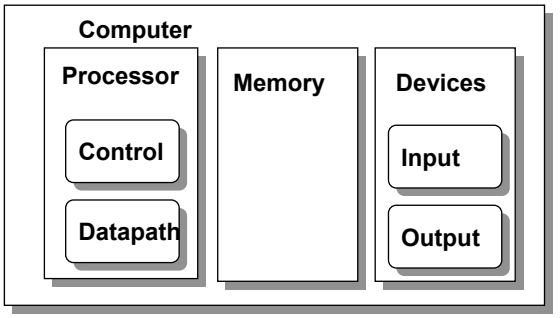
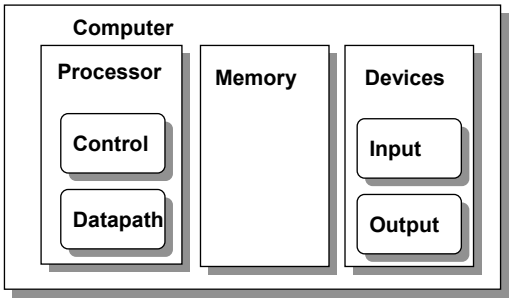
1. Grundkonzepte des Rechneraufbaus



von Neumannsches Rechnerkonzept
Binärkodierungen
Modifikationen und Erweiterungen

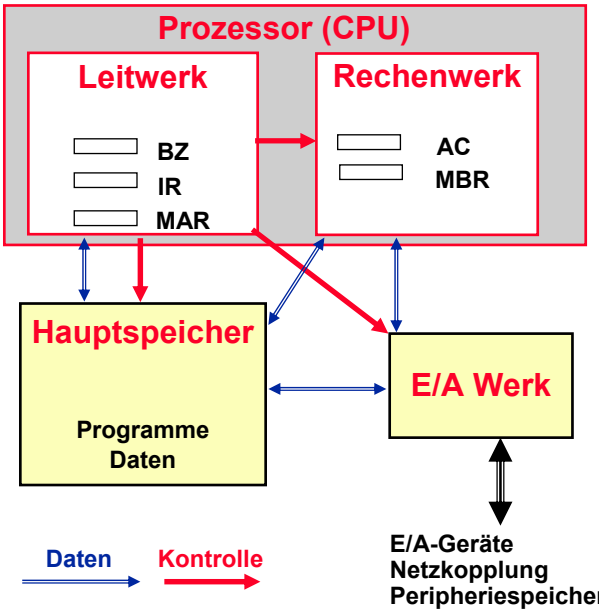
Das von Neumannsche Rechnerkonzept

- Rechner bestehen aus 4 Werken:
  - Kontrolleinheit, Leitwerk
    interpretiert Programme
  - Rechenwerk
    führt Operationen aus
  - Haupt- bzw. Arbeitsspeicher
    enthält Programme
    und Daten
  - Ein/Ausgabewerk
    kommuniziert mit Umwelt, incl. Sekundärspeicher



- Die Struktur des Rechners ist unabhängig vom bearbeiteten Problem (-> Programmsteuerung)
- Programme und Daten stehen im selben Speicher und können durch die Maschine modifiziert werden.
- Der Hauptspeicher ist in Zellen gleicher Größe geteilt, die durch fortlaufende Nummern (Adressen) bezeichnet werden.

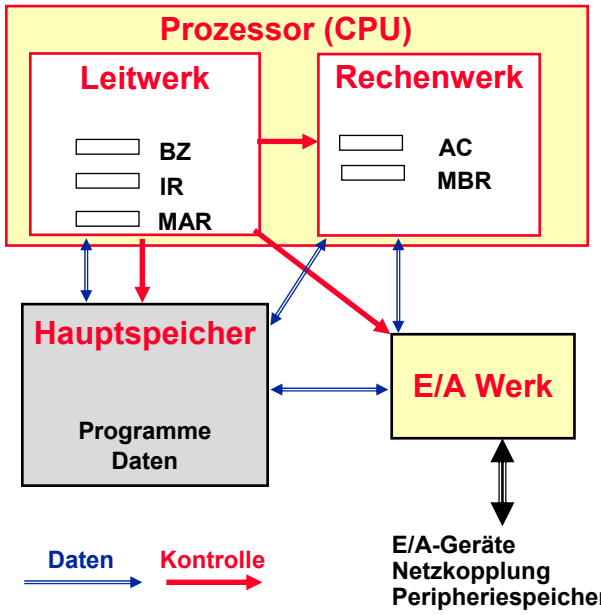
Organisationsplan eines von Neumann-Rechners



Prozessor, Zentraleinheit (CPU = Central Processing Unit) mit

- Rechenwerk (ALU = Arithmetic Logical Unit)
  - Ausführung von Berechnungen
  - Register:
    AC - Akkumulator
    MBR - Speicherpuffer
- Leitwerk, Kontrolleinheit
  - Dekodierung von Befehlen
  - Steuerung der Befehlsausführung
  - Register:
    BZ - Befehlszähler
    IR - Instruktionsregister
    MAR - Speicheradressregister

Organisationsplan eines von Neumann-Rechners



Speicher

- **ROM (Read Only Memory)**
  - Festwertspeicher,
    Daten fest eingebrannt,
    nicht mehr veränderbar
    => Mikroprogrammierung
- **RAM (Random Access Memory)**
  - Speicher mit wahlfreiem Zugriff, Befehle und Daten
  - DRAM (Dynamic RAM): eigentlicher Arbeitsspeicher
  - SRAM (Static RAM): schneller Pufferspeicher (Cache)

  DRAM´s haben eine größere Kapazität, aber höhere Zugriffszeiten als SRAM´s.

