozessoren
- Befehlssätze
- Rechenwerke, Pipelining
- I/O-Prinzipien
- Bussysteme, Timing
- Speicherhierarchie (Cache)
- Multimedia-Befehlssätze (MMX, ISSE)
- Signalprozessoren
- Parallelrechner
- Mikrokontroller
- aktuelle Entwicklung: Intel IA-64 Architektur

Damit Zeit für die Diskussion bleibt, sollten die Vorträge etwa eine Stunde dauern.

Literatur

Die klassischen Lehrbücher zum Mikroprozessoren und Rechnerarchitektur als Ausgangspunkt für die weitere Suche:

- Hennessy, Patterson: "Computer architecture, the hardware/software interface"
- Andrew S. Tanenbaum: "structured computer organization, fourth edition", Prentice-Hall, 1999, ISBN 0-13-020435-8 A-TAN-26062
  (Es gibt das Buch auch auf deutsch ("Computerarchitektur, 4. Auflage"), die Übersetzung ist aber leider nicht so recht gelungen)

25.10.2000 **Norman Hendrich**
Vorbesprechung (Einführung, Literatur, Motivation)

01.11.2000 **Jana Passow, Anne Schick**
computer abstractions and technology

08.11.2000 **Martin Gaitzsch, Mirek Hancl, Davood Kheiri**
the role of performance

15.11.2000 **Tilmann Weidlich, Susanne Möllers**
instructions: language of the machine

22.11.2000 **Mirwais Turjalei, Rene Grohmann**
arithmetic for computers

29.11.2000 **Martin Siepok, Hauke Loock**
the processor: datapath and control

06.12.2000 **Stefan Conrad, Andreas Tyart**
enhancing performance with pipelining

13.12.2000 **Martin Lange, Andre Peters**
large and fast: exploiting memory hierarchy

20.12.2000 **Jan Milz, Ragna Dirkner**
interfacing processors and peripherials

10.01.2001 **Gregor Michalicek, Marcus Schüler**
multiprocessors / single-chip multiprocessors

17.01.2001 **Björn Gaworski, Alexander Stahlberg**
media instructions sets (MMX, ISSE, 3Dnow!)
AMD 3DNow! Datasheet & Technical Manual (PDF)
AMD Extensions to the 3DNow!™ and MMX™ Instruction Sets Manual (PDF)

24.01.2001 **Marcus Schulz, Andrzei Walczak**
Java processors

31.01.2001 **Tarkan Basimer, Vitalij Cornies, Marcus Lehmann**
microcontrollers

07.02.2001 **Nils Andresen, Gunnar Kedenburg**
explicit parallel instruction computing: Intel IA-64

Proseminar 18.066

Mikroprozessoren

Norman Hendrich
Universität Hamburg, Fachbereich Informatik, TECH

`tech-www.informatik.uni-hamburg.de/lehre/ws2000/ps-mikroprozessoren`

Literatur:

D.A.Patterson & J.L.Hennessy:
computer organization & design, the hardware/software interface
Morgan Kaufmann, 1998 (2nd Ed.), 1-55860-491-X
D HEN 25574 (mehrere Exemplare)

A.S.Tanenbaum:
structured computer organization
Prentice Hall, 1999 (4th Ed.), 0-13-020435-8
P TAN 26062

Skripte T1/T2/T3/T4
diverse Datenbücher, insbesondere via developer.intel.com, www.amd.com,
www.motorola,com (Signalprozessoren, µController)

Terminvorschau: Mi, 12-14, C-221

	25.10	Vorbesprechung, Vergabe der Referate	
1	01.11	computer abstractions and technology	Patterson & Hennessy
2	08.11	the role of performance	
3	15.11	instructions: language of the machine	
4	22.11	arithmetic for computers	
5	29.11	the processor: datapath and control	
6	06.12	enhancing performance with pipelining	
7	13.12	large and fast: exploiting memory hierarchy	
8	20.12	interfacing processors and peripherials	
9	10.01	multiprocessors / single-chip multiprocessors	advanced
10	17.01	media instruction sets / MMX, ISSE, 3Dnow!	
11	24.01	signal processors (or: Java processors)	
12	31.01	microcontrollers	
13	07.02	explicit parallel instruction computing: Intel IA-64	

Proseminar: mehrere Lernziele

Inhalt / Thema des Proseminars — 33%

- Mikroprozessoren

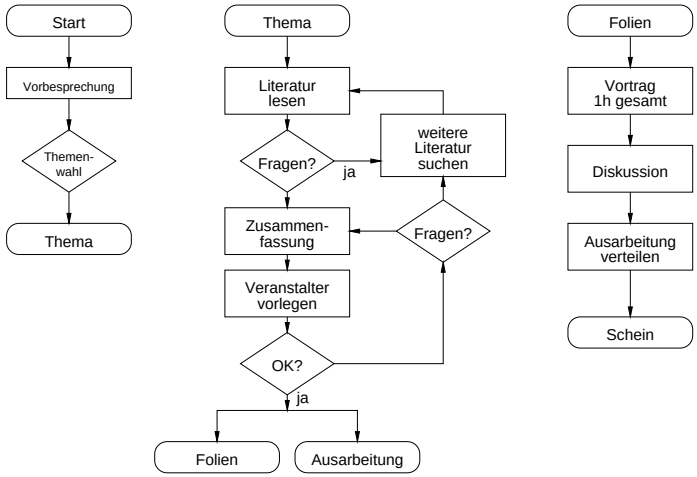
Thema erarbeiten: — 33%

- Literatur lesen und verstehen
- evtl. weitere Literatur suchen und sichten
- umfangreiches Thema zusammenfassen
- Folien und Ausarbeitung erstellen
- dabei selbständiges Arbeiten, evtl. Gruppenarbeit

Vortrag: — 33%

- Vortrag halten, Lampenfieber überwinden
- Diskussion

Proseminar-Algorithmus



Vorführungen, Beamer:

Beamer für Powerpoint & Co:

- im Prinzip möglich (sofern verfügbar)
- muss im RZ angemeldet werden
- rechtzeitig (zwei Wochen vorher) beim Betreuer anfragen
- lohnt für Animationen, Programmdemos, Medienwiedergabe

- ca. 30 Min für Aufbau / Abbauen einplanen
- eigenes Notebook mitbringen

- PC und Macintosh unterstützt
- Beamer verfügen über (mono) Lautsprecher

- Ausarbeitung in jedem Fall notwendig

Vortrag:

- Tafel und Kreide
- Overhead-Folien
- Powerpoint & Co

Faustregeln:

- 3 Minuten pro Folie
  => etwa 15 .. 25 Folien / Stunde Vortrag
- grosse Schrift (>20pt), Querformat
- nicht nur Schlagworte ("Powerpoint-Syndrom")
- sondern möglichst viele Diagramme / Abbildungen
- Backup-Folien bereithalten: Details zu erwarteten Fragen

Ausarbeitung:

wissenschaftliches Schreiben üben:

- als Text ausformulieren
- mit Gliederung und Literaturhinweisen
- Unfang ca. 4-8 Seiten
- möglichst schon beim Vortrag verteilen

- bitte nur portable Dateiformate:
  PDF, Postscript (Apple Laserwriter II), HTML, ASCII-Text
  aber keine "write-only" Formate wie Word
- einfache Folienkopien nur im (begründeten) Notfall

- Literaturliste der Bibliothek:
  "Studieren Lernen Arbeiten"

Mikroprozessoren: Performance 2000

SPEC CPU95 Benchmarks (baseline):	SPECint95	SPECfp95
AMD Athlon 1GHz	42.0	29.4
Intel Pentium-III 800 MHz (VC820)	37.9	27.7
Compaq Alphaserver DS20E	35.7	70.7
HP 9000 C3600	38.4	61.0
Sun Ultra 10/440	15.0	19.8

- keine offiziellen Werte für PowerPC
- Programme laufen weitgehend im L1 Cache
- Alpha sieht bei CPU2000 deutlich besser aus
- aktuell: UltraSPARC-III, alle anderen RISC weit abgeschlagen

[www.spec.org/osg/cpu95, Stand 05/2000]

Mikroprozessoren: "embedded systems"

Mikroprozessoren: x86-Evolution ...

Intel Processor	Date of Product Intro-duction	Perform-ance in MIPs[1]	Max. CPU Frequency at Intro-duction	No. of Transis-tors on the Die	Main CPU Register Size[2]	Extern. Data Bus Size[2]	Max. Extern. Addr. Space	Caches in CPU Pack-age[3]
8086	1978	0.8	8 MHz	29 K	16	16	1 MB	None
Intel 286	1982	2.7	12.5 MHz	134 K	16	16	16 MB	Note 3
Intel386™ DX	1985	6.0	20 MHz	275 K	32	32	4 GB	Note 3
Intel486™ DX	1989	20	25 MHz	1.2 M	32	32	4 GB	8KB L1
Pentium®	1993	100	60 MHz	3.1 M	32	64	4 GB	16KB L1
Pentium® Pro	1995	440	200 MHz	5.5 M	32	64	64 GB	16KB L1; 256KB or 512KB L2
Pentium II	1997	466	266	7 M	32	64	64 GB	32KB L1; 256KB or 512KB L2
Pentium® III	1999	1000	500	8.2 M	32 GP 128 SIMD-FP	64	64 GB	32KB L1; 512KB L2

[Intel P-III databook]

Prozessoren in "embedded systems"

Prozessor	4 .. 32 bit	8 bit	-	16 .. 32 bit	16 bit	32 bit	32 bit	8 .. 64 bit	..32 bit
Speicher	1K .. 1M	< 8K	< 1K	1 .. 64M	?	< 128 M	8 .. 64M	1 K .. 10 M	< 64 M
ASICs	1 uC	1 uC	1 ASIC	1 uP ASIP	DSPs	1 uP, 3 DSP	1 uP, DSP	~ 100 uC, uP, DSP	uP, ASIP
Netzwerk	cardIO	-	RS232	diverse	GSM	MIDI	V.90	CAN,...	I2C,...
Echtzeit	nein	nein	soft	soft	hard	soft	hard	hard	hard
Safety	keine	mittel	keine	gering	gering	gering	gering	hoch	hoch

=> riesiges Spektrum: 4 bit .. 64 bit Prozessoren, DSPs, digitale/analoge ASICs, ...

=> Sensoren/Aktoren: Tasten, Displays, Druck, Temperatur, Antennen, CCD, ...

=> Echtzeit-, Sicherheits-, Zuverlässigkeitsanforderungen

Moore's Law

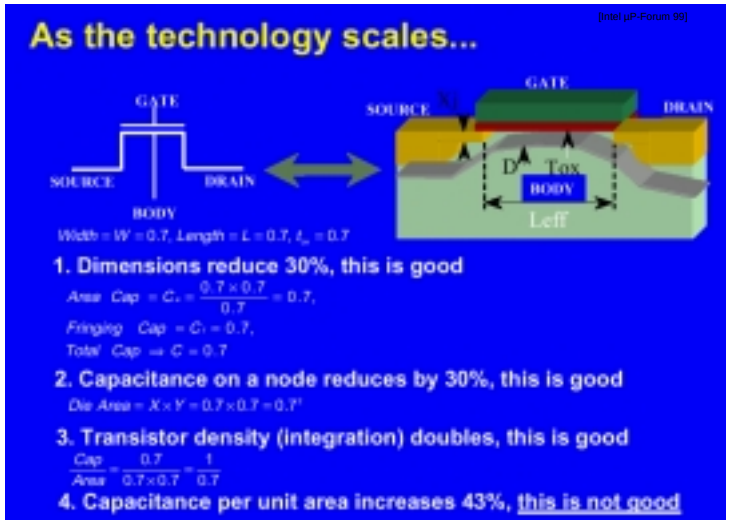
- Planarprozeß ist massiv parallel
- Kosten fast unabhängig von der Anzahl einzelner Elemente

=> Moore's Law: exponentieller Anstieg des Integrationsgrades

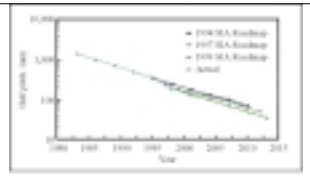
- mehr Funktionen bei gleichen Kosten (gleiche Chipfläche)
- oder gleiche Funktion bei geringeren Kosten
- rein wirtschaftlich bedingt
- solange, bis Kapitalkosten für neue Technologie zu hoch

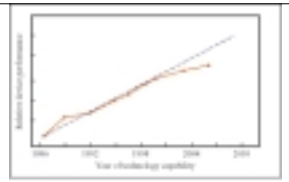
Verbesserungen durch:	(relativer Anteil)
• feinere Lithographie	(50%)
• verbesserte Transistoren / Strukturen	(25%)
• bessere Rechnerarchitektur	(25%)

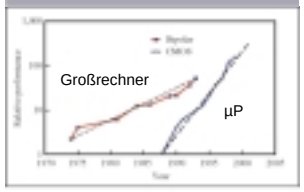
Moore's Law: Transistor-Skalierung



Moore's Law: Lithographie, Hochintegration





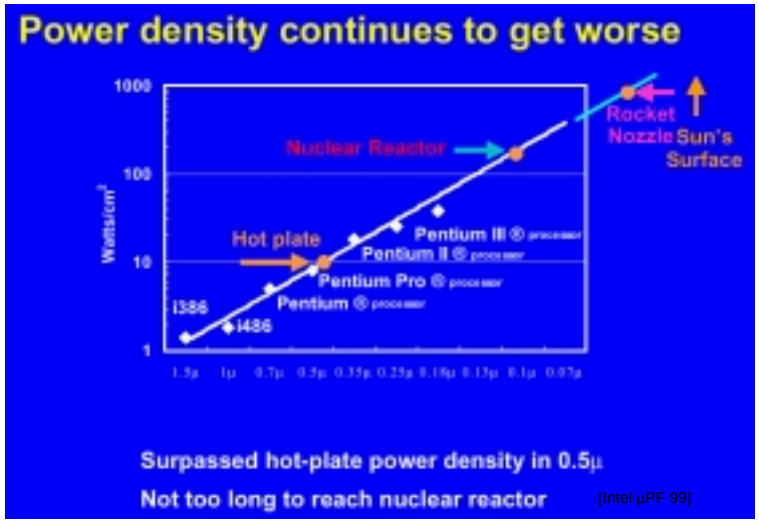


- exponentielles Wachstum
- seit 1970, bis > 2015

- seit 1996 CMOS besser als ECL
- zunehmend Abwärmeproblem

[IBM JR&D 44-3, 2000]

Moore's Law: Leistungsverbrauch



Computer Abstractions and Technology

Anne Schick, Jana Passow

1. November 2000

1 Einführung und Grundbegriffe

Programmierer waren seit Beginn der Computerentwicklung an immer leistungsfähigeren Programmen interessiert. In den 60er und 70er Jahren stellte die begrenzte Speicherkapazität das Hauptproblem dar, neue Speichertechnologien führten jedoch dazu, daß heute nicht mehr die möglichst geringe Speichernutzung im Vordergrund steht, sondern das Verstehen und Nutzen eines hierarchischen Speichermodells sowie paralleler Prozessorarchitekturen. Das Wissen über Computerorganisation ist unverzichtbar, um konkurrenzfähige Compiler, Betriebssysteme oder Anwendungen zu entwickeln.

Es geht also um:

- Basisdefinitionen und -ideen,
- die wichtigsten Hard- und Softwarekomponenten,
- integrierte Schaltkreise.

Abstraktion ist bei der Betrachtung sowohl von Software als auch von Hardware ein fundamentaler Begriff. Will man die Tiefen von Hard- und Software erforschen, dann wird bei jeder nächstniedrigeren Ebene mehr Information sichtbar – oder andersherum gesagt: Um ein einfacheres Modell auf einer höheren Ebene zu zeigen, verbirgt man Details. Diese Stufen heißen Abstraktionen.

2 Historisches: Computer-Generationen

Seit 1952 entstand eine Vielzahl von Computern mit sehr verschiedenen Fähigkeiten und unterschiedlichsten Technologien. Nach der Art der Implementierungstechnologien lassen sich diese Entwicklungen in vier Generationen klassifizieren, wobei jede Epoche 8 bis 10 Jahre dauert. Erst die vierte Computergeneration hält durch den Erfolg des Mikroprozessors fast so lange an, wie alle vorherigen drei zusammen. Dieser Erfolg begründet sich in ebenso rasanten wie revolutionären Verbesserungen.

Generation	Zeitraum	Technologie	wichtigstes neues Produkt
1	1950-1959	Vakuumröhren	kommerzielle Computer
2	1960-1968	Transistoren	Billigere Computer
3	1969-1977	Integrierte Schaltkreise	Minicomputer
4	1978-	LSI und VLSI	Personalcomputer und Workstations

3 Technologie-Entwicklungen und Anwendungen

Die Computer-Industrie ist die einzige in der Geschichte der Menschheit, die in einem solch atemberaubenden Tempo vorangeschritten ist und seit den 40er Jahren zu so beispiellosen Fortschritten geführt hat. Was wäre, wenn man diese Entwicklungskurve auf eine andere Industrie, wie z.B. das Transportwesen, übertragen würde? Dann wären wir heute an einem Punkt, wo wir in 5 Sekunden von einer nordamerikanischen Küste zur anderen reisen könnten, und das für 50 Cent!

Es ist also berechtigt, neben der Agrar- und der Industriellen Revolution von der Computerrevolution zu sprechen.

Natürlich veränderten sich damit auch andere Wissenschaftszweige und Anwendungen, die vor Jahren noch unvorstellbar waren, sind heute Realität. Zu diesen Entwicklungen gehören der Bankautomat, Chips im Auto, Laptops, die Entschlüsselung menschlicher Gene sowie das WWW.

4 Die Software „unter“ dem Programm

“In Paris they simply stared when I spoke to them in French; I never did succeed in making those idiots understand their own language.” (Mark Twain, zitiert nach [1])

Die Sprache einer elektronischer Maschine besteht aus nur zwei „Buchstaben“: Strom an und Strom aus, bzw. 1 und 0. Mit diesen beiden Zeichen läßt sich aber eine unendliche Menge von „Wörtern“, also Daten oder Anweisungen bilden. Nachdem die ersten Pioniere tatsächlich in Binärcode programmiert haben, wurde bald zum einen eine Sprache entwickelt, die der menschlichen ähnlicher war und das Programmieren angenehmer machte (Assembler-Code), und zum anderen Programme, die diese Sprache in Binärcode übersetzten (Assembler).

Im Laufe der Weiterentwicklung des Computers entstanden auf einer noch höheren Sprachebene sogenannte „high-level programming languages“ (z.B. C), die wiederum von Programmen (Compiler) in Assemblersprache übersetzt wurde und noch einfacher und anwenderfreundlicher waren.

Der Vorteil dieser Notationen gegenüber Assemblersprache ergeben sich aus der natürlicheren Ausdrucksweise:

- Programmierer müssen nicht in kryptischen Symbolen denken,
- höhere Programmierproduktivität durch Kürze und Prägnanz,

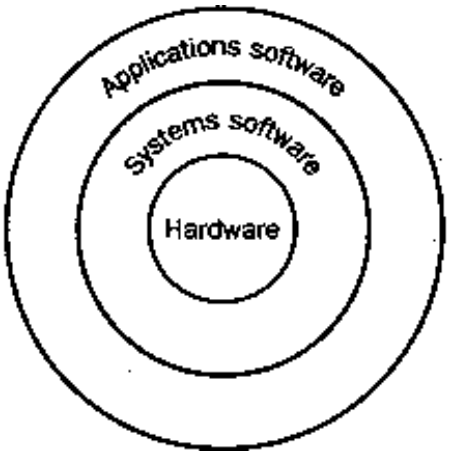
- verschiedene Programmiersprachen entstehen für unterschiedliche Zwecke,
- Programme sind unabhängig von dem Computer, auf dem sie entwickelt wurden, weil es Compiler und Assembler für alle möglichen Rechnertypen gibt.

Um die Effizienz zu steigern wurden immer wiederkehrende oder häufig benutzte Anwendungen in Bibliotheken (subroutine libraries) zusammengefaßt. Eine dieser Bibliotheken kontrolliert z. B. die Dateneingabe und -ausgabe sowie die dazugehörenden Geräte.

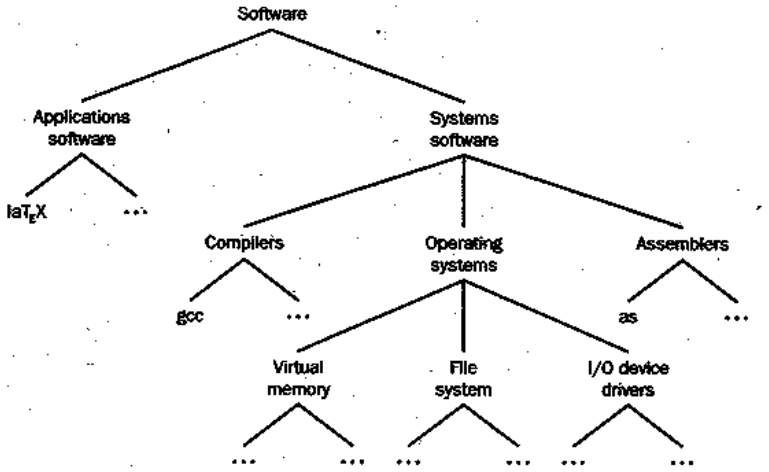
Zudem können mehrere Programme gleichzeitig laufen, wenn ein übergeordnetes Programm (Betriebssystem) die Ausführung überwacht und das nächste Programm startet, sobald die Abarbeitung des ersten abgeschlossen ist.

4.1 Hierarchie von Software und Hardware

Man kann Software also nach Verwendungszweck in Kategorien und Hierarchien unterteilen. Unter Systemsoftware versteht z.B. Betriebssystem, Compiler oder Assembler. Anwendersoftware beinhaltet alle Programme, die der User benötigt um spezielle Aufgaben zu lösen, beispielsweise ein Texteditor.



Allerdings läßt sich bestimmte Software nicht eindeutig plazieren. Der Compiler arbeitet beispielsweise auf beiden Programmebenen. Ein besseres Modell ist das der aufeinander aufbauenden Programme: Jede Software benutzt andere Software, die wiederum mit anderer Software arbeitet, so daß schließlich eine Baumstruktur entsteht.



4.2 Die “instruction set architecture”

Die abstrakte Gliederung von Soft- und Hardware in Stufen ist eine grundlegende Voraussetzungen für das Verstehen und Designen von hochentwickelten komplexen Computersystemen. Das wichtigste Beispiel von Abstraktion ist die Schnittstelle zwischen Software niedrigster Stufe und Hardware, die Prozessorarchitektur (instruction set architecture bzw. architecture). Sie schließt die Eingabe- und Ausgabegeräte genauso ein wie Anweisungen. Diese abstrakte Schnittstelle ermöglicht verschiedene Implementierungen einer Architektur mit unterschiedlichen Kosten und Leistung, auf denen dieselbe Software laufen kann.

5 Die Hardware

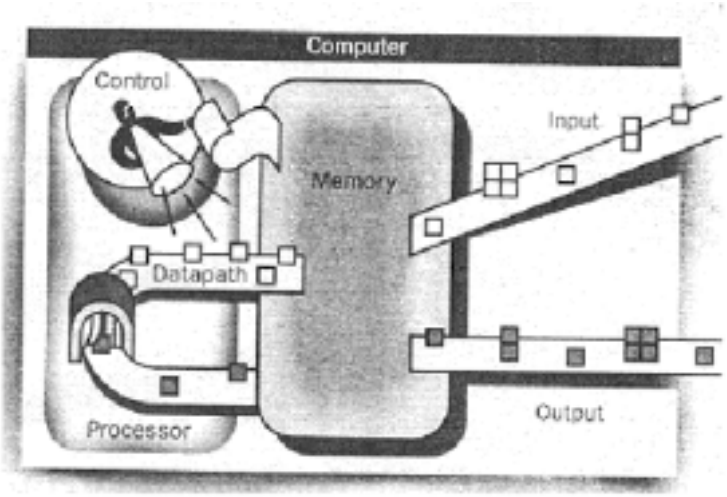
5.1 Das von-Neumann-Modell

Der von-Neumann-Rechner ist die am weitesten verbreitete Bauart.

Bestandteile:

- Rechenwerk und Steuerwerk (diese bilden gemeinsam den Prozessor),
- Speicher,
- Ein- und Ausgabe.

Jeder Teil eines Computers kann in eine dieser Kategorien eingeordnet werden.



Der Prozessor erhält Befehle und Daten vom Speicher, die Eingabe schreibt Daten in den Speicher und die Ausgabe liest die Daten aus dem Speicher. Das Steuerwerk sendet die Signale die die Operationen von Rechenwerk, Steuerwerk, Ein- und Ausgabe steuern.

5.2 Beispiel für ein Eingabegerät – Die Maus

Die Maus ist ein Eingabegerät, daß heute an praktisch jedem Rechner zu finden ist. Das war jedoch nicht immer so. Der erste Prototyp wurde erst 1967 vorgestellt. Ab den 80ern verfügten alle Workstations und viele PC's über dieses Eingabegerät, es wurden grafische Benutzeroberflächen entwickelt und die Maus begann, populär zu werden.

Die Maus ist eigentlich ziemlich einfach aufgebaut. Die mechanische Version sieht folgendermaßen aus:

Im Innern der Maus ist eine große Kugel derart aufgehängt, daß sie an der Unterseite mit der Auflagefläche in Kontakt steht. Ihre Bewegung wird auf ein Paar Räder übertragen, eines für die x-Achse und eines für die y-Achse. Abhängig davon, wie die Maus bewegt wird, wird eins oder beide Räder bewegt. Diese Räder drehen jeweils eine Schlitzscheibe, die den Strahlengang einer Lichtschranke unterbricht. So werden zwei Impulsfolgen erzeugt, die um 90° phasenverschoben sein müssen. Frequenz und Phasenverschiebung sind dann das Maß für Geschwindigkeit und Richtung der Bewegung.

5.3 Beispiel für ein Ausgabegerät – Der Bildschirm

Der Bildschirm ist ein Ausgabegerät, daß an jedem Computer zu finden ist. Es gibt verschiedene Bauweisen.

Auf der Fernsehtechnologie basierend wurden Monitore mit Kathodenstrahlröhre (Bildröhre) entwickelt. Das Bild wird 30 bis 75-mal pro Sekunde neu aufgebaut. Dadurch ist für den Benutzer kein Flackern sichtbar.

Für tragbare Computer wird ein leichter, dünner, energiesparender Bildschirm benötigt. Hierfür werden LCD-Displays eingesetzt (liquid crystal displays). Der Hauptunterschied besteht darin, daß die LCD

nicht die Lichtquelle ist. Eine typische LCD enthält stabförmige Moleküle in flüssiger Form, die eine gedrehte Helix bilden. Diese schirmt das Licht, das von einer Lichtquelle hinter dem Display stammt, ab. Wenn ein Strom angelegt wird, wird das Licht hindurchgelassen und wird auf dem Bildschirm sichtbar. Es gibt für jedes Pixel einen Schalter, um den Strom gezielt kontrollieren zu können und die Darstellung scharfer Bilder zu ermöglichen.

Das Bild wird aufgebaut als eine Matrix von Bildelementen, Pixel genannt, die als eine Matrix von Bits (Bitmap) dargestellt werden können. Abhängig von Bildschirmgröße und Auflösung bewegt sich diese von 512 x 340 bis 1560 x 1280 Pixeln. Die einfachsten Bildschirme (Schwarz-Weiss) brauchen ein Bit pro Pixel. Für Displays die 256 Graustufen erlauben, werden 8 Bits pro Pixel benötigt. Ein Farbbildschirm nutzt pro Grundfarbe (Rot, Blau und Grün) 8 Bit. Damit werden 24 Bit pro Pixel genutzt. Dies ermöglicht die Anzeige von Millionen verschiedenen Farben.

Unabhängig vom Bildschirm, besteht die Hardwareunterstützung des Computers für die Grafik im wesentlichen aus einem raster refresh buffer oder frame buffer, der die Bitmap speichert. Das darzustellende Bild ist im frame buffer gespeichert und wird auf dem Bildschirm entsprechend der refresh rate ausgegeben. Der Sinn der Bitmap ist die gleichmäßige Darstellung auf dem Bildschirm. Die Anforderungen an Monitore ergeben sich aus der hohen Leistungsfähigkeit des menschlichen Auges, das auch geringfügige Veränderungen auf dem Bildschim feststellt.

5.4 Der Prozessor

Der Prozessor ist der aktive Teil des Computers. Er befolgt buchstabengetreu die Programmanweisungen. Der Prozessor wird auch CPU (central processing unit) genannt.

Er setzt sich aus Rechenwerk und Steuerwerk zusammen. Das **Rechenwerk** führt die arithmetischen Operationen durch und das **Steuerwerk** gibt Anweisungen an Rechenwerk, Speicher und I/O-Devices entsprechend den Vorgaben des Programms.

5.5 Speichermedien

Es gibt verschiedene Bauarten von Speichermedien:

- Halbleiterspeicher – für Primärspeicher (Arbeitsspeicher, Cache)
- ferromagnetische – für Sekundärspeicher (Festplatten, Disketten)
- Magneto-optische – CD-ROM, WORM

5.5.1 Primärspeicher

- enthält die Programme zur Laufzeit
- ist stromabhängig – Abschalten des Stroms löscht die Daten
- mehrere DRAMs (dynamic random access memory) bilden den Arbeitsspeicher
- der Cache ist ein schneller Zwischenspeicher, der häufig benötigte Daten enthält

- Speicherkapazität Arbeitsspeicher: mehrere Hundert MB, Cache: ca. 1 MB
- Zugriffszeiten Arbeitsspeicher: 50 ns, Cache: deutlich schneller

DRAM: Der Wortteil RAM bedeutet daß alle Speicherzugriffe, unabhängig von der Position der Daten im Speicher, gleichlang dauern. Im Gegensatz hierzu stehen z.B. Magnetbänder, die nur sequentiellen Zugriff auf die gespeicherten Daten erlauben. Beim Magnetband ist daher die Position der Daten auf dem Band ausschlaggebend für die Zugriffszeit.

5.5.2 Sekundärspeicher

Der Sekundärspeicher behält die gespeicherten Daten auch nach Abschalten des Stroms. Weit verbreitet sind Magnetspeicher wie Disketten und Festplatten.

Die prinzipielle Arbeitsweise von Disketten und Festplatten ist die gleiche. Eine rotierende Platte ist mit einer magnetisierbaren Schicht bedeckt. Floppy Disks enthalten eine flexible Scheibe aus Plastik, Festplatten nutzen dagegen Metallscheiben. Außerdem können Disketten im Gegensatz zu den meisten Festplatten ohne große Umstände aus dem Rechner genommen und transportiert werden. Die Speicherkapazität von Disketten liegt bei 1,44 Megabyte, die von Festplatten bei mehreren Gigabyte. Zugriffszeiten von Festplatten liegen bei 10 ms. Die deutlich längeren Zugriffszeiten im Vergleich zum Halbleiterspeicher werden durch die mechanischen Bauteile der Festplatten verursacht. Die Kosten pro Megabyte sind jedoch für Festplatten deutlich niedriger als für DRAMs (1997 ca. 1/50).

Weitere Speichermedien sind die CD (optical compact disk) sowie das Magnetband. Dieses wird nur noch für Backups genutzt, da es für andere Anwendungen zu langsam arbeitet.

5.6 Integrierte Schaltkreise

- Dutzende bis Hunderte Bauelemente (z.B. Transistoren, Dioden, Widerstände) sind auf einem Siliziumplättchen aufgebracht
- Hochintegrierte Schaltungen (VLSI) enthalten Millionen von Transistoren

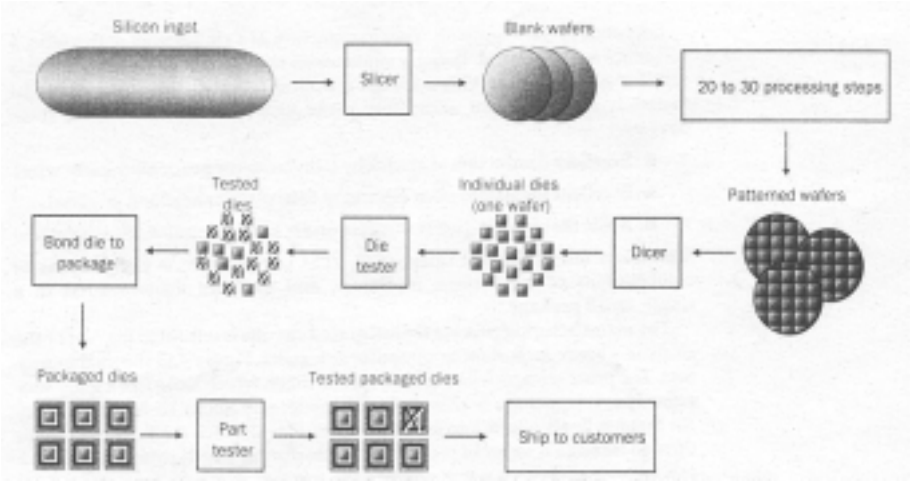
Prozessoren und Speicher haben sich mit einer unglaublichen Geschwindigkeit entwickelt. Die Speicherkapazität von DRAMs hat sich in der Zeit von 1976 bis 1996 alle drei Jahre vervierfacht. Mit jeder Weiterentwicklung hat sich das Preis-Leistungs-Verhältnis weiter verbessert.

Relatives Preis-Leistungs-Verhältnis im Verhältnis zu Vakuumröhren:

1951	Vakuumröhre	1
1965	Transistor	35
1975	Integrierte Schaltkreise	900
1995	Hochintegrierte Schaltkreise	2.400.000

5.6.1 Herstellung integrierter Schaltkreise

Der Herstellungsprozeß für integrierte Schaltkreise ist entscheidend für die Kosten des Chips. Die Fläche des Chips ist dabei der Hauptkostenfaktor.



Die Herstellung beginnt mit einem Siliziumblock, der wie eine riesige Wurst aussieht. (6-12 Zoll Durchmesser und 12-24 Zoll Länge). Der Block wird in Wafer geschnitten, die eine Dicke von höchstens 0,1 Zoll haben. Diese Wafer durchlaufen eine Serie von Bearbeitungsschritten, durch die ein Muster von Chemikalien auf ihnen angeordnet wird. Die chemischen Muster erzeugen die Transistoren, Leiter und Isolatoren.

Ein winziger Fehler auf dem Wafer oder in einem der vielen Bearbeitungsschritte macht den betroffenen Bereich unbrauchbar. Diese Defekte machen es praktisch unmöglich, einen perfekten Wafer zu produzieren. Deshalb werden auf einem Wafer mehrere voneinander unabhängige Komponenten erzeugt. Der Wafer wird zerlegt und es entstehen einzelne Blöcke (dies). Die fehlerfreien 'dies' werden aussortiert und mit den input/output pins verbunden.

Literatur

[1] D.A. Patterson, J.L. Hennessy. *Computer Organization & Design, The Hardware/Software Interface*. Morgan Kaufmann, 1998.

[2] W. Schiffmann, R. Schmitz: *Technische Informatik 1 / Technische Informatik 2*. Springer-Verlag, 1999.

[3] *Duden Informatik*. Duden-Verlag, 1988.

Computer Abstractions and Technology

Einführung

Programmierer waren schon immer an noch leistungsfähigeren Programmen interessiert.
60/70er Jahre: Problem der begrenzten Speicherkapazität
heute: Verständnis von Computerorganisation notwendig, um sinnvoll und konkurrenzfähig zu programmieren

Es geht um:

- Basis-Definitionen und -Ideen
- die wichtigsten Hard- und Softwarekomponenten
- integrierte Schaltkreise

Grundbegriff Abstraktion

...ist bei der Betrachtung von Computersystemen ein fundamentaler Begriff. Software und Hardware kann hierarchisch geordnet werden
Jede tiefere Stufe offenbart mehr Details

Ein einfacheres Modell auf höherer Ebene vernachlässigt Details

$\rightarrow$ Diese Stufen heißen Abstraktionen

Historisches: Computer-Generationen

Generation	Zeitraum	Technologie	wichtigstes neues Produkt
1	1950-1959	Vakuumröhren	kommerzielle Computer
2	1960-1968	Transistoren	Billigere Computer
3	1969-1977	Integrierte Schaltkreise	Minicomputer
4	1978-	LSI und VLSI	Personalcomputer und Workstations

VLSI = very large-scale integrated circuit

Rasante Technologie-Entwicklung

Seit 1985 sind ein halbes Dutzend neue Maschinen entwickelt worden, die alle die Computer-Industrie revolutionierten
$\rightarrow$ Wettrennen führt zu beispiellosem Fortschritt seit den 40er Jahren

Das Transportwesen wäre bei gleichem Wachstum heute an einem unglaublichen Punkt: in 5 Sekunden „coast-to-coast“ für 50 Cent!

Computerrevolution - Einfluß auf andere Wissenschaften

Vervielfachung der menschlichen intellektuellen Kraft und Reichweite beeinflußt auch andere Wissenschaftszweige

Anwendungen, die vor Jahren noch absurd schienen:

- Bankautomat
- Chips im Auto
- Laptops
- Entschlüsselung menschlicher Gene
- World Wide Web

Ausblick - zukünftige Entwicklungen?

Fortschritte in der Computertechnologie streifen alle Bereiche unserer Gesellschaft

Die Science-fiction Anwendungen der Zukunft:

- bargeldlose Gesellschaft
- automatisierte intelligente Highways

Die Software „unter“ dem Programm

“In Paris they simply stared when I spoke to them in French; I never did succeed in making those idiots understand their own language.”
(Mark Twain)

Die Sprache einer elektronischen Maschine aus 0 und 1 (bits) läßt sich eine unendliche Menge von Anweisungen und Daten bilden
$\rightarrow$ Mensch und Maschine können kommunizieren!

Beispiel: Addition Binär 1000110010100000

Addition in Assemblersprache: add A, B
wird vom Assembler in binäre Maschinensprache übersetzt

Assemblersprachen wurden seit den 40er Jahren auf die jeweiligen Maschinensprachen der Systeme ausgerichtet

High-level Programming Languages

Compiler übersetzt Programmcode in Assemblercode

high-level programming languages haben eine natürlichere Notation, aus der sich folgende Vorteile gegenüber Assemblersprache ergeben:

- Programmierer müssen nicht in kryptischen Symbolen denken
- höhere Programmierproduktivität durch Kürze und Prägnanz
- verschiedene Programmiersprachen entstehen für unterschiedliche Zwecke
- Programme sind unabhängig von dem Computer, auf dem sie entwickelt wurden, weil es Compiler und Assembler für alle möglichen Rechnertypen gibt

Subroutine Libraries

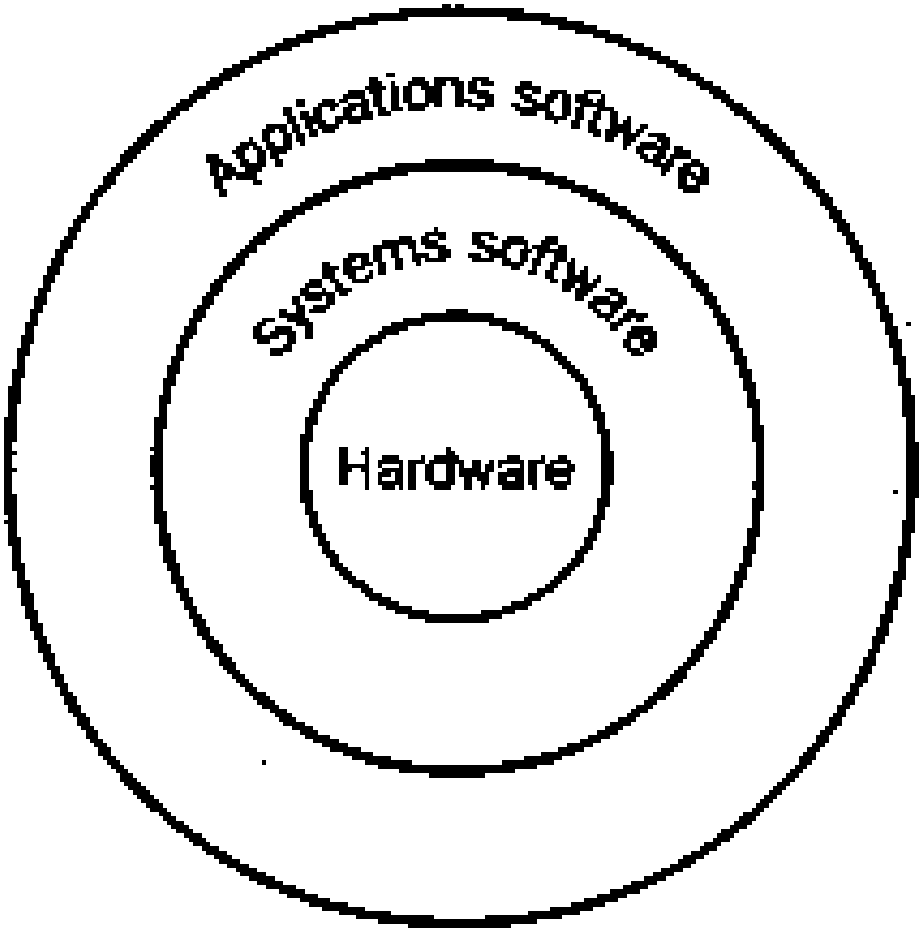
Immer wiederkehrende oder häufig benutzte Anwendungen nennt man Routinen. Sie werden in "subroutine libraries" zusammengefaßt. z.B. Kontrolle der Dateneingabe und -ausgabe sowie dazugehörender Geräte

Betriebssystem (operating system)

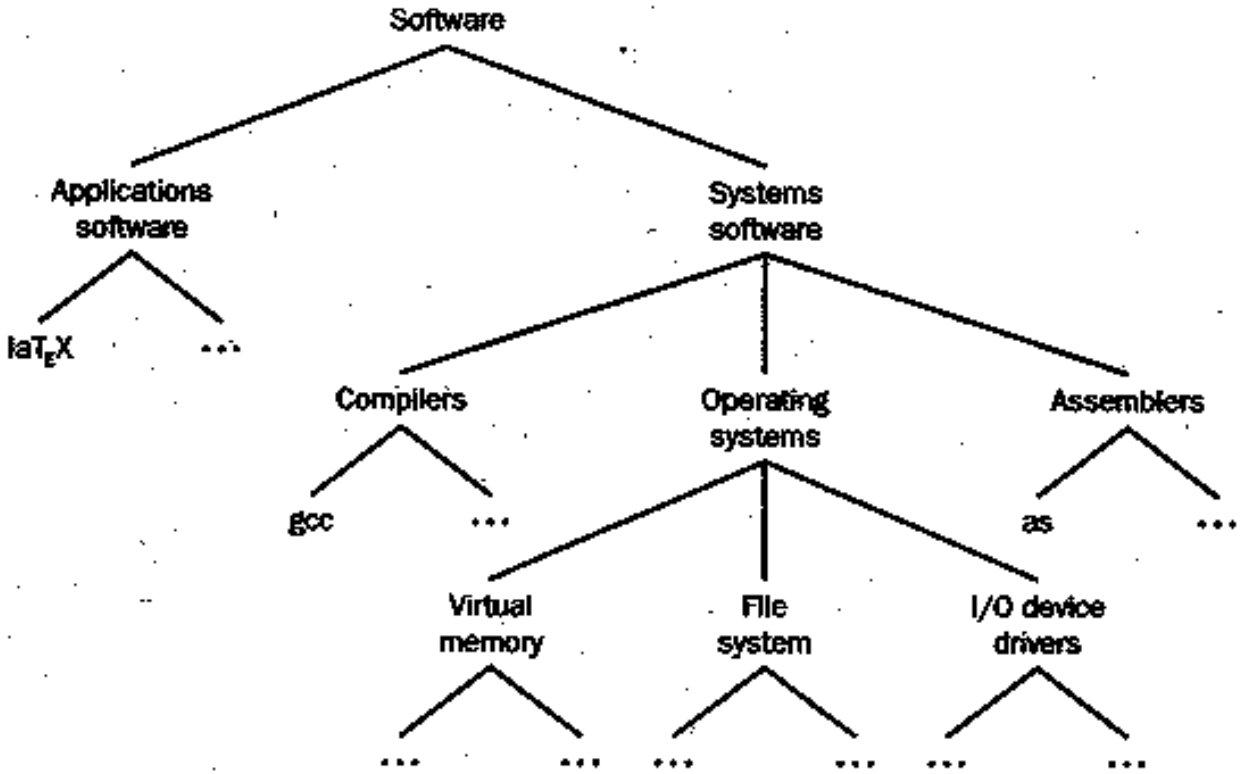
Ein übergeordnetes Programm überwacht die einzelnen Ausführungen und startet sofort das nächste Programm, sobald die Abarbeitung des vorigen abgeschlossen ist.

- enthält Dienstprogramme für immer wiederkehrende Aufgaben
- mehrere Programme können gleichzeitig laufen
- Verwaltung von Rechner-Ressourcen

Hierarchie von Software und Hardware (1)



Hierarchie von Software und Hardware (2)



instruction set architecture

„Sowohl Hardware als auch Software besteht aus hierarchischen Stufen, mit jeder höheren Stufe werden Details der niedrigeren Stufen verborgen. Dieses Prinzip der Abstraktion ist für Hardwaredesigner und Softwaredesigner nützlich, um mit der Komplexität von Computersystem umzugehen.

Eine zentrale Schnittstelle ist die Prozessorarchitektur, die Schnittstelle zwischen Hardware und Software auf unterer Ebene. Diese abstrakte Schnittstelle ermöglicht viele verschiedene Implematationen, auf denen mit unterschiedlichen Kosten und Leistungen dieselbe Software laufen kann.“

Das von-Neumann-Modell

Der von-Neumann-Rechner ist die am weitesten verbreitete Bauart.

Bestandteile:

- Rechenwerk
- Steuerwerk
- Speicher
- Ein- und Ausgabe

Jeder Teil eines Computers kann in eine dieser Kategorien eingeordnet werden.

Die Maus (1)

- weitverbreitetes Eingabegerät
- erster Prototyp 1967 vorgestellt
- ab 1973 wurde der Alto mit Maus ausgeliefert
- ab den 80ern war die Maus im Lieferumfang aller Workstations und vieler Personalcomputer enthalten
- die Entwicklung grafischer Benutzeroberflächen steigerte im folgenden die Popularität der Maus

Die Maus (2)

Aufbau (mechanische Version):

- eine große Kugel bewegt zwei Räder
- ein Rad gibt die Bewegung in x-Richtung, das andere in y-Richtung an
- Daten werden ins System übertragen und geben Länge und Richtung der Bewegung der Maus an
- Position des Mauszeigers wird im System gespeichert

Der Monitor

- Kathodenstrahlröhre (Bildröhre) für gewöhnliche Monitore, LCD-Display für Laptops und Flachbildschirme
- das Bild wird als eine Matrix von Bildelementen gespeichert (Bitmap)
- abhängig von Auflösung und Größe des Bildes sind Bitmap-Größen von 512×340 bis 1560×1280 Pixel möglich
- Speicherbedarf pro Pixel:
  Schwarz-Weiss: 1 Bit, 256 Graustufen: 8 Bit,
  Farbe: 24 Bit (8 Bit pro Grundfarbe)
- der refresh buffer oder frame buffer speichert die Bitmap
- Bitmap wird entsprechend der refresh rate auf dem Bildschirm ausgegeben

Der Prozessor (CPU = central processing unit)

- führt die Programmanweisungen buchstabengetreu aus
- setzt sich aus Rechenwerk und Steuerwerk zusammen
- Rechenwerk: führt die arithmetischen Operationen aus
- Steuerwerk: gibt Anweisungen an Rechenwerk, Speicher und I-/O-Devices

Der Speicher

Es gibt verschiedene Bauarten von Speichermedien

- Halbleiterspeicher – für Primärspeicher (Arbeitsspeicher, Cache)
- ferromagnetische – für Sekundärspeicher (Festplatten, Disketten)
- Magneto-optische – CD-ROM, WORM

Primärspeicher

- ist stromabhängig
- Arbeitsspeicher enthält die Programme zur Laufzeit
- mehrere DRAMs (dynamic random access memory) bilden den Arbeitsspeicher
- der Cache ist ein schneller Zwischenspeicher, der häufig benötigte Daten enthält
- Speicherkapazität Arbeitsspeicher: mehrere Hundert MB, Cache: ca. 1 MB
- Zugriffszeiten Arbeitsspeicher: 50 ns Cache: deutlich schneller

Sekundärspeicher

- behält die gespeicherten Daten auch nach Abschalten des Stroms
- Daten werden bei Festplatten und Disketten auf einer magnetisierbaren rotierenden Scheibe gespeichert
- Speicherkapazität Diskette: 1,44 MB, Festplatte: mehrere GB
- Zugriffszeiten Festplatte: 10 ms
- Kosten pro MB deutlich niedriger als bei Primärspeicher

Integrierte Schaltkreise

- Dutzende bis Hunderte Bauelemente (z.B. Transistoren, Dioden, Widerstände) sind auf einem Siliziumplättchen aufgebracht
- Hochintegrierte Schaltungen (VLSI) enthalten Millionen von Transistoren

Herstellung integrierter Schaltkreise

- ein Siliziumblock in Form einer riesigen Wurst (Länge 12 - 24 Zoll, Durchmesser 6 - 12 Zoll) wird in Wafer geschnitten (Dicke höchstens 0,1 Zoll).
- durch eine Serie von Bearbeitungsschritten wird ein Muster von Chemikalien auf den Wafern angeordnet
- die chemischen Muster erzeugen Transistoren, Leiter und Isolatoren
- um Fehler leichter verschmerzen zu können, werden mehrere voneinander unabhängige Bauteile auf einem Wafer erzeugt
- der Wafer wird in die einzelnen Blöcke zerlegt
- die fehlerfreien Blöcke werden mit I/O-Pins verbunden
  $\rightarrow$ Chip = integrierter Schaltkreis

Die Bedeutung von Performance

Martin Gaitzsch, Mirek Hancl, Davood Kheiri

8.11.2000

A. Einleitung, 'relative' Performance

Performance interessiert sowohl den Systementwickler als auch den Kunden. Auf der Entwicklungsseite kommt z.B. neues Chiplayout der Performance-Optimierung zugute, der Anwender profitiert von Performanceangaben beim z.B. Kauf eines Neusystems (Benchmarks).
Für Performancebestimmung ist Zeit das Maß aller Dinge. Systemadministratoren interessiert der Durchsatz, d.h. wie viele Anfragen an ein System gleichzeitig gestellt werden können, für Einzelplatzsysteme ist die verstrichene Zeit eines einzelnen Programms aussagend. Man unterscheidet zwischen

- Antwortzeit: Zeit zwischen Starten und Beenden eines Programms incl. Wartezeit für I/O (wall-clock time, response time, elapsed time)
- Ausführungszeit: CPU-time
- User CPU-time: Zeit für Programmablauf
- System CPU-time: Zeit für Betriebssystemaktivitäten

'Relative' Performance gibt einen einfachen Leistungsvergleich zwischen zwei Systemen durch Zeitmessung eines gleichen Programms auf beiden Systemen.

$$\frac{Performance_x}{Performance_y} = \frac{Ausführungszeit_y}{Ausführungzeit_x} = n$$

Beispiel: Rechner A benötigt 10s für ein Programm, Rechner B 15s für dasselbe Programm.
Rechner A ist 15/10=1,5mal schneller als Rechner B.

