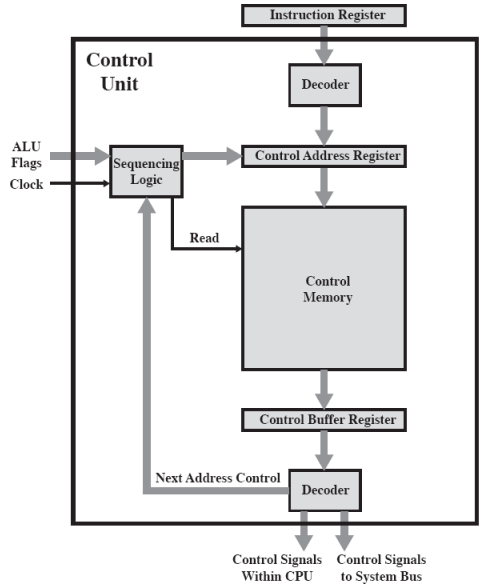
1 Architecture

Machine instruction cycle (5, ggf. weniger)
BH: **B**efehl **h**olen (RISC?)
DE: Befehl **de**kodieren
OP: **Op**erand(en) holen (entfällt bei RISC / Load-Store)
BA: **B**efehl **a**usführen
RS: **R**ück**s**chreiben (RISC?)

1.1 CISC (Complex Instruction Set Computing)



- IR: Makrobefehl
- Dekoder
- CAR: Mikrobefehlsadresse
- CM: Mikroprogrammspeicher
- CBR: Mikrobefehl
- Dekoder

- Seq. Logic: Wählt nächsten Befehl
  1. nächster Mikrobefehl
  2. Mikropr. Sprungziel (alu flags)
  3. nächster Makrobefehl (am Mikroprog. Ende)

\+ **flexibility** (bugfixes BIOS update)
\+ **compatibility** (old generations)
\+ **emulation** of other instruction sets
− most (complex) instructions not used

1.2 RISC (Reduced Instruction Set Computing)

(6 important things)
- Elementary & **small instruction sets**
- **Address calculations explicitly** (no complex addressing modes)
- Operands are given in registers (**Load-Store-Arch**), large universal register sets
- **Hardwired** control units (fast decode, no micro program / CISC)
  $\Longrightarrow$ Instruction & operand fetching in one cycle, decode & op fetch simultan.
- **Pipelining**: One cycle per instruction for EX (not for divisions)
  - Necessary: cooperation of compiler and RISC processor arch.
  - intel archs.: risc / pipelining on micro instructions

Pipelining (+, −, −, 3 limiting factors, 1 extension)
\+ Increased throughput (up to k)
− Increased latency (throughput depends on slowest component, additional FFs)
− Increased energy & transistors & costs: FFs and control logic
- Reduce size of pipeline components: less transistors & additional FFs
  $\rightarrow$ faster, limits: $\lightning$ min one transistor, more stalls (branches), heat
- **Super scalar units** = multiple pipelines / ALUs (e.g. add, mul, ...)
  $\rightarrow$ scheduler, reordering / *dynamic* parallelization $\rightarrow$ Tomasulo / Scoreboard

1.2.1 Optimale Pipelinetiefe

Modell 1 (Architektur) (4 Parameter, Formel)

- T: mittlere Ausführungszeit (ganze Instruktion)
- S: Anzahl Pipelinestufen
- b: Wskt. falsche Sprungvorhersage
- $(S-k)$: Anzahl Wartezyklen

$$G_1(S) = \frac{1}{\frac{T}{S} \cdot (1 + (S-k) \cdot b)}$$

Modell 2 (Architektur + Technologie) (+1 Parameter, Bedeutung, Formel)

- C: Pipeline-Overhead (Speicher-, Setup- & Haltezeiten)
- Verhältnis C/T beschreibt die Technologie

$$G_2(S) = \frac{1}{(\frac{T}{S}+\mathbf{C}) \cdot (1 + (S-k) \cdot b)}$$

Modell 3 (Architektur + Technologie exakter) (was wird präzisiert?)