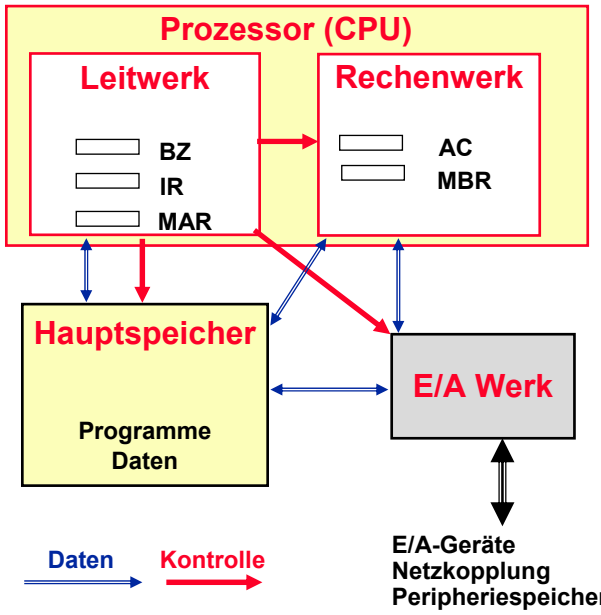
Speicherkenngrößen

- Grundeinheit: 1 Bit (binary digit)
- Speicher = Folge von N Zellen gleicher Größe m (in Bits),
  meist m=8 -> 1 Byte (binary term)
- Oft sind auch Vielfache von Speicherzellen direkt adressierbar:
  - 2 Bytes = Halbwort
  - 4 Bytes = Wort
  - 8 Bytes = Doppelwort
- Anzahl N der Speicherzellen ist meist große Zweierpotenz:
  $N = 2^n$ -> n-Bit-Adressen von 0 bis 2^n-1

$2^{10}=1024$	$\sim 10^3$	=> 1K	Kilo
2^{20}	$\sim 10^6$	=> 1M	Mega
2^{30}	$\sim 10^9$	=> 1G	Giga
2^{40}	$\sim 10^{12}$	=> 1T	Tera
2^{50}	$\sim 10^{15}$	=> 1P	Peta
2^{60}	$\sim 10^{18}$	=> 1E	Exa

z.B.: 16 Bit Adressen => 64 KB Speicher, 32 Bit Adressen => 4 GB Speicher

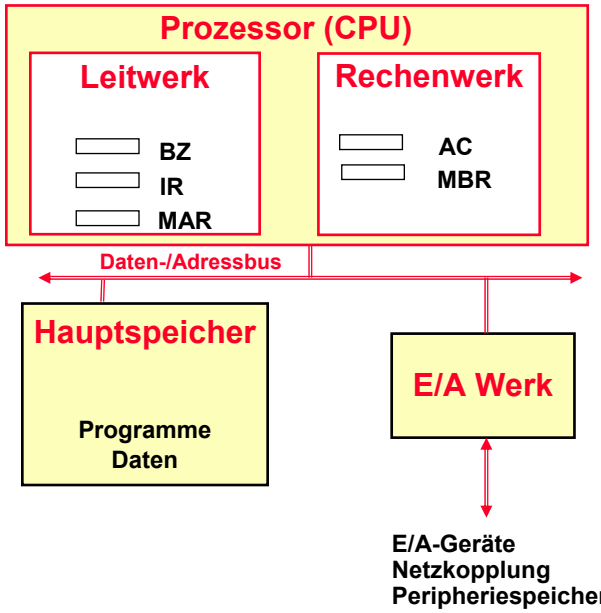
Organisationsplan eines von Neumann-Rechners



Ein-/Ausgabewerk mit parallel arbeitendem Spezialprozessor (PIO - Parallel I/O) mit direktem Speicherzugriff (DMA - Direct Memory Access)

- Ein/Ausgabe von Programmen und Daten
- Netzkopplung: Local Area Network (LAN), z.B. Ethernet
  - Kommunikation zwischen Computern
  - gemeinsame Nutzung von Speicher oder Ein/Ausgabegeräten
  - nicht-lokaler Rechnerzugang
- Peripheriespeicher, Hintergrundspeicher:
  - permanente Datenspeicherung mit magnetisierbaren oder optischen Medien (CDs, DVDs)
  - hohe Kapazität, langsamer Zugriff

Organisationsplan eines von Neumann-Rechners



Busse

- Medien zum Transfer von Daten und Steuerinformationen zwischen mehreren funktionellen Einheiten des Rechners
- Protokolle zur Regelung der Busbenutzung erforderlich
- uni- oder bidirektionale Busse
- bitseriell (1 Leitung) oder parallel (Leitungsbündel)

Arbeitsweise eines von Neumann-Rechners

- Das Programm besteht aus einer Folge von Befehlen (Instruktionen), die nacheinander (sequentiell) ausgeführt werden.
- Von der Folge kann durch bedingte und unbedingte Sprungbefehle abgewichen werden, die die Programmfortsetzung aus einer anderen Zelle bewirken.