B. Taktzyklus, Berechnung Ausführungszeit u. CPU-time

Ein Quarz in einem System liefert konstante Zeitintervalle, sog. Taktzyklen (clock cycles, ticks,...).
Eine Taktperiode ist die Zeit eines Taktzyklus bzw. die Taktrate, welche die Inverse des Taktzyklus ist.
Damit läßt sich die Ausführungszeit für ein Programm berechnen:

Ausführungszeit für ein Programm	=	benötigte Taktzyklen	mal Zeit eines Taktzyklus *oder* geteilt durch Taktrate

Beispiel: Benötigte Taktzyklen für Programm X: 10^{10}, Zeit eines Taktzyklus: 2ns, Taktrate: 500 MHz
Ausführungszeit für Programm X: 10^{10} / 500 MHz = 20s

→ Mehr Performance durch weniger Taktzyklen und der jeweiligen Zeit eines Taktzyklus

Um die CPU-time errechnen zu können muß man wissen, wie viele Taktschritte ein Programm benötigt. Da viele Maschinenbefehle mehr als einen Taktzyklus benötigen (Multiplikation, Gleitkomma-Arithmetik, Speicherzugriffe), wird eine ‘mittlere‘ Anzahl von Taktschritten je Befehl angegeben. Man spricht von clock cycles per instruction, kurz: CPI. Berechnet wird diese durch gewichteten Mittelwert aller individuellen CPIs der Befehle im Programm.

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i * C_i)}{\sum_{i=1}^{n} C_i}$$

Befehle der Klasse i mit CPI_i kommen im betrachteten Programm C_i mal vor.

Beispiel: Befehl A: 1 Taktzyklus, kommt 65mal im Programm X vor.

Befehl B: 2 Taktzyklen, kommt 35mal im Programm X vor.

$$CPI_x = \frac{65*1+35*2}{35+65} = 1{,}35$$

Die CPU-time ergibt sich nun aus folgendem grundlegenden Zusammenhang:

$$CPU\ time = \frac{Instructions}{Pogram}\ X\ \frac{Clock\ Cycles}{Instruction}\ X\ \frac{Seconds}{Clock\ Cycle}$$

⇩ ⇩ ⇩

Befehlsanzahl * CPI * Takperiode

C. Arithmetisches Mittel, Geometrisches Mittel

Mit dem arithmetischen Mittel lassen mehrere Einzelmessungen der Ausführungszeiten einfach zusammenfassen:

$$\frac{1}{n} * \sum_{i=1}^{n} Time_i$$

n = Anzahl der Programme; $Time_i$ = Ausführungszeit de i-ten Programms

Beispiel: Programm 1 läuft auf Rechner A in 1s, auf Rechner B in 10s.

Programm 2 läuft auf Rechner A in 1000s, auf Rechner B in 100s.

Gesamtausführungszeit Rechner A: 1001s, Rechner B: 110s.

Arithmetisches Mittel Rechner A: 500,5s, Rechner B: 55s

→ Besser lassen sich mehrere Einzelmessungen über das geometrische Mittel der Ausführungszeiten zusammenfassen:

$$\sqrt[n]{\prod_{i=1}^{n} Time_i}$$

n = Anzahl der Programme, $Time_i$ = Ausführungszeit des i-ten Programms

Für das obige Beispiel ergibt sich somit ein geometrisches Mittel von 31.6 für beide Rechner.

D. Amdahlsches Gesetz (vereinfacht)

Alt:

$$\text{Optimierte Ausführungszeit} = \frac{\text{optimierter Zeitanteil}}{\text{Verbesserungsfaktor}} + \text{restl. Zeitanteil}$$

Neu:

$$\text{Speedup} = \frac{\text{Performance nach Verbesserung}}{\text{Performance vor Verbesserung}} = \frac{\text{Ausführungszeit vor Verbesserung}}{\text{Ausführungszeit nach Verbesserung}}$$

Konsequenz: Optimiere den häufigsten Fall !

E. Benchmark, Performance-Maße

Zur Messung der Performance eignen sich besonders echte Anwenderprogramme, die fertig vorliegen und nicht verändert werden können. Jedoch sind diese anwenderabhängig, d.h. ein CAD-Programm eines Architekten interessiert einen Programmierer zur Performancemessung seines Systems wenig, eher schon ein Compiler oder Texteditor. Programme zur Messung sollen einfach strukturiert sein, allerdings sind sie betrugsanfällig durch z.B. den Einsatz eines optimierten Compilers. Als allgemein akzeptierten Kompromiß gilt SPEC (System Performance Evaluation Cooperative), gegründet von Apollo/HP, DEC, MIPS und Sun im Jahre 1989. Als Referenz diente bis 1995 eine VAX 11/780, gemessen dient ein Softwarebündel mit unterschiedlichsten Operationen. Durch einen optimierten Compiler war es möglich, den Benchmark der Programme matrix300 und nasa7 um ein vielfaches zu beeinflussen. Als Konsequenz wurden diese 1992 gestrichen und integer- von floating point-Operationen getrennt. Seit 1995 dient eine SunSparcStation 10/40, etw 40mal schneller als die VAX11/780, als Referenzsystem. Neuere Programme wie Webapplikationen sind seitdem dazugekommen.

Die Specratio ergibt sich durch einfaches Dividieren der gemessenen Ausführungszeit durch die des Referenzsystems.

Performance wird des weiteren in folgenden Maßen gemessen:

MIPS, d.h. ‘million instructions per second‘ (native MIPS).

$$\text{native MIPS} = \frac{\text{Anz.d..Befehle}}{\text{Ausf.zeit} * 10 \text{ hoch } 6}$$

Bei optimiertem Compiler mit minimierter CPI spricht man von peak MIPS. Wird gerne von Chipherstellern zu Werbezwecken benutzt.

Bei Vergleich mit einem Referenzrechner spricht man von relative MIPS.

$$\text{relative MIPS} = \frac{\text{Ausf.zeit Referenzrechner}}{\text{Ausf.zeit zu testender Rechner}} * \text{MIPS reference}$$

MOPS, d.h. ‘million operations per second‘

MFLOPS, d.h. ‘million FP operations per second‘

$$\text{MFLOPS} = \frac{\text{Anzahl der Gleitkommaoperationen im Programm}}{\text{Ausf.zeit} * 10 \text{ hoch } 6}$$

Ebenfalls gibt es peak MFLOPS.

Whetstone,Drystone,.... sind synthetische Benchmarks, die speziell im High-End Bereich zum Einsatz kommen.

F. Performance-Optimierung, Zusammenfassung

Performance kann optimiert werden durch höhere Taktrate und/oder geringere CPI. Während letzteres in den Bereich Software fällt und Sache eines Compilers ist, muß für eine höhere Taktrate die Hardware weiterentwickelt werden (Technologischer Optimierung). Einseitig kann selten etwas optimiert werden, es sind oftmals Kompromisse erforderlich.
Performance ist also abhängig vom gewählten Programm, zum besten Vergleich kommen reale Applikationen zum Einsatz. Zeit ist das objektivste Performance-Maß und der häufigste Fall ist zu optimieren (Amdahl'sches Gesetz). Performance-Optimierung ist kompromißgebunden zwischen Taktrate, CPI und Befehlsanzahl. Hersteller geben oft verzerrte Performanceangaben an. Schließlich ist Optimierung auch eine Kostenabwägung.

Quellenangaben:

[1]David A. Patterson, John L. Hennesy. *Computer Organisation & Design, The Hardware/Software Interface*. Morgan Kaufmann, 1998.

Die Bedeutung von

Performance

Martin Gaitzsch, Mirek Hancl, Davood Kheiri

Überblick

Performance richtig

- verstehen
- bestimmen
- angeben

- Entscheidung bei Kauf, Design, Optimierung
- optimale Nutzung der HW bei Programmierung

- besseres Verständnis für Rechnerachitekturen

Performance definieren

Flugzeug	Kapazität	Reichweite (Meilen)	Geschw. (m.p.h.)	Durchsatz
Boeing 777	375	4630	610	228750
Boeing 747	470	4150	610	286700
Concorde	132	4000	1350	178200

Douglas DC-8-50	146	8720	544	79424

Kenngrößen für Performance

- Antwortzeit (wall-clock time, response time, elapsed time): Zeit zwischen Starten und Beenden eines Programms incl. Wartezeit für I/O
- Ausführungszeit: CPU-time
- User CPU-time: Zeit für Programmablauf

- System CPU-time: Zeit für OS – Aktivitäten
- Durchsatz

$$Performance_x = \frac{1}{Ausführungszeit_x}$$

‘relative‘ Performance:

$$\frac{Performance_x}{Performance_y} = \frac{Ausführungszeit_y}{Ausführungszeit_x} = n$$

Beispiel:
Rechner A benötigt 10s für ein Programm,

Rechner B benötigt 15s für dasselbe Programm.
Rechner A ist 15/10 = 1,5 mal schneller als Rechner B.

Taktzyklen

Taktgeber (Quarz) liefert konstante Zeitintervalle, sog. Taktzyklen (clock cycles, ticks,...)
Taktperiode: Zeit eines Taktzyklus (z.B. 2ns) bzw. Taktrate (z.B. 500 Mhz), Inverse des Taktzyklus

Ausführungszeit für ein Programm	=	benötigte Taktzyklen	X	Zeit eines Taktzyklus
Ausführungszeit	=	benötigte	/	Taktrate

für ein Programm Taktzyklen
→ Mehr Performance durch weniger Taktzyklen und der Zeit eines Taktzyklus

CPI: “clock cycles per instruction“

Wieviele Taktschritte benötigt ein Programm ?

Viele Maschinenbefehle benötigen mehr als einen Taktzyklus:

- Multiplikation, Division
- FP-Arithmetik
- Speicherzugriffe

Angabe einer “mittleren“ Anzahl von Taktschritten je Befehl: CPI

Bsp: Performance-Vergleich

- Prozessor A: Taktzyklus = 1ns, CPI = 2.0 für Progr. X
- Prozessor B: Taktzyklus = 2ns, CPI = 1.2 für Progr. X

A und B haben gleichen Befehlssatz.

Welche CPU ist um wieviel schneller ?

CPI-Berechnung

Gewichteter Mittelwert aller individuellen CPIs der Befehle im Programm:

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i * C_i)}{\sum_{i=1}^{n} C_i}$$

Befehle der Klasse i mit CPI_i kommen im betrachteten Programm C_i mal vor.

Bsp: CPI-Berechnung

CPU:

Befehlsklasse	A	B	C
CPI	1	2	3

Compiler:

	Häufigkeit der Klasse		
Code	A	B	C
X	2	1	2
Y	4	1	1

Welcher Code ist schneller ?

Performance-Faktoren

Grundlegender Zusammenhang:

Performance-Optimierung

- höhere Taktrate (Technologie)
- geringere CPI (Compiler)
- weniger Befehle pro Programm bzw. Befehle mit geringerer CPI (Compiler)

einseitige Verbesserung eines Faktors ist selten möglich, meist sind Kompromisse erforderlich.

Der ideale Benchmark

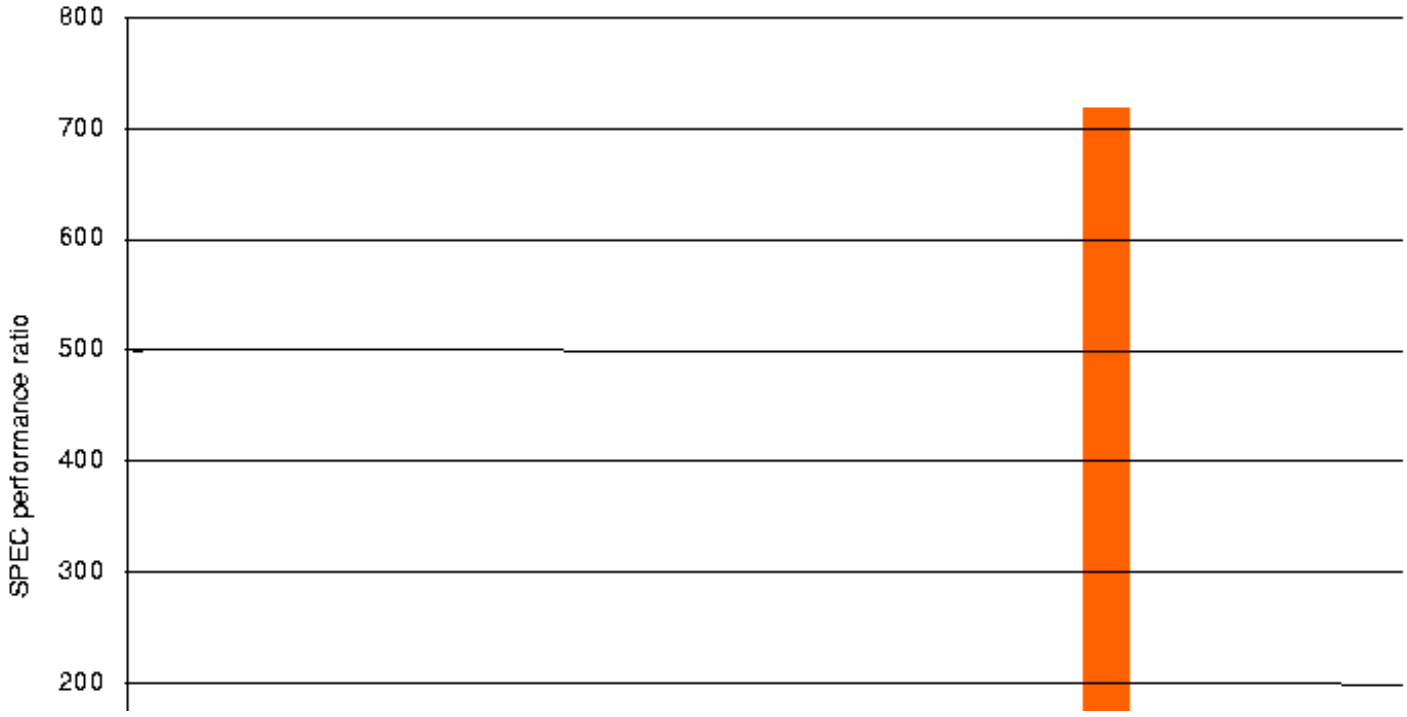
- Echtes Anwenderprogramm
  unbestechlich aber anwenderabhängig

- Einfaches Programm

Einfach nachvollziehbar aber “betrugsanfällig“

- SPEC (System Performance Evaluation Cooperative) allgemein akzeptierter Kompromiß

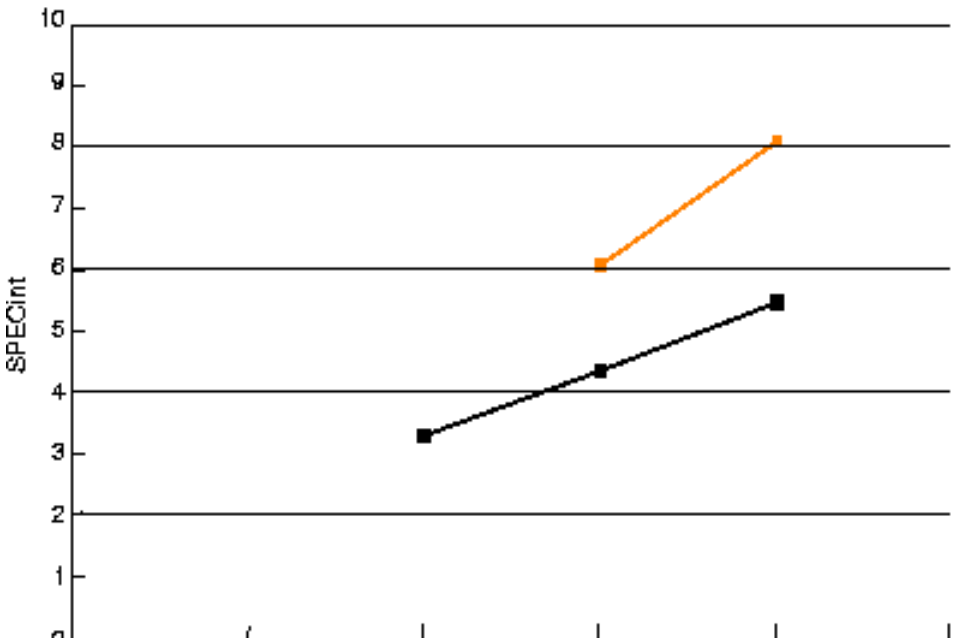
Compileroptimierungen

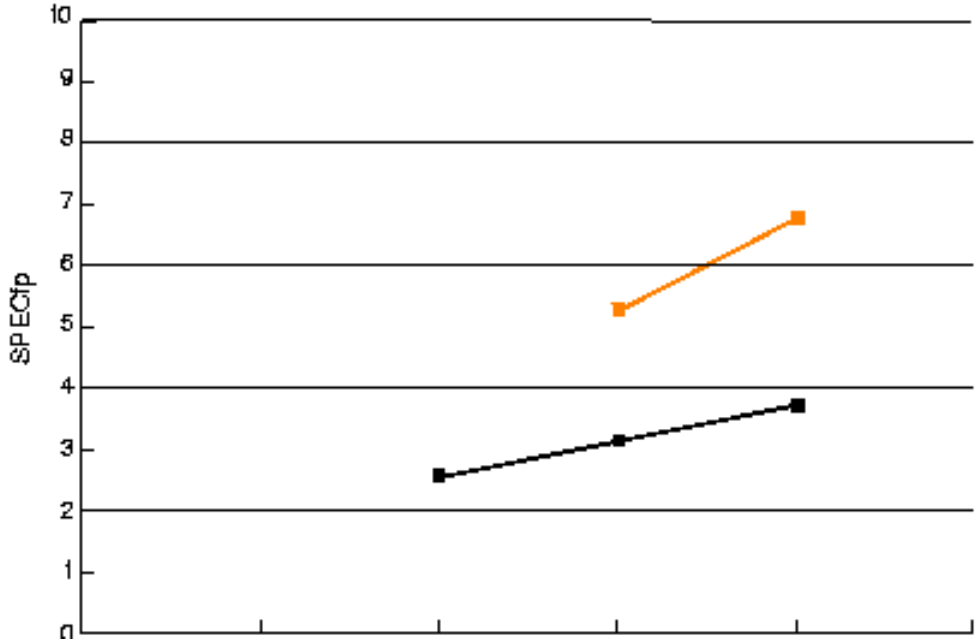


SPEC'95 Benchmark

	Benchmark	Description
int	go	Artificial intelligence; plays the game of Go
	m88ksim	Motorola 88k chip simulator; runs test program
	gcc	The Gnu C compiler generating SPARC code
	compress	Compresses and decompresses file in memory
	li	Lisp interpreter
	ijpeg	Graphic compression and decompression
	perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
	vortex	A database program
FP	tomcatv	A mesh generation program
	swim	Shallow water model with 513 x 513 grid
	su2cor	quantum physics; Monte Carlo simulation
	hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
	mgrid	Multigrid solver in 3-D potential field

Pentium Performance (SPEC’95)





“Gesamt“-Performance

Beispiel:

	Computer A	Computer B
Prog 1	1 s	10 s
Prog 2	1000 s	100 s
Summe	1001 s	110 s

→Der einfachste Weg zur Zusammenfassung mehrerer Einzelmessungen ist das arithmetische Mittel der Ausführungszeiten:

$$\frac{1}{n} * \sum_{i=1}^{n} Time_i$$

N = Anzahl der Programme
$Time_i$ = Ausführungszeit des *i* -ten Programms

Amdahl'sches Gesetz

$$Optimierte\ Ausführungzeit = \frac{optimierter\ Zei\tan teil}{Verbesserungsfaktor} + restl.\ Zei\tan teil$$

Konsequenz:

Optimiere den häufigsten Fall !

Verbreitete Performance-Maße

MIPS
- native: “million instructions per second“
- peak: MIPS für Code mit min CPI
- relative: bezogen auf Referenzrechner

MOPS = “million operations per second“

MFLOPS = “million FP operations per second“

Whetstone, Dhrystone,... synthet. Benchmarks

Beispiel: Native MIPS

Zwei unterschiedliche Compiler generieren für die gleiche Applikation folgende Befehlssequenzen:

	Häufigkeit der Befehlsklassen		
Code von	1 CPI	2 CPI	3 CPI
Compiler 1	$5*10^9$	$1*10^9$	$1*10^9$

Compiler 2	$10*10^9$	$1*10^9$	$1*10^9$

Welcher Code ist schneller ?
Wie sind die entsprechenden native MIPS bei 500 MHz Prozessortakt ?

Zusammenfassung

- Performance abhängig vom gewählten Programm
- Reale Applikationen sind die besten Benchmarks
- Herstellerangaben über Performance oft verzerrt
- Zeit ist das objektivste Performance-Maß
- Optimiere den häufigsten Fall (Amdahl’sches Gesetz)

- Performance-Optimierung erfolgt im Spannungsfeld zwischen Taktrate, CPI und Befehlsanzahl
- Kostenabwägung

6. Adressierungsarten
6.1 Register Adressing

Register Adressing

Der für die Instruction benötigte Operand steht in einem der 32 Register des Prozessors.

6.2 Immediate Adressing

IAderseeing

Der Operand ist eine Konstante im Befehl. Diese Adressierungsart benutzt man um bei Verwendung von kleinen Konstanten, Speicherzugriffe zu vermeiden.

6.3 Base Adressing

Base Adressing

Bei dieser Art von Adressierung befinden sich die benötigten Operanden im Speicher oder kommen von irgendeinem Eingabegerät.
Die Adresse des Operanden ist die Summe der im zweiten Register gespeicherten Basis-Adresse und der Konstanten, die in den hinteren 16 bit des Befehls steht.

7. Subroutinen:
Eine Subroutine (procedure call) ist ein Programmierwerkzeug. Kleine Programme, die man oft braucht, werden in einem bestimmte Speicherbereich abgespeichert und können dann aufgerufen werden.
Sie werden aufgerufen mit dem Befehl Jump and link

jal <Adresse der Subroutine>

Am Ende der Subroutine steht der Befehl Jump return

jr $ra

Das Register ra ist eines der 32 Register des MIPS, in dem bei dem Befehl jal die Adresse des im Programm folgenden Befehls gespeichert wird.
Wenn die Subroutine an ganz anderer Stelle noch mal gebraucht wird, muss sie trotzdem nur einmal programiert werden.
d.H. bei einem etwas umfangreicheren Programm, sollte man auf jeden Fall Subroutinen programmieren. Das eigentliche Programm wird dadurch übersichtlicher und wenn eine Subroutine von mehreren Stellen im Programm benutzt wird spart man Speicherplatz.

Um zu vermeiden daß im Verlauf einer Subroutine Inhalte von Registern, die an anderer Stelle noch gebraucht werden, überschrieben werden, speichert man diese vor Beginn einer Subroutine.
Den Vorgang, vor Beginn einer Subroutine Daten aus den Registern des Prozessors in den Speicher wegzusichern nennt man push.
Nach Beendigung der Subroutine Daten wiederholen nennt man pop.
Es gibt eine Festlegung welche Register zu Beginn jeder Subroutine gesichert werden und welche nicht.

Liste 138

Es gibt ein Register das dafür da ist, festzulegen und zu speichern wo (an welchen Speicheradressen) die Daten gespeichert werden.
Das Register heißt $sp (Stack Pointer).

8 Von der Hochsprache zur Machinensprache.

Folie A%C 156

8.1Compiler

Man programmiert in Hochsprachen, weil es leichter und schneller ist, und mittlerweile Compiler sehr effektiv übersetzen.
Ein Compiler ist ein Programm das Programme von einer Hochsprache in ein Assembler Programm übersetzt.
Mit dem entsprechenden Compiler sind Programme also universell einsetztbar und nicht mehr an den Prozessor gebunden, in dessen Befehlsatz sie programmiert wurden.

8.2Assembler

der Assembler übersetzt die Befehle und Registerbezeichnungen in Maschinensprache.

8.3Linker

Werden in den zu übersetzenden Programm irgendwelche Subroutinen verwendet die nicht Teil des Programms sind, sondern irgendwo anders gespeichert sind, muß die Verbindung zu diesen hergestellt werden. Diese Aufgabe erfüllt der Linker (Binder).

8.4 Loader

Der Lader erhält von der Speicherverwaltung des Betreiebssystem dier Anweisung wo das Programm gespeichert werden soll und rechnet die Adressen des Programms in die entsprechenden Adressen des Speichers um.

9. Zahlen und Buchstaben

Buchstaben können durch Zahlen dargestellt werden. (codieren)
Der wichtigste Zahlencode zur Darstellung von Buchstaben ist der ASCII-Code.
ASCII:American-Standard-Code for Information-Interchange
Der ASCII-Code umfasst 128 Zeichen:

- -große und kleine Buchstaben (52 Zeichen)
- -Ziffern von 1-9 (9 Zeichen)
- -Sonderzeichen und Steuerzeichen (66 Zeichen)

Folie ASCII-Code

Es gibt noch eine erweiterte Form des ASCII Codes, die auch Sonderzeichen wie Ä enthält, die dann 256 Zeichen umfasst und entsprechend 8 bit benötigt.

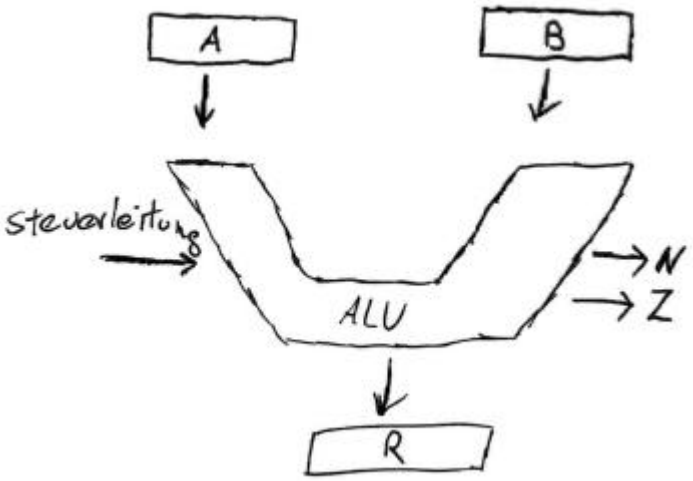
Vortrag von René Grohmann und Mirwais Turjalei, 22.11.2000

Computer Arithmetik

Computer Arithmetik Allgemein

Die ALU: Die Alu ist die Einheit im Computer, die dazu bestimmt ist arithmetische und logische Operationen durchzuführen. Sie arbeitet eng mit so gut wie allen anderen Komponenten des Computers zusammen, unter anderem mit der Steuereinheit, dem Speicher, den Registern und den Input/Output (IO) Komponenten. In diesem Vortrag beziehen wir uns ausschließlich auf die arithmentischen Aufgaben der ALU und nicht auf die logischen. Im Mittelpunkt stehen also Addition, Subtraktion, Multiplikation und Division. Eine einfache ALU besteht aus 2 Eingangs-Registern, einem Ergebnis-Register, Steuerleitungen und so genannten Flags. Bei den Registern handelt es sich jeweils um sogenannte Schieberegister, welche den Zugriff auf einzelne Bits ermöglichen. Die Flags sind einzelne Bits die zum Beispiel angeben, ob ein Ergebnis positiv oder negativ ist, oder ob ein Ergebnis null ist usw. Nun sehen wir uns einmal den groben Aufbau einer einfachen Alu an.

A & B	: Eingangs-Register
R	: Ergebnis-Register
Steuerleitungen	: Bringen die Befehle über die auszuführende Operation in die ALU
N, Z	: Beispiel Flags, N ist 1 wenn das Ergebnis negativ ist und Z wenn das Ergebnis 0 ist.



Integer Arithmetik

Integer Repräsentation im Rechner: Es gibt einige verschiedene Möglichenkeiten Ganzzahlen (Integers) im Rechner darzustellen. Die einfachste Methode ist ausschließlich mit positiven Integers zu arbeiten, dann würden sich mit 8 bit die Zahlen
von 0 bis 255 darstellen lassen. Wie man schnell feststellt ist es sehr unbefriedigend nur mit positiven Zahlen arbeiten zu können, daher gibt es noch ein paar andere Methoden. Zum einen ist da die Sign-Magnitude Methode, bei der das höchstwertige Bit als Vorzeichenbit benutzt wird, ist es eins, ist die Zahl negativ, sonst positiv. Es würden sich so mit 8 Bit die Zahlen von –127 bis +127 darstellen lassen. Das Problem bei dieser Darstellungsform ist nur, das es zum einen 2 Darstellungen für die Null gibt ($1000_2 = 0000_2 = 0_{10}$) und zum anderen die Addition und die Subtraktion bereits sehr kompliziert wären. Dann gibt es noch die 1's Complement (Einer-Komplement) Methode, bei der aus einer positiven Zahl eine negative wird, wenn man die Zahl bitweise invertiert. Es würden sich so ebenfalls mit 8 Bit die Zahlen von –127 bis +127 darstellen lassen, nur das die Addition und Subtraktion schon wesentlich einfacher wären. Allerdings gäbe es weiterhin das Problem mit den 2 Darstellungen für die 0 ($0000_2 = 1111_2 = 0_{10}$). Die wohl beste und somit auch verbreiteste Darstellungsform ist wohl die 2's Complement (Zweier-Komplement) Methode, bei ihr wird aus einer positiven eine negative Zahl, in dem man sie zuerst bitweise invertiert und dann zu der entstandenen Zahl eine 1 addiert. Diese Abbildung funktioniert in beide Richtungen, das heisst man kann auch ohne Probleme aus einer negativen Zahl wieder die zugehörige positive Zahl berechnen. Der Vorteil bei dieses Methode ist das es nur eine Darstellung für die 0 gibt (0000_2) und das die Addition und Subtraktion sehr einfach zu realisieren sind. Es gibt nur eine kleine Unregelmäßigkeit bei dieser Methode die man beachten sollte, da man nur eine Darstellung für die Null hat, bleibt eine Zahl mehr übrig, welches die nächste negative Zahl ist, bei 8 Bit zum Beispiel –128 (10000000_2). Bildet man aus dieser das Zweier-Komplement, entsteht wieder die selbe Zahl. Es lassen sich also im Zweier-Komplement mit 8 Bit die Zahlen von –128 bis +127 darstellen.

Addition und Subtraktion:

Sign-Magnitude Methode: Wie bereits erwähnt ist die Addition bei dieses Methode bereits sehr aufwendig, daher will ich drauf nicht weiter eingehen, sondern nur anhand eines kleinen Beispiels zeigen das es nicht mit einem normalen Addierwerk funktioniert:

```
 0011 (+3)
+1011 (-3)
 1110 (-6)
```

Das 3 + (-3) nicht nicht (-6) sind erkennt wohl jeder, daher brauche ich darauf wohl auch nicht weiter einzugehen.

1's Complement: Hier ist die Addition bereits etwas einfacher, wie man an einem einfachen Beispiel sehen kann, funktioniert dort schon die Benutzung eines normalen Addierwerkes mit einer kleinen Unregelmäßigkeit, auf die ich gleich noch eingehen werde.

```
 0011 (+3)
+1100 (-3)
 1111 ( 0)
```

Dieses Ergebnis ist richtig. Die Unregelmäßigkeit kommt nur Zustande, wenn es bei der Addition einen Übertrag gibt, dann wird allerdings einfach zum entstandenen Ergebnis noch eine 1 Addiert. Dieses nennt man das „end-around carry“.

2’s Complement: Nun zur am einfachsten zu benutzenden Methode, das sogenannte Zweier-Komplement. Hierbei kann die Addition (bzw. auch Subtraktion) mit einem normalen Addierwerk durchgeführt werden. Eventuell entstehende Überträge fallen einfach weg. Man muß nur darauf achten, das wenn das Vorzeichen (höchstwertiges Bit) der Summe anders ist als das beider Summanden, ist es zu einem Überlauf gekommen, das Ergebnis liegt also außerhalb des Wertebereichs und ist somit nicht korreckt. Hier 2 Beispiele

```
 0010 (+2)
+1001 (-7)
 1011 (-5)
```

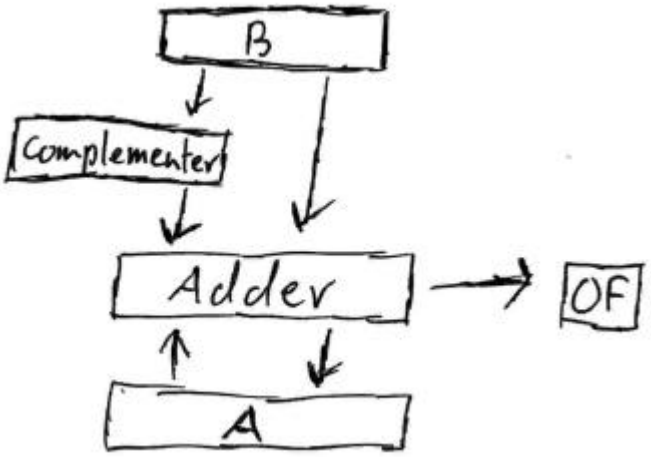
Das ist eindeutig richtig.

```
 0111 (+7)
+0111 (+7)
 1110 (-2)
```

Das Ergebnis wäre nicht richtig, aber da die Vorzeichen der Summanden anders sind als das der Summe, sieht man das es zu einem Überlauf gekommen ist.

Nun will ich noch einmal kurz demonstrieren, wie man sich so eine Addition bzw. Subtraktion nach dem Zweier-Komplement im Rechner vorstellen muß.

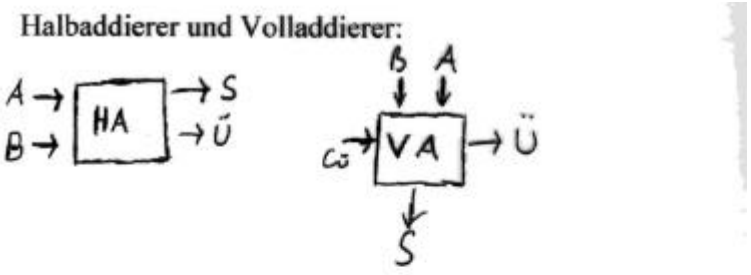
A, B : Sind die Register, das Ergebnis wird nach A zurückgeschrieben
Adder : Ein einfaches Addierwerk
Complementer : Bildet das Komplement
OF : Flag das gesetzt wird, wenn es zu einem Überlauf kommt.

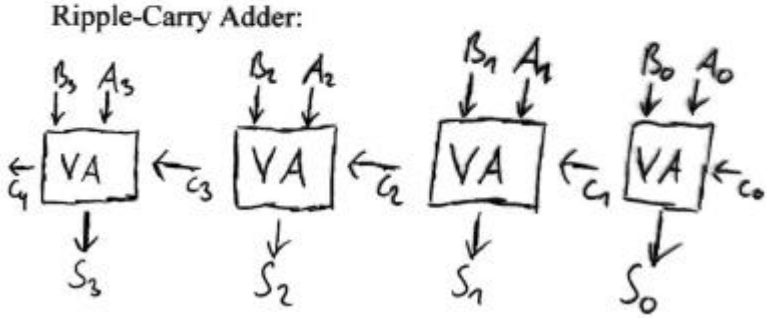


Bei der Addition wird der 2. Summand normal aus Register B vom Addierwerk gelesen, bei der Subtraktion wird erst das Komplement gebildet, womit man dann die Subtraktion auf eine Addition des Komplements zurückgeführt hat.

Addierer: Für diejenigen die noch nicht wissen wie so ein Addierer aufgebaut ist, zeige ich hier noch einmal kurz den Aufbau von Volladdieren, Halbaddieren und einem einfachen Addierwerk.

A,B : Sind immer die Eingangs Bits
S : Das Summen Bit
Ü : Der eventuell entstehende Übertrag der beiden Bits A und B





Der Ripple-Carry Adder ist ein sehr einfacher Addierer, der zwar einen relativ geringen Aufwand an Hardware fordert, aber dafür auch nicht sehr schnell ist. Zu einigen Alternativen komme ich daher später nocheinmal.

Unsigned Integer Multiplikation und Division: Jetzt möchte ich ersteinmal zeigen, wie die Multiplikation und die Division bei Integers ohne Vorzeichen verlaufen. Sie sind schon wesentlich aufwendiger als Addition und Subtraktion**,** was man leicht an den folgenden Algorithmen und Beispielen sehen kann.

Multiplikation

Algorithmus: Initalisiere P mit 0
Wiederhole n mal (n Länge der Faktoren):
(1) Wenn letztes Bit von A ist 1, dann addiere B zu P, sonst addiere 0 zu P
(2) Shift Right mit Carry-Out der Summe ins höchste Bit von P und das niedrigste Bit von P ins höchste Bit von A

Beispiel: (3 * 2)

P	A	B
0000	0011	0010
0010	0011	0010
0001	0001	0010
0011	0001	0010
0001	1000	0010
0001	1000	0010
0000	1100	0010
0000	1100	0010
0000	0110	0010

Nach n Durchläufen steht das Ergebnis jetzt in den Registern A und P, es lautet also: 00000110 (und das enstpricht 6).

Division

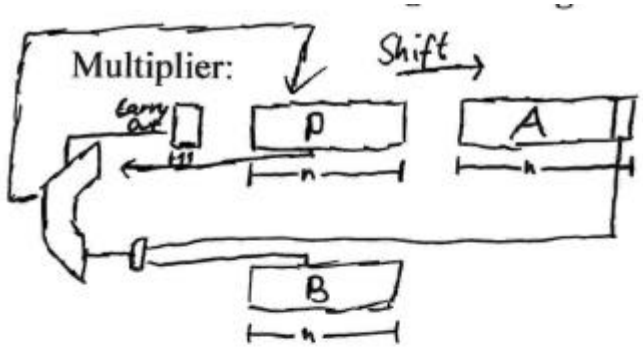
Algorithmus: Initalisiere P mit 0
Wiederhole n mal (n Länge der Faktoren)
(1) Shift Left mit dem höchsten Bit von A ins niedrigste Bit von P
(2) Subtrahiere B von P und schreibe das Ergebnis in P zurück
(3) Wenn das Resultat negativ ist, setze das niedrigste Bit von A gleich 0, sonst gleich 1
(4) Wenn das Resultat negativ ist, hole das alte Ergebnis von P zurück nach P; durch Addition von B

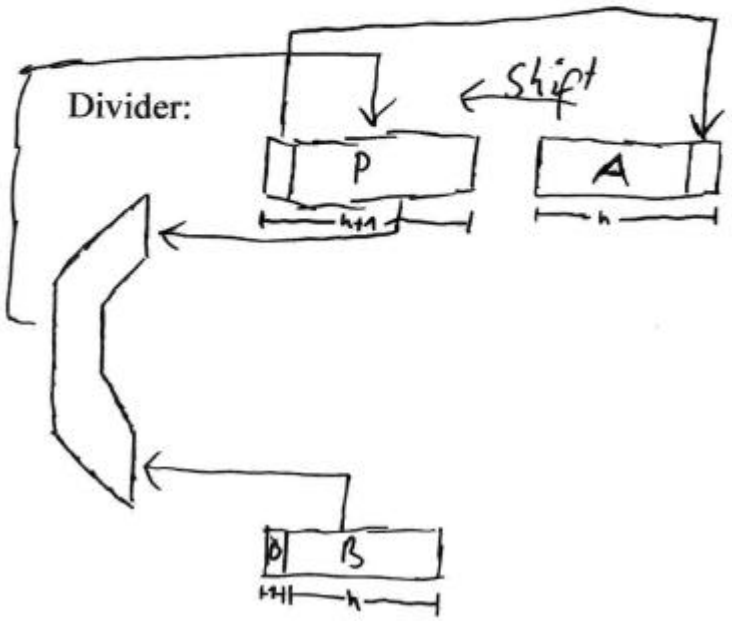
Beispiel: (14 : 3)

P	A	B
0000	1110	0011
0001	110_	
-0011		
-0010	1100	
0001		
0011	100_	
-0011		
0000	1001	
0000		
0001	001_	
-0011		
-0010	0010	
0001		
0010	010_	
-0011		
-0001	0100	
0010		

Das Ergebnis lautet: 0100 Rest 0010 (also 4 Rest 2), das wie man leicht nachrechnen kann völlig richtig ist.

Hier jetzt noch einmal die Unsigned Integer Multiplikation und Division als Hardware Diagramm dargestellt:





Multiplikation & Division bei Signed Integers im Zweier-Komplement: Da die Multiplikation und Division von Integers mit Vorzeichen nocheinmal um ein vielfaches komplizierter sind, zeige ich hier nur einmal ein Beispiel für einen Algorithmus für die Multiplikation, da diese wenigstens noch etwas weniger aufwendig ist als die Division.
Eine einfache Methode wäre zwar Zahlen erstmal in positive unsigned Integers umzuwandeln und dann später die Vorzeichen zu prüfen, nur wäre das sehr rechenintensiv und von daher uneffizient. Ich zeige jetzt erstmal den sogenannten BOOT51:

Booth's Algorithmus: Initalisiere A mit 0, Q_{-1} mit 0,
M mit dem Muliplikant und
Q mit dem Multiplikator
Wiederhole n mal:
(1) Wenn Q_0 gleich 1 und Q_{-1} gleich 0, dann subtrahiere M von A und schreibe das Ergebnis in A zurück
(2) Wenn Q_0 gleich 0 und Q_{-1} gleich 1 dann addiere M zu A und schreibe das Ergebnis in A zurück
(3) Shift Right: A -> Q -> Q_{-1}
$A_{n-1} = A_{n-2}$

Beispiel: (3 * 7)

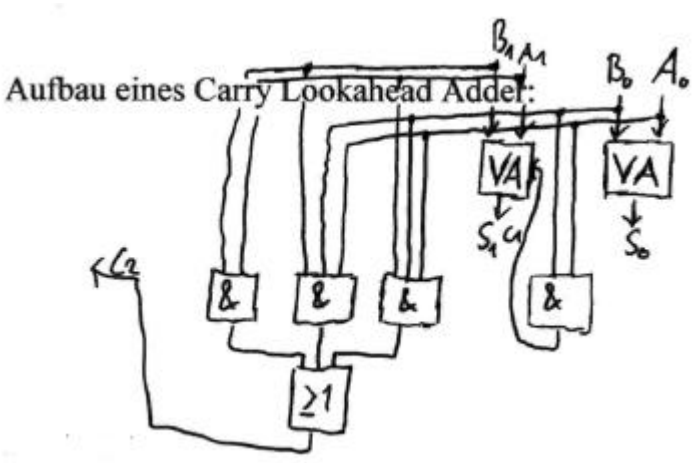
A	Q	Q_{-1}	M
0000	0011	0	0111
1001	0011	0	
1100	1001	1	
1110	0100	1	
0101	0100	1	
0010	1010	0	
0001	0101	0	

Das Ergebnis steht in den Registern A und Q und lautet daher: 00010101 (21), was wiederum richtig ist.

Speeding Up Interger Operations: Zum Schluß will ich noch einmal darauf eingehen, wie es möglich ist die Integer Arithmetik effizienter zu machen. Dies ist ein sehr wichtiger Punkt, da wir später im Abschnitt Floating Point Arithmetik noch sehen werden, das auch diese auf die Intergers zurückgeführt wird. Eine besonders wichtige Rolle bei der effizienteren Gestaltung spielt die Addition, da diese immer wieder eine wichtige Rolle spielt, bei allen Operationen. Daher nun einige Möglichkeiten zur effizienteren Gestaltung der Addition:

Eine Möglichkeit die Addition schneller zu machen ist es einen Carry Lookahead Adder anstelle eines Ripple-Carry Adder zu benutzen. Dieser bringt zwar einen größeren Aufwand an Hardware mit sich, ist dafür aber auch wesentlich schneller. Hier der Aufbau:

A, B: Eingangs Bits
S : Summen Bits
C : Übertrag



Es gibt noch andere Arten von Addierwerken, die jedes auf seine Art, Vorteile mit sich bringen. Es geht bei der Auswahl darum, einen schellen aber auch nicht zu teueren (Hardware Aufwand) Weg zu finden. Zwei weitere Arten von Addierwerken wären zum Beispiel:

Carry – Skip Adder : Ist eine Mischung aus dem Ripple-Carry Adder und dem Carry Lookahead Adder. Die Addition wird hierbei in einzelne Blöcke

zerlegt, wobei der Übertrag immer parallel zum jeweiligen Block berechnet wird.

Carry Select Adder : Die Addition wird in 2 Blöcke zerlegt, die parallel berechnet werden. Der 2. Block wird einmal für den Eingangs-Übertrag mit 1 und einmal mit 0 berechnet und am Ende des ersten Blockes wird bestimmt welches Ergebnis man benutzt. Dies hört sich etwas komplizierter an, aber wer interessiert ist, sollte sich geeignete Fachliteratur besorgen, um die genauen Vorgänge zu studieren.

Was lässt sich außerdem verbessern? Als letztes möchte ich noch erwähnen, das sich natürlich nicht nur die Addition verbessern lässt, sondern es auch wesentlich effizientere Algorithmen für Muliplikation, Division usw. gibt, die dann aber auch etwas komplizierter sind. Kein moderner Rechner arbeitet nach den hier beschriebenen Algorithmen, aber sie sind halt gut dafür geeignet, die Problematik zu beschreiben und ein leichtes Grundverständnis zu vermitteln.

Gleitkomma Arithmetik

Bedeutung der Gleitkommadarstellung

Festkommazahlen stoßen bei vielen Anwendungen, vor allem im mathematisch-naturwissenschaftlichen Bereich, an die Grenze ihrer vernünftigen Verwendbarkeit. Hier entsteht oft das Problem, daß große Zahlenbereiche überdeckt werden, jedoch eine Zahlengenauigkeit von einigen Dezimalstellen genügt. Handelsübliche Taschenrechner stellen Dezimalzahlen im Bereich von bis mit einer Genauigkeit von 8 oder 10 Dezimalstellen dar. Die dargestellten Dezimalzahlen setzen sich zusammen aus einer gebrochenen Mantisse und einem ganzzahligen Exponenten zur Basis 10, beide Größen sind multiplikativ verknüpft. Durch die Wahl der Basis 10 für den Exponenten, das ist gleichzeitig die Basis des Zahlensystems, entspricht eine Änderung des Exponenten um 1 einer Verschiebung des Kommas in der Mantisse um eine Stelle, wie das Beispiel zeigt.

Diese als **wissenschaftliche Schreibweise** bekannte Darstellung erlaubt eine höchst flexible Zahlenbehandlung bei recht geringem Berechnungs- und Anzeigeaufwand. Mit beliebig hohem Aufwand ist es natürlich prinzipiell möglich, den oben beschriebenen Wertebereich auch in Festkommadarstellung zu erreichen, in der Praxis wird jedoch das Zahlenformat schnell unhandlich groß, und zudem ist die Genauigkeit auf den niedrigen Stellen unnötig. Es ist deshalb sinnvoll, für die duale Zahlenverarbeitung ein Format ähnlich der wissenschaftlichen Schreibweise zu verwenden, das die Genauigkeit der Zahl und den zu überdeckenden Wertebereich frei wählbar macht. Man benutzt eine Zahlendarstellung, die neben der Ziffernfolge eine Größenordnungsangabe (Maßstabsfaktor) enthält, die sogenannte **Gleitkommadarstellung**. Die oben eingeführte wissenschaftliche Schreibweise ist der Sonderfall der Gleitkommadarstellung für Zahlen zur Basis 10.