- Daumenregel für das Verhältnis C/T
- Häufig 11FO4-Designs
- # serielle NAND-Gatter $\rightarrow$ 11 (= kritischer Pfad?? s. 103ff)
- Fan-Out max. 4, sonst muss zu viel Ladung durch ein Gatter

$$S_{\text{opt}} = \sqrt{\left(\frac{1}{b} - k\right) \cdot \frac{T}{C}} \quad \text{(M2)}$$

$$= \sqrt{\left(\frac{1}{0.073} - 5\right) \cdot \frac{1}{0.016}} \cong 23 \quad \text{(M3)}$$

Speedup (formula)

$$T_1 = n\sum_{i=0}^{k}\tau_i \leq nk\tau$$
$$T_k = (k+n-1)\max_i\{\tau_i + \text{delay}\} =: (k+n-1)\tau$$
$$S_k = \frac{T_1}{T_k} \leq \frac{nk}{k+n-1}$$
$$\lim_{n\to\infty} S_k \leq k$$

Hazards (3, examples & countermeasures)

- Structural hazards: Ressource unavailable (2 examples)
  - Same memory controller for data and instructions
    → distinct L1 caches / dual port ram
  - No load/store/alu units available → more units
- Control hazards (5 counters): Next instruction unknown (jumps, branches)
  - → Stalling
  - → Static branch prediction (2): branch taken (→ loops), by opcode
  - → Delayed branch (compiler / instr fetch unit brings branch eval forward)
    jump may not depend on instruction that was pulled back!
  - → Dynamic branch prediction (3)
    - Branch **Loop** Buffer (load (block of) instructions in **fifo buffer**, small loops are buffered, could be decoded instructions)
    - Branch **Prediction** Buffer / Branch History Table
      (last instruction-addr-bits adress a table, 1 or 2 (inner loops) bit history)
      (strongly/weakly (not) taken, edges from weak to opposite strong)
    - Branch **Target** Buffer
      (store history & target **adress** or even **decoded** instruction)
  - → Execute both branches, keep one (↯ exponential num. units)
- Data hazards (3, 4 solutions)
  - **RAW** (read after write)
    instr2 reads result before it was written by instr1
    → **Forwarding**: Pass results back to ALU-inputs & other stages
  - **WAR** (write after read) (requires out-of-order-execution & -commit)
    instr2 overwrites register before it was read by instr1 (not in simple pipelines)
    → (Scoreboard) **delay** write back after the operands are read
    → (Tomasulo) **register renaming** = operand copies for each instruction
  - **WAW** (write after write) (requires out-of-order-commit, e.g. faster instr.)
    instr2 writes register before instr1 writes, thereby overwriting result of instr2
    → **delay** read until operands are ready

  General countermeasure: Reordering of instructions (static &) *dynamic* (3):
  - in/out-of-order issue, execution & commit (completion)
- General hazard countermeasure:
  - Simultaneous Multi Threading (Hyper Threading), only mixed code

Scoreboard (4 Phasen, 4 verhinderte Hazards zuordnen)

1. BH+DE1 (in-order-issue, nur ein Befehl ist in dieser Phase)
   - Instruktion dekodieren & auf **strukturelle Hazards** prüfen
   - Warte, wenn Operationen mit selben Zielregister ausstehen (**WAW**)
   - Instruktion an Funktionseinheit senden & Scoreboard-Zustand aktualisieren
2. DE2 (ab hier mehrere Befehle pro Phase möglich)
   - Warte, wenn vorherige Berechnungen der Operanden ausstehen (**RAW**)
   - Operanden lesen & Verarbeitung starten
3. BA (out-of-order-execution)
   - Beim Beenden von Multizyklus-Operationen Scoreboard benachrichtigen
4. RS (out-of-order-commit, aber nicht immer sofort)
   - Warten, wenn Zielregister noch gelesen werden muss (**WAR**)

Scoreboard-Datenstruktur (3)

1. Instruktionsstatus - Schritt der Instruktion
2. Funktionseinheitenstatus (5)
   - Busy
   - Op
   - Fi: Zielregister; Fj, Fk: Quellregister
   - Qj, Qk: Funktionseinheiten, die die Operanden erzeugen
   - Rj, Rk: Ready-Flags der Operanden
3. Registerergebnisstatus - Anzeige, welche Funktionseinheit darauf schreibt

Instrukt.-Zustand	Warten bis ...	Scoreboard-Updates
Issue	not Busy (FU) and not Result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Operanden lesen	Rj and Rk	Rj← No; Rk← No
Execution beendet	"Funktionseinheit fertig"	./.
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rj(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

FU: Funktionseinheit; Result: Registerergebnisstatus; Busy: Instruktionsstatus aktiv

Instruction status

Instruction	j	k	Issue	Read operanc	Executi comple	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6		
MULTI F0	F2	F4	6			
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
6 FU	Mult1	Integer							

(Auszug Beispiel s. 1-65ff)

Tomasulo (Vorteil, "Nachteil" & Unterschiede, 3 Phasen)