  Bedingte Sprünge sind von gespeicherten Werten abhängig.
- Die Maschine benutzt Binärcodes, Zahlen werden dual dargestellt. Befehle und andere Daten müssen geeignet kodiert werden.

Phasen einer Befehlsausführung

1. **Befehlsholphase (Instruction Fetch, IF):**
   Laden eines Befehls aus dem Speicher
2. **Dekodierphase (Instruction Decode, ID):**
   Befehl interpretieren
3. **Operandenholphase (Operand Fetch, OF):**
   Notwendige Operanden aus dem Speicher holen
4. **Ausführungsphase (Execution, EX):**
   eigentliche Ausführung des Befehls

Bitfolgen im Speicher sind nicht selbstidentifizierend. Der Rechner entscheidet aufgrund des zeitlichen Kontextes, ob eine Bitfolge als Befehl, Adresse oder Datum zu interpretieren ist.

Zahlendarstellungen

Sei b>1 eine beliebige natürliche Zahl. Dann heißt
$\Sigma(b) := \{0,...,b-1\}$ das Alphabet des b-adischen Zahlsystems.

Beispiele:

b = 10	**Dezimalsystem**	**$\Sigma(10) = \{0,...,9\}$**
b = 2	**Dualsystem**	**$\Sigma(2) = \{0,1\}$**
b = 8	**Oktalsystem**	**$\Sigma(8) = \{0,...,7\}$**
b = 16	**Hexadezimalsystem**	**$\Sigma(16) = \{0,..,9,A,B,C,D,E,F\}$**
b = 256	**-> ASCII**	**(American Standard Code for Information Interchange)**
b = 60	**-> Zeitrechnung**	

b-adische Darstellung natürlicher Zahlen

Sei b eine natürliche Zahl mit b >1. Sei n eine natürliche Zahl.

Jede natürliche Zahl z mit 0 <= z <= b^n-1 ist eindeutig als Wort der Länge n über Σ(b) darstellbar:

$$z = \sum_{i=0}^{n-1} z_i b^i \quad \text{mit} \quad 0 <= z_i < b$$

Ziffernschreibweise: $z = (z_{n-1}\, z_{n-2} \ldots z_1\, z_0)_b$

Hintergrund: Es gibt b^n Folgen der Länge n über Σ(b). (feste Wortlänge durch "führende Nullen")

besonders wichtig: b = 2

Dualsystem

Mit n binären Ziffern $(Z_{n-1} \ldots Z_1\, Z_0)_2$ **können die Zahlen**

$Z_{n-1} * 2^{n-1} + \ldots + Z_1 * 2^1 + Z_0$

dargestellt werden.

Beispiel: Die Bitfolge 10110001110011 entspricht der Zahl 11379:

$$(10110001110011)_2$$

$$= 1*2^{13} + 0*2^{12} + 1*2^{11} + 1*2^{10} + 0*2^9 + 0*2^8 + 0*2^7 + 1*2^6$$

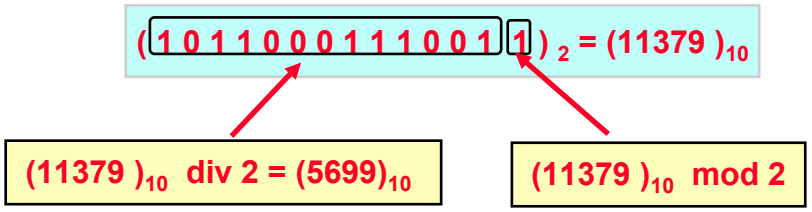
$$+ 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$

$$= 8192 + 2048 + 1024 + 64 + 32 + 16 + 2 + 1$$

$$= (11379)_{10}$$

Umwandlung zwischen Zahlsystemen

Ist eine natürliche Zahl k in einem Zahlensystem zur Basis b dargestellt, so ist die letzte Ziffer gerade k mod b und die übrigen Ziffern stellen k div b dar.



Die Darstellung einer Zahl im Binärsystem gewinnt man daher durch fortgesetztes Dividieren durch 2 und die Berechnung des Restes.

Beispiel (Divisionsrestverfahren)

Die Zahl $(4711)_{10}$ entspricht der Binärzahl $(1001001100111)_2$.