Allgemein gilt: Jede Zahl *Z* eines beliebigen Zahlensystems kann in die folgende halblogarithmische Darstellung gebracht werden

M ist dabei die **Mantisse**, eine gebrochene Zahl kleiner Eins. Die ganze Zahl *B* ist die Basis des verwendeten Zahlensystems und *E* ist der **Exponent**.

Beispiel:

432,5 = 0,4325 * 10^3

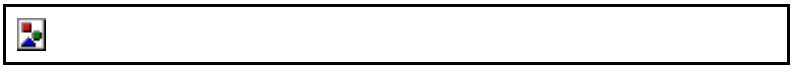
0,0004325 = 0,4325 * 10^-3
(^ : Parameter danach ist exponent)

Die Bezeichnung Gleitkommadarstellung resultiert aus dem variablen Zahlenbereich, den die Mantisse zusammen mit dem **Maßstabsfaktor** □ überstreichen kann. Für die Darstellung von Gleitkommazahlen im Rechner ist die Basis *B* durch das im Rechner verwendete Zahlensystem (Dualsystem) fest vorgegeben. Da die Basis bekannt ist, brauchen rechnerintern nur die Mantisse *M* und der Exponent *E* abgespeichert zu werden. Ein Wort ist für die Zahlendarstellung in zwei Teile zu unterteilen:

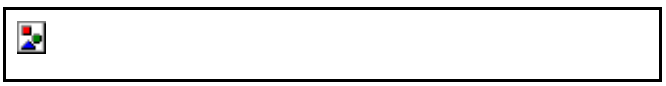
1. Mantissenbereich, der den Zahlenwert der Mantisse *M* enthält
2. Exponentenbereich, der den Zahlenwert des Exponenten *E* angibt

Das bedeutet, daß im Gleitkommaformat weniger Ziffern einer Zahl dargestellt werden können als bei der Festkommadarstellung mit gleicher Wortlänge. Dieser Nachteil wird jedoch in vielen Fällen durch die bessere Ausnutzung der Stellen (keine unnötigen führenden Nullen) wieder ausgeglichen.

Da auch negative Zahlen und Zahlen mit sehr kleinen Beträgen (□) dargestellt werden müssen, sind Mantisse und Exponent vorzeichenbehaftet. Sie können in der bereits bekannten Art als Zweierkomplementzahl dargestellt werden. Um beim Exponenten die Angabe eines Vorzeichens einzusparen, wird im allgemeinen eine sogenannte **Charakteristik** *C* eingeführt. Zum Exponenten *E* wird eine Konstante □ addiert, so daß die Summe aus Exponent und □ immer größer oder gleich Null wird. Diesen so modifizierten Exponenten nennt man die Charakteristik *C*.



Damit erhält man für die Gleitkommazahl *Z*



Beispiel: *Gleitkomma-Darstellung*

Reserviert man bei der Darstellung einer Gleitkomma-Dezimalzahl 2 Stellen für die Charakteristik *C*, so erhält man für *C* den Bereich:

Wird die Konstante gewählt, so ergibt sich für den Exponenten:

Damit ergeben sich folgende Zahlendarstellungen

10,2471 ***** 10^2 = 0,102471 ***** 10^4
Das entspricht einer Mantisse (M) von 0,102471 und einer Charakteristik ($C = E+K_0$) von 54.

0,000572 ***** 10^-1 = 0,572 ***** 10^-4
Das entspricht einer Mantisse von 0,572000 und einer Charakteristik von 46.
(^ : Parameter danach ist exponent)

Für die Darstellung von Gleitkommazahlen muß noch eine Vereinbarung getroffen werden, um die Zahlen auch eindeutig wiedergeben zu können. Eine Zahl läßt sich als Produkt ja auf verschiedene Weise schreiben:

z.B. .

Fordert man, daß die erste Ziffer nach dem Komma von Null verschieden und die Ziffern vor dem Komma identisch Null sein müssen, so ist nur noch die Darstellung möglich. Diese Darstellung führt zu einer optimalen Ausnutzung der für die Mantisse verfügbaren Stellen. Man bezeichnet dies als **normierte Darstellung** der Mantisse. Der Wertebereich der Mantisse in normierter Darstellung ist damit festgelegt. Es gilt:

Abweichend von dieser Normalisierungsfestlegung wird die Zahl 0 durch die Mantisse und beliebigen Exponenten dargestellt. Normalisieren bedeutet im Dualsystem, daß die 1. Stelle nach dem Komma (der Mantisse), die Binärstelle mit der Wertigkeit eine 1 enthält. Der Betrag der Mantisse $|M|$ ist also immer . Bei negativen Zahlen wird die Mantisse im *B*-Komplement dargestellt. Im Dualsystem enthält somit die 1. Binärstelle vor dem Komma bei positiven Zahlen eine 0 und bei negativen Zahlen eine 1. Die Mantisse stellt ein normiertes Festkommawort dar. Eine schematische Gleitkommaformatdarstellung hat die folgende Form:

Mantisse	Exponent
X,XXXX...	**XXXXX**
(m Stellen)	(e Stellen)

Beispiel: *Gleitkomma-Darstellung*

Darstellung im dualen Gleitkommaformat mit , mit .

a) positive Zahl

Damit ist der Exponent E bekannt

Darstellung im gegebenen Format:

b) negative Zahl

wegen *folgt die folgende*
Darstellung im gegebenen Format:

Die Verschiebung des Kommas um eine Stelle nach links oder rechts bedeutet eine Veränderung des Exponenten um 1. **Bei negativen Zahlen wird die Zahl zuerst normalisiert und dann ins 2er-Komplement gewandelt**.

Der wesentliche Vorteil der Gleitkommadarstellung gegenüber der Festkommadarstellung ist wie schon gesagt der viel größere darstellbare Zahlenbereich. Für ein allgemeines Gleitkommaformat mit *m* Stellen Mantisse und *e* Stellen Exponent gelten die folgenden charakteristischen Größen. Dabei wird angenommen, daß die Mantisse im 2er-Komplement dargestellt und normalisiert ist. Für die Konstante *K* wird angenommen, so daß der Zahlenbereich des Exponenten zur Hälfte in positive und negative Zahlen geteilt ist.

1. kleinste positive darstellbare Zahl (normiert)
2. größte positive darstellbare Zahl
3. betragsmäßig kleinste negative darstellbare Zahl
4. betragsmäßig größte negative darstellbare Zahl
   Ihr Betrag ist um größer als die größte positive Zahl
   (wesentliches Merkmal des 2er-Komplements)
5. größte Genauigkeit = Differenz zwischen der kleinsten und zweitkleinsten unnormierten darstellbaren Zahl

Dezimal-Dual-Wandlung im Gleitkommaformat

Die Wandlung von Dezimalzahlen in ein duales Gleitkommaformat ist etwas aufwendiger als bei Festkommazahlen. Sie kann auf verschiedene Arten durchgeführt werden. Im folgenden wird eine Methode gezeigt.

Wandlungsverfahren
Die Wandlung erfolgt durch Umweg über ein Festkommaformat. □für Festkommazahlen beschriebenen Regeln angewendet um die Zahl in ein Festkommaformat zu wandeln. Anschließend wird dieses Wandlungsergebnis durch Wahl eines geeigneten Exponenten in das vorgegebene Gleitkommaformat umgeformt. Dies erfordert eine Verschiebung des Kommas mit einer entsprechenden Anpassung des Exponenten.

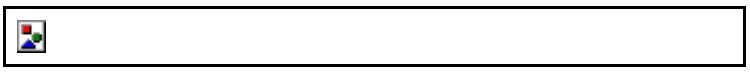
Beispiel: *Wandlungsverfahren*

Gesucht sei zur Zahl □*die duale Gleitkommadarstellung im Format*

mit der Konstanten □.
Es gilt: □
Eine Verschiebung des Kommas um 23 Stellen nach links bedeutet E=23. Damit ergibt sich für die Charakteristik C: □
Die Darstellung im Format ergibt somit: □.
Für den absoluten Wandlungsfehler F in diesem Format gilt:

□

Der relative Wandlungsfehler f ist:



Der Nachteil dieses Verfahrens ist, daß insbesondere bei betragsmäßig großen Exponenten die Übergangsdarstellung im Festkommaformat länglich und damit rechenaufwendig werden kann.

Arithmetik im Gleitkommaformat

Bei den Grundrechenarten in der Gleitkommadarstellung müssen beide Teile des Wortes (Mantisse, Charakteristik) getrennt verarbeitet werden, was die Operationsabläufe etwas aufwendiger macht. Da man sowohl die Mantisse als auch die Charakteristik für die getrennte Verarbeitung als Festkommazahlen betrachten kann, treten keine wesentlich neuen Überlegungen bei der Gleitkommaverarbeitung auf.

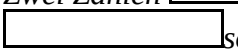
Addition und Subtraktion

Es ist klar, daß nur die Mantissen zweier Zahlen mit gleicher Charakteristik addiert bzw. subtrahiert werden können. Zwei Zahlen im Gleitkommaformat werden *addiert* (voneinander *subtrahiert*), indem die folgenden Operationen ausgeführt werden:

1. Durch Linksverschiebung des Kommas wird der Operand mit der kleineren Charakteristik so umgeformt, daß beide Operanden die gleichen (größeren)

Charakteristiken haben. Der umgeformte Operand ist dann natürlich nicht mehr normalisiert.
2. Die Operation wird durchgeführt, indem die Charakteristik erhalten bleibt und die beiden Mantissen addiert werden (Addition) bzw. das Zweierkomplement des zweiten Operanden addiert wird (Subtraktion).
3. Das Ergebnis wird durch Verschiebung des Kommas, also durch Wahl der geeigneten Charakteristik normalisiert und in das gegebene Format gebracht. Dabei können Abweichungen vom richtigen Ergebnis durch die Einschränkungen des Ergebnisformats auftreten.

Beispiel: *Addition und Subtraktion im Gleitkommaformat*

Zwei Zahlen  *und*  *im Format*  *sollen addiert bzw. subtrahiert werden.*

Addition 

Ergebnis im Format und normalisiert: 

Subtraktion 

Ergebnis im Format und normalisiert: 

Multiplikation und Division

Zwei Zahlen im normalisierten Gleitkommaformat werden miteinander *multipliziert* (durcheinander *dividiert*), indem die folgenden Operationen ausgeführt werden:

1. Die Mantissen der Operanden werden miteinander multipliziert (durcheinander dividiert). Bei negativen Mantissen ist es sinnvoll, die Beträge zu multiplizieren (zu dividieren) und eine getrennte Vorzeichenbetrachtung durchzuführen.
2. Die Exponenten der Operanden werden addiert (bei Multiplikation) bzw. das Zweierkomplement des zweiten Operanden zum ersten addiert (bei Division). Da die Zahlendarstellung mit Charakteristik statt Exponent verwendet wird, sind entsprechend die Charakteristiken zu addieren (zu subtrahieren). Dabei wird jedoch

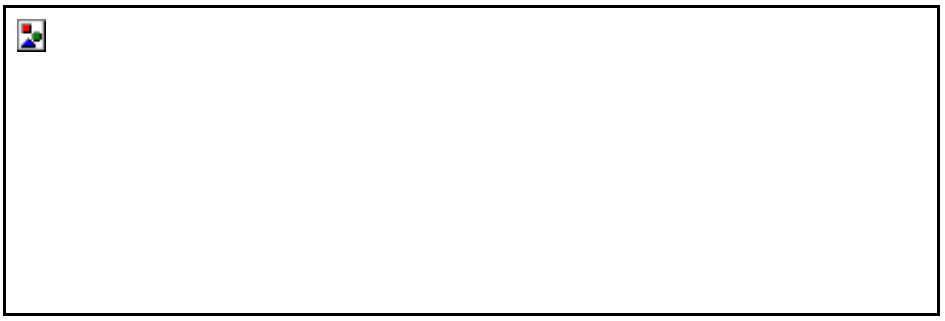
die Konstante zuviel addiert (subtrahiert), was anschließend wieder rückgängig gemacht werden muß.

3. Das Ergebnis wird durch Verschiebung des Kommas, also durch Wahl der geeigneten Charakteristik *C*, normalisiert und in das gegebene Format gebracht. Dabei können Fehler (Abweichungen vom richtigen Ergebnis) durch Einschränkungen des Ergebnisformats auftreten.

Beispiel: *Multiplikation und Division im Gleitkommaformat*

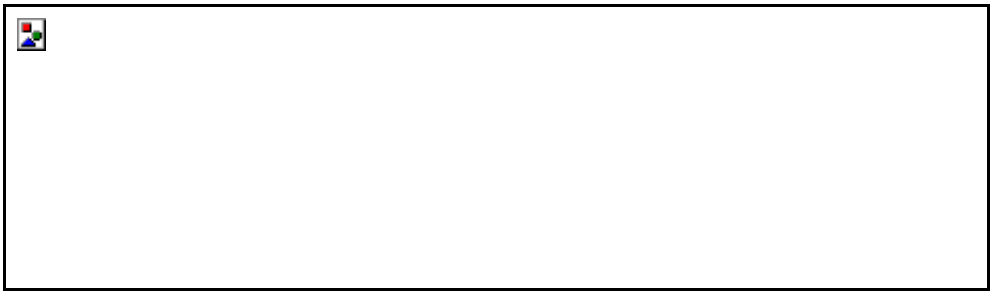
Zwei Zahlen  *und*  *im Format* *sollen multipliziert bzw. durcheinander dividiert werden.*

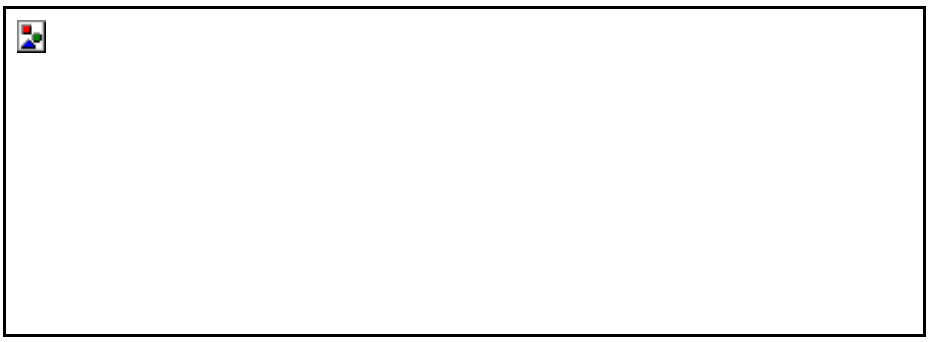
Multiplikation



Ergebnis im Format und normalisiert:

Division





Ergebnis im Format und normalisiert:

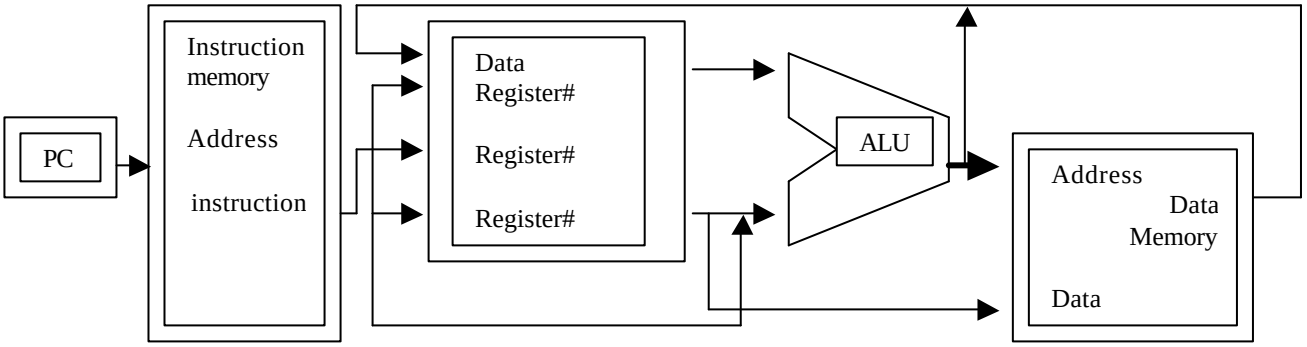
Datapath

Überblick über die Implementation

Um verschiedene Instruktionen, wie MIPS instructions, interger arithmatic-logical instruction und memory-reference instructions zu implementieren muss man für jede dieser Instruktionen

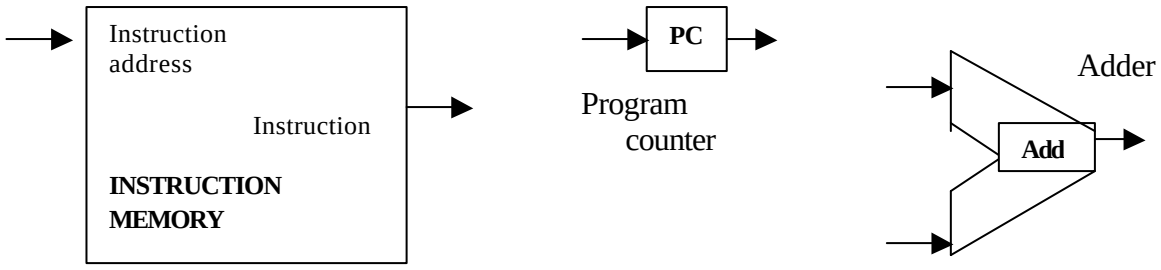
1. Programmbefehl zum Speicher schicken, er muss den Code und die Speicheradresse enthalten
2. Register lesen, meistens zwei, nur bei load word instruction muss ein Register gelesen werden.

Die Beendung der Instruktionen hängt von der Instruktionsklasse (memory-reference, arithmatic- logical, branches) ab.
Die Instruktionsklassen haben Ähnlichkeiten denn alle benutzen die ALU.
Die memory-reference instructions benutzt die ALU für Adressberechnung und die arithmatic-logical Instruktion für Operations Ausführung.
Dannach greift die memory-reference Instruktion auf den Speicher zu um zu speichern oder zu lesen. Die arithmatic-logical Instruktion schreibt Daten von der Alu in die Register.

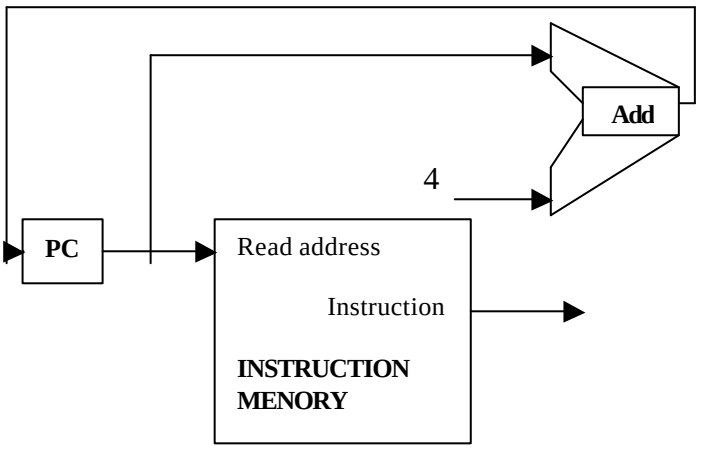


Datenpfad aufbauen

Das erste Element, das man braucht ist ein Platz wo man Programm-Instruktionen speichern kann. Diese Speichereinheit ist ein Zustandselement und wird gebraucht um die Instruktionen zu liefern.
Die Adresse von Instruktionen wird in den program counter (PC) gespeichert. Es ist auch eine state-machine.
Der Addierer (adder),der ist aus einer ALU gebaut. Er inkrementiert den PC zur der Adresse der nächsten Instruktion.



Um eine Instruktion auszuführen, muss man zuerst die Instruktion aus den Instruction-memory holen. Um die nächste Instruktion vorzubereiten, muss man den PC incrementieren,so dass er auf die nächste Instruktion zeigt.

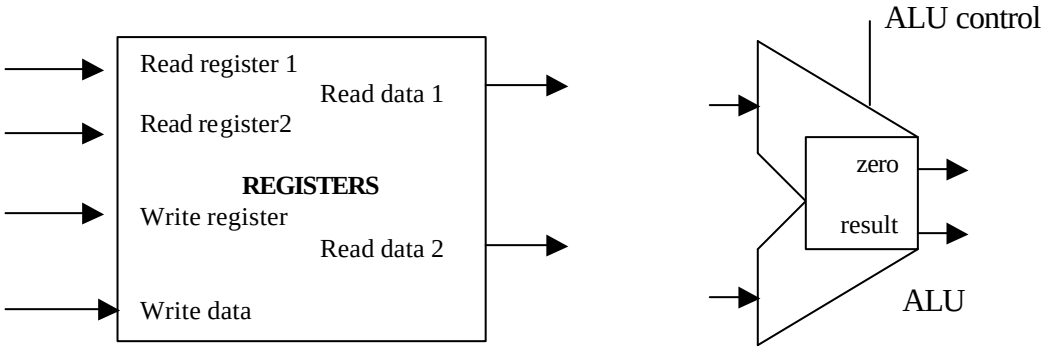


R-format instructions

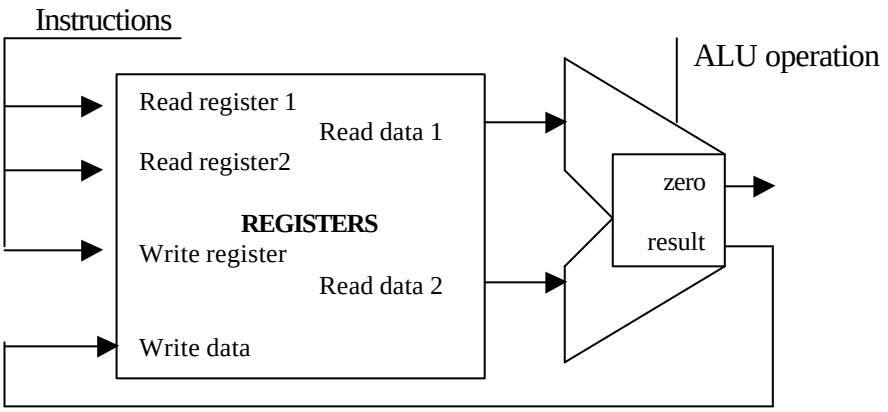
Das ist der Name von einem arithmetischen instuctions format.
Diese Instruktionen lesen 2 Register, führen eine ALU Operation aus und schreiben das Ergibniss. Diese Arithmetic-logic instruction class enthält add, sub, slt and, or.
Die 32 Prozessor register sind in einer Struktur gespeichert, genannt register file. Register file ist einen Sammlung von Registern inder geschrieben und gelesen werden kann. Man braucht eine ALU um die Werte der Register zu lesen. Da R-format instructions 3 Register Operanden haben, braucht man zwei read register um zwei Wörter lesen zu können und einen write Register.
Um ein data-word zu schreiben braucht man noch ein zweiten Register indem die Daten gespeichert werden.
Die write-register werden von den write-control signal gesteuert.
Insgesamt braucht man 4 Inputs (3 für Register Nummer und ein für Daten) und zwei Outputs(beides data Ausgänge).

Die register number Eingänge sind 5 bit weit.

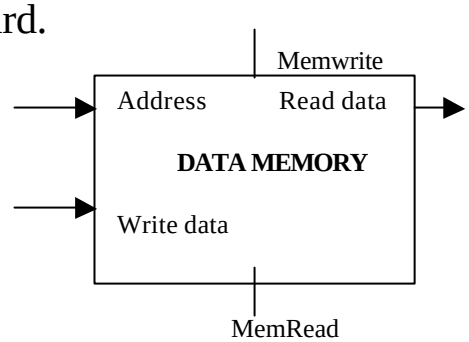
ALU
Die ALU wird gesteuert von einem 3 Bit Signal. Sie nimmt 2*32 Bit an und gibt ein 32 Bit Ergibniss aus.

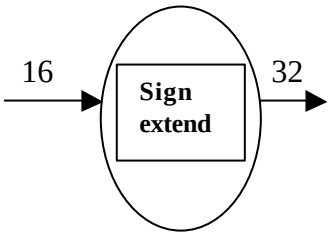


Datapath für R-type instructions



Diese Einheit braucht man um die Befehle laden, speichern zu Implementieren .
Die Data Memory Unit ist eine Zustandsmaschine mit Eingängen für die Adresse und Daten. Und ein Ausgang für das Ergibiss lesen. Es gibt unabhängige control Eingänge für das Schreiben und Lesen. Es kann aber nur ein Signal bei einem clock bearbeitet werden. Die Sign-extension Unit hat einen16 Bit Eingang der in ein 32 Bit gewandelt wird.

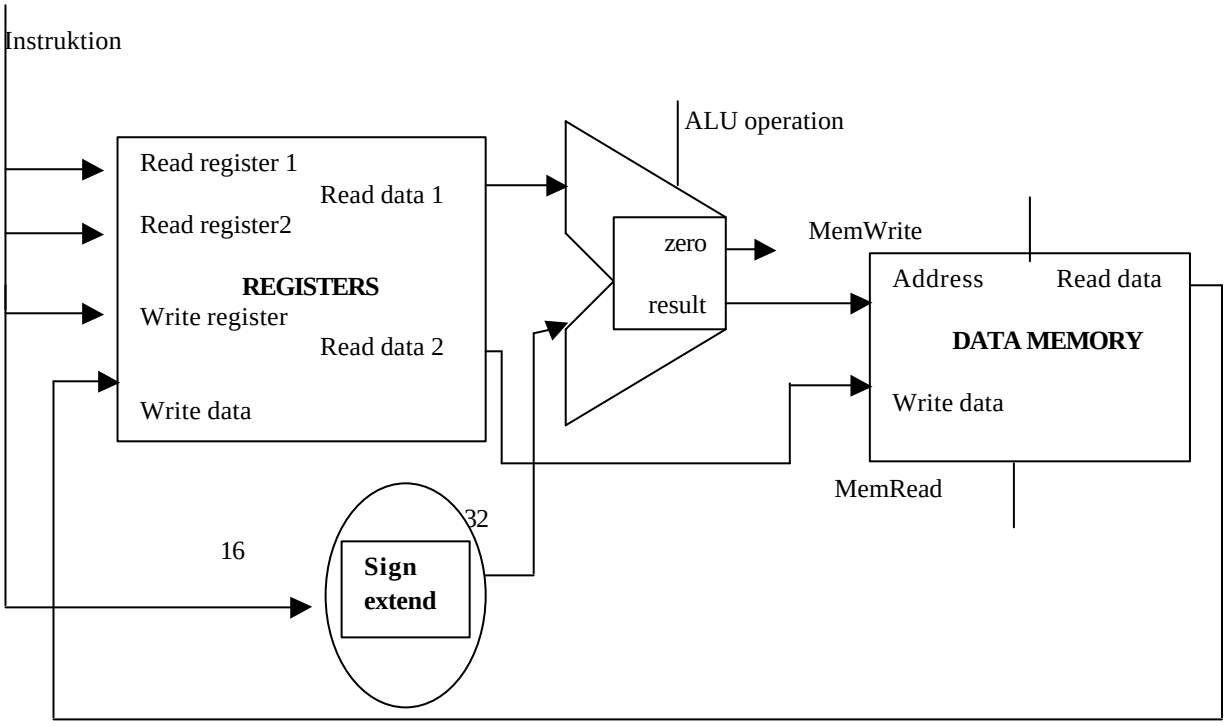




Sign-extension unit

Datapath bestehend aus vorherigen Elementen.
Angenommen, dass die Instruktionen schon geholt sind

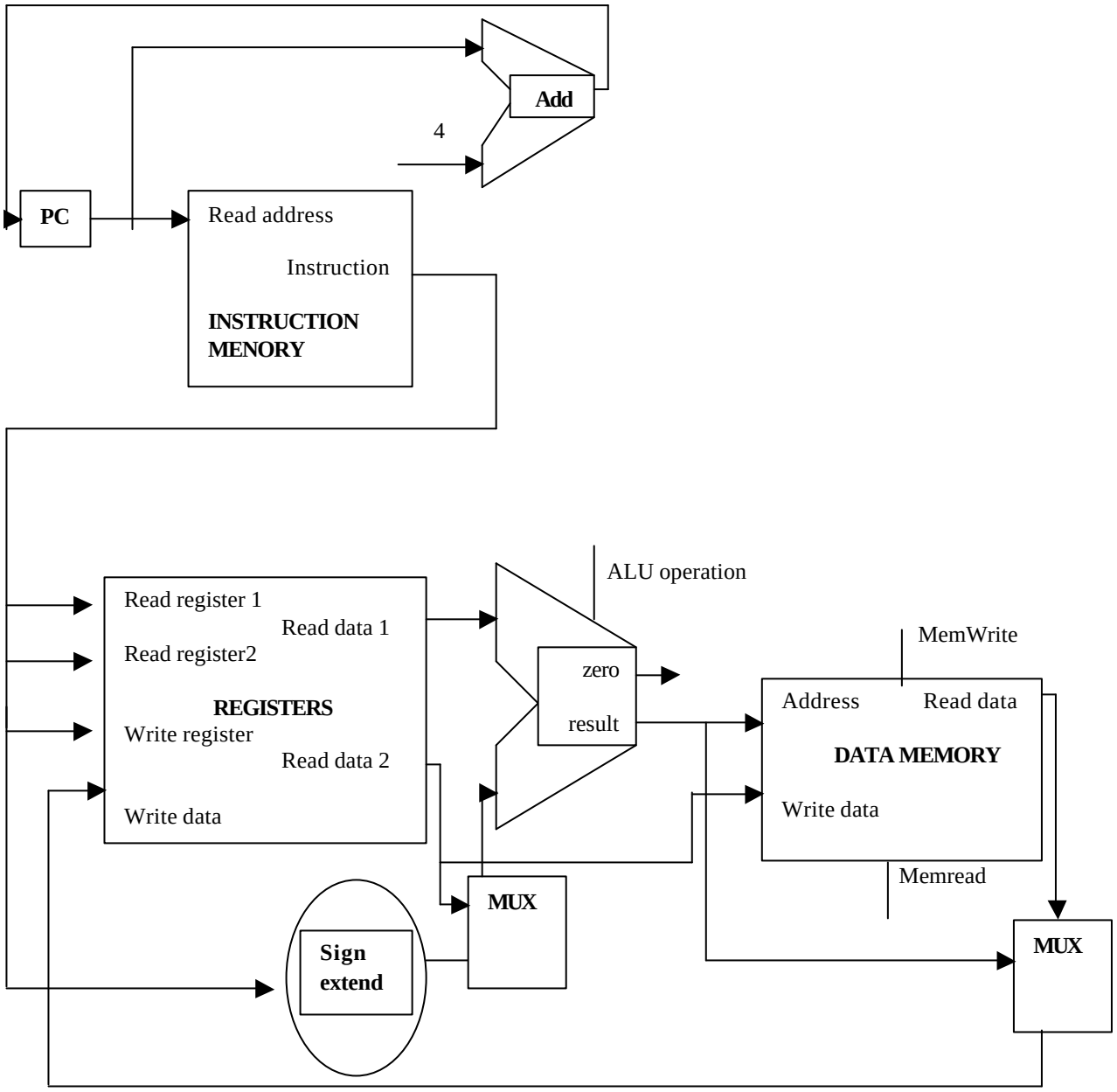
?? die register-number-inputs kommen von den geholten instructions
?? der zweite Eingang bekommt den Wert, der von der Sign-extension-unit erweitert wurde
?? dann berechnet die ALU die Speicheradresse
?? es wird entweder im Speicher geschrieben oder gelesen
?? schlisslich werden die Daten ins Register geladen wenn die nächste Instruktion geladen wird.



Datapath Implementation

Dieser Datapath führt alle Instruktionen in einem clock cyclus aus, d.h. die Datapath Resourcen können nicht mehr als einmal pro Instruktion gebraucht werden. Wenn ein Element zweimal benutzt werden muss, dann muss er verdoppelt werden.
Neu hinzugekommenist der Multiplexor, es ist ein Bauteil,dass dem Ausgang abhängig von den Controlsignal wählt.

PC und instruction-memory:Bei diesem Teil wird die Instruktion geholt und der PC (program counter) incrementiert. Die add-ALU incrementiert den PC während die Haupt-ALU die instruction ausführt.



Mikroprozessoren:

The Processor: Datapath and Control

A Multicycle Implementation

Das Ziel einer Multizyklus Implementierung eines Datenpfads ist es durch die Zerlegung der Befehle in einzelne Schritte und die nacheinander Ausführung dieser Schritte Hardware einzusparen, da die funktionellen Einheiten (Speicher; Register, ALU) während eines Befehls mehrfach genutzt werden können sofern sie in unterschiedlichen Zyklen liegen, und zu erreichen, dass Befehle unterschiedlich lange Dauern können (bei der single cycle implementation dauern alle Befehle so lange wie alle Schritte des längsten Befehls zusammen). Dadurch benötigt man im Unterschied zur single cycle implementation nur eine Speichereinheit und nur eine ALU und es werden einige Register hinzugefügt, um Zwischenergebnisse aufzunehmen, die während eines späteren Zyklus benötigt werden. Die Register, die hierfür benötigt werden sind: das Instruction register (IR) und das Memory data register (MDR), welche die Ausgabe des Speichers zwischenspeichern (Befehl bzw. Daten), die Register A und B, die die Ausgabe des Hauptregisters zwischenspeichern und das ALUOut register, das die Ausgabe der ALU zwischenspeichert. Außer dem Instruction register, das seinen Inhalt bis zum Ende des Befehls halten muss, brauchen diese Register ihren Inhalt immer nur bis zum nächsten Zyklus zu speichern.

Da nun eine ALU alle Daten erhalten muss, die bei einer single cycle implementation an mehrere gingen, müssen nun zusätzlich für den ersten ALU Eingang ein Multiplexor, der zwischen den Ausgaben des Registers A und des Program Counters (PC) wählt, und für den zweiten ALU Eingang ein Multiplexor, der zwischen der Ausgabe des Registers B, der Konstante 4, dem sign-extended Feld und dem shifted offset Feld wählt.

Durch die oben genannten Änderungen im Vergleich zur single cycle implementation werden ein Speicher und zwei Addierer gespart und durch kleinere Teile (Register, Multiplexor) ersetzt, was insgesamt zu einer Verringerung der Hardwarekosten führt.

Da dieser Datenpfad mehrere Zyklen zum Ausführen eines Befehls benötigt, muss nun noch eine Kotrolleinheit hinzugefügt werden, die bestimmt wann welches Register mit welchem Wert beschrieben werden soll.

Kontrollsignale:

Actions of the 1-bit control signals

Signal name	***Effect when deasserted***	***Effect when asserted***
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None	Contents of memory at the location specified by the Address input is put on Memory data

		output.
MemWrite	None	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None	The output of the memory is written into the IR.
PCWrite	None	The PC is written; the source is controlled by PCSource.
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active.

Actions of 2-bit control signals

Signal name	***Value***	***Effect***
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25-0] shifted left 2 bits and concatenated (verknüpft) with PC + 4[31-28]) is sent to the PC for writing.

Zerlegung der Befehle in Zyklen

Die Befehle sollen nun so zerlegt werden, dass die Operationen, die verrichtet werden, möglichst gleichmäßig über die Zyklen verteilt werden, wodurch dann auch der Zyklus kurz gehalten werden kann. So soll ein Zyklus z. B. höchstens eine ALU Operation oder einen Speicherzugriff oder einen Hauptregisterzugriff enthalten, wodurch der Zyklus nur so lang wie die längste dieser Operationen wird und gleichzeitig festgelegt wird was in einen Zyklus passt. Hierbei geschehen alle Operationen während eines Zyklus gleichzeitig ausgeführt und die Zyklen zur Ausführung eines Befehls nacheinander.

Die ersten beiden Schritte sind für alle Befehle gleich. Da noch nicht bekannt ist um welchen Befehl es sich handelt, können nur Operationen ausgeführt werden, die bei allen Befehlen identisch sind oder die, wenn sie nicht von dem entsprechenden Befehl benötigt werden, einfach ignoriert werden können. Der Vorteil diese Operationen früh auszuführen liegt darin, dass dann die Ergebnisse schon bereit liegen, falls sie gebraucht werden sollten. Die Schritte zur Ausführung eines Befehls sind:

1. Instruction fetch step: IR = Memory[PC];

PC = PC + 4

Der Befehl wird aus dem Speicher gelesen und in das Instruction register geschrieben, der Programm Counter wird erhöht.

2. Instruction decode and register fetch step: A = Reg[IR[25-21]];

B = Reg[IR[20-16]];

ALUOut = PC + (sign-extend (IR[15-10]) << 2);

Das Register wird gelesen und die Werte werden in A und B gespeichert, die Zieladresse des Branch Befehls wird berechnet (Ergebnis kann ignoriert werden, falls es kein Branch Befehl ist) und in den ALUOut gespeichert, wo sie im nächsten Zyklus abgerufen werden können, falls dies nötig wird.

3. Execution, memory address computation, or branch completion: Dies ist der erste Schritt bei

dem die Operationen von der Art des Befehls abhängen:

Memory reference: ALUOut = A + sign-extend (IR[15-0]);

Die ALU berechnet die Adresse für einen Speicherzugriff aus A und der Ausgabe der sign extension unit und speichert das Ergebnis im ALUOut.

Arithmetic-logical instruction (R-type): ALUOut = A op B;

Die ALU führt die arithmetisch-logische Operation, die im Befehl angegeben ist, mit den Werten aus A und B (aus Schritt 2.) aus und speichert das Ergebnis im ALUOut.

Branch: if (A == B) PC = ALUOut;

Die ALU überprüft A und B auf Gleichheit (falls A und B gleich sind wird die Branchzieladresse aus Schritt 2. die Adresse für den nächsten Befehl).

Jump: PC =PC [31-28] || (IR[25-0]<<2)

Der PC wird mit der Jump Adresse ersetzt.

4. Memory access or R-type instruction completion step:

Memory reference: Laden: MDR = Memory [ALUOut];

oder

Speichern: MDR [ALUOut] = B;

Bei der Operation Laden werden Daten aus dem Speicher in das Memory data register geschrieben. Bei der Operation Speichern werden Daten in den Speicher geschrieben. In beiden Fällen wird die Adresse aus Schritt 3. benutzt.

Arithmetic-logical instruction: Reg[IR[15-11]] = ALUOut;

Das Ergebnis aus dem 3. Schritt wird ins Register geschrieben.

5. Memory read completion step: Reg[IR[20-16]] = MDR;

   Daten, die im 4. Schritt in das Memory data register geschrieben wurden, werden in das Hauptregister geschrieben.

Schritte zur Ausführung eines Befehls:

Step name	***Action for R-Type instructions***	***Action for memory reference instructions***	***Action for branches***	***Action for jumps***
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/ register fetch	A = Reg [IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut = A op B	ALUOut = A + sign-extended(IR[15-0])	if (A == B) then PC = ALUOut	PC = PC [31-28] || (IR [25-0]<<2)
Memory access or R-type completion	Reg[IR[15-11]] =ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Definieren der Kontrolleinheit

Die Kontrolleinheit für den Multicycle Datapath muss die Operationen eines Schrittes festlegen und anzeigen, welcher Schritt dann folgt. Es gibt zwei Methoden, um dies zu erreichen:

Die erste Methode ist die Finite state machine. Diese besteht aus einer Anzahl von Zuständen und Richtungen für Zustandsänderungen (grafische Darstellung). Die Richtung für eine Zustandsänderung wird aus dem gegenwärtigen Status und durch Eingaben, die folgen müssen um zu einem anderen Status zu gelangen, gegeben. Die ersten beiden Schritte der Finite state machine sind für alle Befehle gleich,danach hängen die Schritte vom auszuführenden Befehl ab. Nach dem letzten Schritt eines Befehls beginnt die Finite state machine wieder von vorne. Der Prozeß je nach Befehl einen anderen Status auszuwählen nennt sich decoding.

Die zweite Methode ist die des Microprogramming. Da die Finite state machine für den kompletten Befehlssatz des MIPS zu komplex und zu fehleranfällig wird, wird ein kleines Programm mit sogenannten microinstructions geschrieben, das festlegt, welche Kontrollsignale auf welche Weise geschaltet werden und welche microinstruction als nächstes ausgeführt wird. Das Mikroprogramm wird möglichst einfach gehalten, um es besser schreiben und lesen zu können und um zu verhindern, dass fehlerhafte microinstructions entstehen. Eine microinstruction besteht aus sieben Feldern, von

denen die ersten sechs die Kontrollfelder sind und das siebte angibt welche microinstruction als nächstes ausgeführt wird. Um die nächste microinstruction auszuwählen gibt es drei Möglichkeiten: 1. die Ausführung als Sequenz, es folgen die nächsten microinstructions des Mikroprogramms der Reihenfolge nach (häufig Standardeinstellung); 2. zu der microinstruction gehen, die beginnt den nächsten Befehl auszuführen (Fetch); 3. die nächste microinstruction wird anhand der Eingaben in die Kontrolleinheit und einer Tabelle (dispatch table), die die dazugehörigen folgenden microinstructions enthält, ausgewählt. Ein Mikroprogramm kann als eine Textrepräsentation einer Finite state machine angesehen werden.

Die sieben Felder einer microinstruction:

Field name	***Function of Field***
ALU Control	Specify the operation being done by the ALU during this clock; the result is always written in ALUOut.
SRC1	Specify the source for the first ALU operand.
SRC2	Specify the source for the second ALU operand.
Register control	Specify read or write for the register file, and the source of the value for a write.
Memory	Specify read or write, and the source for the memory. For a read, specify the destination register.
PCWrite control	Specify the writing of the PC.
Sequencing	Specify how to choose the next microinstruction to be executed.

Werte, die ein Feld einer microinstruction annehmen kann:

Field name	***Values for field***	***Function of field with specific value***
Label	Any string	Used to specify labels to control microcode sequencing. Labels that end in a 1 or 2 are used for dispatching with a jump table that is indexed based on the opcode. Other labels are used as direct targets in the microinstruction sequencing. Labels do not generate control signals directly but are used to define the contents of dispatch tables and generate control for the Sequencing field.
ALU control	Add	Cause the ALU to add.
	Subt	Cause the ALU to subtract; this implements the compare for branches.
	Func code	Use the instruction´s funct field to determine the ALU control.
SRC1	PC	Use the PC as the first ALU input.
	A	Register A as the first ALU input.
SRC2	B	Register B as the second ALU input.
	4	Use 4 for the second ALU input.
	Extend	Use output of the sign extension unit as the second ALU input.
	Extshft	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read	Read two registers using the rs and rt fields of the IR as the register numbers, putting the data into registers A and B.
	Write ALU	Write the register file using the rd field of the IR as the register number and the contents of ALUOut as the data.
	Write MDR	Write the register file using the rt field of the IR as the register number and the contents of the MDR as the data.

Memory control	Read PC	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	Write memory using the ALUOut as address; contents of B as the data.
PCWrite control	ALU	Write the output of the ALU int the PC.
	ALUOut-cond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	Jump address	Write the PC with the jump address from the instruction.
Sequencing	Seq	Choose the next microinstruction sequentially.
	Fetch	Go to the first microinstruction to begin a new instruction.
	Dispatch i	Dispatch using the ROM specified by i (1 or 2).

Das Mikroprogramm für die Kontrolleinheit:

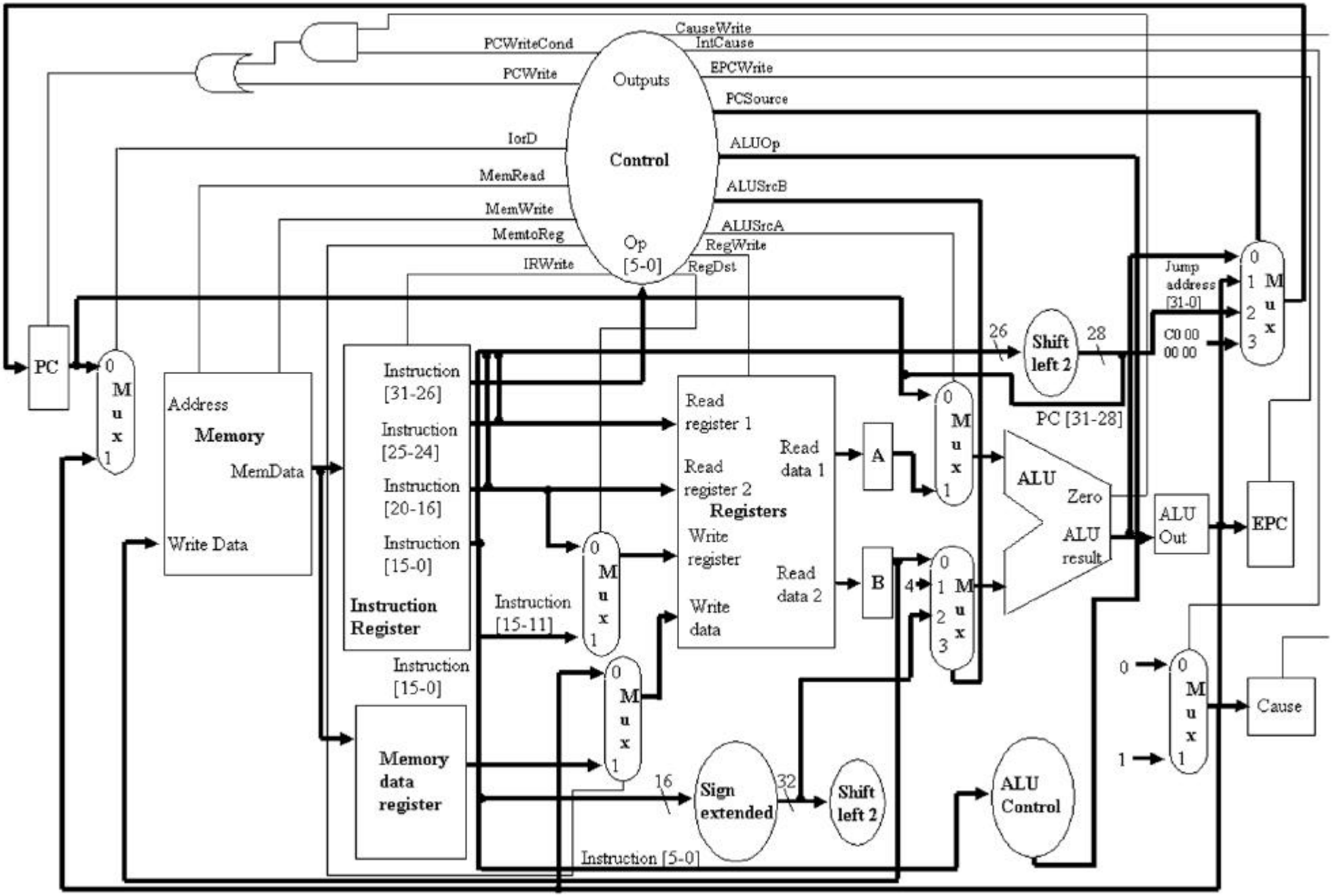
Label	***ALU Control***	***SRC1***	***SRC2***	***Register control***	***Memory***	***PCWrite control***	***Sequencing***
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

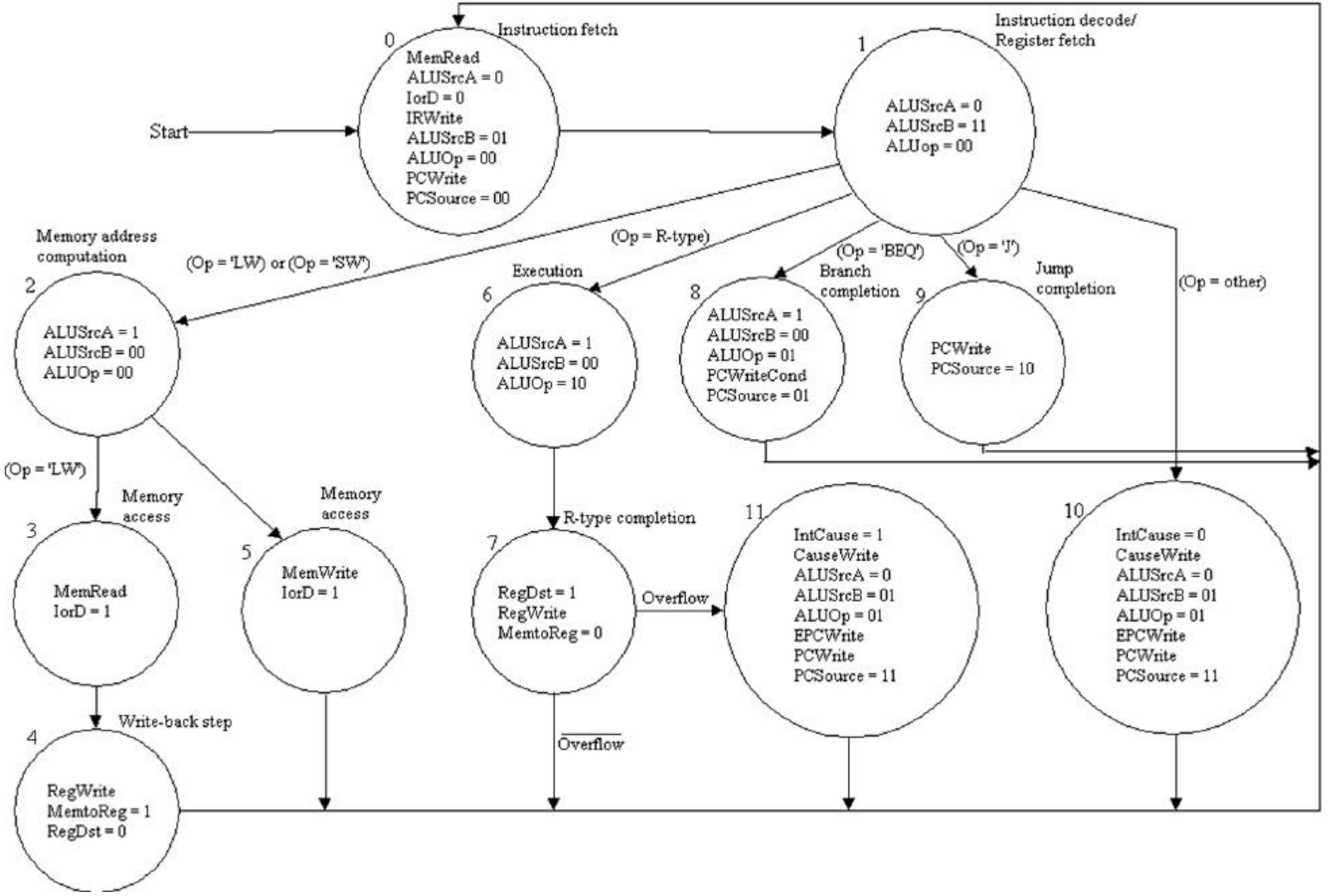
Label wird dazu benutzt eine microinstruction als Ziel einer anderen microinstruction anzugeben (Name einer microinstruction).