+ Verhindert **WAR** und **WAW** durch **Register Renaming** (Iwann Strukturhazard)
− **mehr** (Schatten-)**Register**, komplexere **Kontrolllogik** (assoziative Speicherung)
- **Reservierungsstation**: Operandenzwischenspeicher für Funktionseinheit
- Registerreferenzen in Instruktionen zu Reservierungsstationszeigern übersetzen
+ Forwarding: Resultate von Berechnungen werden über gem. Datenbus verteilt

1. BH+DE (in-order-issue, nur ein Befehl ist in dieser Phase)
   - Warte bis Reservierungsstation frei (**strukturelle Harards**)
   - Reservierungsstation belegen & initialisieren
2. BA (out-of-order-execution, selbstständig, ab hier mehrere Befehle pro Phase)
   - Warte ("am BUS"), wenn ein Operand noch nicht vorhanden ist (**RAW**)
   - Starte die Berechnung
3. RS (out-of-order-commit)
   - Ergebnis an BUS anlegen (mit (Instruktions-?)Quelladresse statt Zieladdr.)
   - Reservierungsstation freigeben

Tomasulo-Datenstruktur (3)

1. Instruktionsstatus - Schritt der Instruktion
2. Funktionseinheitenstatus (5)
   - Busy
   - Op
   - Fi: Zielregister (fehlt in VL?!); ~~Fj, Fk: Quellregister~~
   - Qj, Qk: Reservierungsstationen, die die Operanden erzeugen
   - Vj, Vk: Werte der Operanden ~~Readyflags~~
3. Registerergebnisstatus - Anzeige, welche Funktionseinheit darauf schreibt

Befehlsstatus

Befehl		j	k	Befehl install.	Befehl beendet	Ergebnis rückschr.
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULT	F0	F2	F4	3		
SUB	F8	F6	F2	4	7	8
DIV	F10	F0	F6	5		
ADD	F6	F8	F2	6	10	

	Busy	Adresse
Load1	No	
Load2	No	
Load3	No	

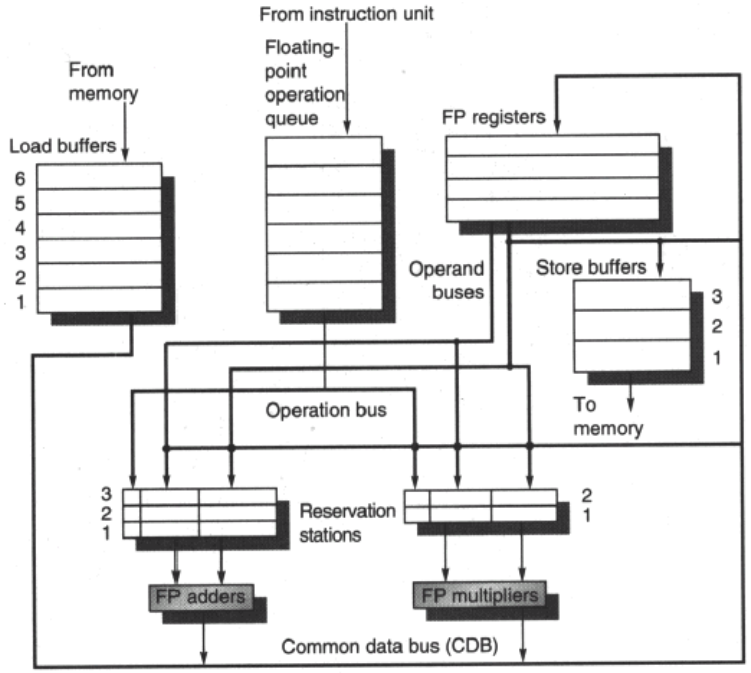
Resevierungs-Station

Zeit	Name	Busy	Op	S1 V_j	S2 V_k	RS für j Q_j	RS für k Q_k
0	Add1	No					
0	Add2	Yes	ADD	M()-M()	M(45+R3)		
0	Add3						
5	Mult1	Yes	MULT	M(45+R3)	R(F4)		
0	Mult2	Yes	DIV		M(34+R2)	Mult1	

Registerergebnisstatus

Takt		F0	F2	F4	F6	F8	F10	F12	F30
10	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2		

(Auszug Beispiel s. 1-82ff)



1.3 Caches

- **Hierarchie** mit **Inklusionsbedingung**, ausnutzen von:
- **zeitliche Lokalität** (mehrfache Zugriffe, z.B. Akkumulator in Schleife)
- **räumliche Lokalität** (stride-one-access, mehrere Daten in selber Cacheline)
- Victim Cache: Kleiner vollassoziativer Cache zur Verringerung von Cache-Thrashing Effekten, welches durch ungünstige Zugriffsoffsets verursacht wird. Wirkt wie Vergrößerung der (also einer *einzelnen*) Cachemenge.