4711	div 2	=	2355	Rest	1
2355	div 2	=	1177	Rest	1
1177	div 2	=	588	Rest	1
588	div 2	=	294	Rest	0
294	div 2	=	147	Rest	0
147	div 2	=	73	Rest	1
73	div 2	=	36	Rest	1
36	div 2	=	18	Rest	0
18	div 2	=	9	Rest	0
9	div 2	=	4	Rest	1
4	div 2	=	2	Rest	0
2	div 2	=	1	Rest	0
1	div 2	=	0	Rest	1

Oktal- und Hexadezimalsystem

einfache lokale Umwandlung zwischen Dual- / Oktal- und Hexadezimalsystem

kompakte Darstellung von Binärzahlen

Dezimal-system	Binärsystem	Oktalsystem	Hexadezimal-system
123	0111 1011	173	7B
1984	0111 1100 0000	3700	7C0
1994	0111 1100 1010	3712	7CA
2000	0111 1101 0000	3720	7D0
2004	0111 1101 0100	3724	7D4
43690	1010 1010 1010 1010	125252	AAAA

Das Hexadezimalsystem

Je 4 Bits entsprechen genau einer Hexadezimalziffer:

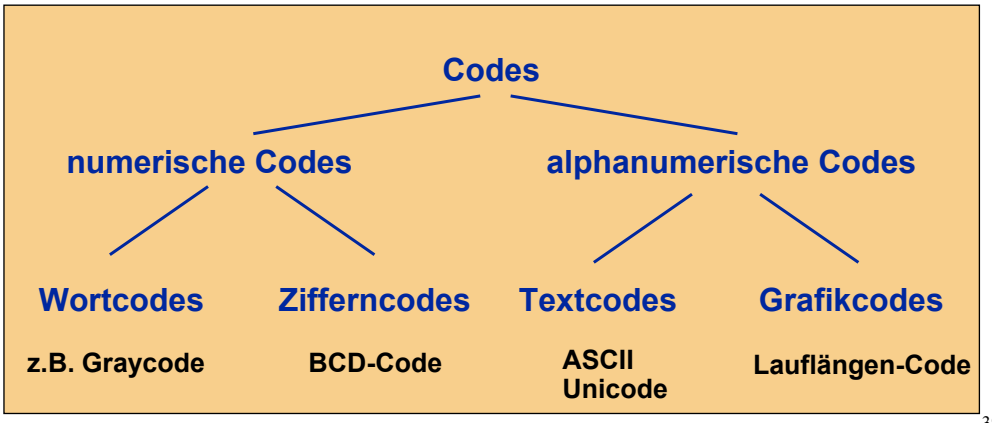
0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

Wie das folgende Beispiel zeigt, ist der Grund für die Verwendung des Hexadezimalsystems die übersichtlichere Darstellung einer Bitfolge durch Hexadezimalziffern:

1100	1011	1110	0110	0011	0000	1111	1010	0101	0111	0011	1011	1000	1101	1011	0000
C	B	E	6	3	0	F	A	5	7	3	B	8	D	B	0

Codes

DIN 44300: Ein Code ist eine Vorschrift für die eindeutige Zuordnung (Kodierung) der Zeichen eines Zeichenvorrats zu denjenigen eines anderen Zeichenvorrats.



Numerische Codes

Wortcodes:
Kodierung von Zahlen als Ganzes

Beispiel:
Graycode: Von einer Zahl zur nächsten ändert sich genau ein Bit im Code

0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Bildungsvorschrift?

Zifferncodes:
einzelne Kodierung von Ziffern

Beispiel:
BCD (Binary Coded Digits):
4 Bits pro Dezimalziffer

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Die Bitmuster 1010 …1111 werden nicht verwendet.

Alphanumerische Codes

Textcodes

Kodierung von Buchstaben, Satzzeichen, Sonderzeichen, Steuerzeichen etc.

- ASCII – American Standard Code for Information Interchange

  7 Bits pro Zeichen -> 128 Zeichen

- Unicode (Internationaler Standard)

  16 Bits -> 65536 Codepunkte

  0 ... 127 entspricht ASCII

Grafikcodes

Kodierung von Bildpunkten,

etwa 4 Bits für 16 Graustufen oder 8 Bits for 256 Graustufen bei Schwarz-Weiß-Bildern

Beispiel: Lauflängenkodierung

Speichere in zwei Bytes

1. Anzahl aufeinanderfolgender Bildpunkte mit gleicher Graustufe
2. Graustufe

=> Kompression bei vielen benachbarten gleichen Bildpunkten

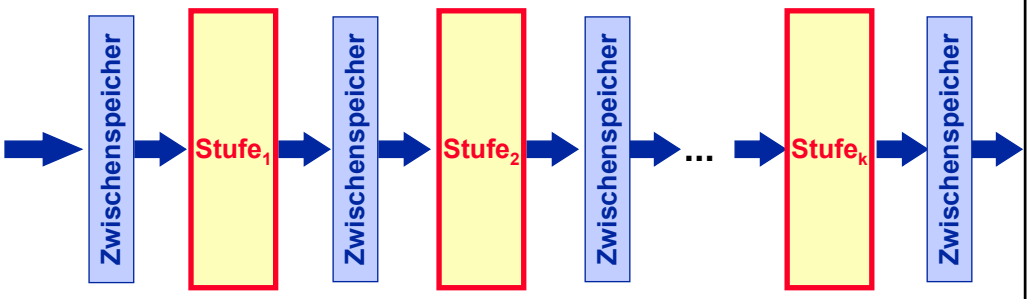
Modifikationen und Erweiterungen des von Neumann-Konzeptes