Die Schwierigkeit beim erstellen einer Kontrolleinheit ist es die Kontrolle klein und schnell zu halten und trotzdem alle Möglichkeiten zu berücksichtigen.

Ausnahmefehler/Exception

Eine Exception ist irgendeine unerwartete Veränderung in der Abarbeitung der Kontrolleinheit. Wenn dieser Fall eintritt, was passieren kann, wenn für einen Status kein Folgestatus definiert ist oder wenn ein arithmetischer Überfluss (Arithmetic Overflow) bei der ALU auftritt,wird die Adresse, die den Fehler ausgelöst hat im Exception program counter (EPC) gespeichert und dann die Kontrolle an das Betriebsystem übergeben. Dieses führt dann für diesen Fall vordefinierte Operationen durch und danach wird das Programm entweder beendet oder an der vorher gespeicherten Stelle fortgeführt.


CauseWrite
PCWriteCond
IntCause
PCWrite
Outputs
EPCWrite
PCSource
IorD
Control
ALUOp
MemRead
ALUSrcB
MemWrite
ALUSrcA
MemtoReg
Op
[5-0]
RegWrite
IRWrite
RegDst
Jump
address
[31-0]
26
Shift
left 2
28
C0 00
00 00
PC
PC [31-28]
Address
Memory
MemData
Write Data
Instruction
[31-26]
Instruction
[25-24]
Instruction
[20-16]
Instruction
[15-0]
Instruction
Register
Instruction
[15-11]
Instruction
[15-0]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALU
Out
EPC
Memory
data
register
16
Sign
extended
32
Shift
left 2
ALU
Control
Instruction [5-0]
0
1
Cause
M
u
x


Start
0
Instruction fetch
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
Instruction decode/
Register fetch
ALUSrcA = 0
ALUSrcB = 11
ALUop = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
2
Memory address
computation
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00
6
Execution
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
Branch
completion
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
Jump
completion
PCWrite
PCSource = 10
(Op = 'LW')
3
Memory
access
MemRead
IorD = 1
5
Memory
access
MemWrite
IorD = 1
7
R-type completion
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
4
Write-back step
RegWrite
MemtoReg = 1
RegDst = 0
Overflow

Performanceverbesserung durch Pipelining

am Beispiel des MIPS-Prozessors

Wozu Pipelining?

Pipelining beschleunigt den Prozessor durch Erhöhung des Befehlsdurchsatzes.
Dies wird durch die Parallelausführung mehrerer Instruktionen erreicht.

Pipelined Datapath

Der Data path (siehe Rückseite) wird in fünf Phasen zerlegt:

1. IF = Instruction fetch: Ein Befehl wird aus dem Speicher geladen.
2. ID = Instruction decode: Der Befehl wird interpretiert: Die Kontrollsignale werden erzeugt und die Register gelesen.
3. EX = Execution: Die ALU Operation wird ausgeführt
4. MEM = Memory access: Speicherzugriff
5. WB = Write Back: Ergebnis wird gegebenenfalls in das Register geschrieben.

Zwischen den Phasen müssen Pipeline Register eingefügt werden, damit die Daten jeweils weiter gegeben werden können.

Im Pipelining können Probleme auftreten, wenn Befehle von vorherigen Ergebnissen abhängig sind. Man unterscheidet Datenkonflikte und Sprungkonflikte.

Datenkonflikte

Das Ergebnis einer Berechnung wird von einem darauffolgenden Befehl benötigt, bevor es in ein Register gespeichert worden ist. Durch Forwarding (Daten werden aus einem Pipeline Register direkt zur ALU umgeleitet) lässt sich dieses Problem meistens lösen.
Wird das Ergebnis eines Load-Befehls gebraucht, lässt sich dieses nicht gleich forwarden. Die Pipeline muss in die Ex-Phase ein NOP (No OPeration) eingefügt werden.

Sprungkonflikte

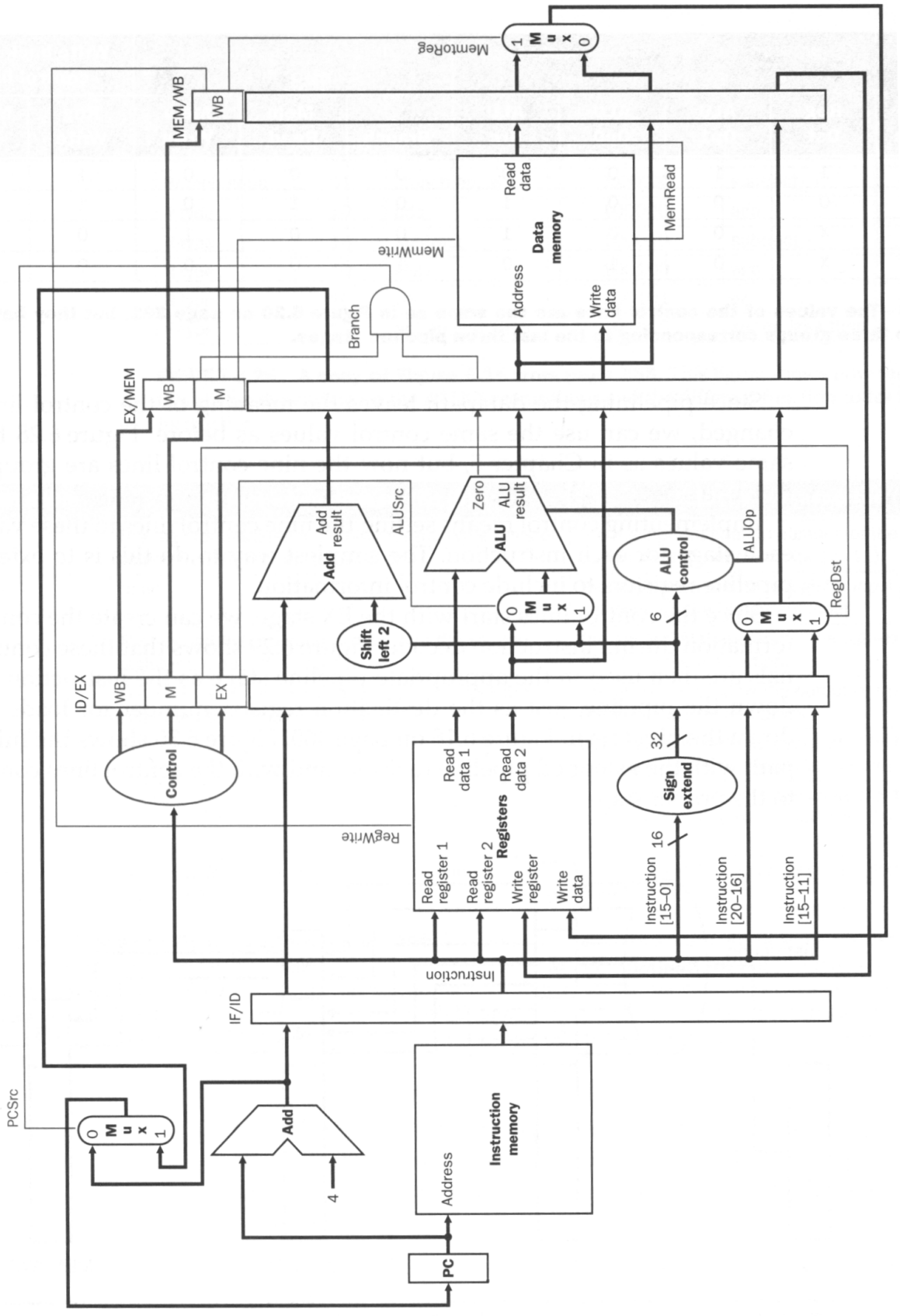
Kommt in einer Befehlsfolge ein Sprungbefehl vor, ist noch nicht klar, bei welchem Befehl weitergemacht werden soll. Mittels Branch Prediction (Vorhersage unter Berücksichtigung, ob die letzen Male gesprungen oder nicht gesprungen wurde) wird versucht, Verzögerung durch Falschraten zu reduzieren. Die Hardware wird so gestaltet, dass die Entscheidung möglichst früh fällt, damit bei Irrtum möglichst wenig Befehle umsonst aufgeführt werden.

Ausblick: Superscalar und Dynamic Pipelining

Beim Superscalar Pipelining wird versucht, durch Duplizierung der Hardware in jeder Phase mehrere Befehle auszuführen.
Dynamic Pipelining versucht Konflikte durch nichtlineares Bearbeiten zu lösen. Die Befehle werden auf unterschiedliche Funktionseinheiten verteilt und dort gleichzeitig bearbeitet. Eine Commit Unit sortiert hinterher die Befehle wieder in die richtige Reihenfolge.

Verwendete Literatur

David A. Patterson and John L. Hennessy:
Computer Organization & Design – The Hardware/Software Interface
Morgan Kaufmann, 1998 (2nd Ed.), 1-55860-491-X


MemtoReg
MEM/WB
WB
Read data
Data memory
Address
Write data
MemRead
MemWrite
Branch
EX/MEM
WB
M
Add result
Add
ALUSrc
Zero
ALU result
ALU
ALU control
ALUOp
RegDst
Shift left 2
ID/EX
WB
M
EX
Control
Read data 1
Read data 2
Registers
Read register 1
Read register 2
Write register
Write data
RegWrite
Sign extend
32
16
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Instruction
IF/ID
PCSrc
Add
4
Instruction memory
Address
PC

Performanceverbesserung durch Pipelining

am Beispiel des MIPS-Prozessors

Stefan Conrad Andreas Tyart

1. Ein Überblick über Pipelining

Gibt es für verschiedene Aufgaben unterschiedliche Hardware, lässt sich die Performance durch Parallelarbeit weiter steigern.

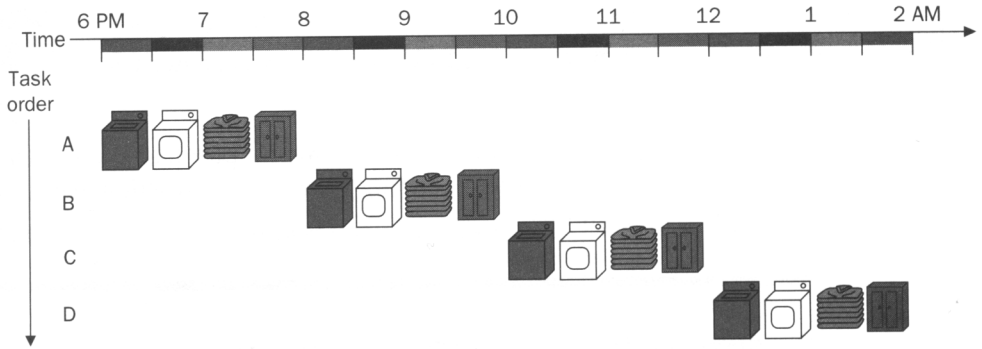
1.1. Ein Beispiel aus dem Alltag

Wäsche waschen: Es gibt vier Arbeitsschritte

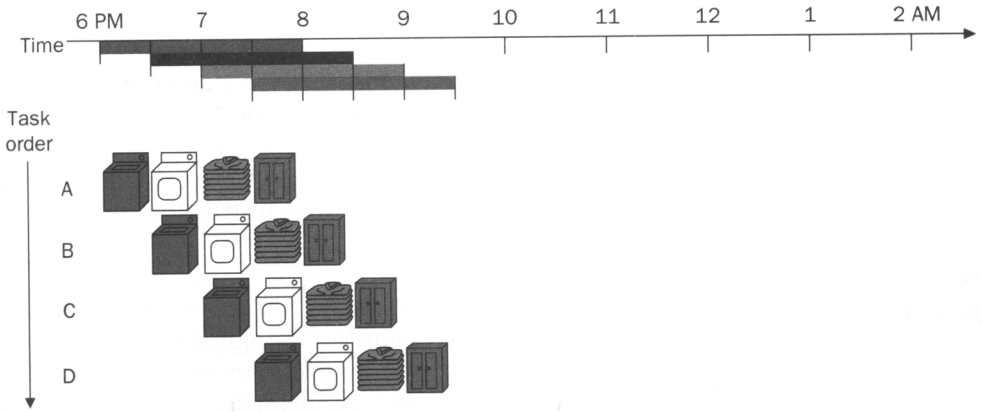
- Waschmaschine
- Trockner
- Bügelautomat
- Wegsortieren

4 Personen wollen waschen.

1.2. Hintereinanderausführung



1.3. Parallelarbeit (Pipelining)



→ Die Arbeitszeit für ein Waschdurchgang ist gleich geblieben, aber die gesamte Waschdauer ist mehr als halbiert.

2. Datapath beim Pipelining

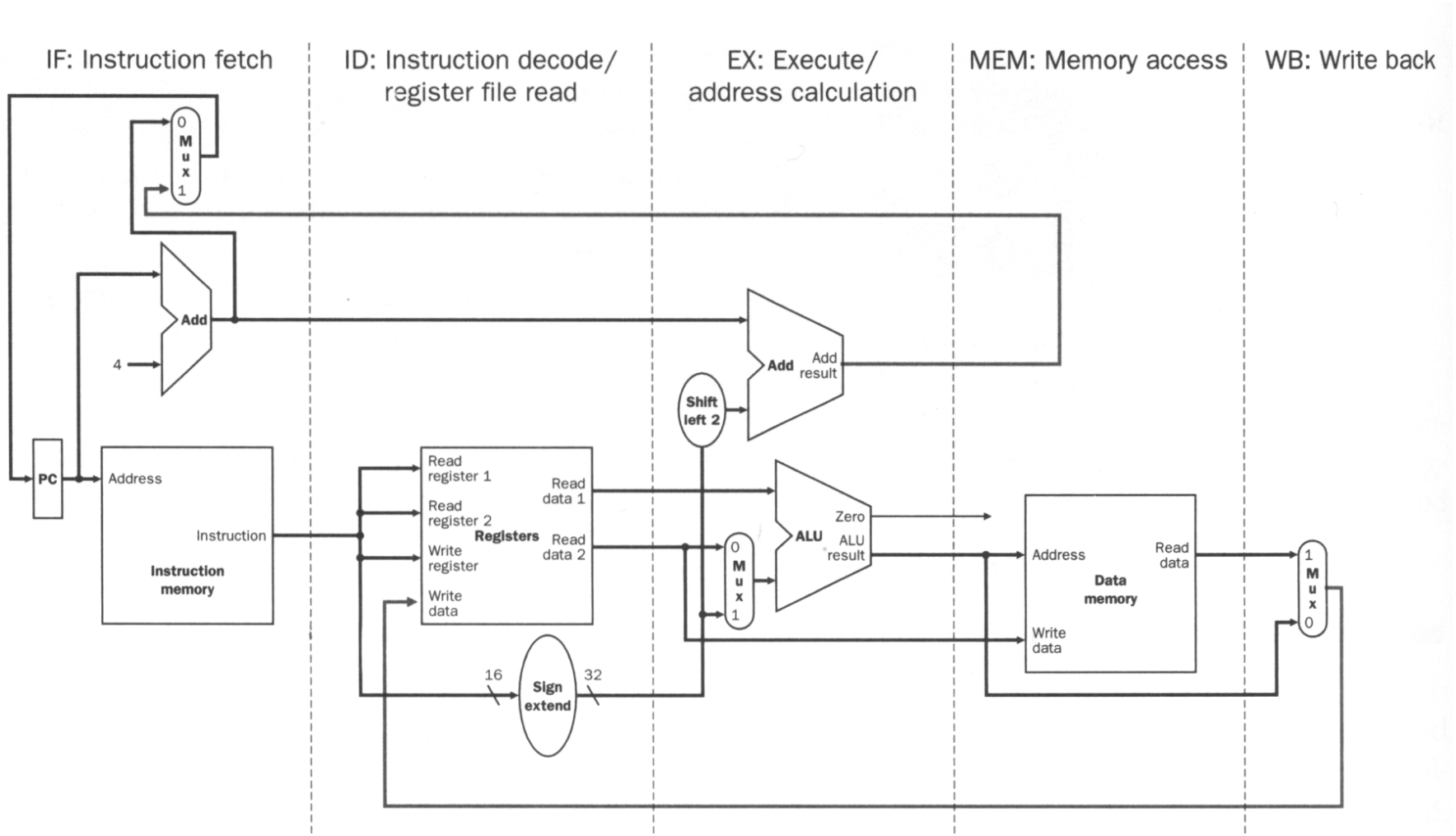
2.1. Einteilung der Phasen beim MIPS

Wir betrachten hier nur die Befehle lw, sw, add, sub, and, or, slt und beq

Instruction class	Instruction fetch	Register read	ALU Operation	Data access	Register write	Total time
Load	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
Store	2 ns	1 ns	2 ns	2 ns		7 ns
R-Format	2 ns	1 ns	2 ns		1 ns	6 ns
Branch	2 ns	1 ns	2 ns			5 ns

Jede Phase muss die gleiche Länge haben, ebenso jede Instruktion.
→ Phasenlänge 2 ns, Dauer für die gesamte Instruktion: 10 ns

2.2. Datapath beim Pipelining

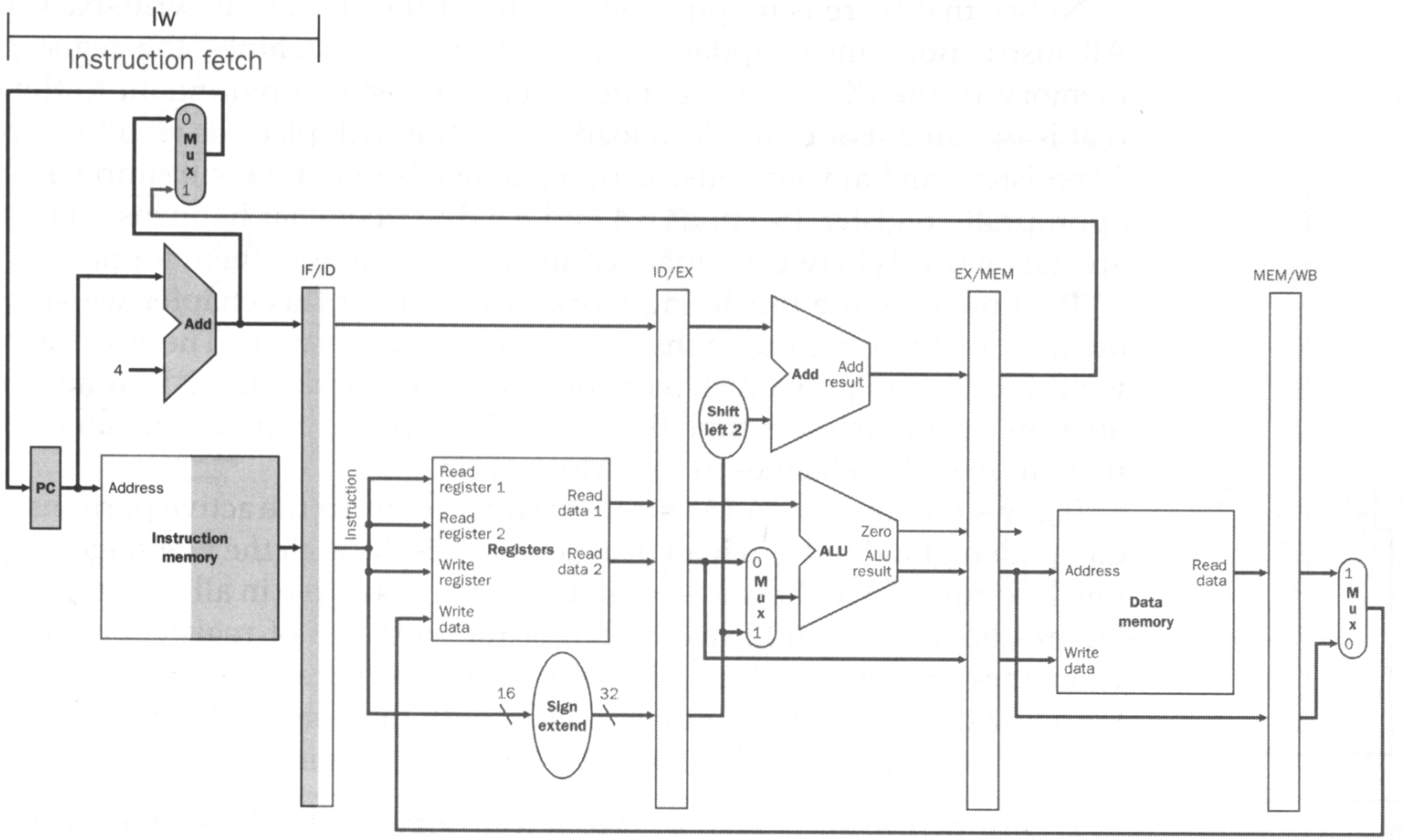


2.3. Pipeline Register

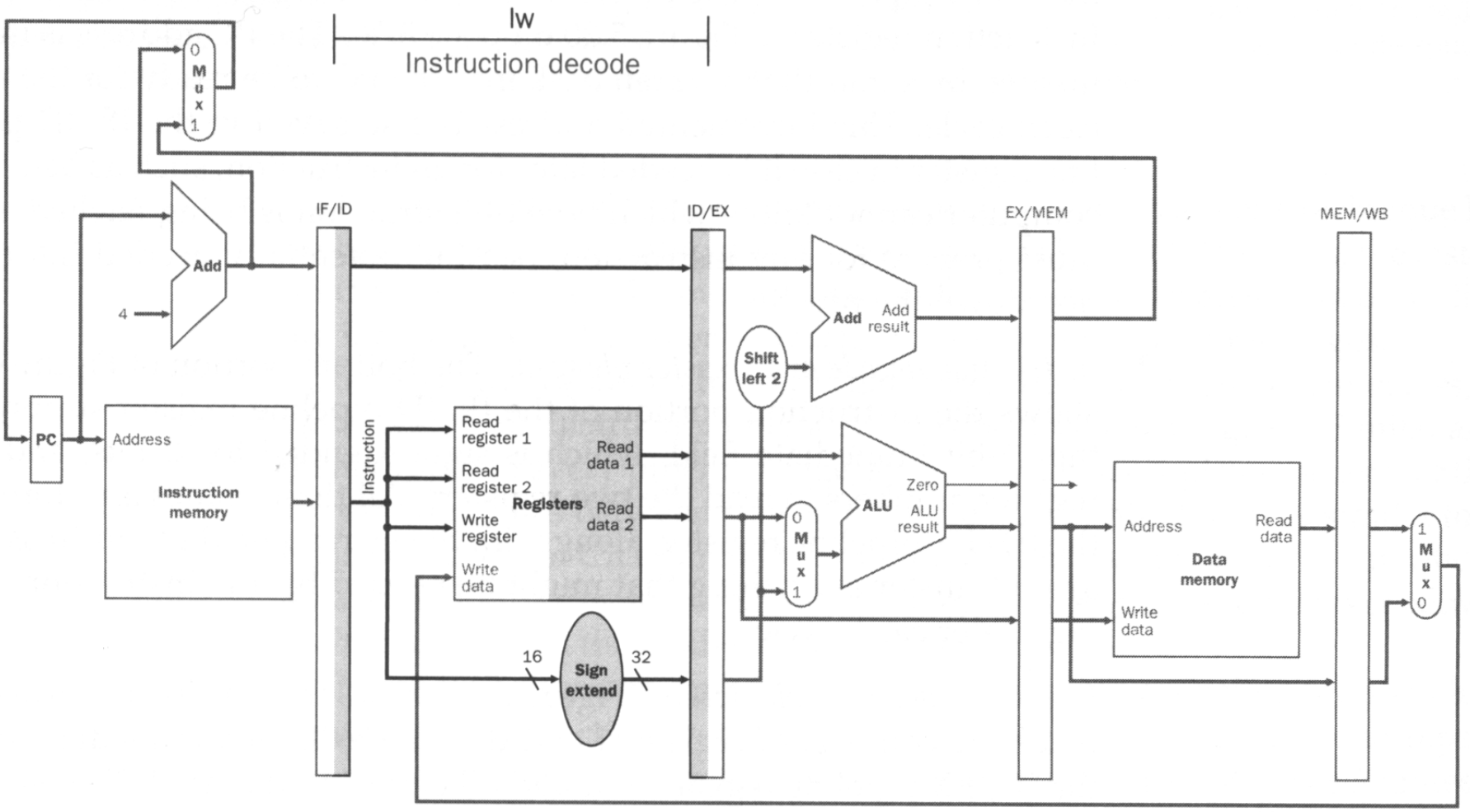
- Problem: Wohin mit den Zwischenergebnissen?
- Lösung: Pipeline Register für die Zwischenspeicherung wie bei Multicycle
- Analogie: Waschkörbe zwischen Waschmaschine, Trockner, Bügelautomat und Schrank

2.4. Beispiel load-Instruktion

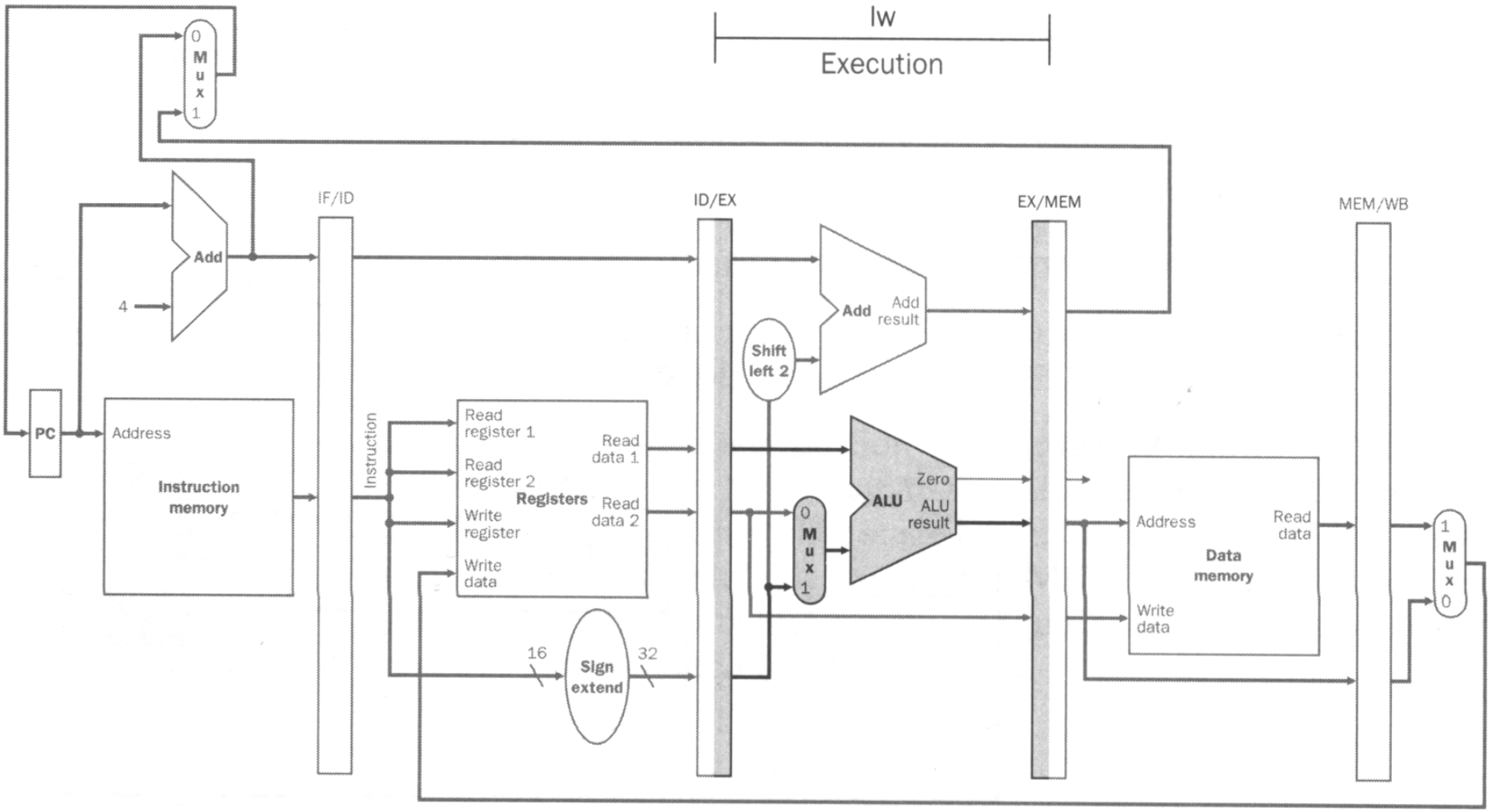
2.4.1. IF: Instruction load (Instruktion laden)



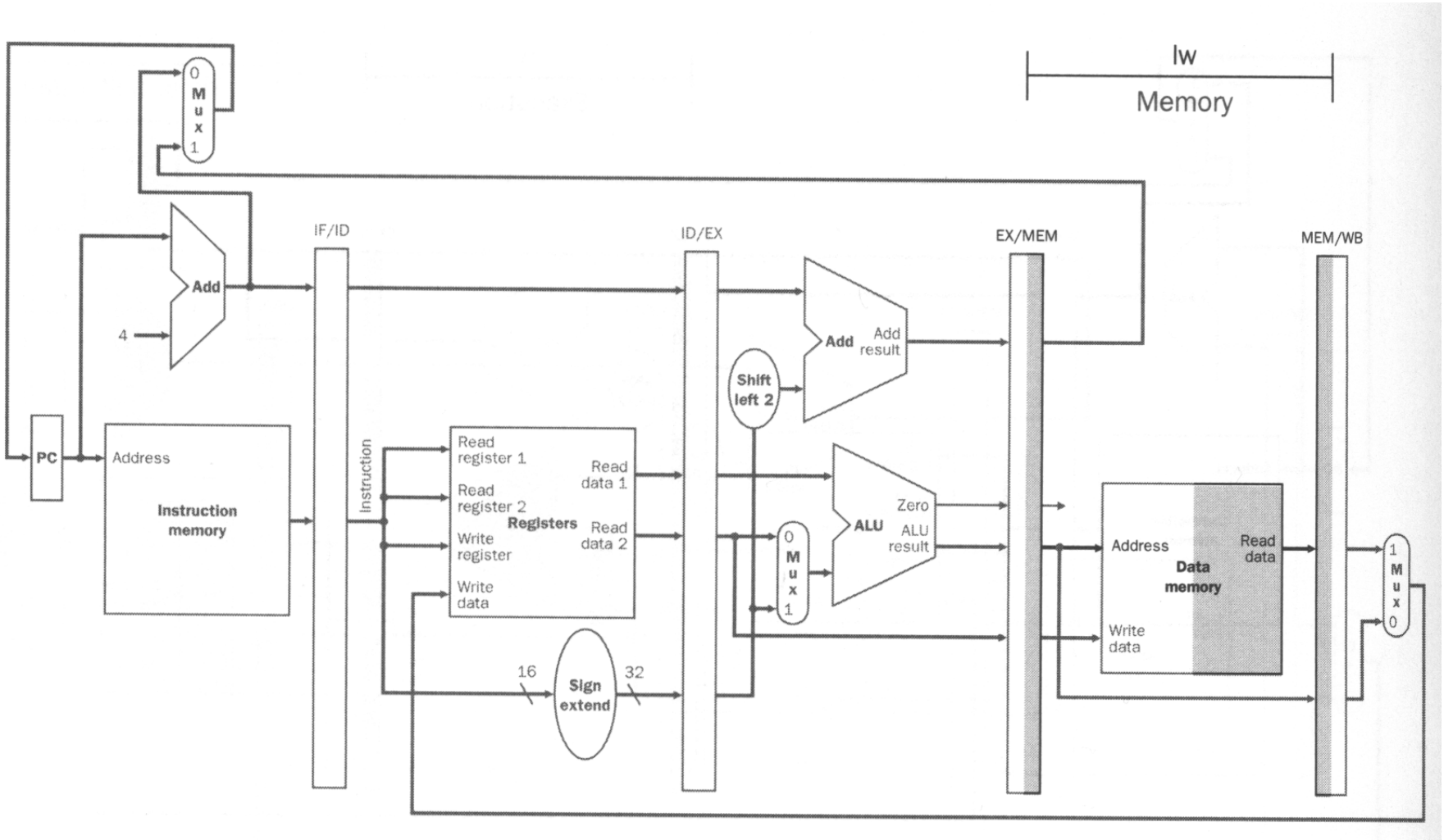
2.4.2. ID: Instruction decode (Instruktion dekodieren)



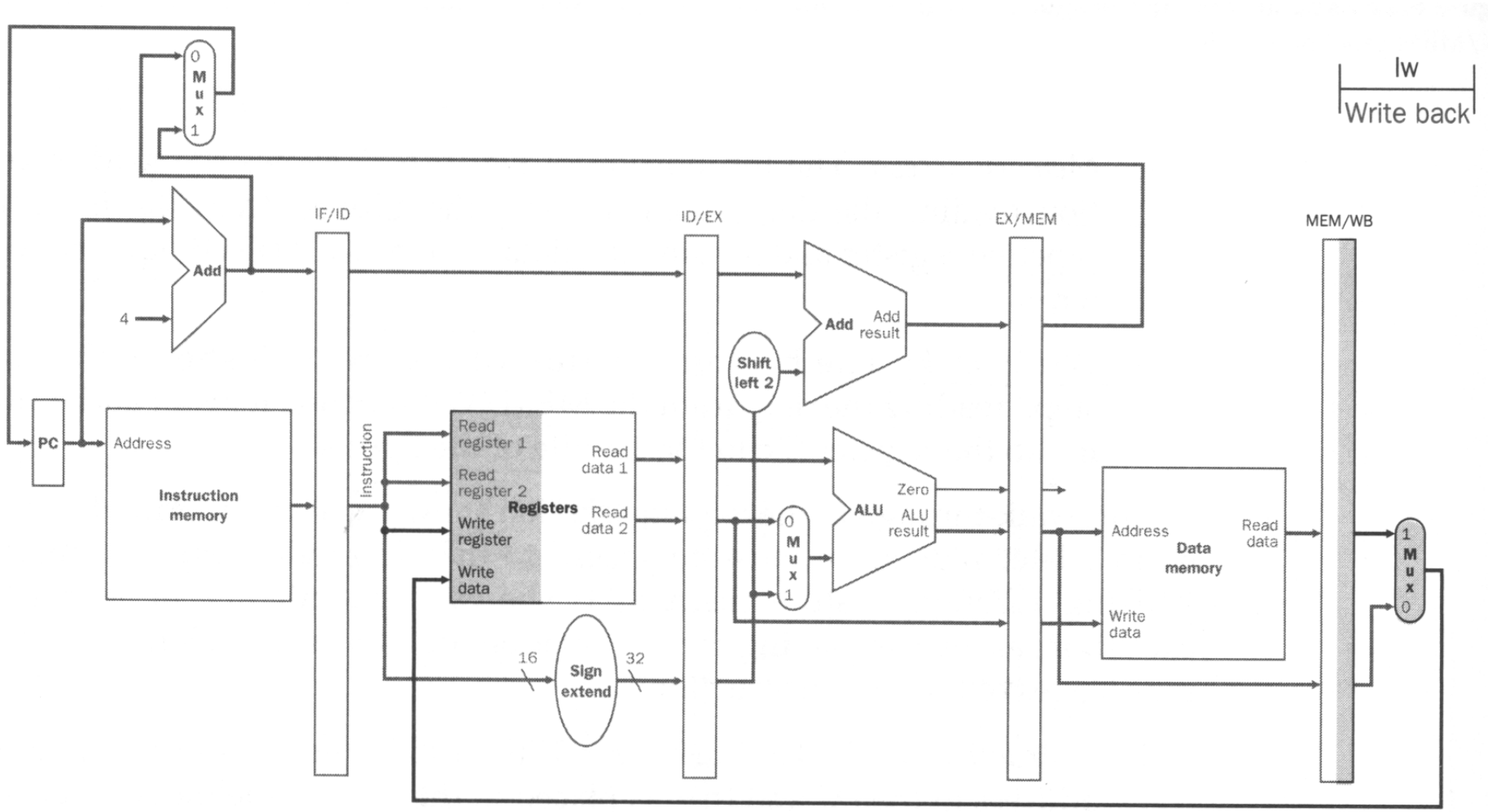
2.4.3. EX: Execute (Ausführen)



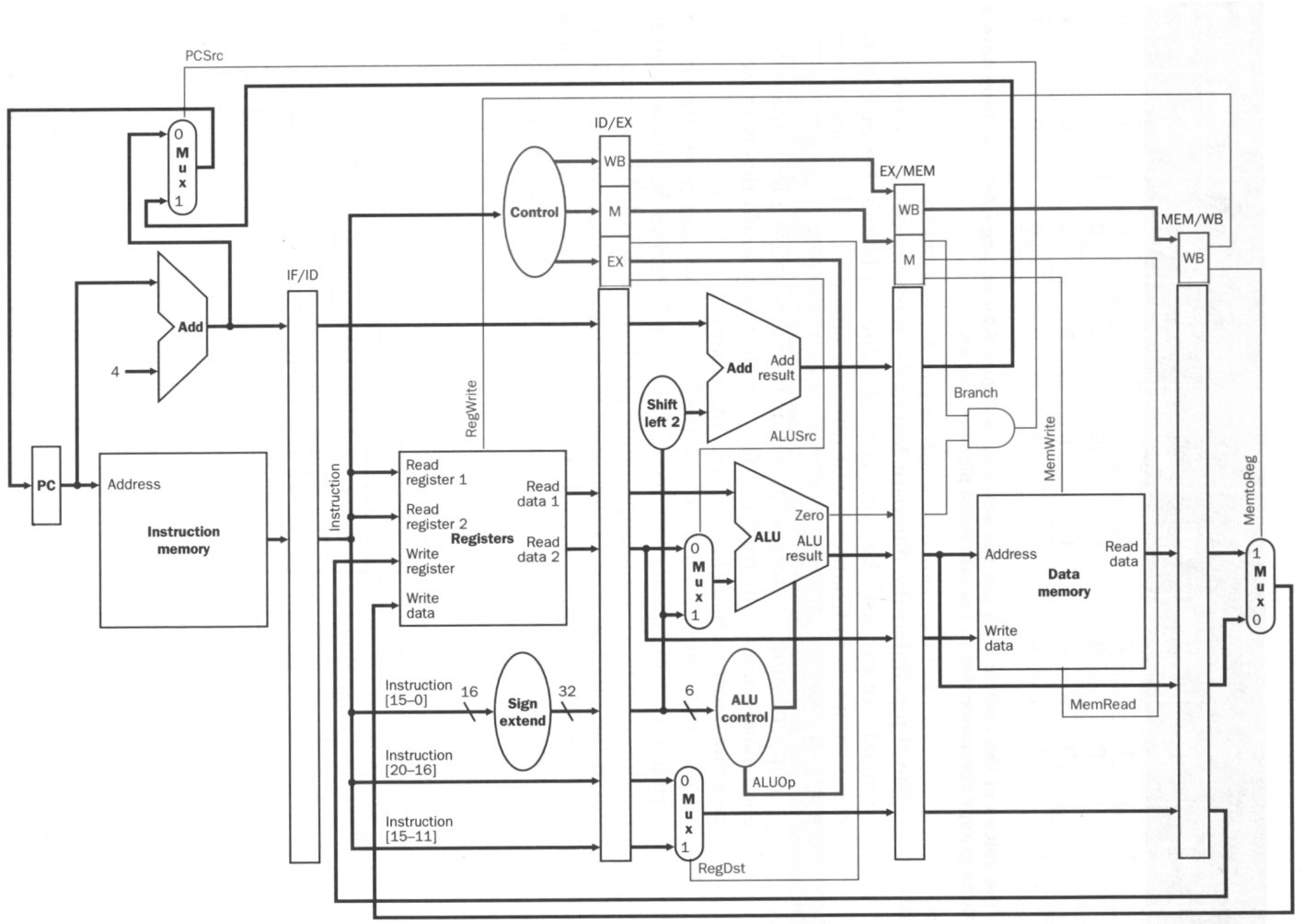
2.4.4. MEM: Memory access (Speicheroperation)



2.4.5. WB: Write back (Ergebnis speichern)



2.5. Pipelined Control



3. Die Probleme der Pipeline :

Datenkonflikte :

Diese entstehen, wenn Befehle von dem Ergebnis vorheriger abhängig sind.
Möglichkeiten diese zu umgehen :

- der Compiler ordnet die Befehle so, daß die Abhängigkeit umgangen wird (evtl. nop)
- Forwarding - die Ergebnisse werden benutzt, bevor sie im Zielregister sind
- Stalls - einige Abhängigkeiten lassen sich nicht durch Forwarding lösen, so daß einzelne Phasen sich im „Leerlauf“ befinden müssen

Sprungkonflikte:

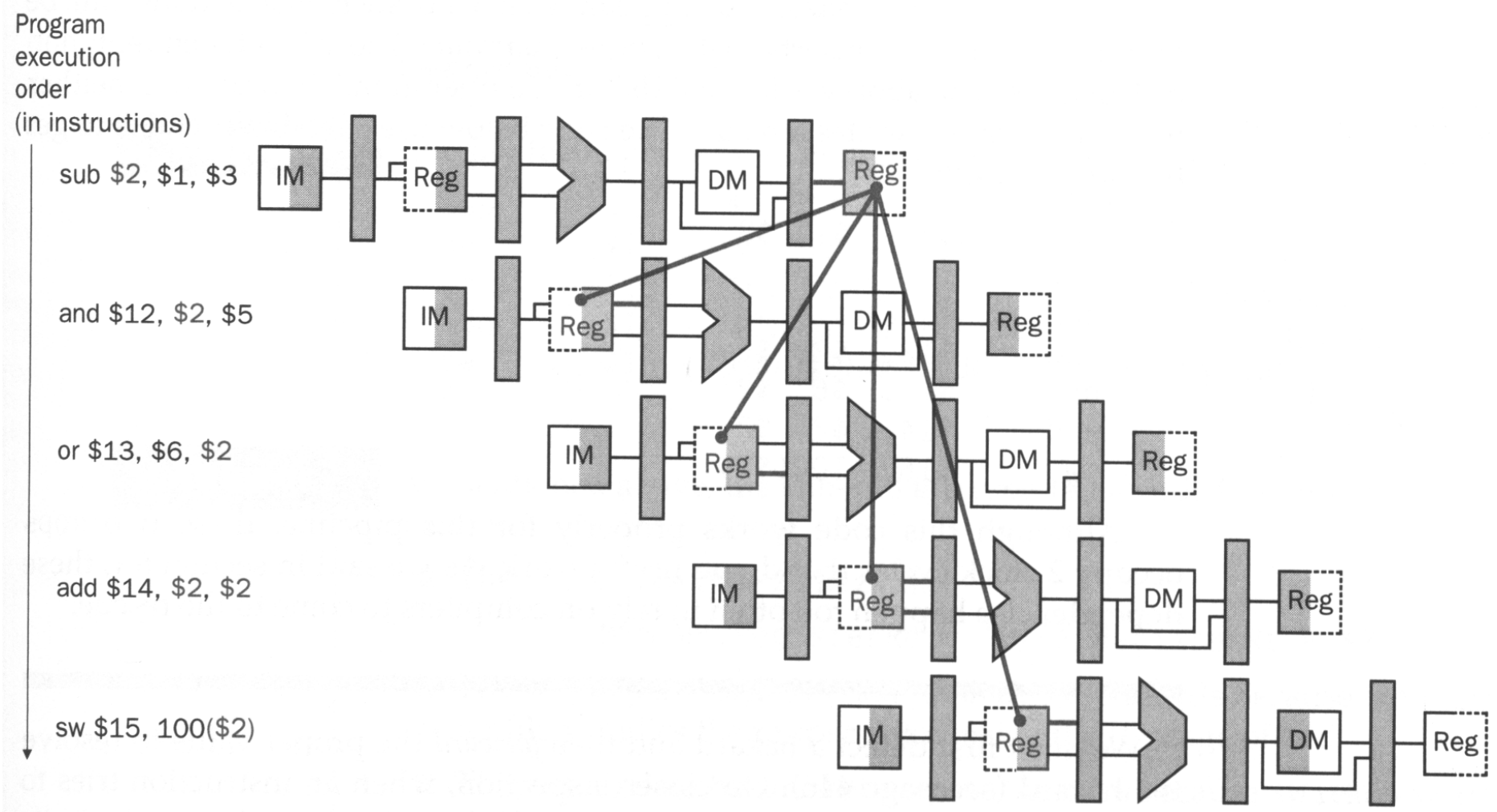
Wenn in der Befehlsfolge ein Sprungbefehl (z.B. beq) auftaucht, ist nicht klar, mit welchem Befehl es weitergeht. Hier gibt es keine Patentlösung wie bei den Datenkonflikten, sondern nur Schadenbegrenzung:

- Reducing the Delay of Branches
- Assume Branch not taken
- Dynamic Branch Prediction

3.1. Datenkonflikte (Data Hazards) :

Da viele Befehle von einander abhängig sind, bekommt man schnell Probleme.
Die folgende Befehlsfolge zeigt solche Abhängigkeiten.:

```
sub   $2, $1,  $3
and $12, $2,  $5
or    $13, $6,  $2
add $14, $2,  $2
sw  $15, 100, $2
```


Program
execution
order
(in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM
Reg

3.2. Lösungsmöglichkeiten :

3.2.1. Die Softwarelösung :

Der Compiler läßt es nicht zu, daß solche Abhängigkeiten auftreten.

Umstellung des Programmcodes, einfügen von anderen Befehlen oder nop (no operation).

```
sub   $2, $1,  $3
nop
nop
and $12, $2,  $5
or    $13, $6,  $2
add $14, $2,  $2
sw  $15, 100, $2
```

Die Softwarelösung :

- Vorteile :
  - Keine Änderung der Hardware notwendig

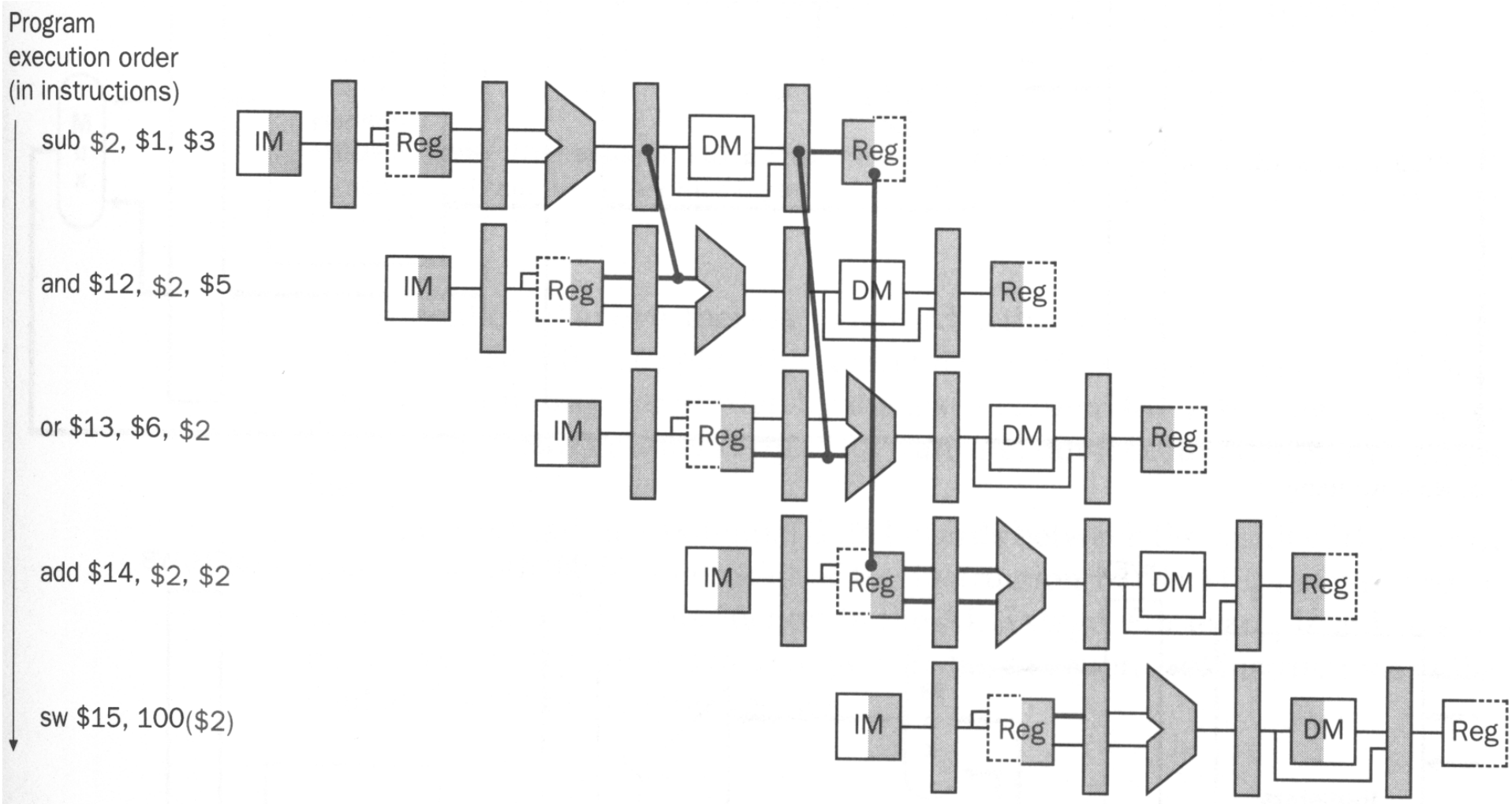
- Nachteile :
  - Solche Abhängigkeiten entstehen zu oft, als es sinnvoll ist, die Problemlösung nur dem Compiler zu überlassen.
  - Die nop- Befehle haben bedeuten Leerlauf des Prozessor.