Organisationsformen

- direkt abbildend (Position = **Block**adresse % #Cachezeilen)
  Blockadresse = Adresse ohne Byteadresse innerhalb der Cachezeile
- voll-assoziativ (Position beliebig)
- *n*-fach assoziativ (direkte Abbildung auf Menge, innerhalb vollassoziativ)

s Mengen

N # Cachezeilen insgesamt

n # Cachezeilen pro Menge / Assoziativitätsgrad / # notwendige Komparatoren

- Adressaufteilung (Bitanzahlen berechnen können):
  1. Tag
  2. Index der Cachemenge ($\rightarrow$ **Indexbits**, *nicht* bei vollassoziativen Caches)
  3. Byte innerhalb der Cachezeile
- Bei großen Caches ist ein geringer Assoziativitätsgrad (8) ausreichend
- AMD: Kein(?) Victim Cache, dafür höher Assoziativitätsgrad im L3 Cache

Aktualisierungsstrategien (2(+2))

- Bei Hit ("normal"):
  - write through
  - write back (bei Verdrängung, Ausgabe, Zugriff anderer Prozessor $\rightarrow$ dirty bit)
- Bei Miss (nicht wirklich *Aktualisierung*)
  - write around (e.g. non-temporal writes)
  - write allocate (normales Verhalten)

Ersetzungsstrategien (4, **replacement problem**, nur assoziative Caches)

- LRU, LFU, FIFO, (pseudo) random (letzteres v.a. bei großen Caches gut)

3 Cs (guess how many..., Unterschied letzte beiden laut Fey fließend)

- Compulsory (erster Zugriff auf einen Block)
- Capacity (Konflicht auch bei höherem Assoziativitätsgrad)
- Conflict (Konflicht durch höheren Assoziativitätsgrad vermeidbar)

Cache-Bewertung / Leistungsbetrachtung (Formel, 3 Komponenten)

- $T_a = T_h + m \cdot T_m$: mittlere / Nachlade / Cache-Speicherzugriffszeit
- Reduzierung eines Faktors bewirkt Erhöhung eines anderen $\rightarrow$ guter Kompromiss
- m: Fehlerrate
  - Erhöhung der Kapazität $\rightarrow$ $\lightning$ Cache-Zugriffszeit
  - Erhöhung des Assoziativitätsgrades $\rightarrow$ $\lightning$ Cache-Zugriffszeit
  - Victim Cache
  - Prefetching
- T_h: Cache-Zugriffszeit
  - kleinere Caches $\rightarrow$ $\lightning$ Fehlerrate
  - einfachere Organisationsform $\rightarrow$ $\lightning$ Fehlerrate

- T_m: Nachladezeit
  - Sekundär-Cache (T_m wird selbst zu Formel)
  - Nicht-blockierende Caches (Stalls nicht bei Miss, sondern wenn Operand benötigt, ggf. andere Instruktionen, siehe Scoreboard/Tomasulo)

Ausnutzen von Cache-Effekten (4,5 & verbesserte Lokalität)

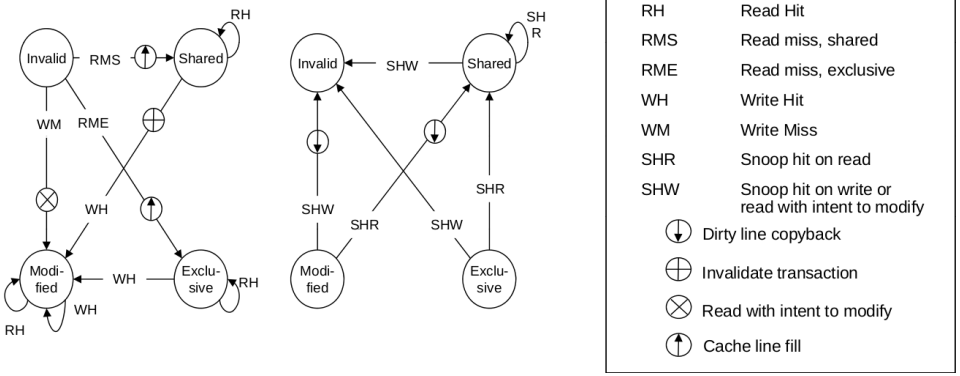
- Speicher umordnen → Matrix transponieren, spatial
  → Array-Merging: Verbundarray statt mehrere Arrays (↯ Cache-Trashing) spatial
  Je nur ein Element des structs benötigt → umgekehrte Wirkung
- Schleifenvertauschung: row major access (sonst ↯ Cache-Trashing) spatial
- Schleifenfusion: temporal
- Blocking (und das geht mit 3 Schleifen (jj i j, blocktraversal col major)!) temporal