- Fließbandverarbeitung
- Parallelisierung auf verschiedenen Ebenen (Rechen-/Kontrollwerke, Speicher)
- Verfeinerungen der Speicherstruktur

  ==> Virtualisierung, Hierarchisierung

Fließbandverarbeitung

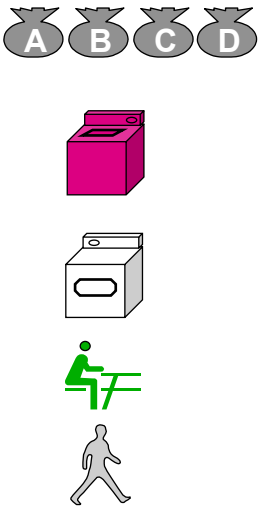
- zentrales Merkmal von RISC-Prozessoren
- überlappende Ausführung mehrerer Instruktionen
- aufeinanderfolgende Phasen der Instruktionsabarbeitung werden in separaten Verarbeitungseinheiten durchgeführt.



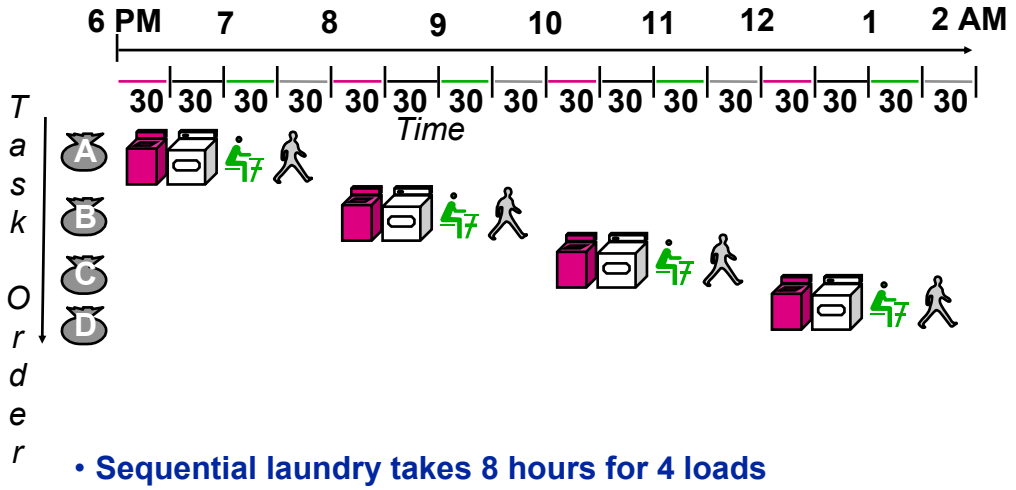
Pipelining is Natural!

David Pattersons Erklärung der Fließbandverarbeitung (aus Course CS 152, Berkeley Univ. 1997)

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- "Folder" takes 30 minutes
- "Stasher" takes 30 minutes to put clothes into drawers