3.2.2. Forwarding

- Daten früherer Berechnungen sind in Pipeline-Registern, bevor sie im Hauptregister sind

- Erkennen von Abhängigkeiten : (Einteilung in zwei Klassen):
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

- Sobald die Abhängigkeit erkannt ist, muß noch die Hardware entsprechend erweitert werden, dann können auch Pipeline Register als Alu-Input genutzt werden.


Program
execution order
(in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM
Reg

3.2.2.1 Technische Umsetzung: Die Forwarding Unit

1. Testen, ob und wenn ja, welche Abhängigkeit vorliegt, ggf. forwarden :
   1. Ex Hazards :

      if (EX/MEM.RegWrite
      and (EX/MEM.Register.Rd ≠ 0)
      and (EX/MEM.Register.Rd=ID/EX.RegisterRs)) ForwardA= 10

      if (EX/MEM.RegWrite
      and (EX/MEM.Register.Rd ≠ 0)
      and (EX/MEM.Register.Rd=ID/EX.RegisterRt)) ForwardB= 10
   2. Mem Hazards :

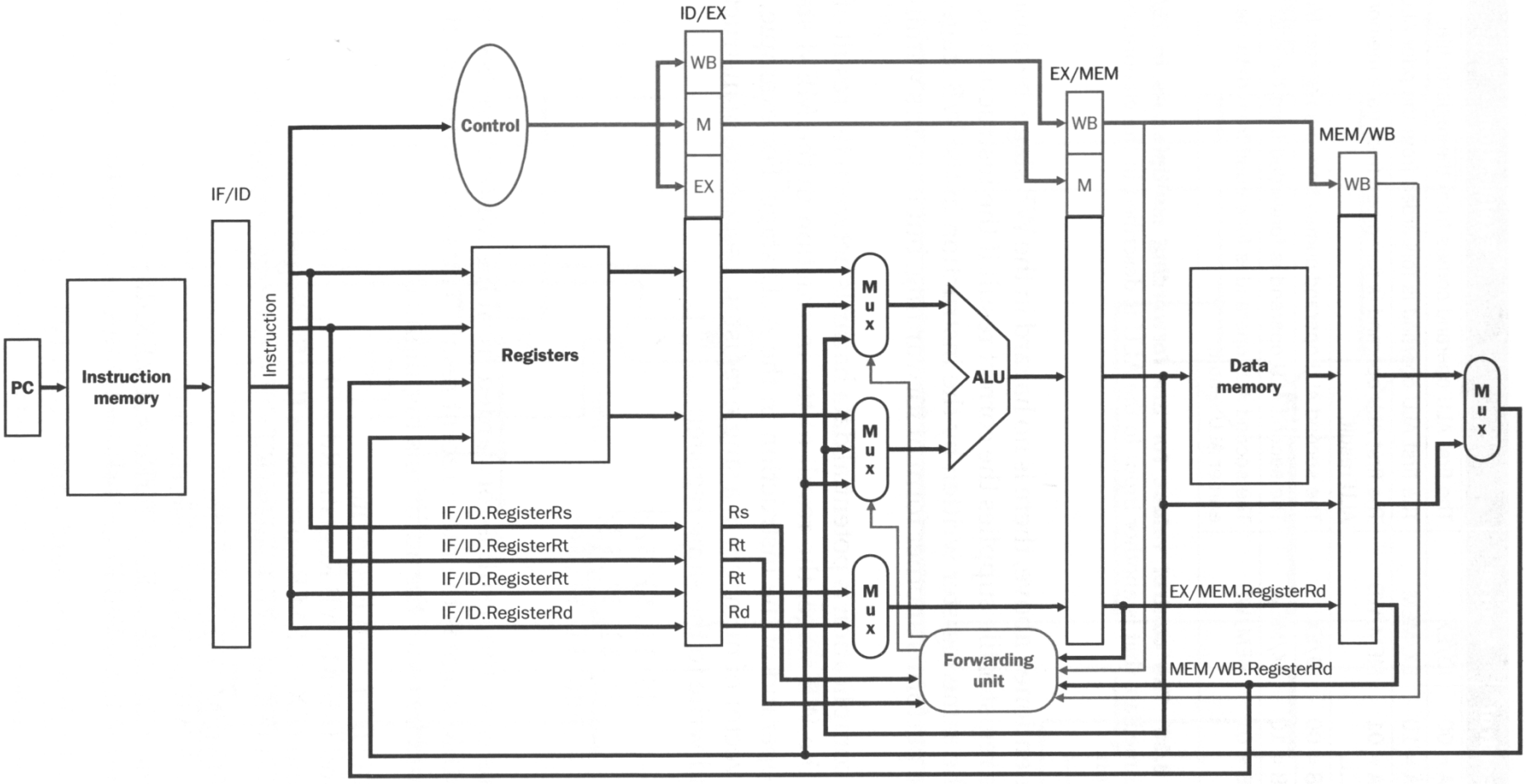
      if (MEM/WB.RegWrite
      and (MEM/WB.Register.Rd ≠ 0)
      and (MEM/WB.Register.Rd=ID/EX.RegisterRs)) ForwardA= 01

      if (MEM/WB.RegWrite
      and (MEM/WB.Register.Rd ≠ 0)
      and (MEM/WB.Register.Rd=ID/EX.RegisterRt)) ForwardB= 01

2. Kontrollwerte für die Forwarding-Multiplexors

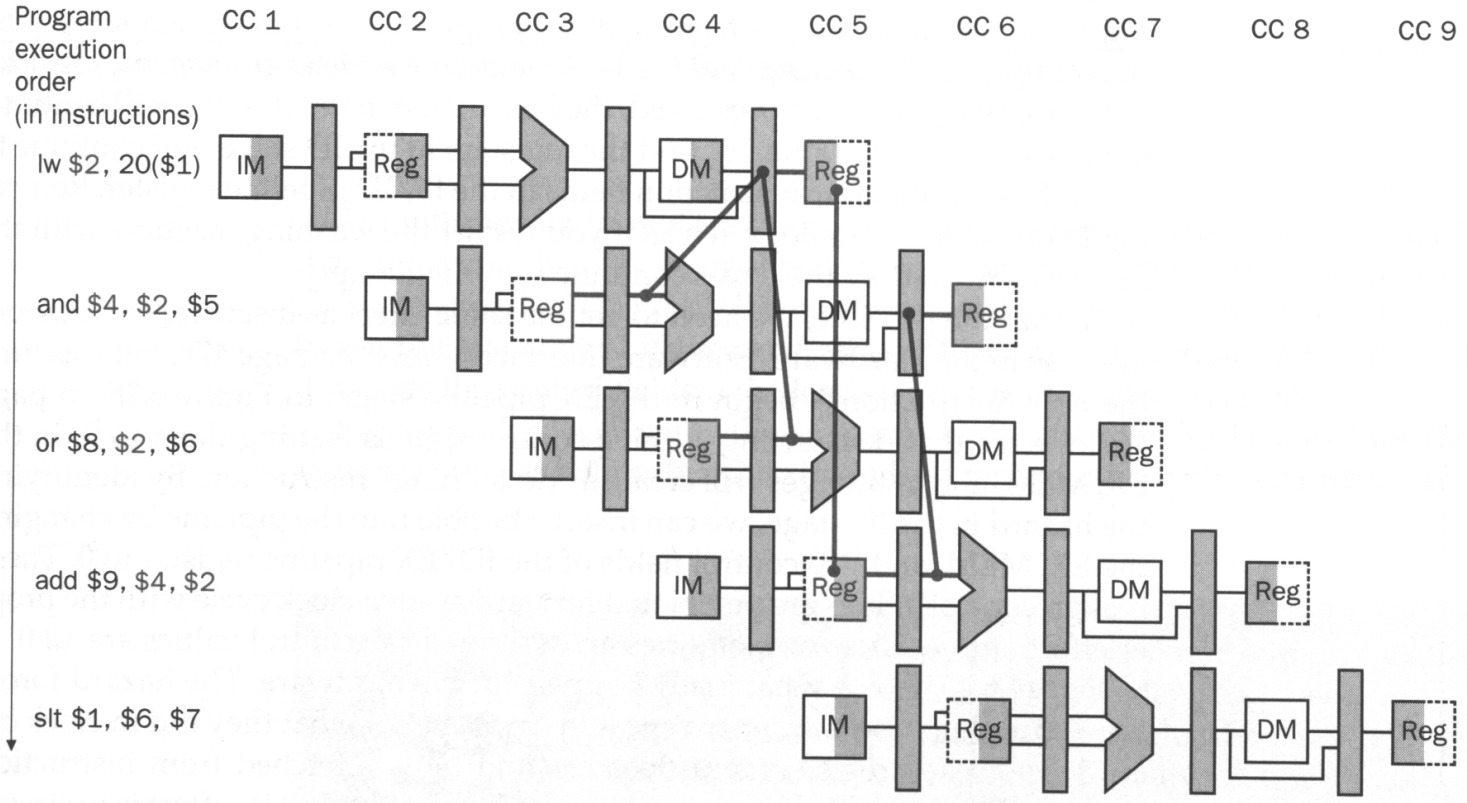
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result
ForwardB = 00	ID/EX	The second ALU operand comes from the register file
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result

3. Das Schaltbild



3.2.3. Datenabhängigkeit und Stall

- Folgt auf einen load-Befehl ein Befehl, der den zu schreibenden Register lesen will, so kann man diese Abhängigkeit nicht so einfach mit Forwarding lösen.



- Die Pipeline muß angehalten werden („stall“)

3.2.4. Hazard detection unit

- Die Hazard Detection Unit muß o.g. Abhängigkeiten erkennen und entsprechend reagieren.

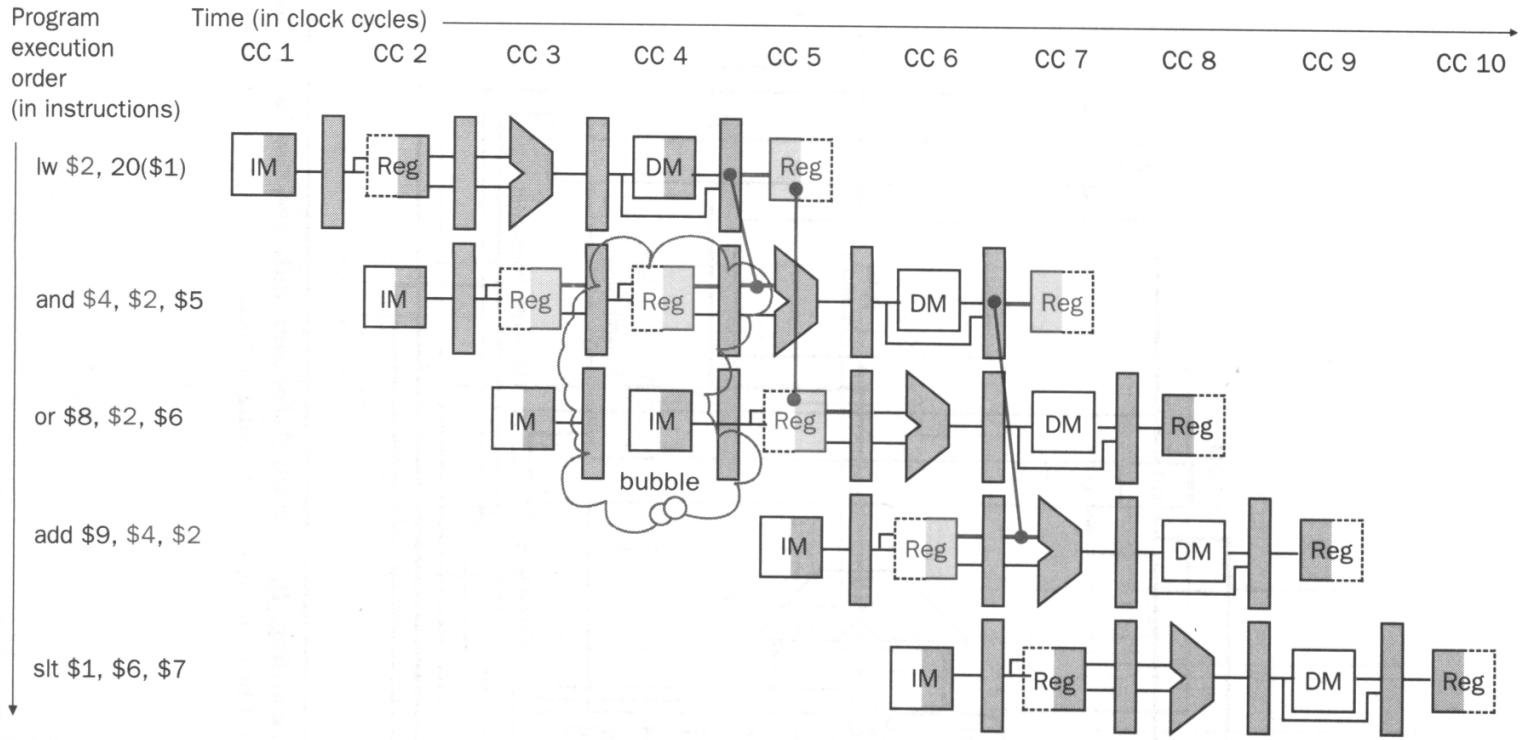
- Die Erkennung :

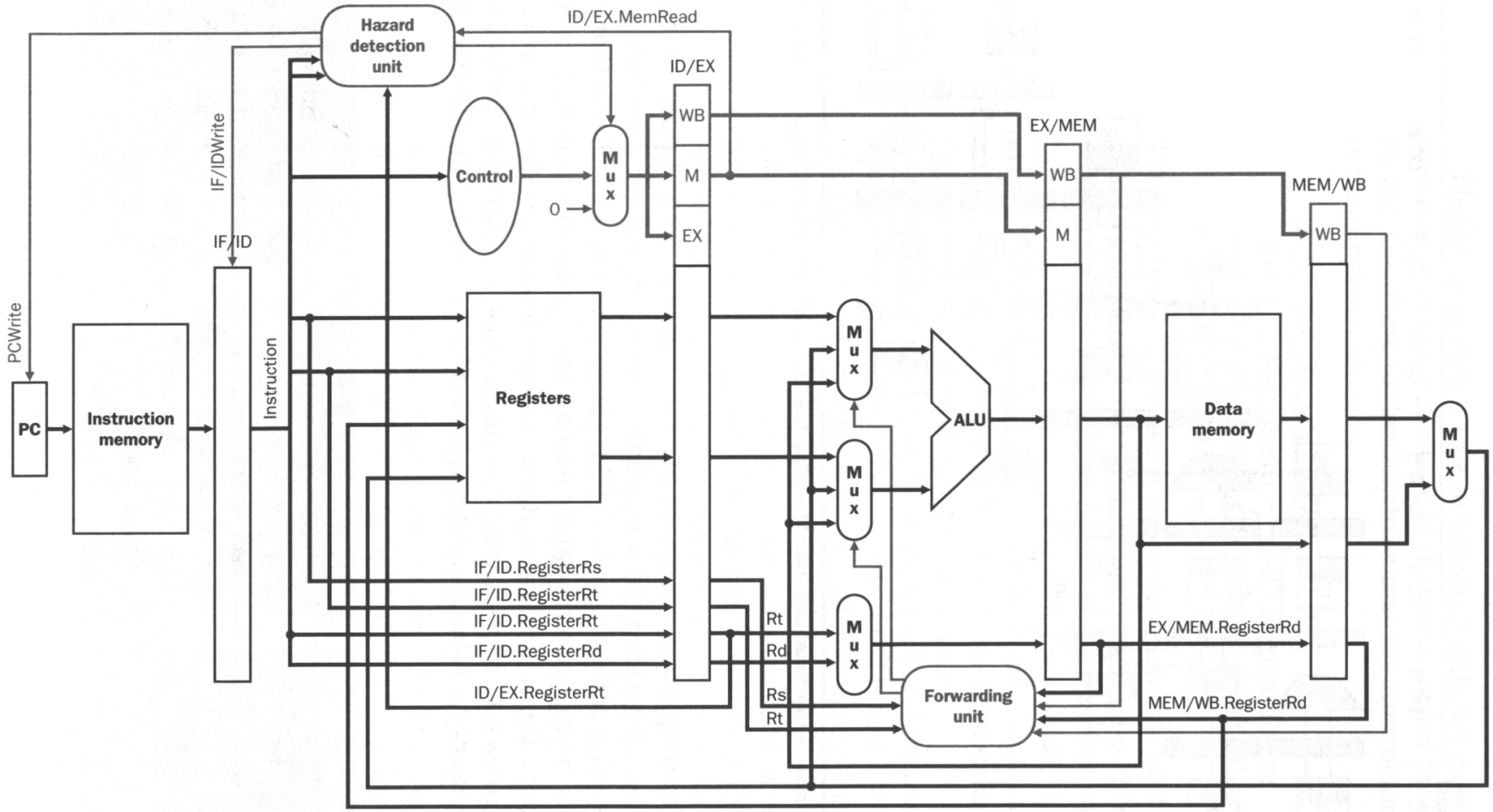
```
if ( ID/EX.MemRead and
((ID/EX.RegisterRt=IF/ID.RegisterRs) or
 (ID/EX.RegisterRt=IF/ID.RegisterRs)))
stall the Pipeline
```

3.2.5. Die technische Umsetzung:

- Wenn der Befehl in der ID-Phase gestoppt werden soll, muß auch der Befehl in der IF-Phase aufgehalten werden, da er sonst verlorengehen würde.

- In der EX-Phase wird praktisch ein „nop“ eingefügt.




Hazard detection unit
ID/EX.MemRead
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
IF/IDWrite
Control
M u x
0
IF/ID
PCWrite
PC
Instruction memory
Instruction
Registers
M u x
ALU
Data memory
M u x
M u x
M u x
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Rs
Rt
Forwarding unit
EX/MEM.RegisterRd
MEM/WB.RegisterRd

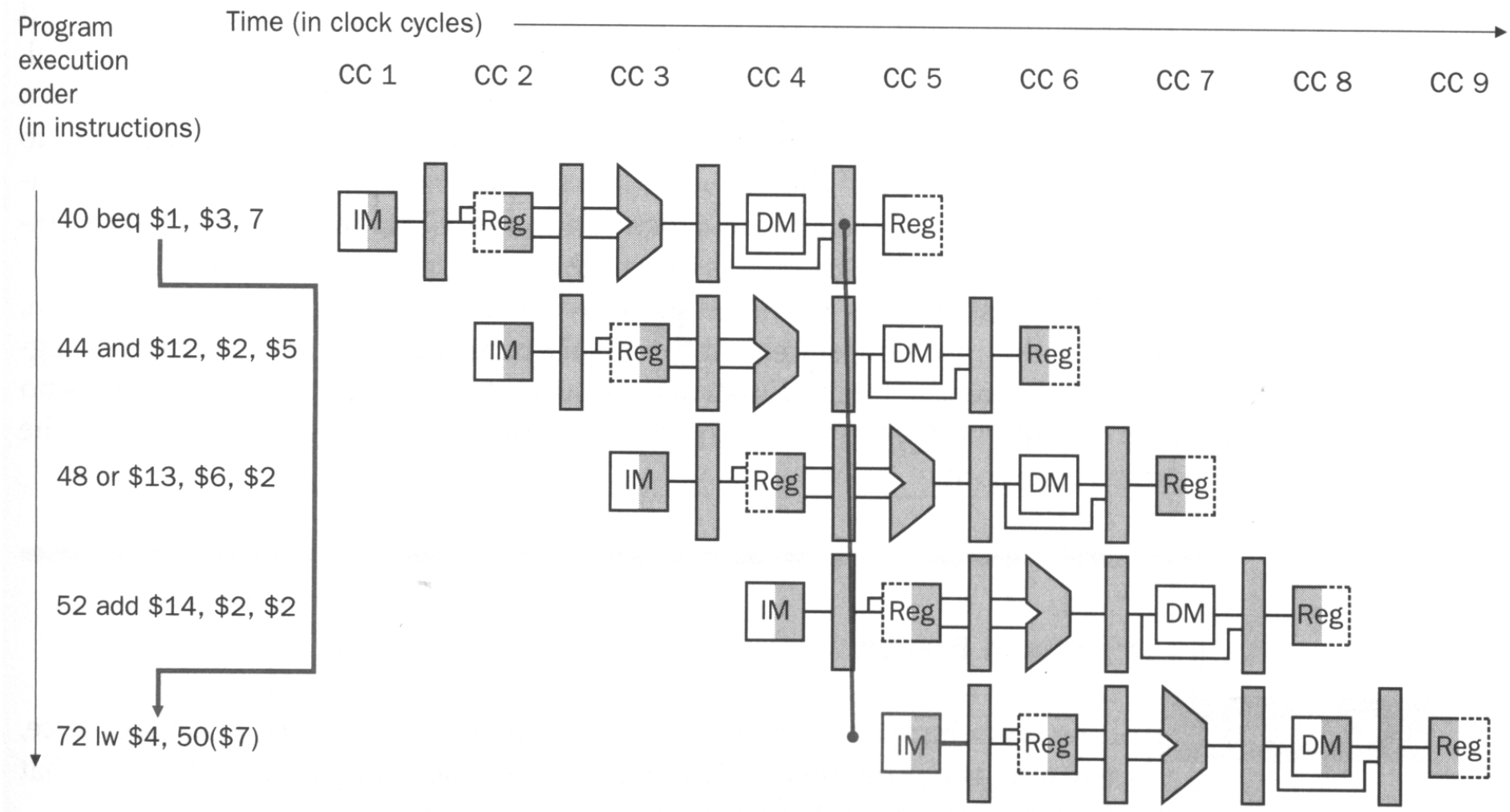
3.3. Sprungkonflikte (Branch Hazards)

Hier gibt es keine Möglichkeit, das Problem zu beheben wie bei den Data Hazards.
Tritt ein Branch-Befehl auf kann man :

- warten, bis entschieden ist an welcher Stelle weitergemacht wird

oder

- raten (besser, versuchen vorherzusagen), wo es weitergeht.
  - rät man richtig, macht man einfach weiter
  - hat man unrecht, müssen alle falschen Befehle gelöscht werden.


Program execution order (in instructions)
Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
40 beq $1, $3, 7
44 and $12, $2, $5
48 or $13, $6, $2
52 add $14, $2, $2
72 lw $4, 50($7)
IM
Reg
DM
Reg

3.4. Reducing the Delay of Branches

- Die Branch Entscheidung wird in die ID-Phase vorverlegt :
  - Erhöht, den Hardwareaufwand
  - ist schneller
  - bei falscher Vorhersage muß nur ein Befehl gelöscht werden.

3.5. Branch Prediction

- „assume Branch not taken“ : Die einfachste Art der Sprungvorhersage ist davon auszugehen, die Verzweigung wird nicht genommen

- dynamik Branch Prediction : diese Sprungvorhersage ist besser, da sie sich auf vorherige Ereignisse beruft
  - die einfache Variante ist der 1-Bit Branch-prediction-Buffer
  - genauer ist die 2-Bit Methode

3.6. Sprungentscheidung: Am Beispiel der Schleife

Situation: In einer Schleife wird eine Verzweigung immer neun aufeinander folgenden Durchläufen genommen und dann einmal nicht

- assume branch not taken : 10 % Trefferquote

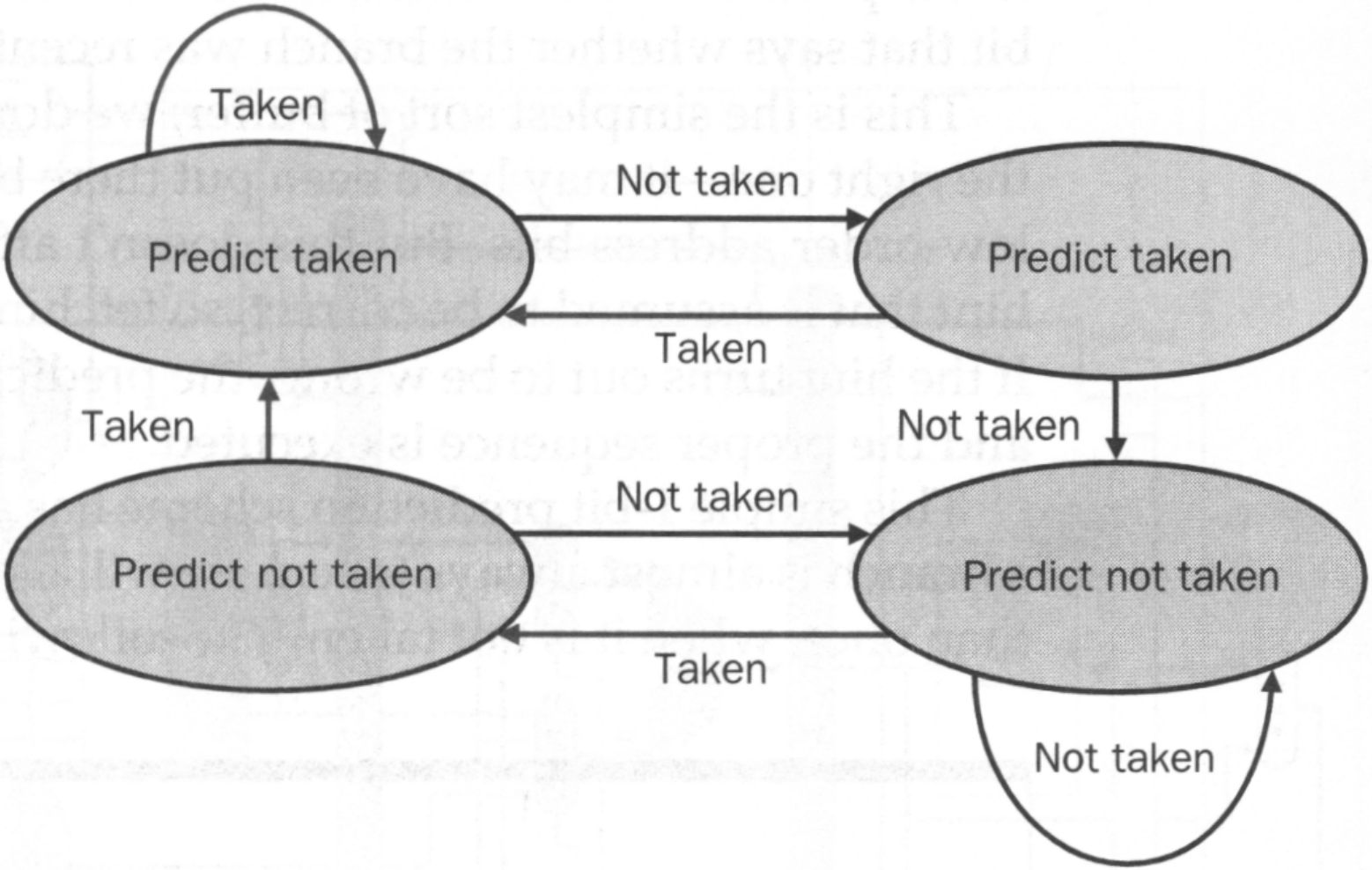
(da immer nur blind geraten wird)

- 1-Bit Methode : 80 % Trefferquote

(es wird nur die letzte Entscheidung berücksichtigt)

- 2-Bit Methode : 90 % Trefferquote

(es wird mehr als nur die letzte Entscheidung betrachtet)


Taken
Predict taken
Not taken
Predict taken
Taken
Taken
Not taken
Predict not taken
Not taken
Predict not taken
Taken
Not taken

4. Exceptions

4.1. Einige Gründe für Exceptions:

- Arithmetischer Overflow
- I/O-Gerät Anfrage
- Undefinierte Instruktion
- Technische Fehlfunktion

Reaktion:

- Die Adresse der die Exception verursachenden Instruktion wird im EPC-Register gespeichert
- Nachfolgende Instruktionen gestoppt
- Vorherige Instruktionen beendet
- Exceptionhandler aufgerufen.

4.2. Probleme bei Exceptions

- Es können mehrere Exceptions gleichzeitig auftreten → Prioritäten vergeben
- Zuordnung der Exceptions zu der sie verursachenden Instruktion

5. Zusammenfassung

5.1. Performancegewinn durch Pipelining

Typisches Beispiel für durchschnittliche Zeit zwischen zwei Instruktion:

Singlecycle: 8 ns
Multicycle: 6,3 ns
Pipelining: 2,34 ns

→ Pipelining ist 3,4 mal schneller als Singleycle
→ und 2,7 mal schneller als Multicycle

5.2. Ausblick: Weitere Verbesserungen

5.2.1. Superpipelining

→ Aufteilung des Datapath in mehr Phasen

Je mehr Phasen, je mehr Instruktionen kann man gleichzeitig ausführen.

Dabei ist darauf zu achten, dass die einzelnen Phasen ausgeglichen sind.

5.2.2. Superscalar Pipelining

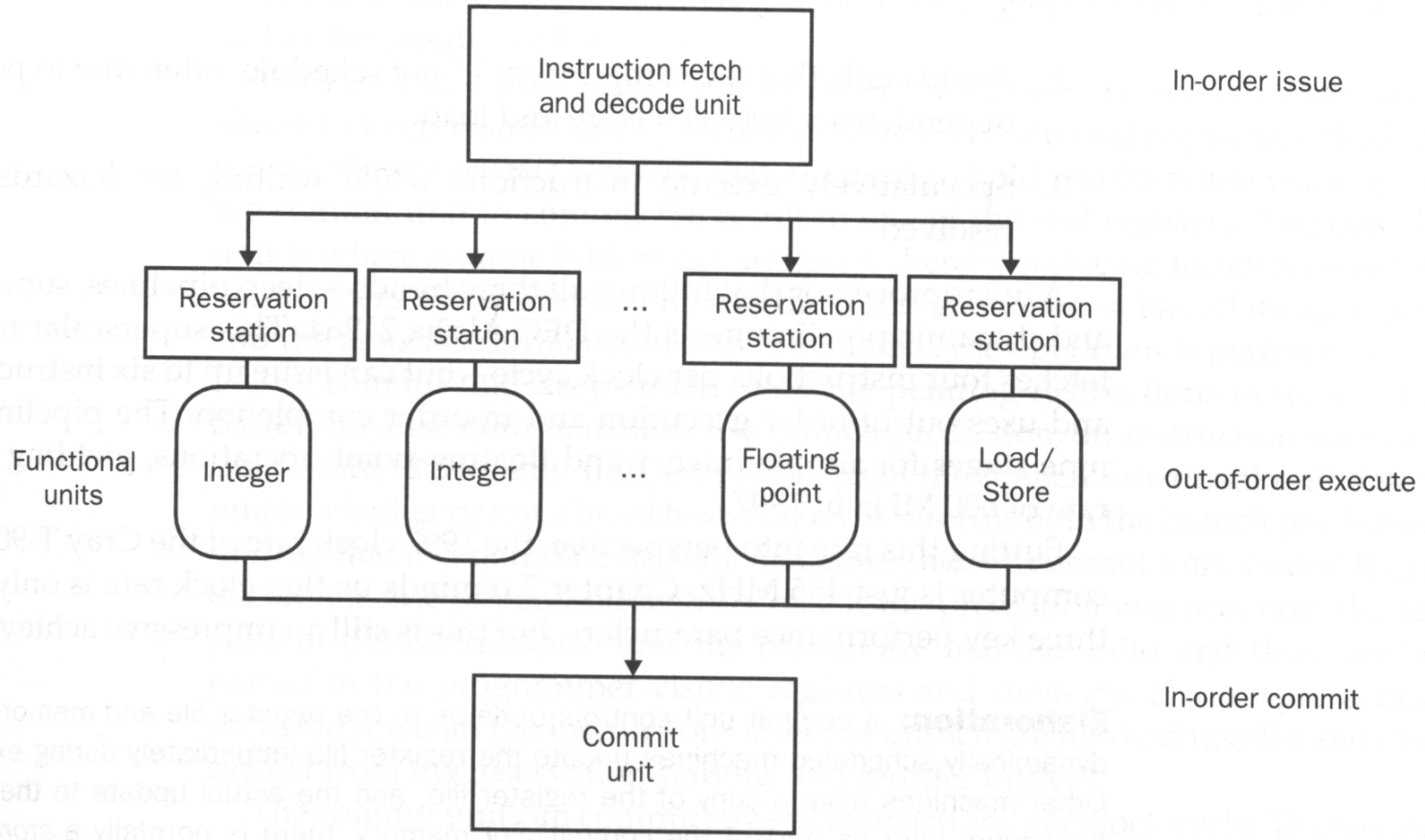
→ Durch Multiplizierung der Hardware können mehrere Instruktionen gleichzeitig ausgeführt werden.

Probleme:

- Durch Abhängigkeiten kann oft doch nur wenige oder sogar nur eine Instruktion ausgeführt werden
- Deutlich komplexere Hardware

5.2.3. Dynamic Pipelining

→ Versuch, Pipeline Konflikte durch “Out of order”-Bearbeitung zu vermeiden.



5.3. Zusammenfassung

- Durch Parallelarbeit erhöht Pipelining die Durchsatzrate an Instruktionen
- Daten- und Sprungkonflikte verringern die theoretisch erreichbare Geschwindigkeitsverbesserung und erhöhen die Komplexität des Datapath
- Datenkonflikte lassen sich größtenteils durch Forwarding lösen
- Bei Sprungkonflikten kann man durch Branchprediction die Verzögerung reduzieren.
- Der Übergang zu längeren Pipelines, Superscalar- und Dynamischem Pipelining hat stark zu den Performancegewinnen der letzten Jahre beigetragen.

Large and Fast: Exploiting Memory Hierarchy

Cache und Speicherhierarchie

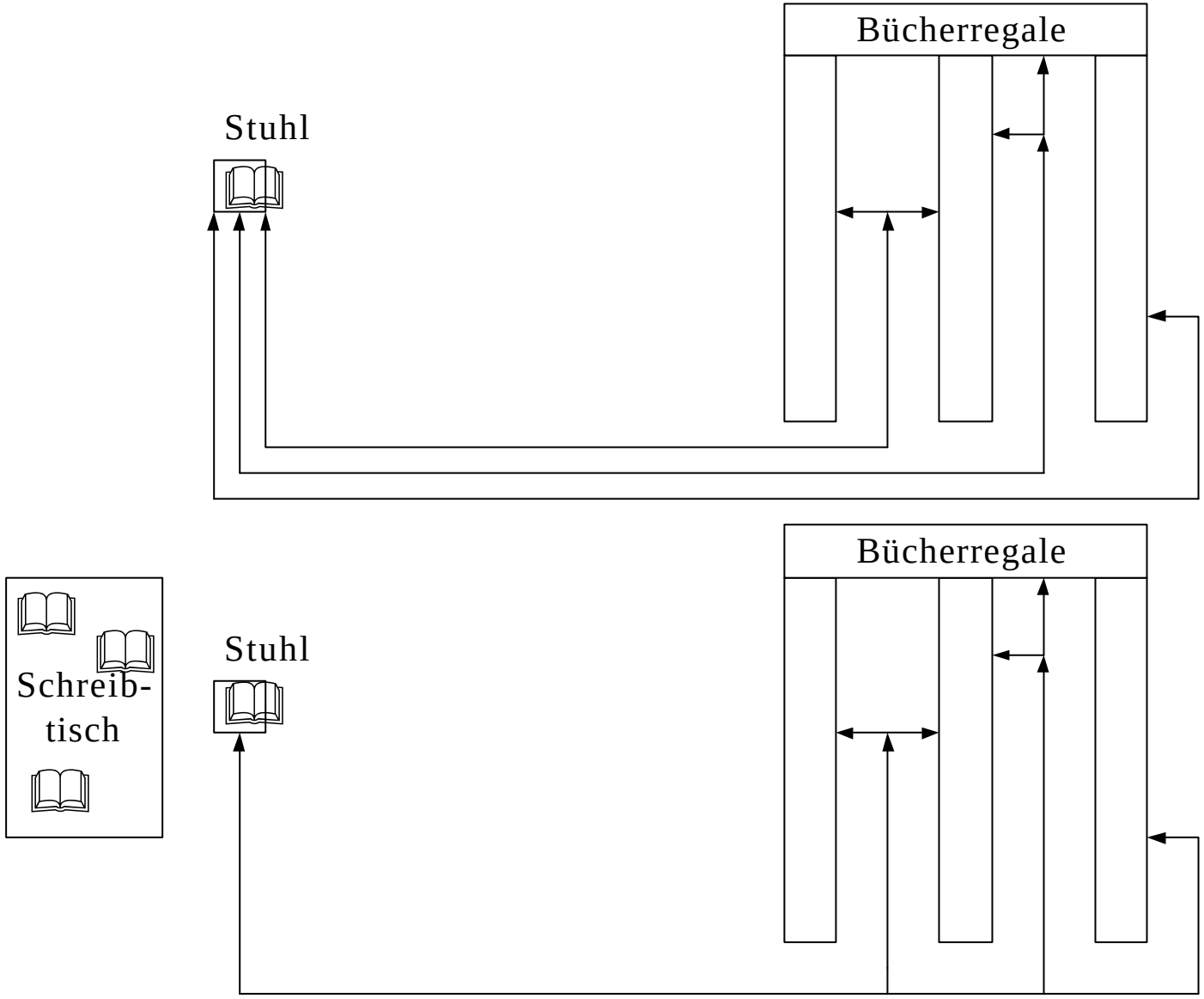
Mi 13.12.2000

Inhalt :

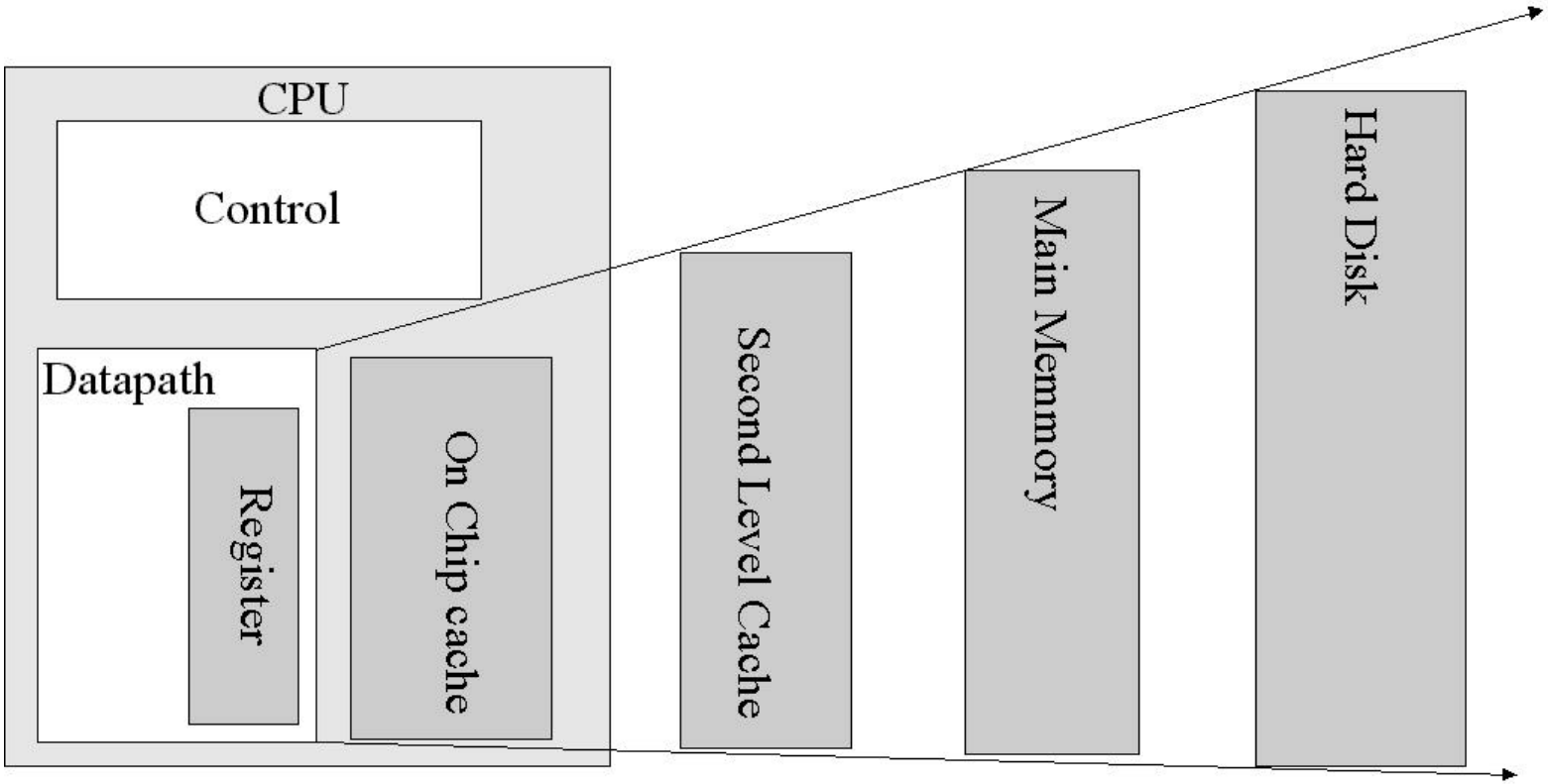
- Einleitung
- Grundlagen der Speicherhierarchie
- Cache: Aufbau und Verwaltung
- Virtueller Speicher
- Messen und Verbessern der Cache Performance
- Umsetzung am PC (Beispiele)
- Historisches

Idealer Weise würde man eine unbestimmt große Speicherkapazität anlegen, so daß jedes einzelne Wort sofort verfügbar ist...
Wir sind durch die Erkennung der Konstruktionsmöglichkeiten gezwungen, eine Speicherhierarchie zu finden, in der jede Stufe eine größere Kapazität als ihr Vorgänger hat, hieran gekoppelt ist allerdings eine längere Zugriffszeit.