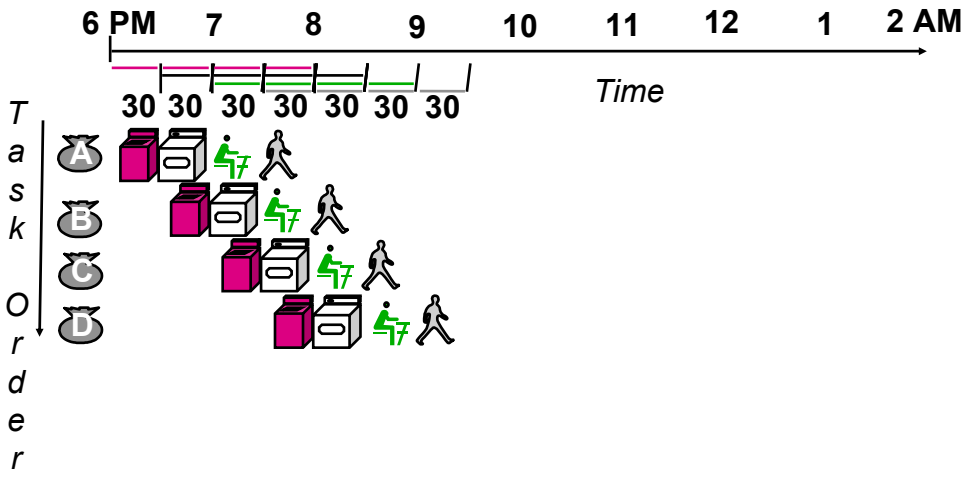


Sequential Laundry



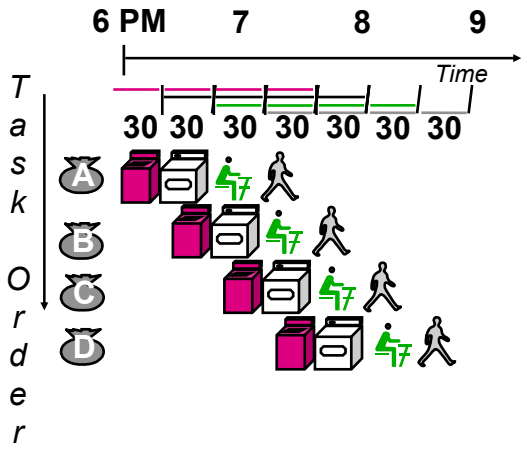
- Sequential laundry takes 8 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons

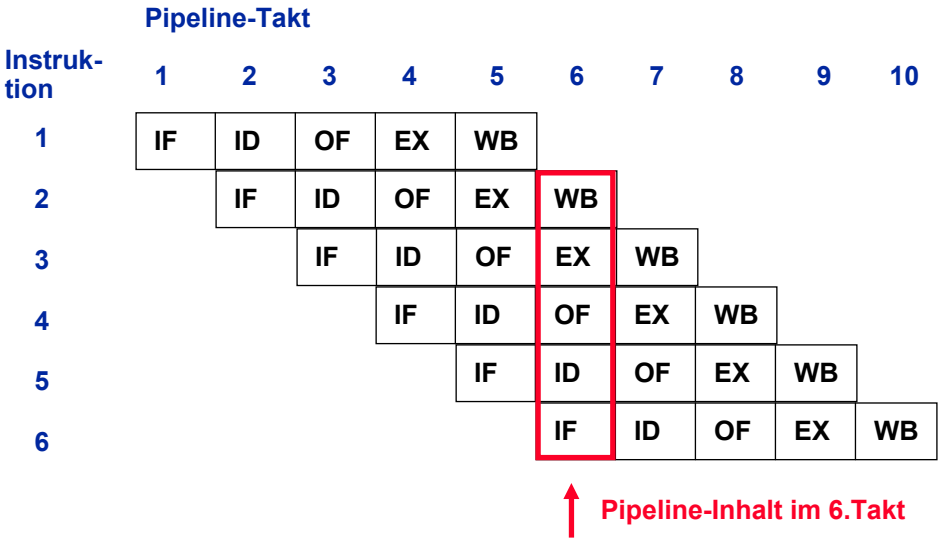


- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for Dependences

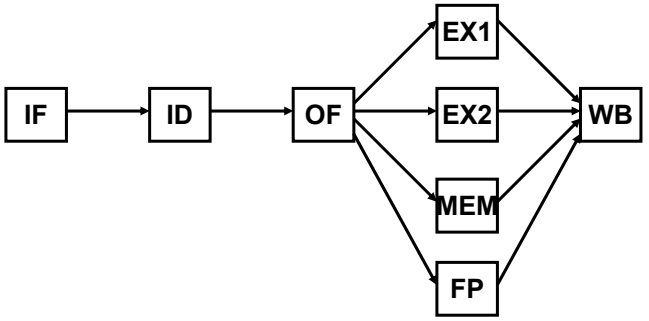
Phasen der Befehlsausführung

- IF - Instruction Fetch Befehlsbereitstellung
  - Laden der Instruktion aus dem Speicher (Cache)
- ID - Instruction Decode Befehlsdekodierung
  - Analyse der Instruktion, Vorbereitung der Ausführung
- OF - Operand Fetch Holen der Operanden
  - Laden der Operandenregister
- EX - Execution Ausführung
  - Ausführung der Instruktion im Rechenwerk
- WB - Write Back Zurückschreiben
  - Rückschreiben des Ergebnisses in Zielregister

Befehlsbearbeitung in einer 5-stufigen Pipeline

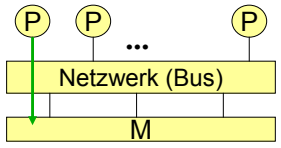
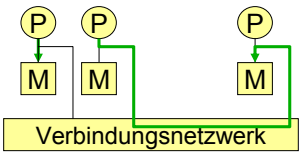
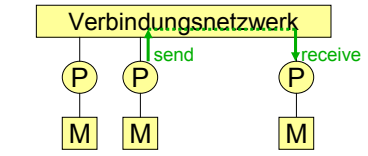


Superskalare Architekturen



- Pipeline mit mehreren Funktionseinheiten
- Funktionseinheiten wie MEM (Speicherzugriff) oder FP (Gleitkommaoperation) benötigen mehr Takte als die übrigen Stufen.

Parallelrechner

Speicher / Adressraum	global	physikalisch verteilt
gemeinsam	SMP (symmetrischer Multiprozessor)  **UMA - uniform memory access**	DSM (distributed shared memory)  **NUMA - non-uniform memory access**
verteilt		Multicomputer, DM(distributed memory)  **(N)UCA - (non-)uniform communication architecture**

Virtualisierung des Speichers

Grund:

Programme wurden so groß, dass sie nicht mehr in den Hauptspeicher passten

=> Modularisierung

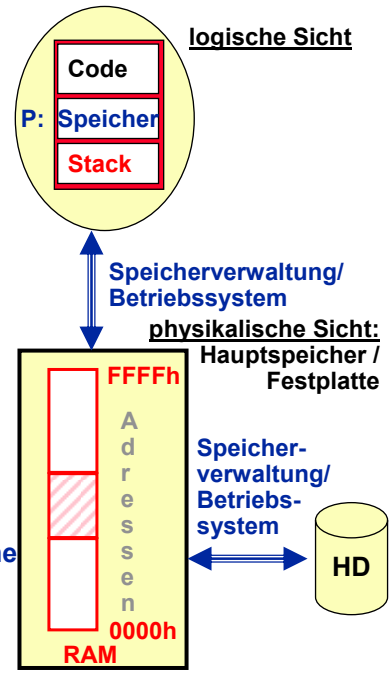
Methode:

Hin- und Herladen von Programmteilen zwischen Hintergrundspeicher und Hauptspeicher durch das Betriebssystem

Programmadressen sind virtuelle Adressen, die vom Betriebssystem in physikalische Adressen umgesetzt werden.

Vorteile:

- virtueller Adressraum größer als physikalische
- Programm ist unabhängig von Datenplatzierung im Hauptspeicher
- Vereinfachung von Mehrbenutzerbetrieb



Von Neumannscher Flaschenhals

Die Schnittstelle zwischen Prozessor und Speicher wird von Neumannscher Flaschenhals (engl. bottleneck) genannt.

Problem

Daten werden vom Prozessor schneller verarbeitet als sie aus dem Speicher gelesen oder in den Speicher geschrieben werden können.

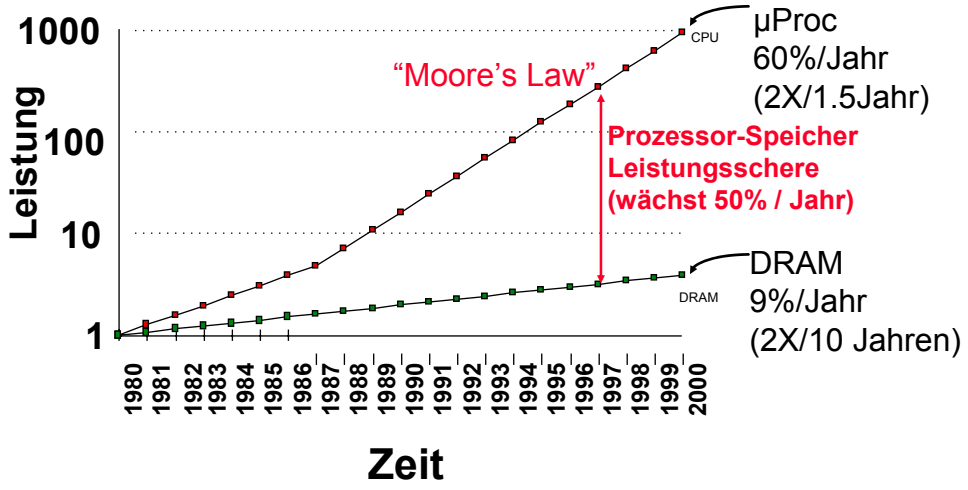
Die Leistungsdiskrepanz zwischen Prozessor und Speicher wird immer größer.

	Kapazität	Geschw.
Logik:	2x in 3 Jr	2x in 3 Jr
DRAM:	4x in 3 Jr	2x in 10 Jr
Disk:	4x in 3 Jr	2x in 10 Jr

DRAM

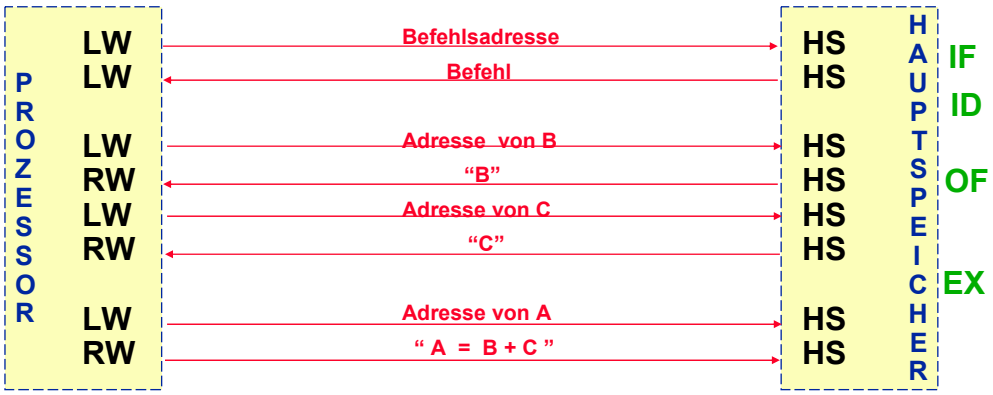
Jahr	Größe	Taktzeit
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns