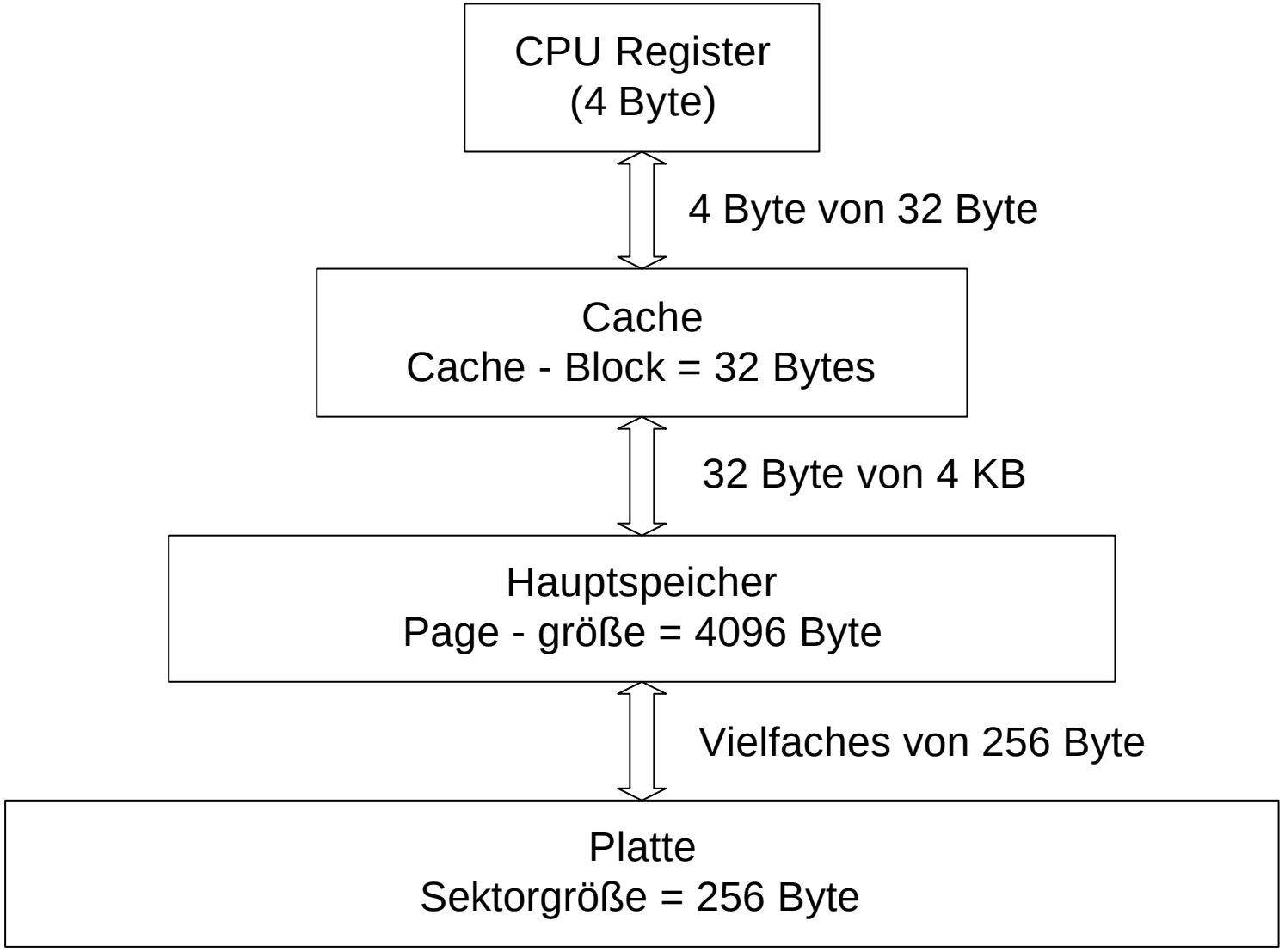
A.W. Burks, H. H. Goldstine, and J. von Neumann
Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946

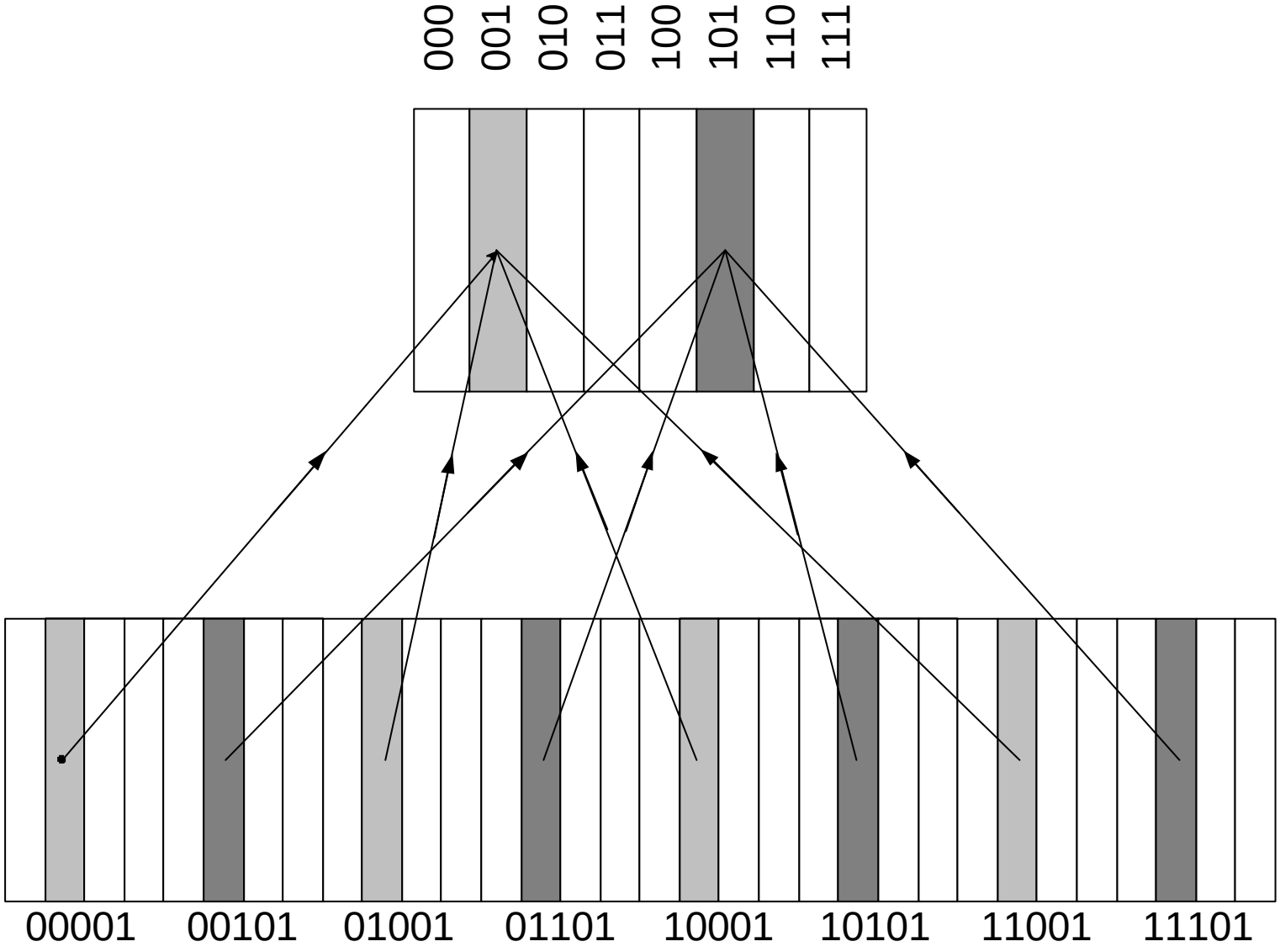

Bücherregale
Stuhl
Bücherregale
Schreib-
tisch
Stuhl

Speichertyp	Technologie	Größe	Zugriffszeit
Register		1 KB	1 ns
Cache	Statisches Halbleiter - RAM	512 KB	10 ns
Hauptspeicher	Dynamisches Halbleiter - RAM	128 MB	50 ns
Magnetische Platte	Festplatte	40 GB	10 ms
Optische Platte	CD-ROM	650 MB	300 ms
Archivband	Magnetband	100 MB - 5 GB	5 sec - 30 Min



Speed	1ns	1-3 ns	10 ns	10 ms
Size	100	Kilo	Mega	Giga
Cost	Highest			Lowest


CPU Register
(4 Byte)
4 Byte von 32 Byte
Cache
Cache - Block = 32 Bytes
32 Byte von 4 KB
Hauptspeicher
Page - größe = 4096 Byte
Vielfaches von 256 Byte
Platte
Sektorgröße = 256 Byte


000
001
010
011
100
101
110
111
00001
00101
01001
01101
10001
10101
11001
11101

Mi 13.12.2000

Decimal address of reference	Binary address of reference	Hit or miss in Cache	Assigned cache block (where found or placed)
22	10110two	miss	(10110two mod 8)=110two
26	11010two	miss	(11010two mod 8)=010two
22	10110two	Hit	(10110two mod 8)=110two
26	11010two	Hit	(11010two mod 8)=010two
16	10000two	miss	(10000two mod 8)=000two
3	00011two	miss	(00011two mod 8)=011two
16	10000two	hit	(10000two mod 8)=000two
18	10010two	miss	(10010two mod 8)=010two

INDEX	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

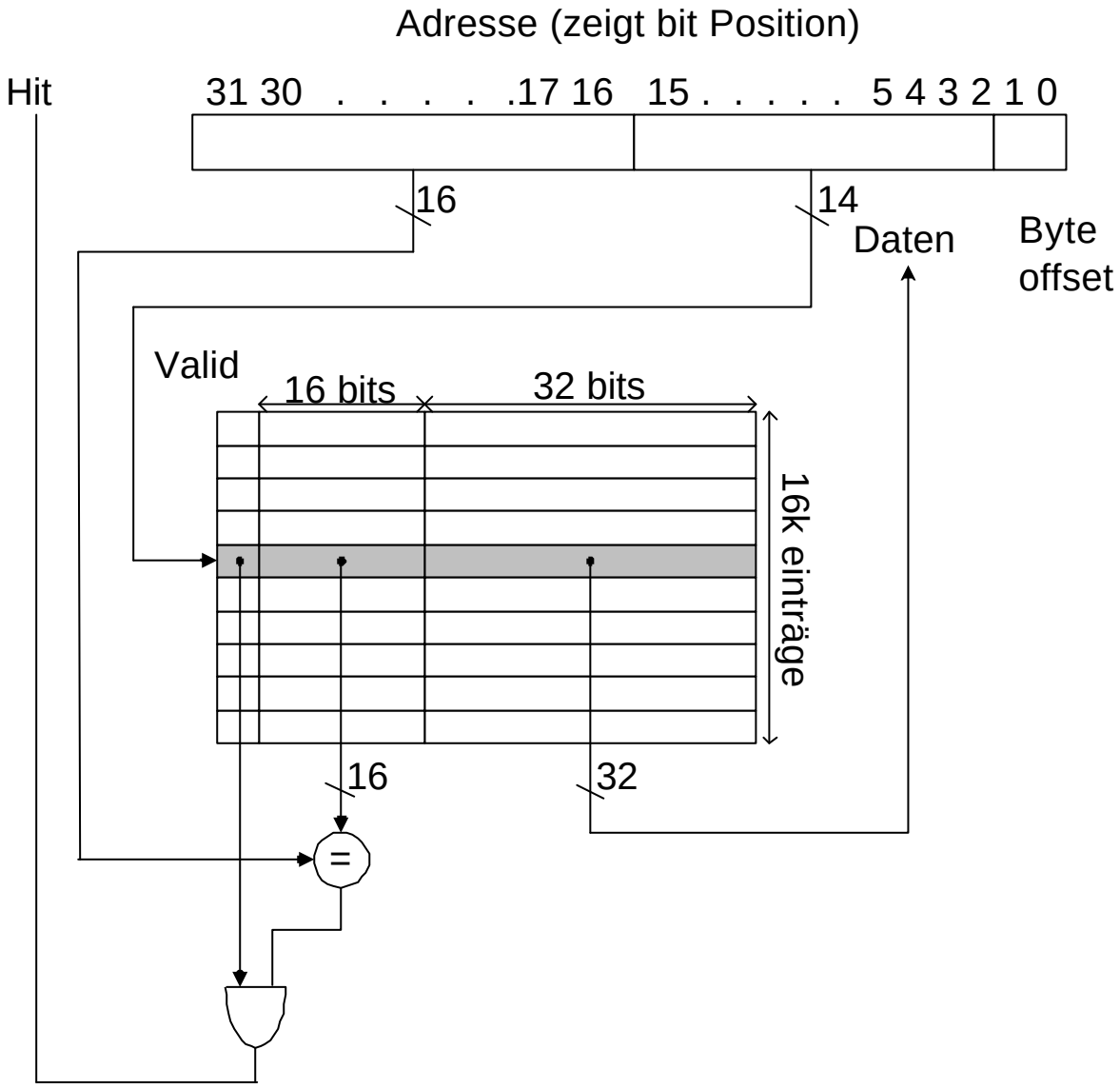
V:= Valid

Prinzipielle Funktionsweise:
Lesezugriff
1. Lese Datum aus dem Arbeitsspeicher unter Adresse *address*
2. CPU überprüft, ob eine Kopie der Hauptspeicherzelle *address* im Cache abgelegt ist.

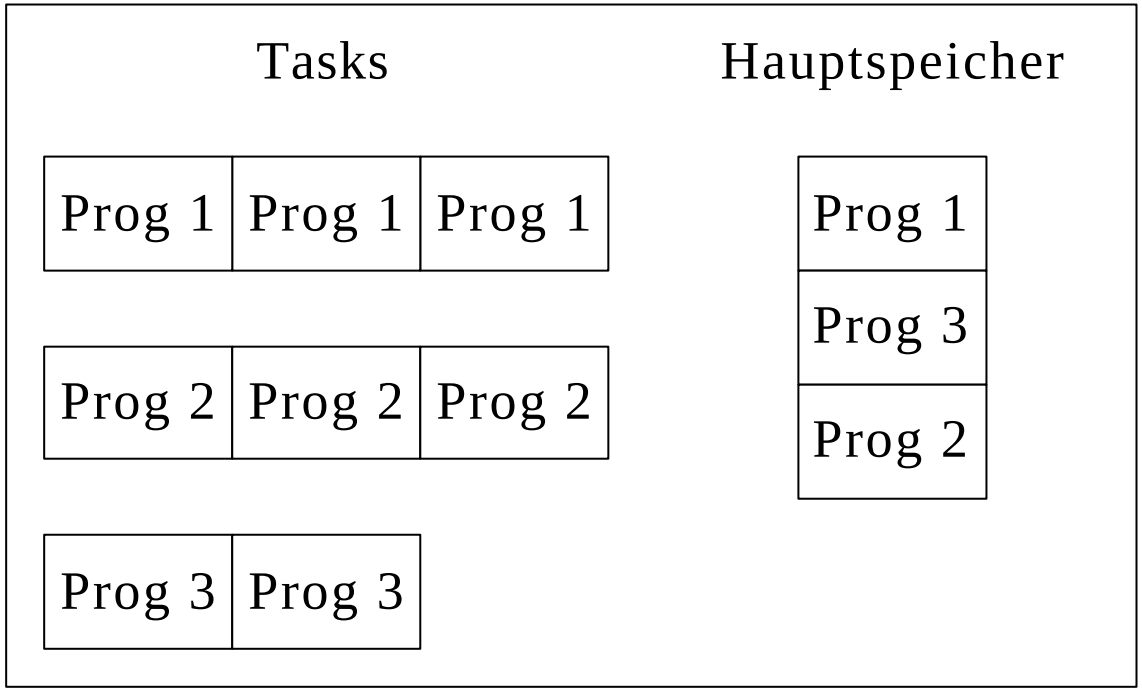
- Falls ja (**cache hit**)
  - so entnimmt die CPU das Datum aus dem Cache.
    Die Überprüfung und das eigentliche Lesen aus dem Cache erfolgt in einem Zyklus, ohne ein Wartezyklus einfügen zu müssen.
- Falls nein (**cache miss**),
  - so greift die CPU auf den Arbeitsspeicher zu
  - lädt den umgebenden Block des Datums in den Cache und
  - lädt das Datum von dort in die CPU.

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *address* im Cache abgelegt ist.

- Falls nein (**cache miss**)
  - CPU schreibt das Datum in die Hauptspeicherzelle mit Adresse *address.* Der Inhalt des Cache wird nicht verändert.
- Falls ja (**cache hit**):
  - die Kopie der Hauptspeicherzelle im Cache wird aktualisiert
  - **write-through Verfahren**: Hauptspeicherzelle wird sofort aktualisiert
  - **write-back Verfahren**: Hauptspeicherzelle wird erst später aktualisiert, nämlich wenn die Kopie aus dem Cache verdrängt wird.

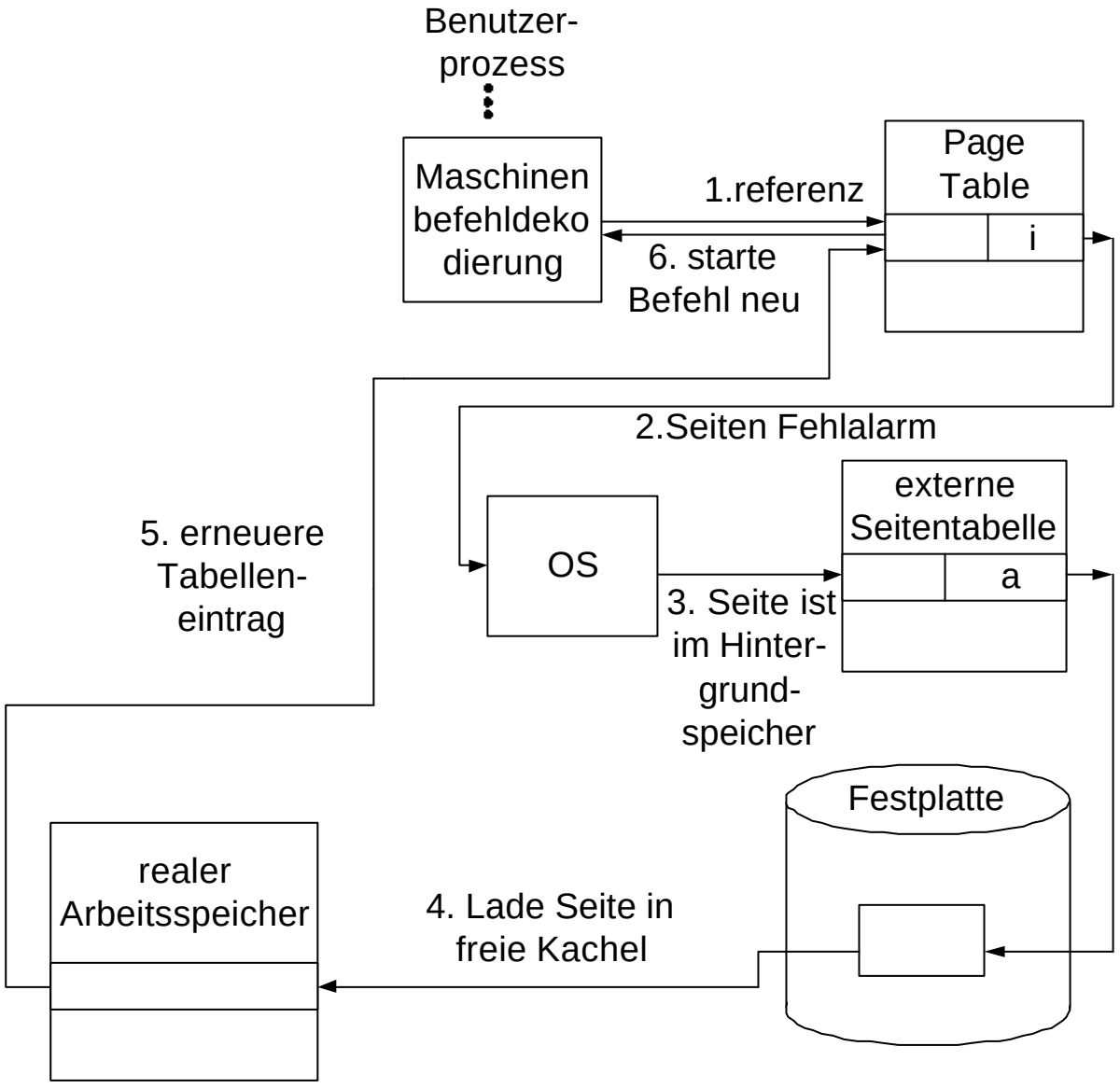

Adresse (zeigt bit Position)
Hit
31 30 . . . . .17 16 15 . . . . . 5 4 3 2 1 0
16
14
Daten
Byte
offset
Valid
16 bits
32 bits
16k einträge
16
32
=

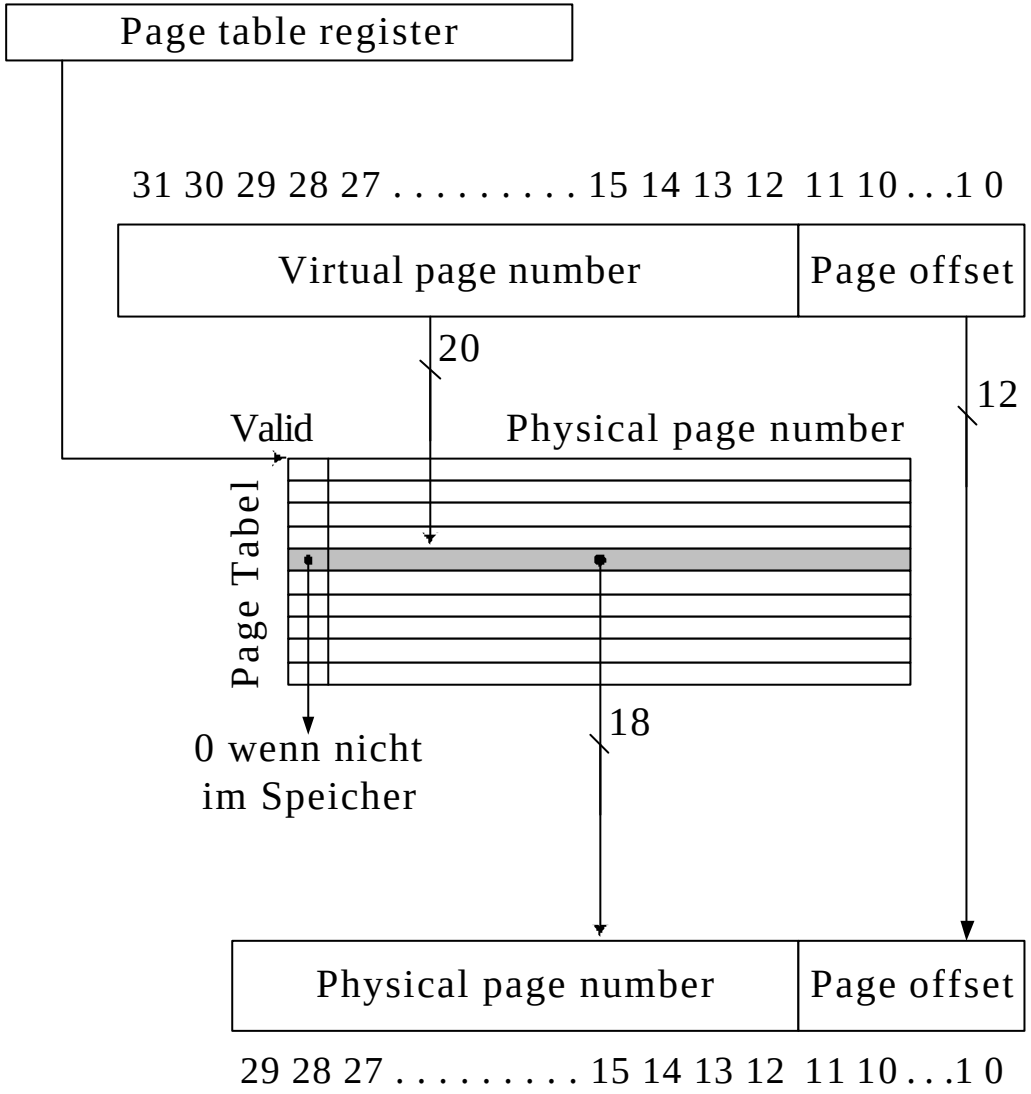
Mi 13.12.2000


Tasks
Hauptspeicher
Prog 1
Prog 1
Prog 1
Prog 2
Prog 2
Prog 2
Prog 3
Prog 3
Prog 1
Prog 3
Prog 2

Mi 13.12.2000

Da der virtuelle Speicher größer als der dem Prozess verfügbare Hauptspeicher sein kann, kann es vorkommen, dass eine Seite nicht im Hauptspeicher zu finden ist, d.h. sie ist auf die Festplatte ausgelagert. In diesem Fall muß die Seite von der Festplatte geholt, und in eine freie Kachel kopiert werden.




Page table register
31 30 29 28 27 . . . . . 15 14 13 12 11 10 . . .1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page Tabel
18
0 wenn nicht
im Speicher
Physical page number
Page offset
29 28 27 . . . . . 15 14 13 12 11 10 . . .1 0

Mi 13.12.2000

Messen und Verbessern der Cache Performance

Ansätze:

1.) Reduzierung der Fehlerrate, durch die reduzierung der Wahrscheinlichkeit, dass 2 verschiedene Memory Blöcke auf die selbe Stelle im Cache zugreifen.

2.) Reduzierung der Fehlerwartezeit, durch hinzufügen einer weiteren Ebene in Hierarchie.(multilevel Caching)

Aufteilung der CPU Zeit :

CPU time = (CPU execution clock cycles + Memory-stall clock cycles) x Clock cycle time

Memory-stall clock cycles = read-stall cycles + write-stall cycles

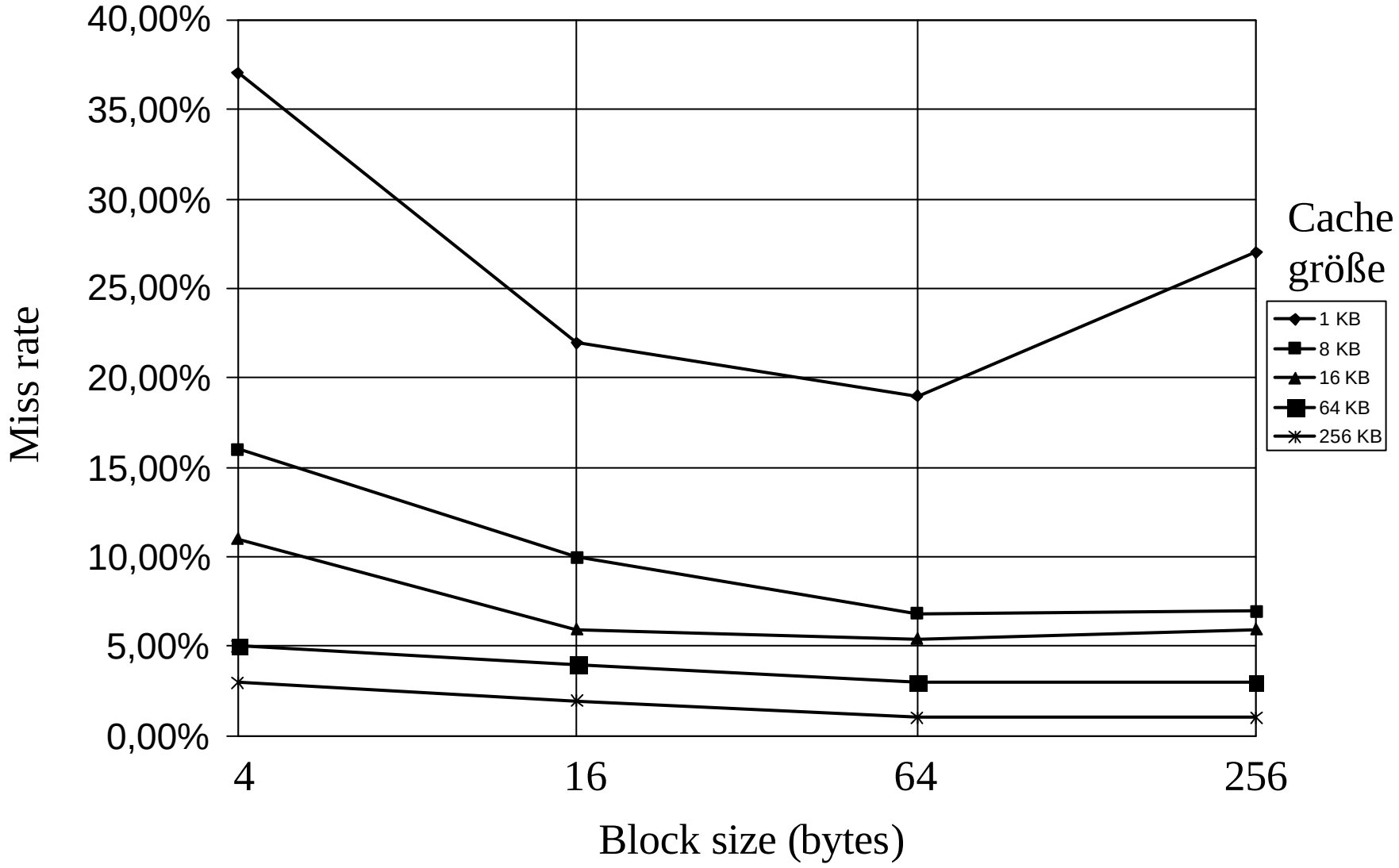
Read-stall cycles = ($\frac{READS}{PROGRAM}$ x Read miss rate x Raed miss penalty)

Write-stall cycles = ($\frac{WRITES}{PROGRAM}$ x Write miss rate x Write miss penalty) + write buffer stalls

Unter der Anahme das die Schreibpuffer Unterbrechung unwesentlich ist, können wir das Lesen und Schreiben durch das Verwenden einer einzelnen Miss-rate und Miss penalty vereiniegen

Memory-stall clock cycles = $\frac{MEMORY\ \ ACCESSES}{PROGRAM}$ x Miss rate x Miss penalty

Wir können auch schreiben:


40,00%
35,00%
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
Miss rate
4
16
64
256
Block size (bytes)
Cache größe
1 KB
8 KB
16 KB
64 KB
256 KB

Einfach assoziativer Cache
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Zweifach assoziativer Cache

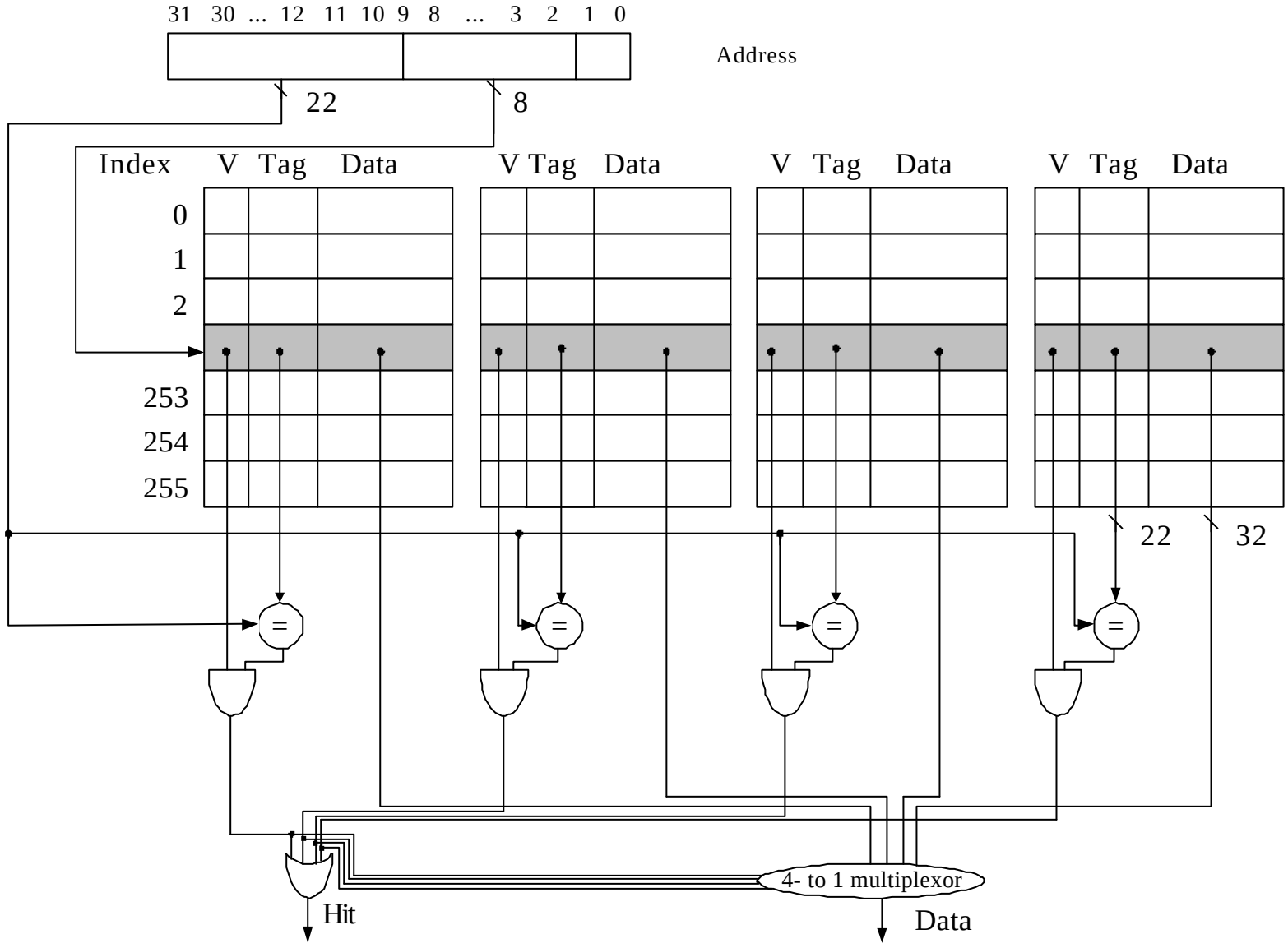
Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Vierfach assoziativer Cach

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Achtfach assoziativer Cach (full associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data


31 30 ... 12 11 10 9 8 ... 3 2 1 0
Address
22
8
Index
V Tag Data
V Tag Data
V Tag Data
V Tag Data
0
1
2
253
254
255
22
32
=
=
=
=
4- to 1 multiplexor
Hit
Data

Merkmale	Power PC 604	Intel Pentium Pro
Jahr :	1997	1997
Nutzung : (Firmen)	Apple, IBM	Dell, DEC, Gateway, Hp, Intergraph, Micron
Modelle	Mac(7600,8500,9500) IBM(RS/6000)	High end Pc desktops,Server
Taktfrequenz :	100 - 250 MHz	150 - 250 MHz
2nd Level Cache :	256-KB später 512KB	256-KB oder 512-KB on die
Virtuelle Adresse :	52 Bit	32 Bit
Physikalische Adr.:	32 Bit	32 Bit
Page Size :	4KB, ausw. & 256MB	4KB, 4MB
1st Level Cache:	16 KB each	8 KB each
Block size :	32 byte	32 byte
Write police :	Write back or write through	Write back
Replacement :	LRU	LRU Annäherung

Technik	Größe	Zeit	Beschreibung
Vakuum-Röhren	20 Register	> 1950	
Quecksilber delay lines	0,5 Kbit	> 1950	Elektrische Signale konvertiert in Vibrationen werden durch eine Quecksilberröhre geleitet. Am anderen Ende wird das Signal ausgelesen und zurück in die Röhre geleitet.
Eisenkern	32 Kbit	1950	Nutzte einen Eisenkern der magnetisiert werden konnte. (wie bei Harddisk)
Dram	1Kbit	1960	Integrierte Schaltkreise (derzeitig in Gebrauch)

Proseminar Mikroprozessoren Interfacing I/O Devices to the Memory, Processor and Operating System

Von Jan Milz

Gliederung

1. Einführung (What is I/O ?)
2. I/O Performance – Messungen
3. Typen und Eigenschaften von I/O Geräten
4. Busse
5. Functions the OS must provide
6. Giving Commands to I/O Devices
7. Communicating with the Processor – Polling, Interrupts,
8. DMA
9. DMA and Memory
10. Zusammenfassung
11. Future Directions in I/O Systems

Einführung

What are I/O Devices ?

Wir haben gesehen wie ein word oder ein Datenblock durch ein Busprotokoll übertragen wird. Es bleiben weitere Aufgaben offen, die verwirklicht werden muessen, wie z.B. Daten von einem Geraet in die Speicheradresse eines Programms zu transferieren. Folgende Fragen treten auf:

- Wie wird eine Benutzer I/O Anfrage in ein devicecommand umgewandelt und wie erfolgt die Kommunikation?

- Wie werden Daten in und aus dem Speicher gelesen?

- Welche Rolle spielt das Betriebssystem?

Drei Charakteristika eines I/O Systems:

1. Das I/O System teilen sich viele Programme, die auf den Prozessor zurückgreifen.

2. I/O Systeme benutzen Interrupts für I/O Operationen. Da Interrupts auf die Kernel-Ebene zugreifen, muss das OS sie steuern

3. Die Kontrolle eines I/O Gerätes ist sehr komplex, da viele Vorgänge gleichzeitig gesteuert werden muessen und die Anforderungen einer korrekten Kontrolle oft sehr detailliert sind

I/O Geräte

I/O-Performancemessungen

Will man die Performance von Disksystemen messen, muß man wissen, daß die Benchmarks, mit denen gemessen wird, von anderen Systemeinheiten beeinflußt werden, z. B. Speicher, Prozessor (im Gegensatz zu Prozessorbenchmarks).
Gemessen wird in MHz = 10^6 HZ.
Die Transferrate wird in MB/s gemessen : 1MB = 10^6B = 1.000.000 B
(Hauptspeicher : 1 MB = 2^20 B = 1.048.576 B).

Supercomputer I/O

Das Maß der Supercomputer I/Os ist die Anzahl der Bytes pro Sekunde zwischen Hauptspeicher und Diskette.

Transaktionprozedur I/O

Der Durchgang wird zeitabhängig gemessen.
Da die Zugänge klein sind, wird die I/ORate als Maß genommen (Maßeinheit: Anzahl der Diskettenaufrufe pro Sekunde).
Die Meßwerte sind kritisch zu betrachten und die Messung ist kostenintensiv.
Beispiel : die Benchmarks TPC-C , TPC-D
Beide stellen die Anfragenbearbeitung gegen die Datenbasis. Gemessen wird die Leistung in Transaktionen / Minute o. Sekunde , wobei ein komplettes System an der Messung beteiligt ist.
Der TPC-D wird bei komplexen Anfragen (typisch bei decision-support-Anweisungen) benutzt.
Der TPC-C (z.B. für Onlinebanking) enthält 9 Typen Databaserecords, 5 verschiedene Transaktionen und eine Benutzersimulation.

Dateiensysteme I/OBenchmarks

Messungen der Unix-Dateisysteme an einer arbeitenden Umgebung ergaben :
- 80% Zugriffe auf Dateien mit weniger als 10 KB
- 90% Zugriffe auf Daten mit sequentieller Adresse
- 67% Leseprozesse, 27% Schreibprozesse, 6%LeseVeränderеSchreibprozesse

Aus den Ergebnissen wurden synthetische Dateisystemen entwickelt => Benchmarks.

Typen und Eigenschaften von I/OGeräten

Es gibt eine sehr große Bandbreite von Typen, die sich nach drei Punkten einordnen lassen:
1. Verhalten : Input (nur lesen), Output (nur schreiben), Storage(lesen u. schreiben)
2. Partner : Mensch oder Maschine
3. Datenrate: höchste Datentransferrate zwischen I/OGerät und Hauptspeicher/Prozessor
   => Höchstrate des I/OGerätes muß berücksichtigt werden

Beispiele für I/OGeräte mit Blick auf Interaktion mit dem Prozessors

Maus

Es gibt zwei Arten von Mäusen, die einen generieren eine Serie von Impulsen, die anderen haben Zähler für die Höhe und Breite. Der Prozessor liest in bestimmten Abständen die Stände und überträgt sie auf den Mauszeiger.
Der Zeiger bewegt sich abrupt, da der Prozessor die Stände aber sehr oft liest, scheint sich der Zeiger stetig zu bewegen.
Beide Arten haben einen oder mehrere Knöpfe, und das System muß merken, wenn sie gedrückt wurden, ev. mit einer Unterscheidung zwischen drücken und gedrückt halten.
Mäuse werden von der Software kontrolliert, sind daher einstellbar.

Magnetische Disketten

Es gibt zwei Arten von Disketten : Floppy Disks (weich) und harte Disketten
Beide Typen beruhen auf dem gleichen Prinzip: eine rotierende Platte mit magnetischer Oberfläche und einem bewegbaren read/write-Kopf, wodurch die Speicherung stromunabhängig ist.
Früher wurden große Disketten mit vielen Bits hergestellt, was die Elektrokosten aufwog, da dieses Modell die niedrigsten Kosten pro Bit hatte.
Heute haben kleinere Disketten Vorteile bei der Leistung und dem Volumen/Byte. Sie sind billiger zu produzieren und lassen sich in hohen Stückzahlen absetzen.
=> 1997 kostete ein MB $0,10 - $0,20 , egal wie groß die Diskette war
Heute ist es weit verbreitet die Disk mit einer konstanten Bitdichte zu bespielen. Das heißt, daß die Dauer in welcher ein Sektor unter den Kopf gelangt, variiert (außen schneller).
Also muß ebenfalls die Rate, mit der gelesen und geschrieben wird, ebenfalls variieren.

Netzwerke.

- einige Eigenschaften : -Entfernung : 10m - 10.000km
  -Geschwindigkeit : 0,001 MB/s - 100 MB/s
  -Topologie : Bus, Ring, Stern, Baum
  -Leitungen : Punkt-zu-Punkt oder geteilte (multi drop) Leitungen

Beispiel: Langstrecken-Netzwerke
Die Entfernung beträgt 10- 10.000 km.
Das erste war ARPANET (Advanced Research Projects Agency of US.Government)
Es schickte 56 Kbits/s und nutzt die schon vorhandenen Punkt-zu-Punkt-Verbindungen, nämlich Telefonleitungen.
Der Hostcomputer kommuniziert mit einem IMP(Interface Message Prozessor) , der teilt die Nachricht in ein Kbit-Pakete, wobei die Adresse in jedem Paket enthalten ist, und schickt sie über verschiedene Kabel zu ihrem Bestimmungsort.
An jedem Knoten werden die Pakete gespeichert, bis sie beim Ziel-IMP wieder zusammengesetzt werden und zum Empfängerhost geschickt werden.
Die meisten Netzwerke nutzen diesen „packet-switched"-Ansatz, bei dem einzelne Pakete sich einen individuellen Weg zum Ziel suchen.
=> ARPANET ist der Vorläufer des Internets.

Es wurde eine Protokoll-Familie (TCP / IP) entwickelt, um verschiedene Netzwerke zu verbinden. Beim IP (Internet Protokoll) tauschen zwei Hosts ihre Adressen aus, aber man hat keine Garantie, daß der Transfer geklappt hat. Beim TCP (Transmission Control Protokoll) ist sicher gestellt, daß alle Pakete ankommen.
Sie arbeiten zusammen : IP-Pakete umschließen die TPC-Pakete

Busse

Sie verbinden die I/OGeräte mit Prozessor und Speicher und die Einheiten untereinander.
Ihr Vorteil ist die Vielseitigkeit, da neue Geräte können leicht angeschlossen werden
Der Nachteil ist, daß ein Engpaß entsteht, der den I/ODurchgang limitiert.

Leistungsoptimierung:

Die Geschwindigkeit der Busse kann nicht stark erhöht werden, da sie hauptsächlich von physikalischen Faktoren, Länge des Busses und Anzahl der I/OGeräte, abhängt.
Die Leistung dagegen ist steigerungsfähig, hat aber ev. negative Auswirkungen.
Beispiel :
Schnellere Antwort auf I/OOperationen erreicht man durch Minimieren der Zeit des Buszugangs. Andrerseits muß für hohe I/ODatenraten die Bandbreite maximiert werden, z. B. durch mehr Buffer und den Transport von größeren Datenblöcken, aber beides verzögert den Zugang des Busses !
=> Die beiden Ziele schneller Buszugang und hohe Bandbreite führen zu entgegengesetzten Designansprüchen
Ein Bus enthält Kontrollleitungen und Datenleitungen.
Kontrollleitungen stellen Anfragen, geben Bestätigungen und identifizieren den Typ der Daten auf der Datenleitung, welche die Information zwischen Quelle und Ziel transportiert.
Informationen bestehen aus Daten, komplexen Kommandos oder Adressen.
Beispiel : Diskette schreibt einen Sektor in den Speicher
Die Datenleitung transportiert sowohl die Speicheradresse als auch die Daten selber (nacheinander). Die Kontrollleitung zeigt die Art der Information an; sie ändert sich also während des Prozesses.
Einige Busse haben zwei Datenleitungen, eine für die Adressen, die andere für die Informationen.
Andere Busse nutzen die Kontrollleitung zusätzlich für ein Protokoll.

Bustypen

1. Prozessor-SpeicherBus
- kurz, schnell, maximiert die Prozessor-Speicherbandbreite
2. I/OBus
- können länger sein, keine festgelegten Endgeräte, viele verschiedene Bandbreiten
3. BackplaneBus
- wurden entworfen, um Prozessor, Speicher und I/OGeräten zu erlauben zu kommunizieren

Einige Hochleistungssysteme nutzen eine Kombination aus allen drei Bussen, wobei der P-S-Bus einen oder mehrere Adapter für den Standardbackplanebus hat , und die I/OBusse an den Backplanebus angestöpselt werden können.
Der Vorteile ist, daß der P-S Bus unabhängig von den anderen schneller gemacht werden kann und eine Ausweitung des Systems unabhängig vom P-S Bus möglich ist.
Beispiel : IBM RS/6000 , Silicon Graphics multiproz

Synchrone und asynchrone Busse

Synchrone Busse haben eine Uhr in der Kontrollleitung und ein festes Protokoll in Abhängigkeit zur Uhr.
Das Protokoll ist einfach umzusetzen, da es wenig Logik benötigt, daher ist der Bus schnell.
Nachteile: 1. Alle Geräte des Busses müssen aufeinander abgestimmt sein.
2. Wegen Taktproblemen können schnelle Busse nicht lang sein.
P-S Busse sind häufig synchron, da sie mit wenigen, nahen Geräten mit hoher Frequenz arbeiten.

Asynchrone Busse sind in der Länge unbeschränkt; es lassen sich viele verschiedene Geräte anschließen.
Zum Koordinieren des Datenaustausches wird ein Handschlagprotokoll benutzt, das heißt, daß beide Geräte erst zum nächsten Schritt übergehen, wenn beide ihre Zustimmung gegeben haben. Realisiert wird es durch zusätzliche Kontrollleitungen.
Asynchrone Busse arbeiten wie abgeschlossene Maschinen, die nur etwas tun, wenn sie wissen, daß der andere einen bestimmten Status hat .
I/OBusse sind oft asynchron, damit neue Geräte leicht angeschlossen werden können.

Erhöhen der Busbandbreite
Viel der Bandbreite wird durch die Wahl zwischen asynchron und synchron entschieden.
1. Wird die Datenbusbreite erhöht, dauert der Transfer von mehreren Wörtern weniger Buszyklen.
2. Durch getrennte Adress- und Datenleitungen können Adresse und Information in einem Zyklus geschickt werden.
3. Blocktransfers : Man erlaubt dem Bus viele Wörter ohne Adresse oder Bestätigung zwischendurch in Back-to-Back Bussen zu transportieren
Die Nachteile sind mehr Busleitungen, erhöhte Komplexität und / oder erhöhte Antwortzeit bei Anfragen, die ev. warten müssen, während ein längerer Transfer läuft

Verteilen des Buszugriffs
Wie kann ein Gerät sich einen Bus reservieren ?
Entscheidend bei großen I/OSystemen ist es, daß die I/OGeräte ohne große Einmischung des Prozessors arbeiten
Es wird also eine Kontrollinstanz benötigt, die den Zugriff auf den Bus regelt.
Der Busmaster kontrolliert alle Anfragen und schickt die Busse los.
Der Prozessor muß trotzdem in der Lage sein, Busse zu rufen oder zu schicken, wobei er dann
der Busmaster ist => einfachstes System
Nachteil : Der Prozessor ist an jeder Transaktion beteiligt.
Bei mehreren möglichen Busmastern muß entschieden werden, wer den Bus zuerst benutzen darf.

Busvermittlung (Arbitration)
Es gibt viele verschiedene Schemata für die Vermittlung, z. B. durch spezielle Hardware oder ausgeklügelte Protokolle.
Ein Gerät signalisiert eine Anforderung; nach Gebrauch des Busses wird wieder ein Signal gegeben, daß der Bus frei ist.
Die meisten Multiplemasterbusse haben ein extra Leitungen für Anfragen und Freigabe
Haben nicht alle Geräte ihre eigenen Anfragenleitungen, braucht man eine zusätzliche Releaseline.

Allgemein berücksichtigen Schemata zwei Faktoren :
1. Priorität : die Geräte bekommen verschiedene Prioritäten zugewiesen
2. Fairness : auch das Gerät mit der niedrigsten Priorität soll irgendwann bedient werden

Vermittlungsschemata können in vier Klassen eingeteilt werden :
1) Daisy Chain Arbitration

Die Grant(=Freigabe)leitung läuft vom Vermittler zum Gerät mit höchster Priorität, dann zum nächsten, bis zum letzten.
Möchte ein Gerät den Bus benutzen, schickt es ein Signal über die Anfrageleitung und wartet auf eine Übertragung auf der Grantline, die anzeigt, daß der Bus zugeteilt ist.
Dann wird das Grantsignal unterbrochen und die Anfrageleitung freigegebenen.
Das Gerät benutzt den Bus und setzt danach die Releaseline zum Signalisieren von : Fertig!
Dadurch, daß die Grantline unterbrochen wird, verhindert man, daß Geräte hoher Priorität einem Gerät niedriger Priorität, das glaubt die Freigabe erhalten zu haben, den Bus weg-schnappen können.
Fairness ist nicht implementiert.

2) Die zentrale parallele Vermittlung nutzt multiple Anfrageleitungen, wodurch die Geräte ihre Anforderungen unabhängig stellen. Der zentrale Vermittler sucht ein aus und teilt diesem den Bus zu.
Der PCI, ein Standard-Backplanebus nutzt dieses Schema.

3) Die geteilte Vermittlung durch Selbsternennung nutzt ebenfalls multiple Anfrageleitungen. Jedes Gerät, das den Bus nutzen will, sendet einen Identifizierungscode zum Bus und kommt unabhängig von der Priorität an die Reihe.

4) Bei der geteilten Vermittlung durch Kollisionsentdeckung fordert jedes Gerät unabhängig den Bus an. Gehen mehrere Anfragen auf einmal ein, kommt es zur Kollision, die nach einem festgelegtem Schema abgebaut wird.

<u>Busstandards</u>
Die meisten Computer lassen sich erweitern, daher hat die Industrie verschiedene Busstandards
geschaffen. Manchmal wird ein I/OBus einer besonders populären Maschine de facto Standard, wie z. B. : IBM PC-AT Bus, oder ein Standard wird von einer kleinen Gruppe für ein bestimmtes Problem festgelegt, z. B. SCSI (Small Computer System Interface)
- PCI (ursprünglich von Intel entwickelt) nutzt einen general purposed Backplanebus
- SCSI hat Backplanebusse und P-S Busse

Functions the OS must provide

Das Betriebsystem spielt eine große Rolle beim handling von I/O Vorgängen. Es ist das **Interface zwischen Hardware und Software**.

Die oben genannten Charakteristika führen zu vielen verschiedenen

Funktionen, die das OS zur bereitstellen muss:

- Das OS garantiert, dass ein Benutzerprogramm nur auf die Teile eines Gerätes zugreift, für die der Benutzer auch die Zugriffsrechte hat.
- Das OS stellt Unterprogramme (routines) bereit, die den Gerätezugriff steuern.
- Das OS steuert die Interrupts die von I/O devices ausgehen.
- Es versucht gleichwertigen Zugriff auf die geteilten I/O Ressourcen zu ermöglichen.

Um diese Funktionen zu Gunsten der Benutzerprogramme zu performen, muss das OS in der Lage sein mit den I/O Geräten zu kommunizieren und Benutzerprogramme am direkten Zugriff hindern.

Drei Arten der Kommunikation werden benötigt:

1. Das OS muss den I/Os Kommandos geben können. Diese sollen nicht nur Operationen wie read oder write enthalten, sondern auch eine Dateiensuche (diskseek) o.ä. implizieren können.

2. Das device muss in der Lage sein, dem OS eine vollständige Operation oder einen Fehler zu melden.

   – beendete Suche

3. Daten muessen zwischen Speicher und I/O device bewegt werden.
   - ein ausgelesener Block während eines diskreads muss von der harddisk in den Speicher bewegt werden.

Giving Comands to I/O Devices

Um einem I/O device einen Befehl zu geben, muss der Prozessor es adressieren. Es gibt zwei Methoden das Device zu adressieren.

- memory-mapped I/O
- special I/O instructions

Durch **memory-mapped I/O** werden Teile des Adressenspeichers genutzt, um durch reads and writes in den Adressen Kommandos an die devices zu realisieren.

Dabei wird z.B. ein write genutzt, um Daten an das Geraet zu senden, wo diese als Kommando interpretiert werden.

Wenn der Prozessor die Adresse und die Daten in den Memory Bus schickt, ignoriert der Speicher die Operation, weil die Adresse einen Teil des Speichers anzeigt, der für I/O reserviert ist.

Der device-controller bemerkt den Vorgang und leitet die Daten (however) an die Geräte weiter.

Benutzerprogramme können hier nicht stören, da der I/O-Adressspeicher durch das OS geschützt ist.

Es sind mehrere seperate I/O Operationen notwendig, um eine solche Programmabfrage zu verwirklichen.

Der Prozessor muss den Status der Geräte abfragen können.

Beispiel: DEC LP11
Dieser Drucker hat zwei Register. Eines für Statusinformationen und eines für Daten, die gedruckt werden sollen. Das Statusregister enthält ein done-bit, das gesetzt wird, wenn ein Buchstabe gedruckt wurde und ein error-bit, das gesetzt wird, wenn der Drucker Störungen oder kein Papier mehr hat. Alle Daten, die gedruckt werden sollen, werden in das Datenregister geschrieben. Der Prozessor muss warten, bis das done-bit
gesetzt ist, um einen weiteren Buchstaben in den Buffer zu laden. Außerdem muss er das error-bit checken, um zu stoppen, falls ein Fehler auftritt. Jede dieser Aktionen braucht eine Deviceadresse.

Special I/O Instructions:

Diese Instructionen können sowohl die devicenumber als auch das command word spezifizieren. Der Prozessor spricht die Deviceadresse über den I/O bus an.
Um unerlaubten Zugriff zu verhindern, können I/O Instruktionen nur in supervisor oder kernel mode gesteuert werden.

- z.B. Intel 80x86 oder IBM 370 benutzen dedicated I/O instructions.

Der periodische Check eines Statusbits (beim eben genannten Drucker) wird **polling** genannt.

Communicating with the Processor

Polling:

- der einfachste Weg der Kommunikation zwischen device und Prozessor.
- Informationen werden im Statusregister platziert und werden vom Prozessor interpretiert.

→ Der Prozessor hat die gesamte Kontrolle aber auch die ganze Arbeit.

Nachteile:

- Verschwendung von Prozessorleistung, da Prozessoren viel schneller als I/O devices sind.
- Polling muss kontinuierlich weiterlaufen, da keinen Signale vom Gerät selbst ausgehen. Das OS(Prozessor) muss warten bis das Gerät auf den poll reagiert.

Interrupts

- Unterbrechung des Prozessors durch device, das aktiv werden will.
- OS gesteuert.
- Es gibt eine Interrupt-Hierarchy bei vielen parallel auftretenden Interrupts (Prioritäten)

→ Der Prozessor braucht nicht mehr zu pollen, kann sich auf das auszuführende Programm konzentrieren.

Interrupt-Algorithmus:

1) Unterbrechungssignal tritt auf (interrupt recognition).

2) Per Hardware wird geprüft, ob Berechtigung vorliegt. Wenn nicht, interrupt wirkungslos. (interrupt masking)

3) Wenn ja, wird aktives Programm(Zählerstand, Registerinhalte, Status) gesichert. (save status)

4) Unterbrechungsursache wird ermittelt und

   - Der Unterbrecher erhält Rückmeldung, dass der interrupt angekommen ist (interrupt acknowledge)
   - Ein zweites Programm wird gestartet und erledigt notwendige Systemarbeiten. (interrupt sevice routine)

5) Wenn Interrupt erledigt, wird altes Programm wieder geladen und fortgesetzt. (retore and return)

Der Prozessor soll aus der Kette Speicher↔Prozessor↔Device
Entfernt werden.
Zusätzliche hardware wird benötigt:

DMA - Direct memory acces

- Direktspeicherzugriff: unmittelbarer Zugriff eins peripheren Gerätes auf den Speicher.
- Sitzt zwischen I/O controller und Speicher
- Es gibt eine speziellen **DMA-controller**, der ohne die CPU Daten vom Arbeitspeicher über den Datenbus auf ein Gerät oder umgekehrt transportieren kann.
- Interrupts werden nur noch gebraucht, um dem Prozessor Fehler zu melden.

Drei Schritte während eines DMA- Transfers:

1) Der Prozessor teilt dem DMA das device, die auszuführende Operation, die Speicheradresse, aus der die Daten gelesen werden sollen, und die Anzahl der bytes, die transferriert werden sollen, mit.

2) Der DMA startet die Operation im device und bestimmt den bus. Wenn die Daten verfügbar sind(device oder Speicher), werden sie übertragen. Wenn der Vorgang mehr als einen Transfer benötigt, wird der nächste vorbereitet während der erste läuft. Manche DMAs haben einen eigenen Speicher, um flexibel mit Verzögerungen(delays) umgehen zu können, während sie auf den Bus warten.

3) Wenn der Transfer komplett ist, interruptiert der Kontroller den Prozessor.

DMA and the Memory Sytem

- Beziehung zwischen Speicher und Prozessor ändert sich. Ohne DMA greift der Prozessor direkt auf den Speicher zu.
- Ein neuer Weg zum Speicher entsteht, der nicht über Adressen und oder Cache läuft.
- Übertragungsschwierigkeiten treten auf, die mit Hilfe bestimmter Hard- und Software gelöst werden müssen.

In einem System mit virtuellem Speicher können Daten doppelt auftreten:
Der DMA greift direkt auf den Hauptspeicher zu, während der Prozessor den Cache mit einbezieht. So können gleiche Speicheradressen vom DMA und vom Prozessor unterschiedliche Werte annehmen(stale data problem or coherency problem).

Multiprozessoren

Gregor Michalicek Marcus Schüler

Vorteile gegenüber Singleprozessoren

- Multiprozessoren sind zuverlässiger. Einige Multiprozessorsysteme können trotz defekter Hardware weiterhin korrekt weiterarbeiten, wenn nämlich ein einzelner Prozessor in einem System mit n Prozessoren ausfällt, arbeitet es mit n-1 Prozessoren weiter

- Fällt ein Single-Chip Rechner aus, muß er in einem teuren und zeitlich aufwendigen Prozeß ersetzt werden.
  Ein Multiprozessorsystem kann in einzelnen Schritten durch Austausch einzelner Komponenten repariert oder erweitert werden, ohne eine komplette Systemunterbrechung herbeizuführen.

- Multiprozessoren besitzen die höchste absolute Leistung – schneller als jeder Single-Chip-Prozessor

- Ein Multiprozessor aus mehreren einzelnen Prozessoren ist effektiver als ein high-performance Singleprozessor mit einer neu entwickelten Technologie.

- Mehr Rechenleistung wird einfach durch die Erhöhung der Anzahl der Prozessoren erreicht. Der Verbraucher ordert soviel Prozessoren wie das Budget erlaubt und erhält die dazu vergleichbare Leistung.

Warum setzen sich Multiprozessoren nicht durch ?!

- Nur wenige wichtige Programme wurden bisher für Multiprozessoren umgeschrieben.
- Viele Programme können prinzipiell nicht durch Multiprozessoren beschleunigt werden.
- Programmierung von Programmen für Multiprozessoren ist schwierig, da der Programmierer z.B. die zugrundeliegende Hardware genau kennen muß.
- Programme sind nicht einfach von einem Multiprozessor auf den nächsten übertragbar.

Wie stark kann ein Programm durch Multiprozessoren beschleunigt werden?

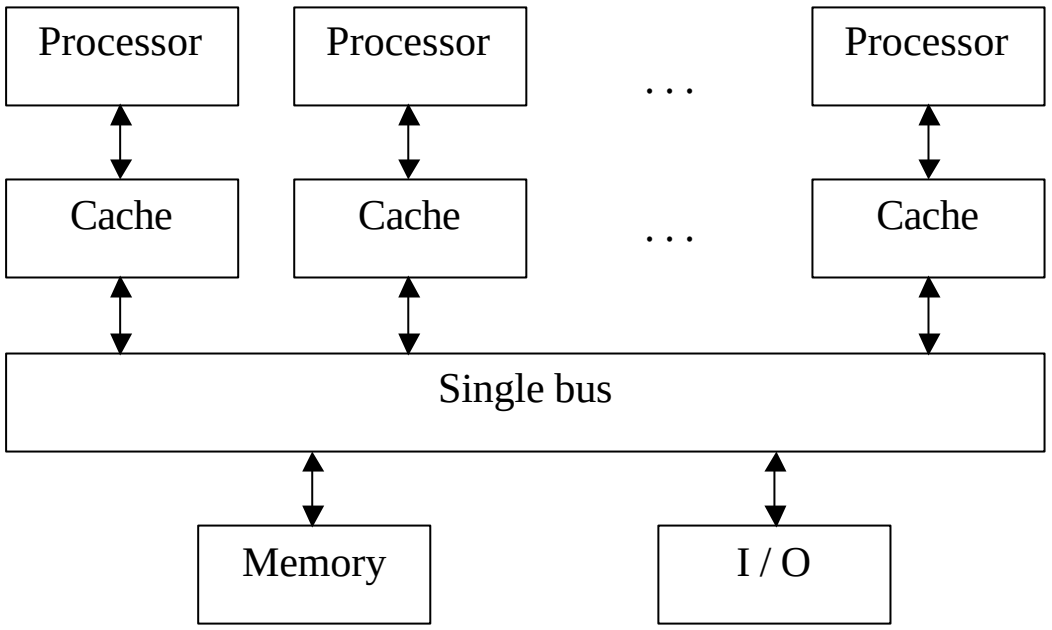
- Amdahl's Gesetz :

$$Zeit\_nach\_Optimierung = \frac{optimierter\_Anteil}{Verbesserungsfaktor} + restliche\_Zeit$$

- Es können nur die Teile eines Programms durch Multiprozessoren beschleunigt werden, die parallel berechnet werden können.
- Sequentielle Programmteile können nicht beschleunigt werden.

>>> Ein Programm darf keine sequentiellen Teile enthalten, damit es mit steigender Anzahl der Prozessoren (in einem Multiprozessor) linear beschleunigt werden kann.

Single-bus Multiprozessor



Wie viele Prozessoren sind sinnvoll?

- Die Bandbreite des Busses beschränkt die sinnvolle Anzahl an Prozessoren.
- Die Menge an Daten, die jeder Prozessor über den Bus schickt beschränkt die sinnvolle Anzahl an Prozessoren.
- Der Bus wird durch die Caches entlastet.
- Single-bus Multiprozessoren bestehen normalerweise aus 2 bis 32 Prozessoren
- Die maximale sinnvolle Anzahl an Prozessoren bei Single-bus Multiprozessoren scheint mit der Zeit abzunehmen.

Eigenschaften von Single-bus Multiprozessoren (1997)

Name	Maximale Anzahl an Prozessoren	Prozessor	Maximale Größe des Speichers / System	Kommunikations-Bandbreite / System
Compaq ProLiant 5000	4	Pentium Pro 200 MHz	2 GB	540 MB / s
Digital AlphaServer 8400	12	Alpha 21164 440 MHz	28 GB	2150 MB / s
HP 9000 K460	4	Pa-8000 180 MHz	4 GB	960 MB / s
IBM RS/6000 R40	8	PowerPC 604 112 MHz	2 GB	1800 MB / s
SGI Power Challenge	36	MIPS R10000 195 MHz	16 GB	1200 MB / s
Sun Enterprise 6000	30	UltraSPARC 1 167 MHz	30 GB	2600 MB / s

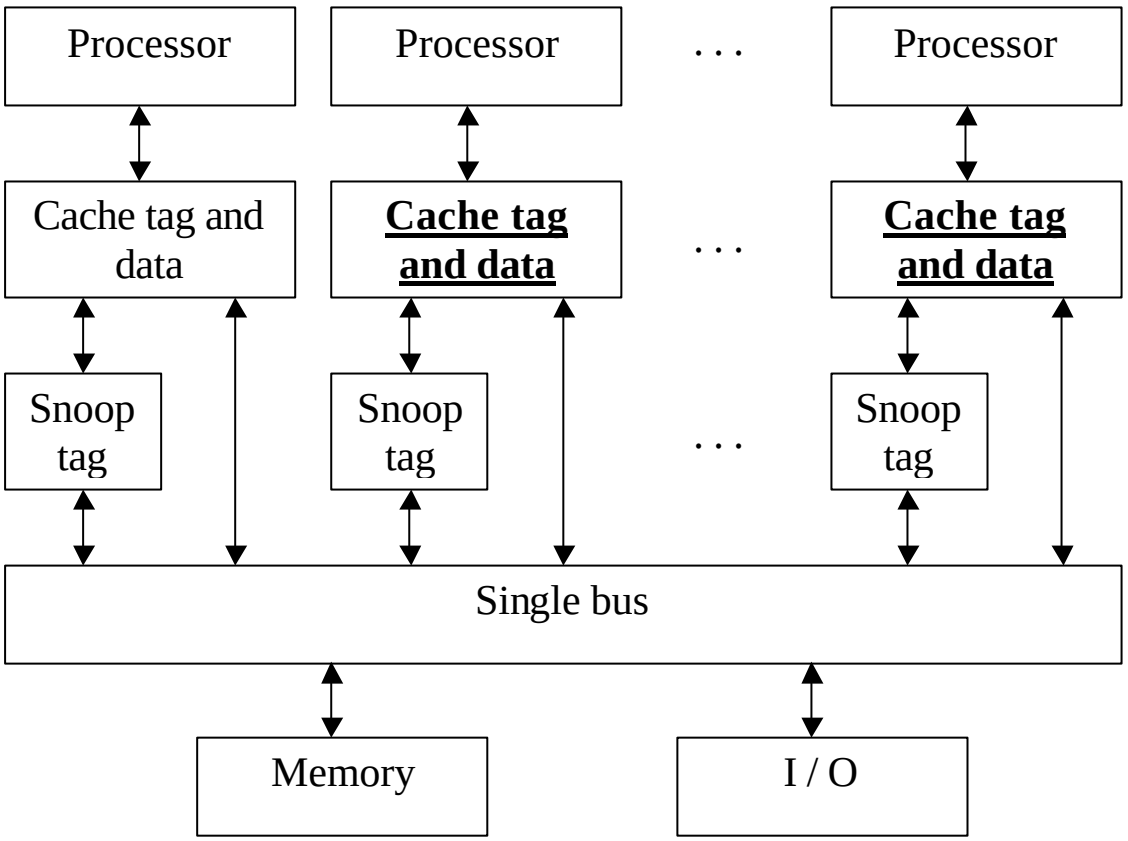
---

Cache Coherency bei Multiprozessoren

Problem : Wie kann sichergestellt werden, daß die Daten in den einzelnen Caches nicht veraltet sind?!

- Ein Prozessor könnte, z.B. durch eine Berechnung, einen neuen Wert für eine Variable erhalten und diesen in seinem Cache und dem Speicher speichern. In den anderen Caches würde der alte Wert gespeichert bleiben.

Cache Coherency durch Snooping



Zwei Arten von Snooping-Protokollen :

1. Write-invalidate :

- Der schreibende Prozessor erklärt die anderen Kopien der betroffenen Daten für ungültig und schreibt die neuen Daten dann in seinen Cache.

2. Write-update :

- Der schreibende Prozessor schreibt die neuen Daten über alle Kopien der betroffenen Daten.

- Normalerweise wird ein *write-invalidate* Protokoll in Verbindung mit *write-back* Cache verwendet, um den Bus zu entlasten.

Was passiert, wenn zwei Prozessoren im gleichen Taktzyklus auf den gleichen geteilten Speicherblock schreiben wollen?!

1. Einem der beiden Prozessoren wird zuerst der Bus zugeteilt.
2. Der erste Prozessor schreibt seine Daten.
3. Dem zweiten Prozessor wird der Bus zugeteilt.
4. Der zweite Prozessor schreibt seine Daten.

- Durch den Bus wird also ein sequentielles Verhalten beim Schreiben erzwungen. Nur so ist es möglich, daß das Schreiben mehrerer Prozessoren in den gleichen Speicherblock korrekt funktioniert.

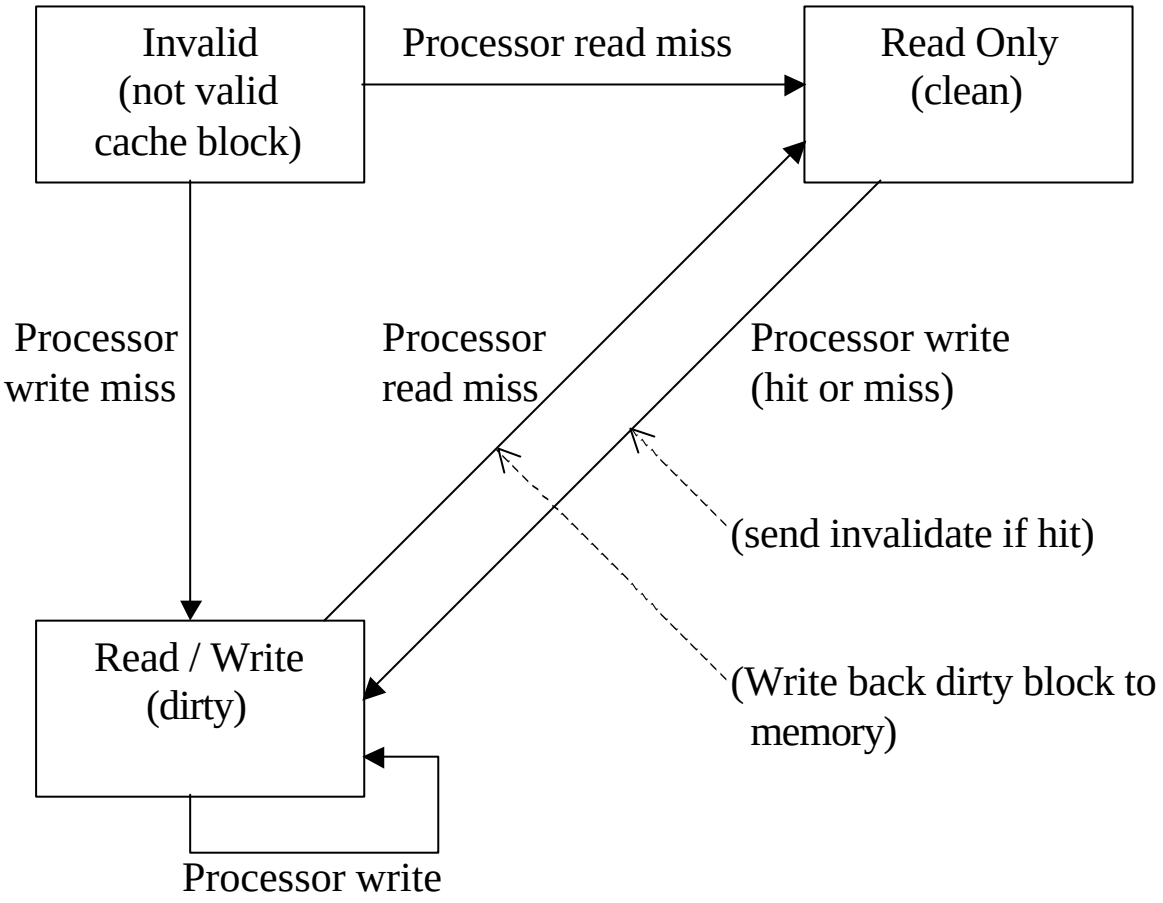
Beispiel für ein Cache Coherency Protokoll

- Jeder Cache-Block ist in einem von drei verschiedenen Zuständen :

  1. ***Read Only*** : Dieser Cache-Block ist nicht beschrieben (*clean*) und kann geteilt werden.
  2. ***Read / Write*** : Dieser Cache-Block ist beschrieben (*dirty*) und kann nicht geteilt werden.
  3. ***Invalid*** : Dieser Cache-Block enthält keine gültigen Daten.

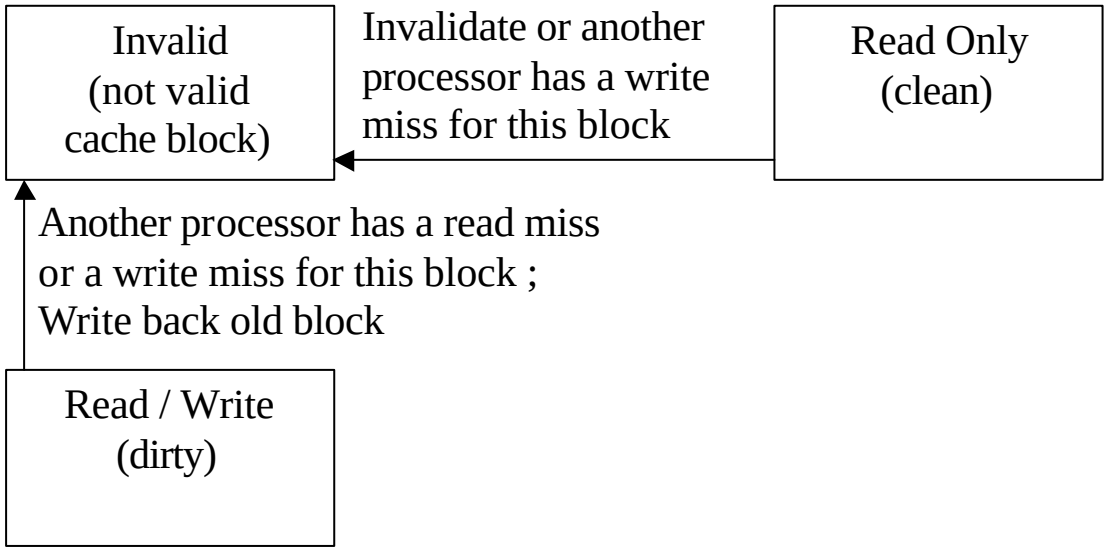
Beispiel für ein Cache Coherency Protokoll (Fortsetzung)

a) Cache-Status Veränderungen durch den zugehörigen Prozessor :



---

b) Cache-Status Veränderungen durch einen anderen Prozessor :



Beispiel für ein Cache Coherency Protokoll (Fortsetzung)

Wann kommt es zu Cache-Status-Veränderungen?

- Der Cache-Status wird verändert, wenn ein Leseversuch nicht funktioniert (*read miss*).
    - Der Cache-Status wird verändert, wenn ein Schreibversuch nicht funktioniert (*write miss*).
    - Der Cache-Status wird verändert, wenn ein Schreibversuch funktioniert (*write hit*).

Was passiert bei einem *read miss*?

1. Der Status des Cache-Blocks wird auf *Read Only* gesetzt.
2. Wenn der alte Cache-Block auf *Read / Write* gesetzt war, so schreibt der Cache den alten Block in den Speicher.
3. Alle Caches der anderen Prozessoren überprüfen, ob sie eine Kopie des betroffenen Cache-Blocks haben, der auf *Read / Write* gesetzt ist. Ist dies der Fall, so setzen sie ihn auf *Invalid* (oder *Read only*).

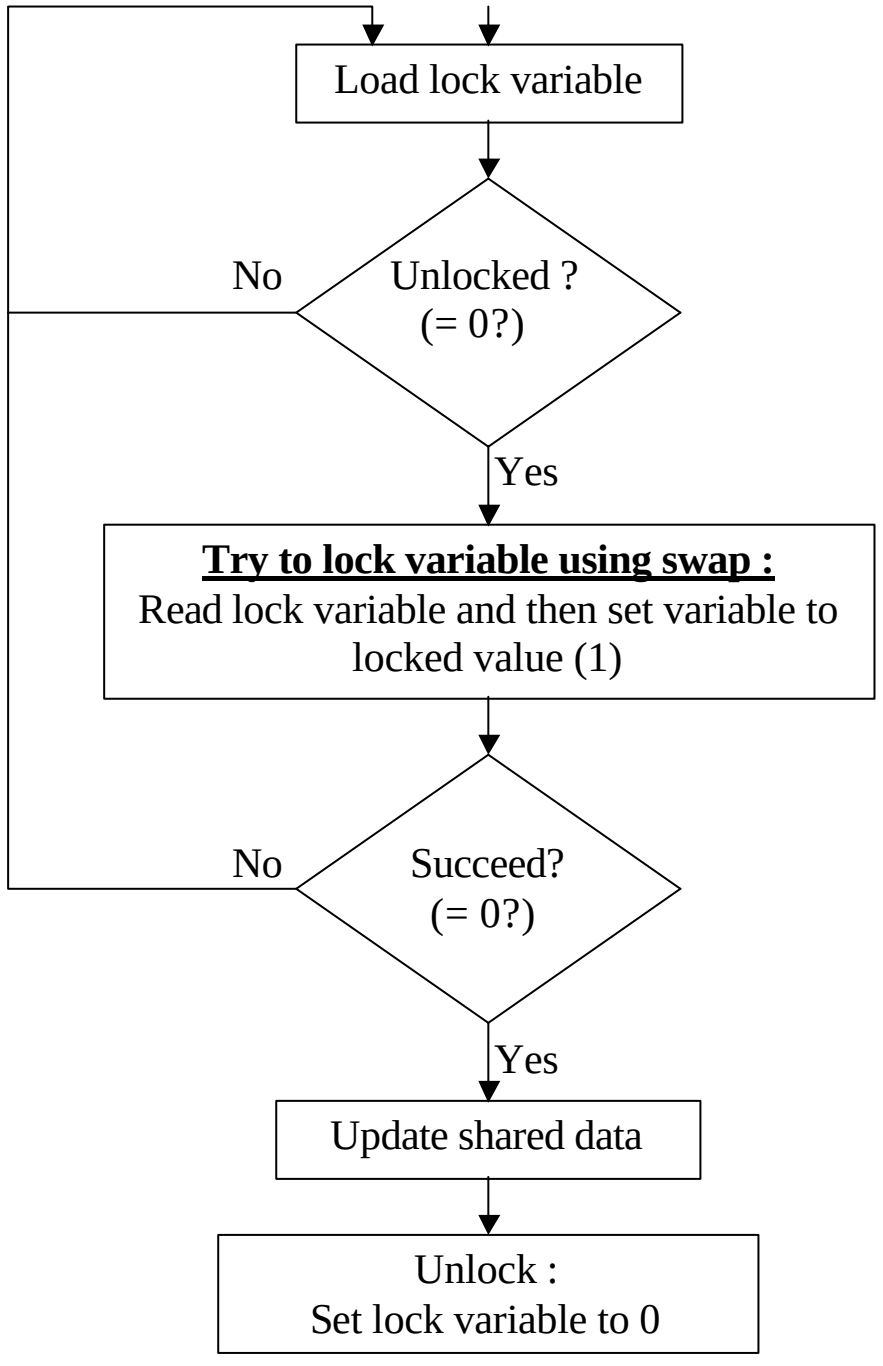
Was passiert, wenn der Prozessor auf einen Cache-Block schreiben will?

1. Der Prozessor sendet ein *Invalidate*-Signal über den Bus. Die Caches der anderen Prozessoren überprüfen, ob sie eine Kopie des Blocks haben und setzen diese, falls vorhanden, auf *Invalid*.
2. Der schreibende Prozessor schreibt in seinen Cache-Block und setzt ihn auf *Read / Write*.

Durch *Cache Coherency* synchronisieren

- Es müssen oft Prozesse miteinander koordiniert werden, die in einem Programm laufen.

- Synchronisierung der Prozesse wird normalerweise durch *lock variables* erreicht. Diese sorgen dafür, daß immer nur ein Prozeß auf die geteilten Daten zugreifen kann.



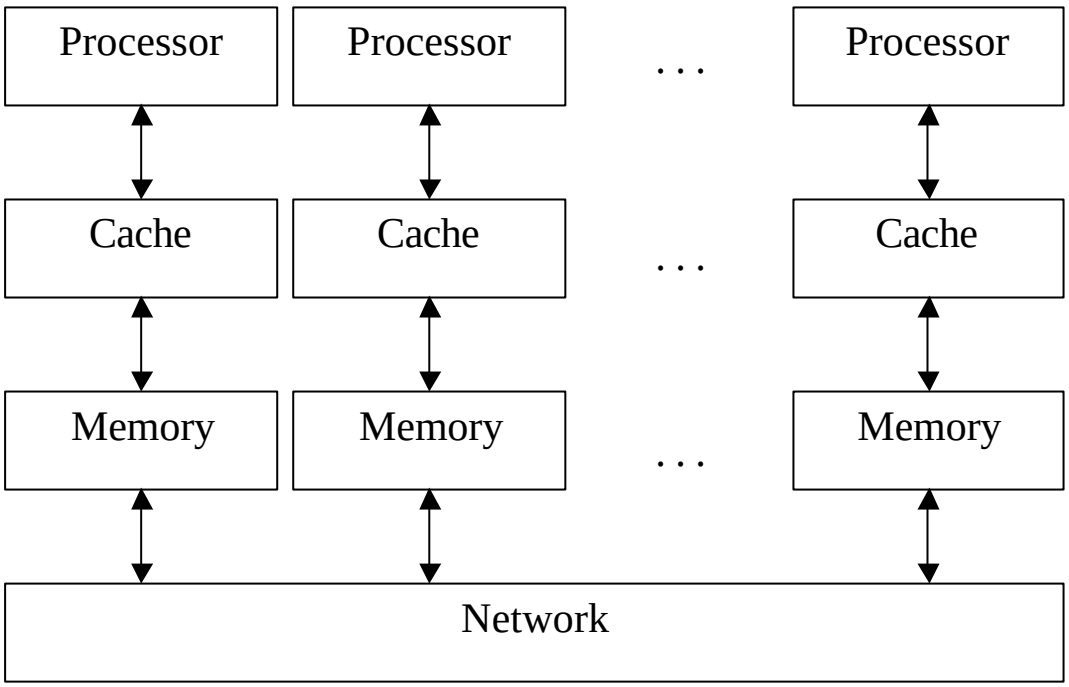
Durch *Cache Coherency* synchronisieren (Fortsetzung)

Step	Processor P0	Processor P1	Processor P2	Bus activity
1	Has lock	Spins, testing if lock = 0	Spins, testing if lock = 0	None
2	Sets lock to 0 and 0 sent over bus	Spins, testing if lock = 0	Spins, testing if lock = 0	Write-invalidate of lock variable from P0
3		Cache miss	Cache miss	Bus decides to service P2 cache miss
4		(Waits while bus busy)	Lock = 0	Cache miss for P2 satisfied
5		Lock = 0	Swap : reads lock and sets to 1	Cache miss for P1 satisfied
6		Swap : reads lock and sets to 1	Value from swap = 0 and 1 sent over bus	Write-invalidate of lock variable form P2
7		Value from swap = 1 and 1 sent over bus	Owns the lock, so can update shared data	Write-invalidate of lock variable from P1
8		Spins, testing if lock = 0		None

Vorteile und Nachteile

- *Spin waiting* entlastet den Bus.
- Der Bus wird stark belastet, wenn der *lock* freigegeben wird. Dies wirkt sich besonders stark aus, wenn der Multiprozessor aus vielen Prozessoren besteht.

Durch ein Netzwerk verbundene Multiprozessoren



Warum kein Single-bus Multiprozessor?

- Ein Bus kann nicht gleichzeitig eine hohe Bandbreite, eine lange Länge und eine niedrige Zugriffszeit bieten.

  >>> Es können nur wenige Prozessoren sinnvoll durch einen Bus verbunden werden.

- Bei einem Single-bus Multiprozessor wird der Bus für jeden Speicherzugriff usw. beansprucht. Bei einem Multiprozessor, der durch ein Netzwerk verbunden ist, wird das Netzwerk nur zur Kommunikation zwischen den Prozessoren beansprucht.

- Viele Prozessoren kann man gut durch ein Netzwerk miteinander verbinden.

Eigenschaften von Multiprozessoren, die durch ein Netzwerk verbunden sind (1997)

Name	Maximale Anzahl an Prozessoren	Prozessor	Größe des Speichers / System	Kommunikations-Bandbreite / Link
Cray Research T3E	2048	Alpha 21164 450 MHz	512 GB	1200 MB / s
HP /Convex Exemplar X-class	64	PA-8000 180 MHz	64 GB	980 MB / s
Sequent NUMA-Q	32	Pentium Pro 200 MHz	128 GB	1024 MB / s
SGI Origin2000	128	MIPS R10000 195 MHz	128 GB	800 MB / s
Sun Enterprise 10000	64	UltraSPARC 1 250 MHz	64 GB	1600 MB / s

---

Organisation des Speichers

- **Shared memory :** Ein Adress-Raum für den ganzen Speicher, Kommunikation mit *loads* und *stores*
- **Multiple private memories :** Jeder Prozessor hat seinen eigenen Adress-Raum, Kommunikation mit *sends* und *receives*
- **Centralized memory :** Speicher liegt physikalisch zusammen, jeder Prozessor hat die gleiche Zugriffszeit
- **Distributed memory :** Speicher ist physikalisch unterteilt, jeder Prozessor hat Speichermodule in seiner Nähe

loads und stores vs. sends und receives

- Die meisten großen Multiprozessoren haben *distributed memory*.
- Sends und receives sind auf Hardwareebene leichter zu realisieren, als loads und stores.
- Sends und receives machen es dem Programmierer leichter, die Kommunikation zwischen den Prozessoren zu optimieren.
- Loads und stores führen normalerweise zu weniger überflüssiger Kommunikation. Dies entlastet das Netzwerk.
    - Es ist effizienter, Daten anzufordern, wenn sie gebraucht werden, als Daten zu erhalten, die möglicherweise gebraucht werden könnten.

    >>> Systeme mit distributed shared memory (DSM)

- Es ist möglich, durch Software die Illusion eines gemeinsamen Adress-Raumes zu schaffen

    >>> shared virtual memory

Cache Coherency bei Multiprozessoren, die durch ein Netzwerk verbunden sind

- Bus-snooping Protokolle funktionieren nicht, da die Prozessoren nicht durch einen gemeinsamen Bus miteinander verbunden sind.
- Eine Alternative zu bus-snooping sind Verzeichnisse (*directories*).

Verzeichnisse (*directories*)

- Protokolle, die auf Verzeichnissen aufbauen, haben ein Verzeichnis, daß den Status jedes Speicherblocks des Hauptspeichers enthält.

- Information in einem Verzeichnis :

  - Welche Caches haben Kopien des Speicherblocks

  - Ist der Block *dirty* / *clean*

  - ...

- Verzeichnisse können auf verschiedenen Speichern vervielfältigt auftreten.

  >>> Dies verringert die Anzahl der Zugriffe auf einen Speicher.

Negative Auswirkungen bei einheitlichem Adress-Raum und Verzeichnissen

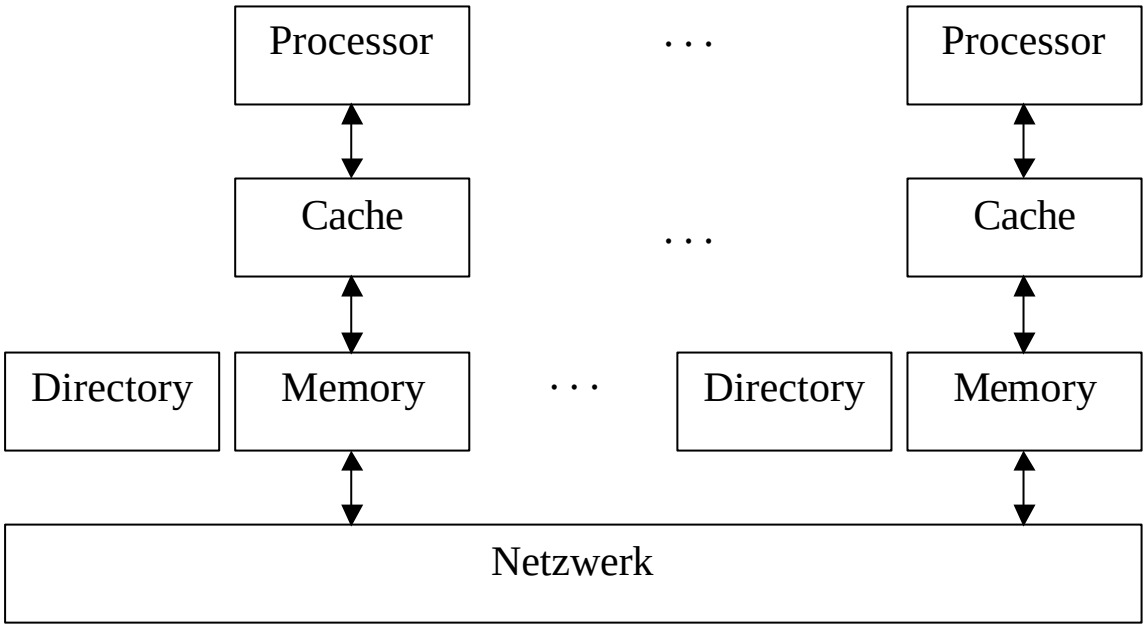
- Verzeichnisse können über verschiedene Speicher verteilt sein.

  >>> Zugriffe auf das Verzeichnis dauert lange, da über das Netzwerk zugegriffen werden muß.

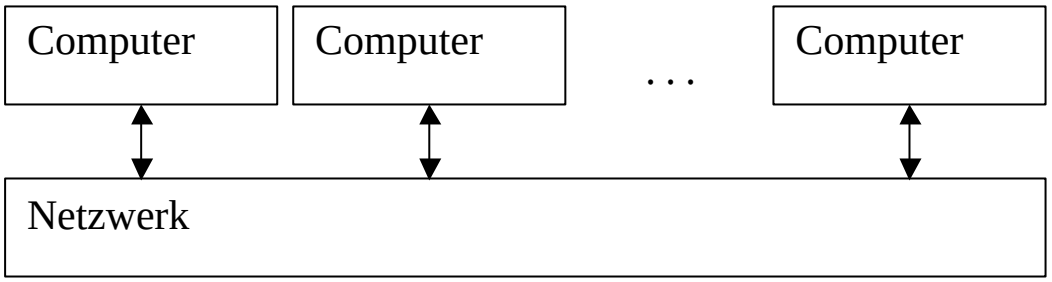
  >>> Das Netzwerk wird durch Zugriffe auf das Verzeichnis stark belastet.

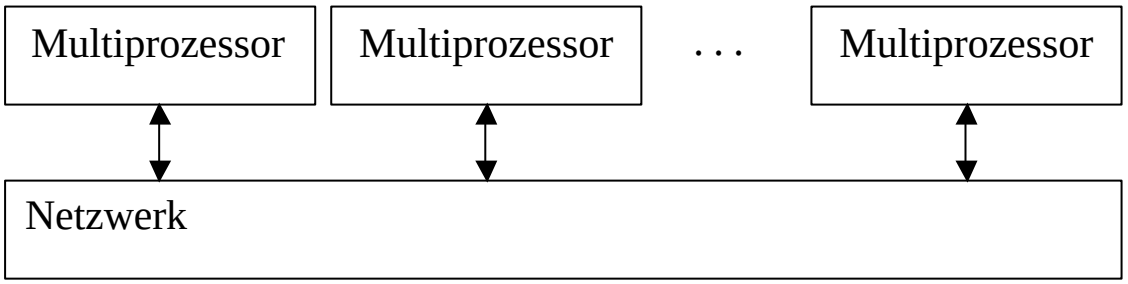
  - Diese negativen Auswirkungen führen nicht bei allen Programmen zu deutlichen Leistungseinbrüchen und können durch gute Programmierung vermindert werden.

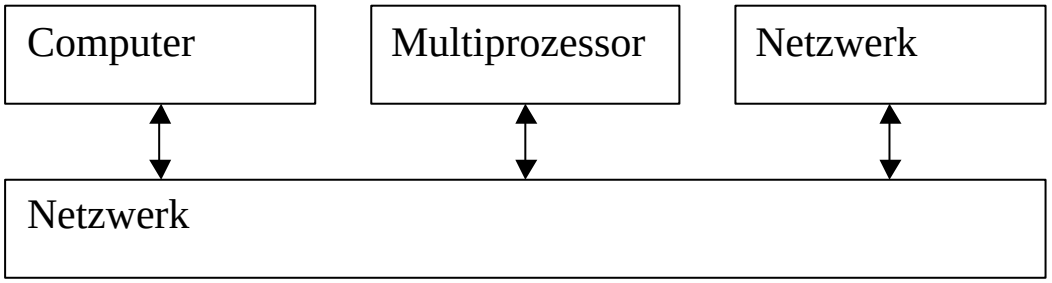
Durch ein Netzwerk verbundene Multiprozessoren mit Verzeichnissen



Cluster


Computer
Computer
. . .
Computer
Netzwerk


Multiprozessor
Multiprozessor
. . .
Multiprozessor
Netzwerk


Computer
Multiprozessor
Netzwerk
Netzwerk

Vorteile von Clustern

- Der Ausfall eines Systems hat nur geringe Auswirkungen auf den gesamten Cluster.

  >>> Cluster sind als Systeme geeignet, die immer und zu jeder Zeit funktionieren müssen.

- Es können sehr viele einzelne Systeme als Cluster sinnvoll miteinander verbunden werden.

- Cluster können leicht mit zusätzlichen Systemen erweitert werden.

- Große Cluster sind billiger, als große Multiprozessoren.

Nachteile von Clustern

- Cluster sind meistens durch den I /O Bus der einzelnen Systeme vernetzt. Dieser Bus hat nur eine sehr geringe Bandbreite.

- Die Betreuung eines Clusters kostet sehr viel mehr, als die Betreuung eines einzelnen Multiprozessors.

- Jedes System eines Clusters hat seinen eigenen Speicher. Ein Programm hat also nicht den gesamten Speicher des Clusters zur Verfügung, sondern nur den Speicher eines Systems.

Nutzung der Vorteile von Clustern und Multiprozessoren

- Cluster, die aus kleineren SMP’s bestehen weisen Vorteile von Clustern und Multiprozessoren auf.

Eigenschaften von Clustern (1997)

Name	Maximale Anzahl an Prozessoren	Prozessor	Maximale Größe des Speichers / System	Kommunikations-Bandbreite / Link
HP 9000 EPS21	64	PA-8000 180 MHz	64 GB	532 MB / s
IBM RS/6000 HACMP R40	16	PowerPC 604 112 MHz	4 GB	12 MB / s
IBM RS/6000 SP2	512	Power2 SC 135 MHz	1024 GB	150 MB / s
Sun Enterprise Cluster 6000 HA	60	UltraSPARC 167 MHz	60 GB	100 MB / s
Tandem NonStop Himalaya S70000	4096	MIPS R10000 195 MHz	1024 GB	40 MB / s

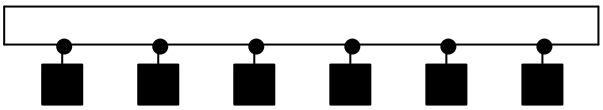
- Alle hier aufgelisteten Cluster, bis auf den IBM RS/6000 SP2, bestehen aus SMP’s.

Netzwerke

- Wie kann man Prozessor-Speicher Einheiten miteinander verbinden?

  - ***Fully connected network*** : Jede Einheit hat zu jeder anderen Einheit eine direkte Verbindung.

  - ***Ring*** : Jeder Prozessor-Speicher Knoten ist mit zwei anderen verbunden.



Ringe vs. vollkommen verbundene Netzwerke

- Vollkommen verbundene Netzwerke sind sehr teuer und sind ab einer gewissen Größe nicht mehr realisierbar.

- Ringe sind billig und haben keine Größenbegrenzung.

- Vollkommen verbundene Netzwerke bieten sehr viel mehr Leistung, als Ringe.

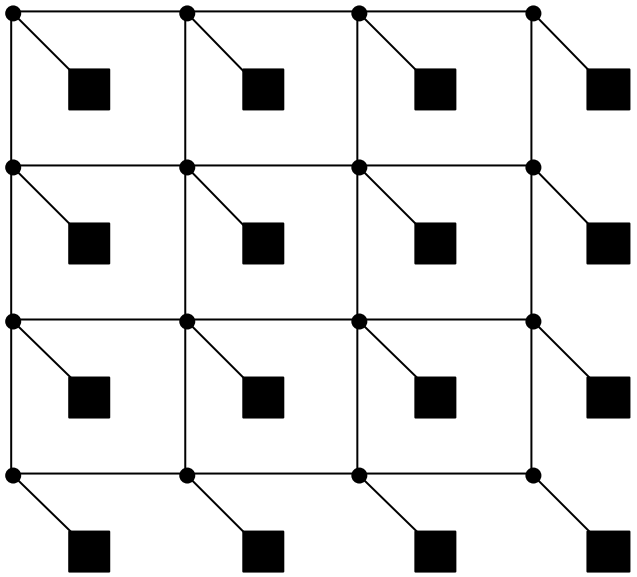
- Es werden meistens Netzwerke gebaut, die geringere Kosten, als ein vollkommen vernetztes Netzwerk haben, aber eine höhere Leistung, als ein Ring.

Alternativen

- Gitter (*2-D grids*)

- *n-cubes*

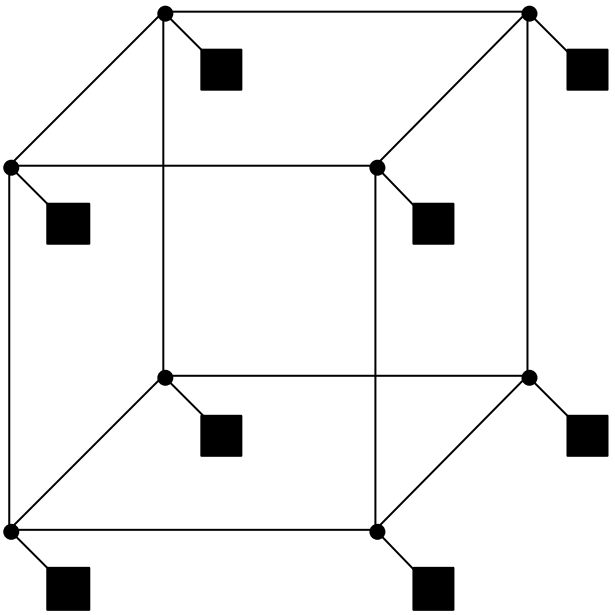
2-D grid



- 2-D Gitter mit 16 Knoten

---

n-cube



- n-cube mit 8 Knoten ($8 = 2^3 \rightarrow n = 3$)

Leistung bei Netzwerken

- ***total network bandwidth*** : Die Bandbreiten aller Verbindungen im Netzwerk werden addiert.

- ***bisection bandwidth*** : Das Netzwerk wird in zwei Teile geteilt, wobei jeder Teil die Hälfte der gesamten Knoten enthält. Leistung ergibt sich als Summe der Bandbreiten der Verbindungen, die von einem Teil in den anderen verlaufen.

    - Das Netzwerk muß immer an der ungünstigsten Stelle geteilt werden, da die geringste Bandbreite meist die Gesamtleistung des Netzwerks bestimmt.

Weitere Alternativen

- Man muß nicht an jeden Knoten einen Prozessor anschließen. Der Knoten kann durch ein *switch* gebildet werden.

    - Ein *switch* ist kleiner, als eine Einheit aus Prozessor, Speicher und *switch*.

    >>> *Crossbar*

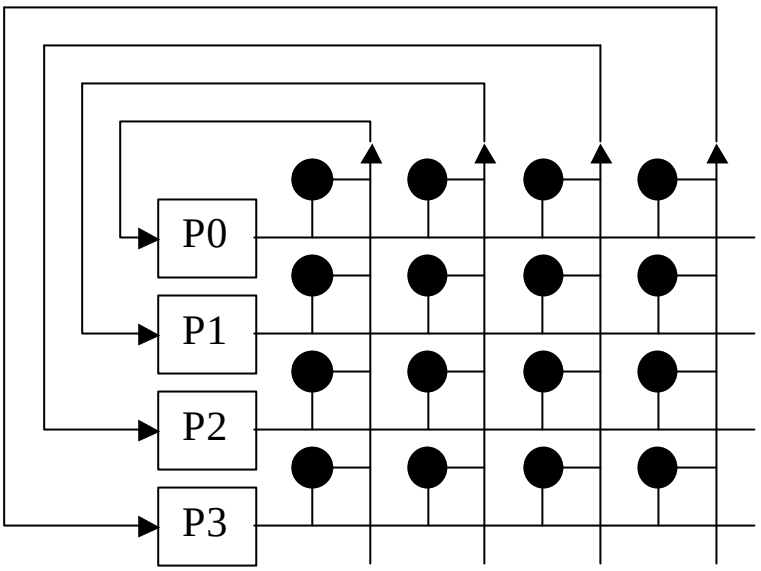
    >>> *Omega network*

Probleme bei Netzwerken

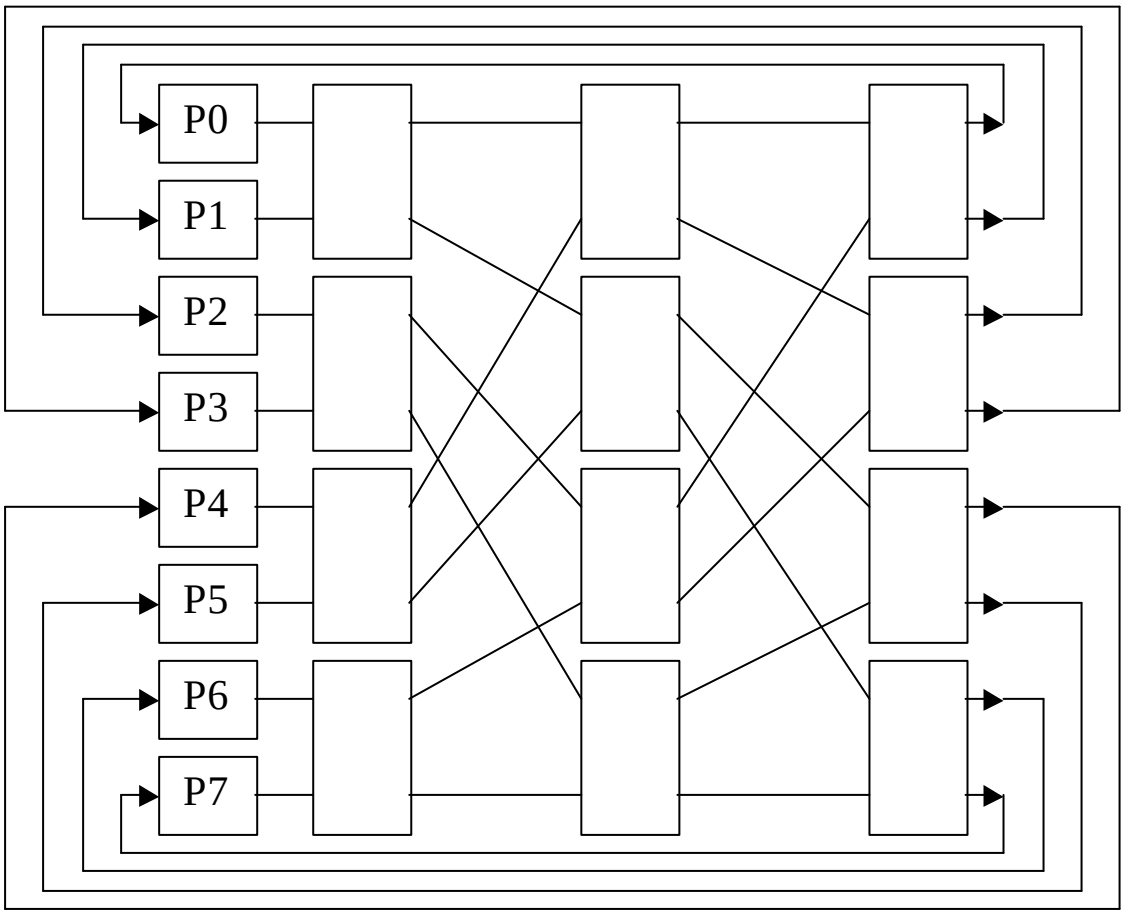
- Lange Verbindungen können nur durch großen Aufwand mit einer hohen Taktrate laufen.