1000:1! 2:1!



von Neumannscher Flaschenhals

Beispiel: Ausführung der Anweisung "A: = B + C"
=> Problem: Verwendung von Adressen für Operanden und Ergebnisse



↳ **Prozessor /Speicherschnittstelle kritisch für Leistung des Gesamtsystems**
Dies war früher wegen hoher Gatterschaltzeiten unproblematisch, aber durch die Fortschritte der Halbleitertechnik wird die Diskrepanz zwischen der Prozessorgeschwindigkeit und den Speicherzugriffszeiten immer größer.

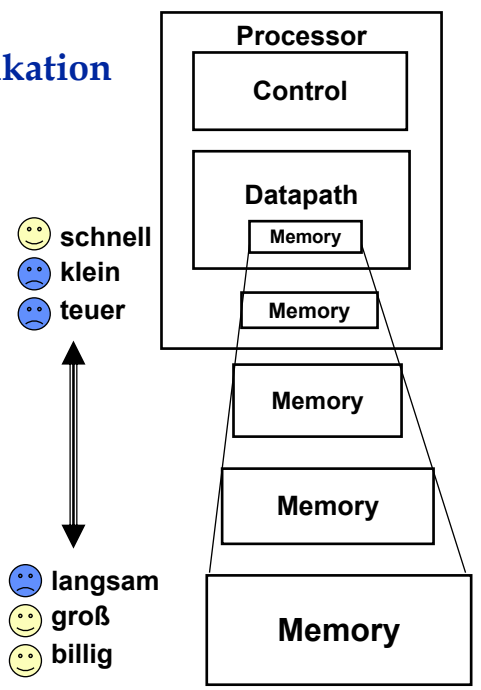
"Beschleunigung" der Prozessor/Speicherkommunikation

Ziel: Arbeitsspeicher so schnell wie schnellste verfügbare Speicherchips und so groß wie die größten

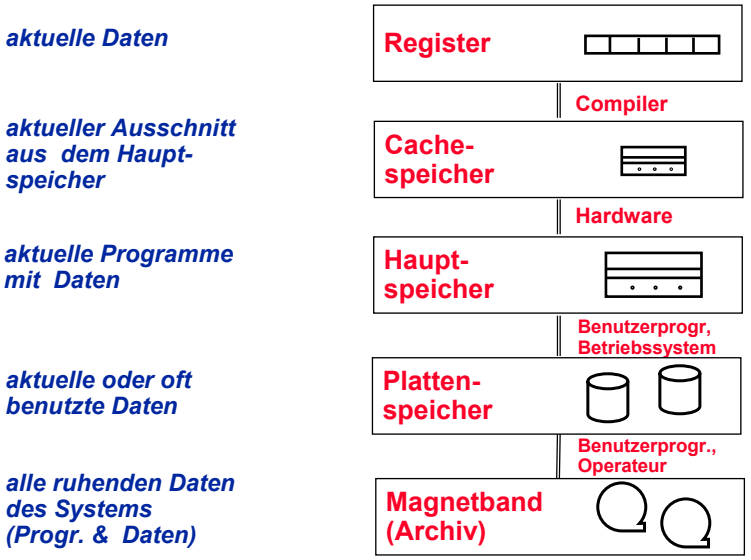
Lösung:
hierarchische Speicherorganisation

- Prozessor kommuniziert mit top-down organisierter Folge von Speichern
- mit wachsender "Entfernung" vom Prozessor
  - steigen Größe und Zugriffszeit
  - fällt der Preis (pro Bit)
- **Ausschnittshierarchie:**

  Alle Daten einer Ebene sind auch in der darunterliegenden Ebene gespeichert.

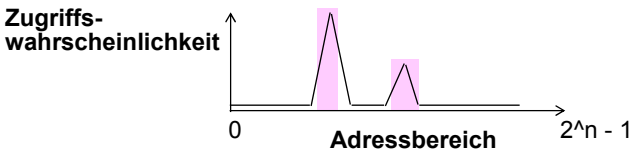


Speicherhierarchie



Lokalität

- **Lokalitätsprinzip: 90/10-Regel**
  90% der Daten eines Programms werden aus 10% des vom Programm insgesamt benötigten Speicher geladen.



- **Temporale Lokalität:**
  - **Halte zuletzt benutzte Daten beim Prozessor**
- **Räumliche Lokalität:**
  - **Bewege Blöcke benachbarter Speicherzellen Richtung Prozessor**