- Netzwerke, die auf dem Papier übersichtlich und elegant aussehen, sind nicht automatisch leicht zu realisieren.

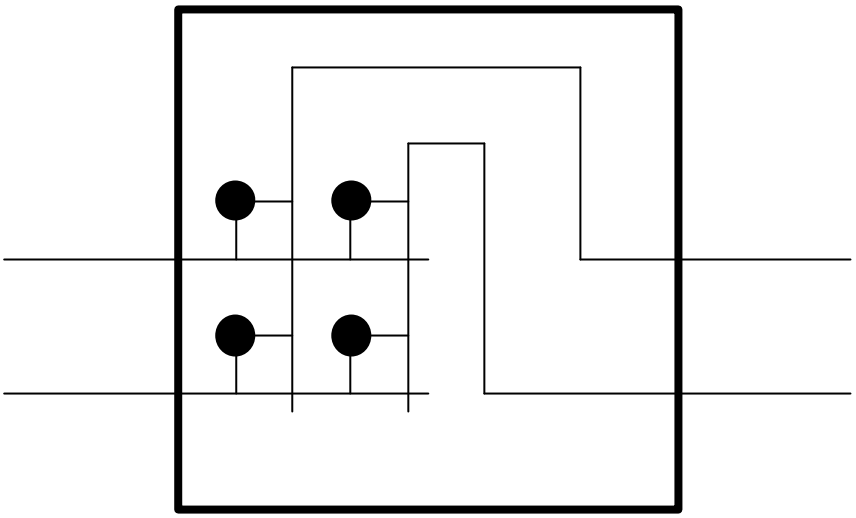
Crossbar



Omega network



Omega network switch box



Trugschluß : Praktisch erreichbare Leistung ist nahe der Spitzenleistung

- 1. Definition : Spitzenleistung = Spitzenleistung eines Prozessors * Anzahl der Prozessoren

- 2. Definition : Spitzenleistung ist die Leistung, die garantiert nie erreicht wird.

Name	Spitzenleistung in MFLOPS	Leistung bei 3D FFT PDE in MFLOPS
Cray YMP (8 Prozessoren)	2666	1795 (67 %)
IBM SP2 (64 Prozessoren)	14636	1093 (7 %)

Trugschluß : Amdahl's Gesetz gilt nicht für Multiprozessoren

- Ein Programm, daß auf einem Multiprozessor mit 1000 Prozessoren in der gleichen Zeit das tausendfache von dem berechnet, was das Programm auf einem Prozessor berechnet, ist nicht automatisch 1000 mal so schnell.

- Programme mit sequentiellen Programmteilen können nicht auf n Prozessoren n-mal schneller sein.

SIMD: Media processing

"Media processing" mit dem PC ?!

- steigende Anforderungen für Audio, Video, Image, 3D
- grosse Datenmengen
- geringe Genauigkeit (8 bit .. 16 bit)
- x86-FPU ausgereizt

=> Trick: vorhandene ALUs/Datenpfade für SIMD verwenden

Befehlssatzerweiterungen:

• MMX	"multimedia extension"	1996
• 3Dnow!		1998
• ISSE	"internet SIMD streaming extension"	1999
• AltiVec	(PowerPC G4, Macintosh)	1999

SIMD: Flynn-Klassifikation

SISD *"single instruction, single data"*

=> jeder klassische PC

SIMD *"single instruction, multiple data"*

=> Feldrechner/Parallelrechner
=> z.B. Connection-Machine 2: 64K Prozessoren
=> eingeschränkt: MMX&Co: 2-8 fach parallel

MIMD *"multiple instruction, multiple data"*

=> Multiprozessormaschinen
=> z.B. vierfach PentiumPro-Server

MISD :-)

SIMD: Literatur

MMX: *"The MMX technology page has been removed"*

- developer.intel.com/drg/mmx/manuals/
- developer.intel.com/drg/mmx/appnotes/
- Linux "parallel-processing-HOWTO"
- IEEE Micro 8/96 S.42, c't 01/97 S.228ff

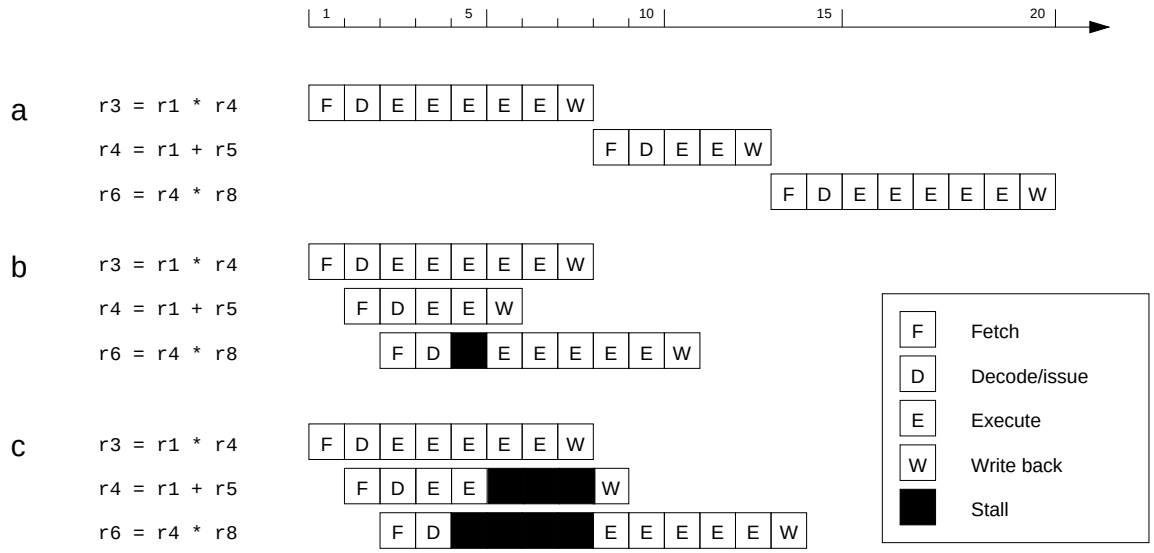
ISSE: Intel website:

- developer.intel.com/software/idap/resources/technical_collateral/pentiumiii/
- c't 04/00 S.314 (ISSE/3Dnow/Altivec)

3D Now! AMD website:

- www.amd.com/K6/K6docs/, www.amd.com/swdev/
- c't 15/98 S.186 ff
- IEEE Micro 3/4-99 S.37ff

Befehlspipeline: in order / out of order

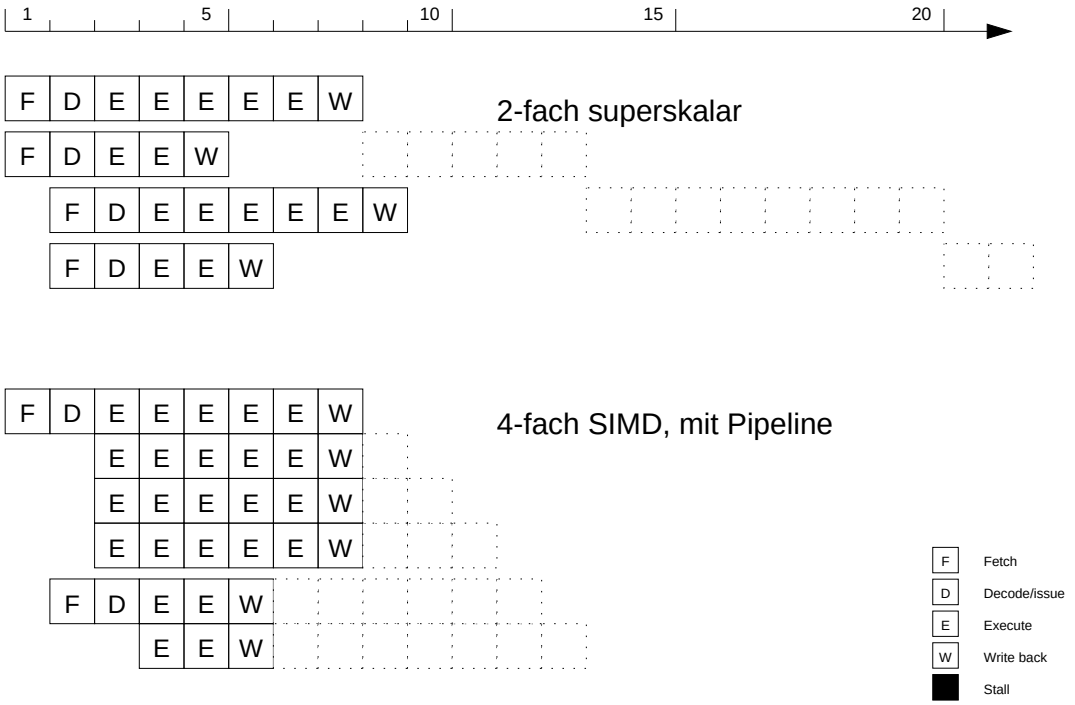


a) serielle Befehlsbearbeitung

b) pipeline, out-of-order completion

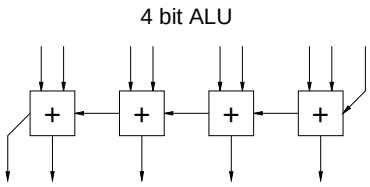
c) in-order-completion

Superskalar, SIMD



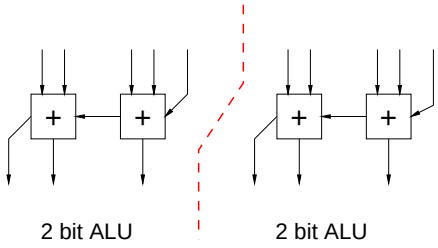
MMX: Grundidee

- 32/64-bit Datenpfade sind "overkill"
- ALUs aber leicht parallel nutzbar:
- carry-chain auftrennen



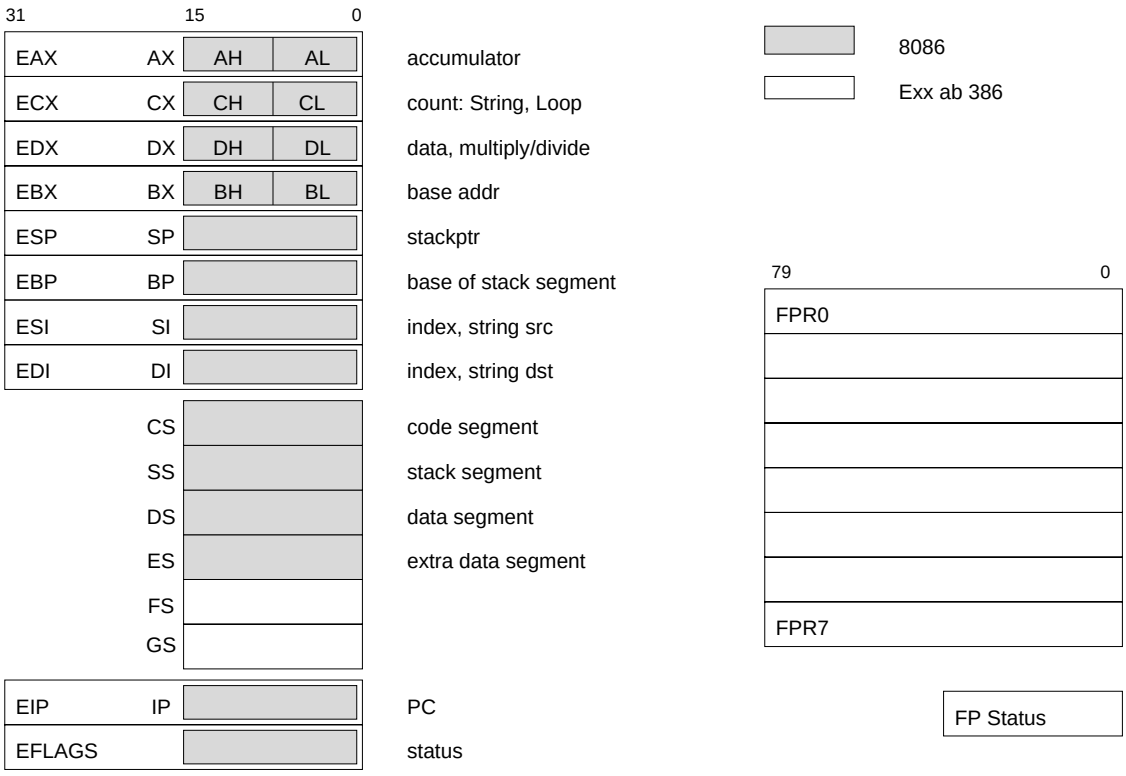
=> SIMD leicht implementierbar
~10% area on Pentium/MMX

=> Performance 2x .. 8x für MMX Ops
=> Performance 1.5x .. 2x für Apps



- MMX press release 03.05.1996
- Nutzen vs. Marketing ?!

x86: Register



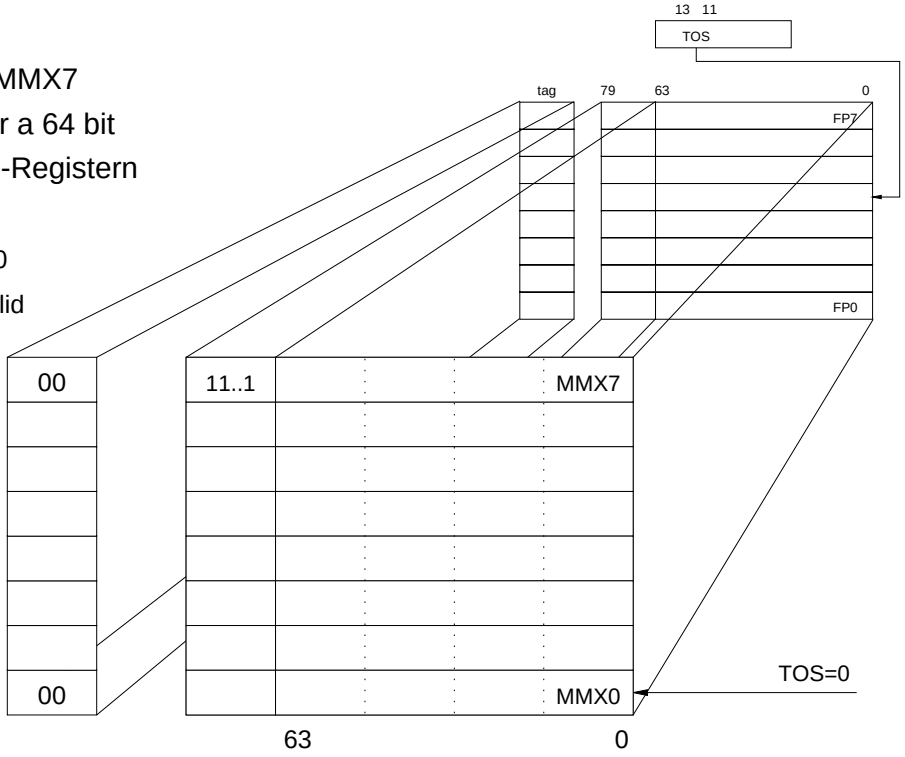
MMX: Entwurfsentscheidungen

Kompatibilität zu alten Betriebssystemen / Apps:

- keine neuen Register möglich => FP-Register nutzen
- keine neuen Exceptions => Überlauf ignorieren
  => saturation arithmetic
- bestehende Datenpfade nutzen => 64 bit
- möglichst wenig neue Opcodes
- alte Prozessoren und neue Software => Code doppelt
  => MMX DLLs
- Test-Applikationen: => 16 bit dominiert
  (audio/image/MPEG-1/3D-Graphik/...)
- keine Tools => Assembler
- optimierte Libraries verfügbar

MMX: Register

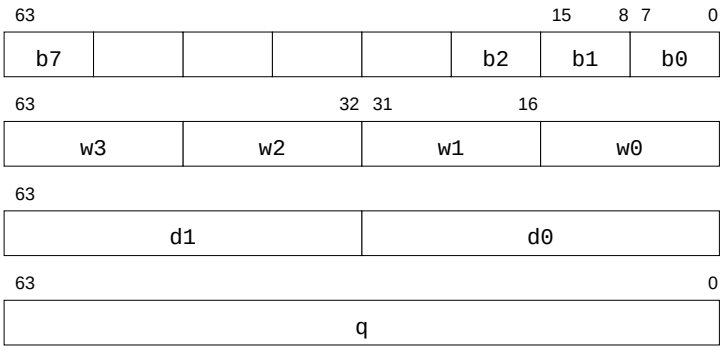
- MMX0 .. MMX7
- 8 Register a 64 bit
- in den FP-Registern
- FP NaN
- FP TOS = 0
- tag 00 = valid



MMX: Datenformate

64-bit Register, 4 Datentypen:

- packed byte *8 / packed word *4 / packed doubleword *2 / quadword
- Zugriff abhängig vom Befehl

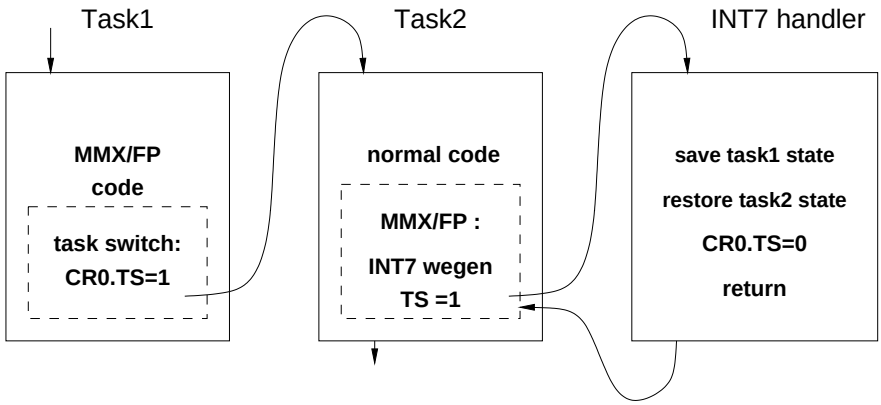


MMX: Befehlssatz

EMMS (FSAV / FRESTOR)	clear MMX state (handle FP regs)
MOVD mm1, mm2/mem32	move 32 bit data
MOVQ mm1, mm2/mem64	move 64 bit data
PACKSSWB mm1, mm2/mem64	pack 8*16 into 8*8 signed saturate
PUNPCKH mm1, mm2/mem64	fancy unpacking (see below)
PACKSSDW mm1, mm2/mem64	pack 4*32 into 4*16 signed saturate
PAND mm1, mm2/mem64	mm1 AND mm2/mem64 / auch OR/XOR/NAND
PCMPEQB mm1, mm2/mem64	8*a==b, create bit mask / auch GT
PADDB mm1, mm2/mem64	8*add 8 bit data
PSUBD mm1, mm2/mem64	2*sub 32 bit data / signed wrap
PSUBUSD mm1, mm2/mem64	2*sub 32 bit data / unsigned saturate
PSLL mm1,mm2/mem64/imm8	shift left mm1 / auch PSRA/PSRL
PMULL/HW mm1, mm2/mem64	4*mul 16*16 store low/high 16 bits
PMADDWD mm1, mm2/mem64	MAC 4*16 -> 2*32
insgesamt 57 Befehle	(Varianten B/W/D S/US)

MMX: Multitaskting . . .

Interaktion mit Betriebssystem / Taskwechsel:

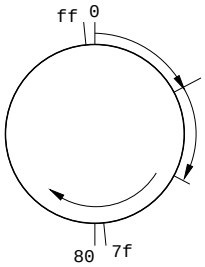


- FP-Register nur bei Bedarf sichern
- vorhandene FP INT7 Routine funktioniert auch für MMX
- keine Anpassung des Betriebssystems notwendig

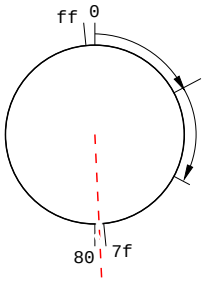
MMX: "Saturation Arithmetic"

was soll bei einem Überlauf passieren?

- wrap-around
  ..., 125, 126, 127, -128, -127, ...



- saturation
  ..., 125, 126, 127, 127, 127, ...
- Zahlenkreis "aufgeschnitten"
- gut für DSP-Anwendungen



paddw (wrap around):

a3	a2	a1	7FFFh
+	+	+	+
b3	b2	b1	0004h
a3+b3	a2+b2	a1+b1	8003h

paddusw (saturating):

a3	a2	a1	7FFFh
+	+	+	+
b3	b2	b1	0003h
a3+b3	a2+b2	a1+b1	7FFFh

MMX: "packed multiply add word"

für Skalarprodukte:

pmaddwd

a3	a2	a1	a0
*	*	*	*
b3	b2	b1	b0
a3*b3+a2*b2		a1*b1+a0*b0	

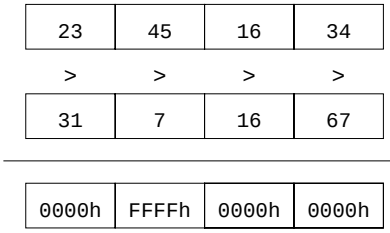
```
vector_x_matrix_4x4( MMX64* v, MMX64 *m ) {
  MMX64  v0101, v2323, t0, t1, t2, t3;

  v0101 = punpckldq( v, v );  // unpack v0/v1
  v2323 = punpckhdq( v, v );  // unpack v2/v3

  t0    = pmaddwd( v0101, m[0] );  // v0|v1 * first 2 rows
  t1    = pmaddwd( v2323, m[1] );  // v2|v3 * first 2 rows
  t2    = pmaddwd( v0101, m[2] );  // v0|v1 * last 2 rows
  t3    = pmaddwd( v2323, m[3] );  // v2|v3 * last 2 rows

  t0    = paddd(    t0, t1 );      // add
  t2    = paddd(    t2, t3 );      //
  v     = packssdw( t0, t2 );      // pack 32->16, saturate
}
```

MMX: "parallel compare"

Vergleiche / Sprungbefehle:

- schlecht parallelisierbar
- Pipeline-Abhängigkeiten

pcmpgtw:



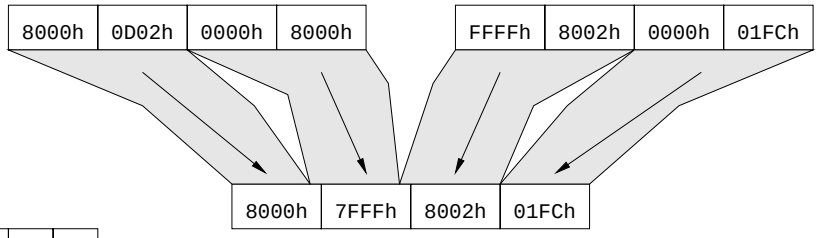
=> keine Sprungbefehle in MMX

=> compare-Operationen setzen Bitmasken

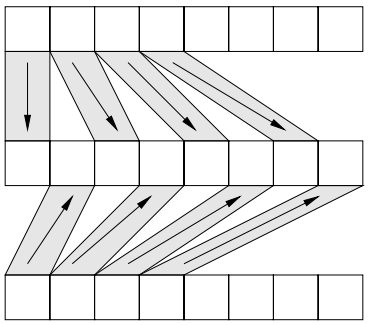
- Bitmasken für logische Ops verwendbar
- Beispiel: chroma-keying

MMX: packssdw / punpckhbw

packssdw: pack with saturation 32 -> 16 signed data:



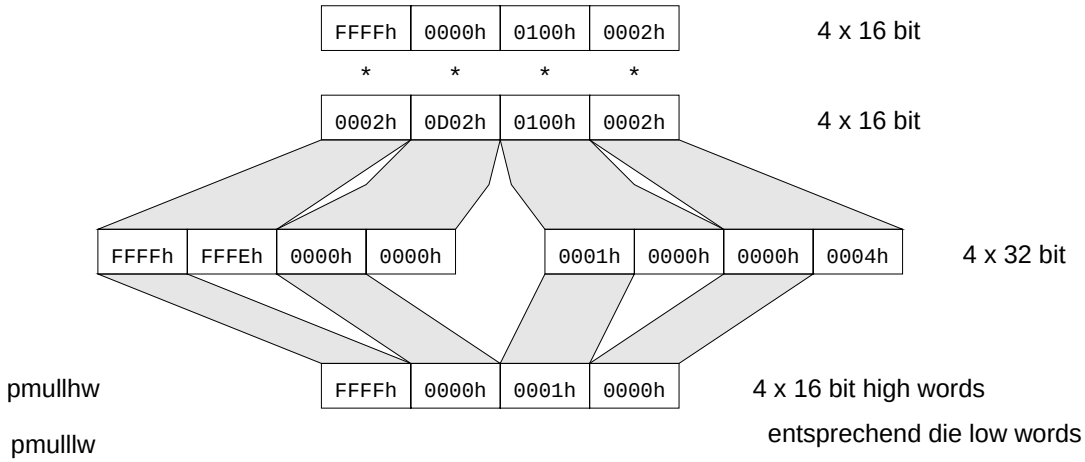
punpckhbw:



punpcklbw: lower 32 bits

MMX: pmulllw / pmullhw

pmull[lh]w: multiply 4 words, write low/high byte of results:



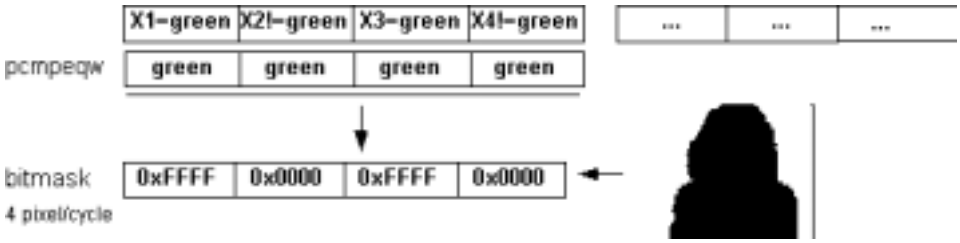
mit Packbefehlen kombinieren, wenn 32-bit Resultate gewünscht

MMX: Chroma Keying (1)

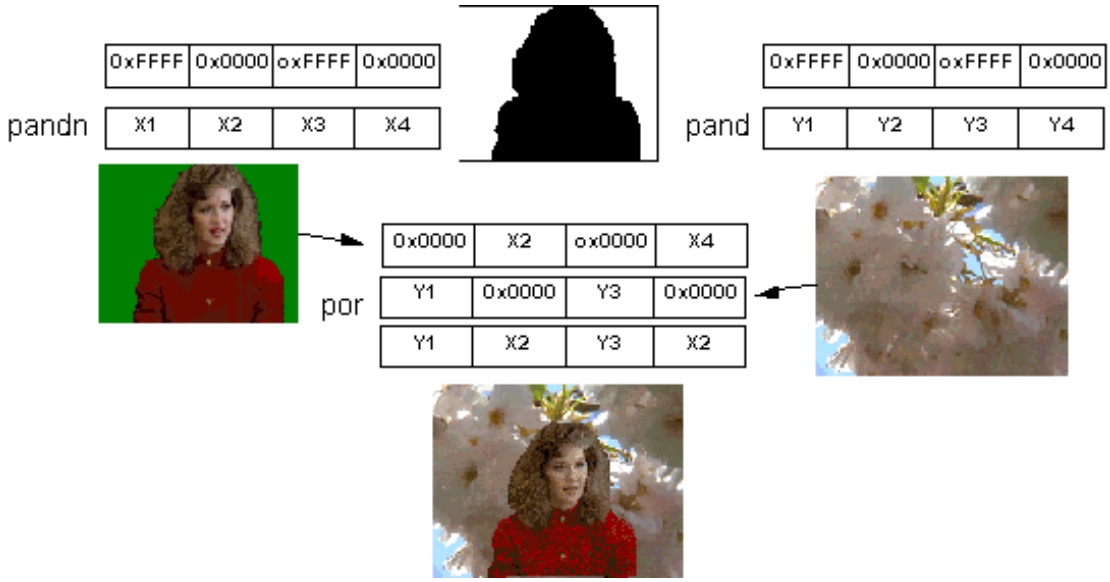
"Wetterbericht":

- MMX berechnet 4 Pixel / Takt
- keine Branch-Befehle

- Schritt 1: Maske erstellen (high-color: 16 bit/pixel)



MMX: Chroma Keying (2)



MMX: Zufallszahlen

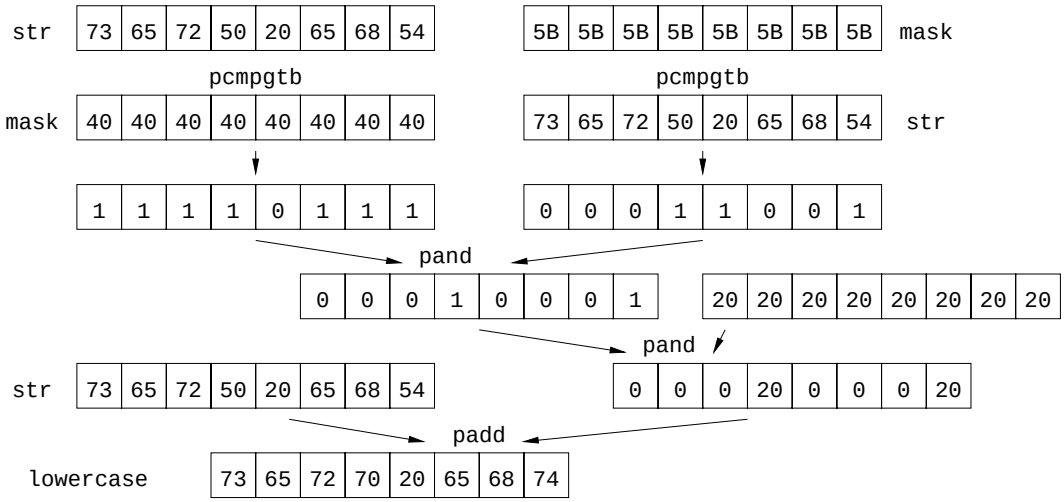
```
x(t) = (x(t-1) * 47989) & 0xFFFF;
```

```
QuadWord  DithMultVal = 0x4f314f314f314f31;
QaudWord  DithRegInit = 0x4f31994d2379bb75;

Init:
  MOVQ    mm0, DithRegInit;

Loop:                          // x(t) -> x(t+1)
  PMULLW  mm0, DithMulVal      // 3 clocks
  MOVQ    [result64], mm0      // 1 clocks
```

- PMULLW latency 3, throughput 1 (on Pentium)
- bis zu vier Zufallszahlen pro Takt (U/V pipelines genutzt)

MMX: toLowerCase()

String lower-to-upper-case conversion:



(aber Probleme mit Umlauten...)

[aus Intel MMX appnote]

3Dnow! Motivation

- stark wachsende Bedeutung von 3D-Spielen
- 32-bit Gleitkommaoperationen nötig für Geometrie-Transformationen
- FPU im AMD K6 vergleichsweise langsam
- MMX unterstützt nur Integer-Datentypen

=> SIMD-Befehle für 32-bit float Datentypen

- schnelle Add/Mult/MAC/Sqrt-Befehle
- muß ohne OS-Unterstützung nutzbar sein
- MMX-Register verwenden
- MMX zwei-Operanden Adressierung
- je zwei float-Datenwerte pro MMX-Register

=> 3Dnow! Spezifikation
(vergleiche Motorola Altivec / Intel ISSE)

3Dnow! Entscheidungen

SIMD-Befehle für 32-bit float Datentypen:

- MMX-Register verwenden, zwei Datenworte pro Register
- zwei-Adress-Befehle
- keine Status-Flags, keine Exceptions
- MMX-Befehle nutzbar (logische, Vergleiche, ...)
- belegt nur einen einzigen x86 Opcode (0F0F ... subobcode)

möglichst wenig Chipfläche:

- keine Unterstützung für NaN/INF/...
- nur round-to-nearest-even Modus, +- 1LSB
- Saturation-Arithmetik statt Überlauf
- Approximation für Division und Quadratwurzel

3Dnow! Prefetch

Speicherzugriffe in Multimedia-Applikationen:

- reguläre Speicherzugriffsmuster
- ungewöhnliche Lokalität
- viele Daten werden (pro Frame) nur einmal benötigt
- aber regelmässig (in jedem Frame)
- Performance stark von optimaler Cache-Ausnutzung abhängig

=> prefetch-Befehl

- quasi normaler Ladebefehl, aber ohne Zielregister
- gewünschte Daten werden in L1/L2-Cache geladen
- löst keine Exceptions / Page Faults aus

=> "memory streaming"
=> auch für andere Anwendung gut nutzbar (etwa Numerik)

3Dnow! Division / Quadratwurzel

- Rechenwerk für Division / Sqrt ist sehr aufwendig
- möglichst wenig Chipfläche für 3Dnow!
- teilweise nur geringe Genauigkeit benötigt
- etwa Shading/Beleuchtungsberechnung für 3D-Graphik

=> Division und Quadratwurzel per Approximation

- erster Befehl liefert 14/15 bit Approximation
- aus Lookup-Table und Interpolation
- mit vollem Takt

- zusätzliche Befehle für Newton-Iteration
- quadratische Konvergenz: zwei Iterationsschritte für volle Genauigkeit
- wenig Hardwareaufwand
- voll in Pipeline integriert, maximaler Durchsatz

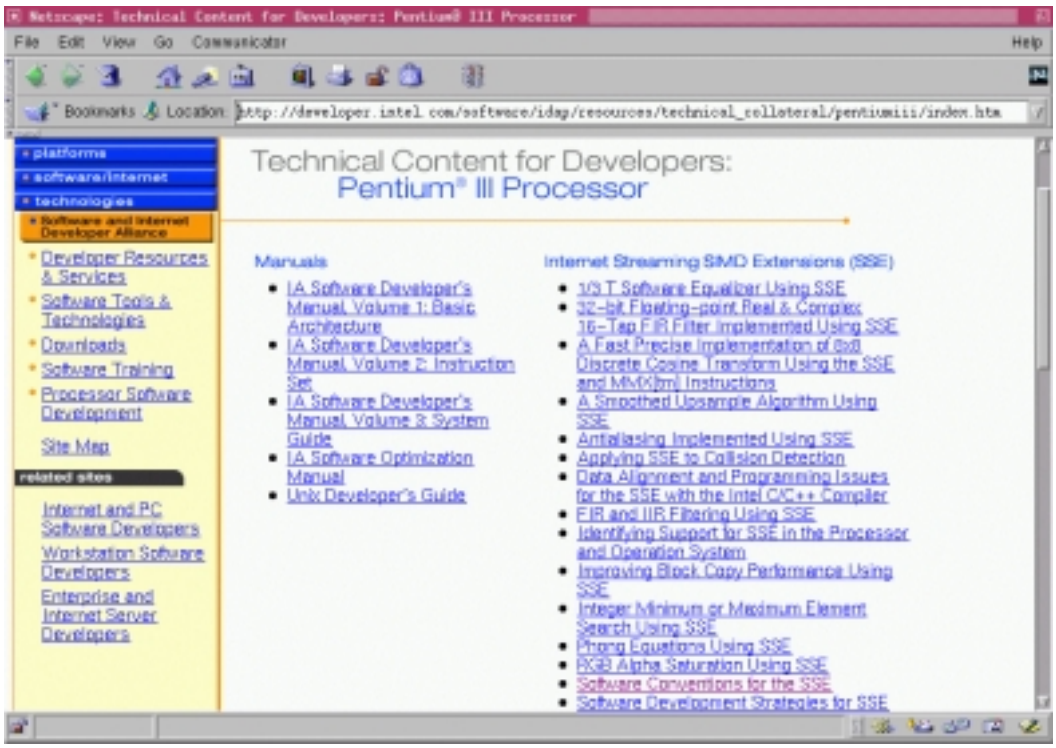
3D Now! Apfelmännchen

```
Function IterPasD
     (I,R :Double; Grenze, Tiefe :Paratyp):Paratyp;
     var A,B,C:double;
  Begin
   Count:= 0;
   A:=0; B:=0;
   Repeat
      C:= SQR(A) - SQR(B) + R;
      B:= 2*A*B + I;
      A:= C;
      INC (Count);
   Until (abs (A) >Grenze) or (Abs (B) > Grenze)
           or (Count=Tiefe);
  IterpasD:=Count;
  End;
```

3D Now! Apfelmännchen

```
;   Quadriere (A + jB)**2 = A**2 - B**2 + j 2*A*B
;   Entry   MM0                     ;A          |  B
;           MM1                     ;1          | -1
;           MM2                     ;R          |  I
;
loop:
            MOVQ    MM3,MM0         ;MM3=A      | B
            MOVQ    MM4,MM0         ; oh weh
            PSLLQ   MM3,32          ; das Vertauschen ist
            PSRLQ   MM4,32          ; sehr mühsam ...
            POR     MM3,MM4         ;MM3=B      | A

            PFMUL   MM3,MM0         ;MM3= A*B   | A*B
            PFMUL   MM0,MM0         ;MM0= A**2  | B**2
            PFMUL   MM0,MM1         ;MM0= A**2  | -B**2
            PFACC   MM0,MM3         ;MM0= A**2 - B**2     | A*B+A*B
            PFADD   MM0,MM2         ;MM0= A**2 - B**2 + R | 2*A*B+I
                                    ;   = A(n+1) |    = B(n+1)
            PF2ID   MM4,MM0         ;iA = INT(A) | iB = Int(B)
            MOVQ    iA,MM4
;  Sieh nach, ob A oder B > GRENZE ist
            ...
            dec     CX              ; iteration counter
            jnz  loop
```


ISSE: Homepage / Literatur



ISSE: Entwurfsentscheidungen

- Markt fordert 3D
- mindestens doppelte FP-Performance notwendig

2-fach oder 4-fach SIMD?

- 128-bit machbar (FP bereits 80-bit)
- bereits 2 64-bit ALUs auf dem Prozessor

=> 4-fach SIMD

"already register-starved IA32 architecture"

=> neue Register, 128-bit

=> erfordert OS-Unterstützung

- 70 neue Befehle
- sowohl "packed" als auch "scalar ISSE instructions"

ISSE: "Streaming"

typisch für Medienverarbeitung:

- hohe Datenmenge / Datenrate
- geringe Lokalität: viele Daten (Pixel) werden nur 1x benötigt

=> Cache-"Pollution"
=> herkömmliche Cache-Strategien nutzlos
=> ALUs müssen auf die Daten warten

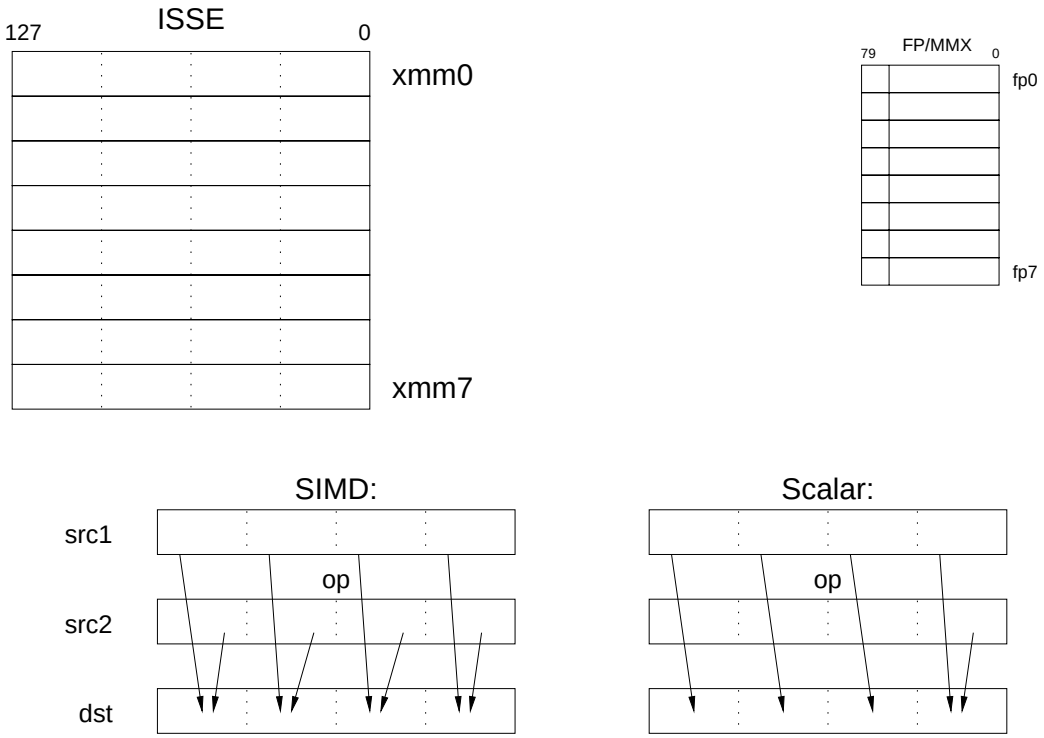
→ Performance leidet (extrem)

- 1GHz, 8x SIMD, 100 nsec Speicher: 800 OPs / 1 Zugriff

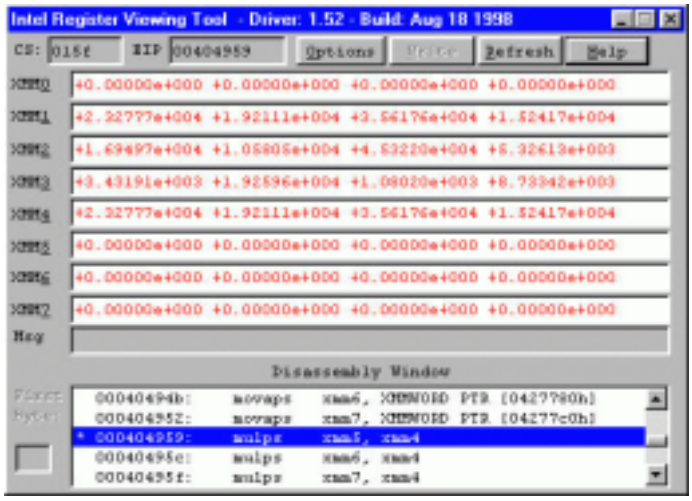
Streaming:

- Cache-Nutzung anpassen
- Prefetch: Daten rechtzeitig anfordern
- Speicherlatenz fast perfekt versteckt (für Media-Apps.)

ISSE: Register



ISSE: Register Viewing Tool



Softwareentwicklung für MMX / ISSE / 3Dnow:

- nur rudimentäre Compiler- und Tool-Unterstützung
- oft handoptimierter Assembler wg. bester Performance

ISSE: Programmierung

Intel VTune Performance Enhancement Environment:

- optimierender Compiler mit ISSE-Unterstützung:

• Intrinsics	C-Funktionen, Compiler inlining
• Vector Class Library	Klassen, inlining durch Compiler
• Vectorization	optimierender Compiler
• Intel Performance Library Suite	

- erfordert 16-Byte Alignment aller Datentypen
- umfangreiche Profiling-Tools

- sehr teuer

ISSE: Programmierung mit "Intrinsics"

```
float xa[SIZE], xb[SIZE], xc[SIZE];
float q;

void do_c_triad() {
  for( int j=0; j < SIZE; j++ ) {
    xa[j] = xb[j] + q*xc[j];
  }
}
```

ISSE-Programmierung mit "Intrinsics" und VTUNE:

```
#define VECTOR_SIZE 4
__declspec(align(16)) float xa[SIZE], xb[SIZE], xc[SIZE];
float q;

void do_intrin_triad() {
  __m128 tmp0, tmp1;

  tmp1 = _mm_set_ps1(q);
  for( int j=0; j < SIZE; j+= VECTOR_SIZE) {
    tmp0 = __mm_mul_ps( *((__m128 *) &xc[j]), tmp1 );
    *(__m128 *) &xa[j] =
                  _mm_add_ps(tmp0, *((__m128 *) &xb[j];
  }
}
```

ISSE: AoS / SoA

Array of Structures:

- Daten lokal
- Anordnung schlecht für SIMD

```
struct
{
  float A, B, C;
} AoS_data[1000];
```

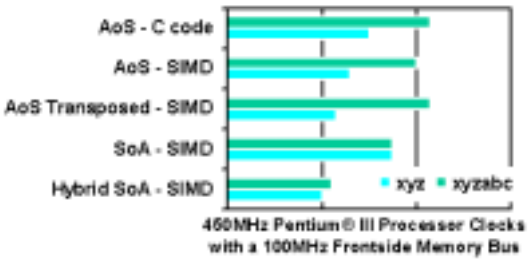
Structure of Arrays:

- Anordnung optimal für SIMD
- aber im Speicher "verstreut"

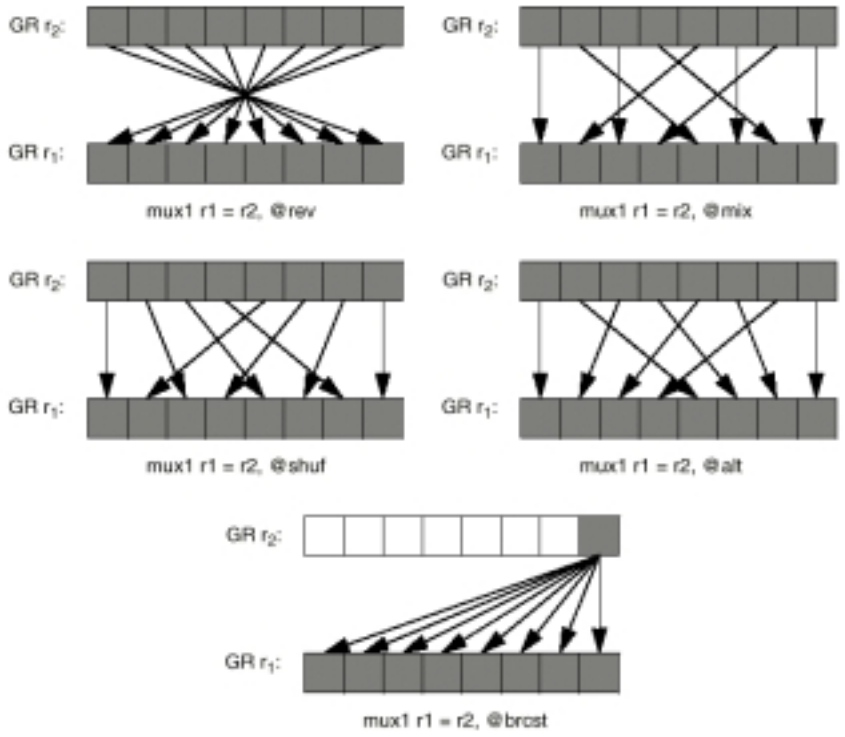
```
struct
{
  float A[1000],B[1000],C[1000];
} SoA_data;
```

=> Hybrid SoA - SIMD

```
struct
{
  float A[8],B[8],C[8];
} Hybrid_data[125];
```



ISSE2: mux1-Befehl (IA64)



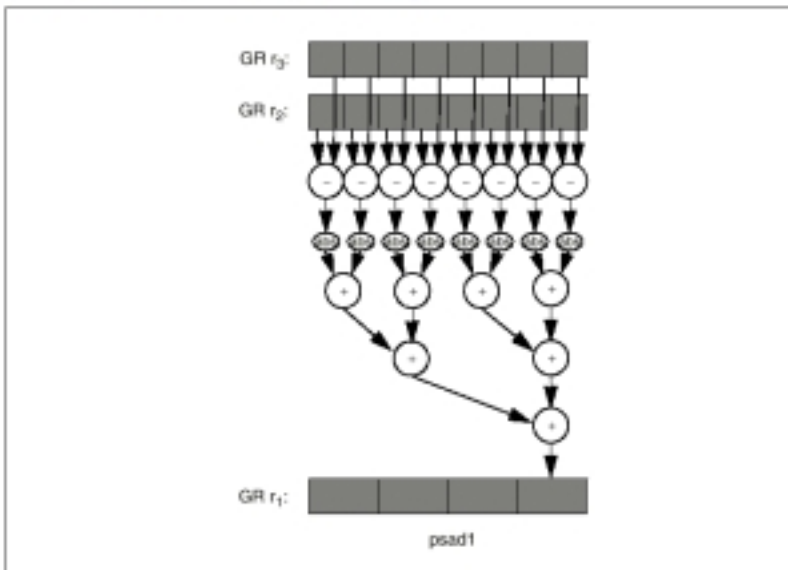
ISSE2: psad1-Befehl (IA64)

Parallel Sum of Absolute Difference

Format: (qp) psad1 $r_1 = r_2, r_3$

Description: The unsigned 8-bit elements of GR r_2 are subtracted from the unsigned 8-bit elements of GR r_3. The absolute value of each difference is accumulated across the elements and placed in GR r_1.

Figure 7-36. Parallel Sum of Absolute Difference Example



ISSE2: pavg2-Befehl (IA64)