Cache-Kohärenz

- **Verzeichnis-Protokolle** (für Cache-Inhalte)
  - Bestandteil zentraler und lokaler Cache-Controller
  - — Flaschenhals (Kommunikation)
  - \+ Effektiv in groß-skalierten System weil viel davon parallel geht
    Flaschenhals bleibt aber, also irgendwie strange...
- **Snoopy-Protokolle**
  - dezentrale Kontrolle auf alle lokalen Cache-Controller
  - Abhören der gesamten Kommunikation zur Erkennung gemeinsam genutzter Cache-Zeilen
  - Neuschreiben einer Zeile an alle Controller broadcasten
  - Geeignet für BUS-gekoppelte Multiprozessoren (→ Broadcast)
  - — Hoher Datenverkehr auf BUS, ↯ Skalierung
    - ∗ **Write-invalidate**: Bsp. MESI, nur invalidieren
    - ∗ **Write-update**: Schreiben zieht sofortigen Broadcast nach sich

MESI-Protokoll (Write-invalidate Snoopy Protokoll, 4 Zustände)

- **Modified** (nur dieser cache, verändert)
- **Exclusive** (nur dieser cache, unverändert)
- **Shared** (mehrere caches, unverändert)
- **Invalid**



(SHW in Modified muss zurückschreiben, mgl. nur Teil der Cacheline modifiziert)

2 Multikern-Architekturen

Motivation & Moore's Law

- Bis etwa 2002:
  - höhere Transistorleistung → höhere Frequenz
  - höhere Transistordichte → komplexere Architekturen
  - geringerer Energieverbrauch pro Logikop. → Leistungsdissipation beschränkt
- Hitzeentwicklung zu hoch
  - $P_{\text{diss}} = C \cdot \rho \cdot f \cdot V_{\text{dd}}^2$
  - P_{diss} Leistungsverbrauchdichte
  - C Gesamte Kapazität
  - ρ Transistordichte
  - f Taktfrequenz (hängt mit V_{dd}^2 zusammen)
  - V_{dd}^2 Versorgungsfrequenz
- Superskalares Prinzip ausgereizt, z.B. Sprungvorhersage 95% Treffer

Allgemein & Pollack

- **homogen** und **heterogen** (Cell, Beschleuniger-Systeme)
- Pollack-Regel: Rechenleistungszuwachs $\cong \sqrt{\text{Komplexitätszuwachs}}$
- "inverse Pollack-Regel": Besser Anzahl Prozessoren verdoppeln
  - \+ einzelner Kern abschaltbar
  - \+ einzelnen Kern mit optimaler Spannung und Frequenz betreiben (*niedriger*)

+ Rechenlast gleichmäßiger verteilen, keine Hotspots über Die
+ Zuverlässigkeit und Leckströme bei niedriger Temperatur geringer
+ heterogene Kerne möglich → Spezialprozessoren → Effizienz

↯ Transistorleistung wird nicht weiter steigen wie bisher
- Leckströme (zu wenig Atome zwischen den Transistoren)
- Verringerung der Versorgungsspannung notwendig
- Frequenz kann nur moderat ansteigen

Amdahl

$$S(N) = \frac{1}{T_{seq} + (1 - T_{seq})/N} \qquad \text{(Paralleler) Speedup, } N \text{ Threads}$$

$$S(N, M) = \frac{1}{T_{seq}/M + (1 - T_{seq})/N} \qquad M \text{ parallelisierte Programme}$$

- Nicht immer so schlimm, weil mehrere Anwendungen (Desktop) / gut parallelisierbare Anwendungen (Simulation)
- Amdahl führt zu Sättigung, Realität zu Verlangsamung durch sync. Overhead

2.1 Das Roofline-Modell

Bestandteile (3 mit Herkunft, Verbesserungen & Bestimmung)
- **Arithmetische Intensität (AI)**: FLOPS / Byte → logscale x-Achse
  - Eigenschaft der Implementierung
  - Verbesserung: Blocking, loop fusion, row-major / stride-1-access
  - Aus Zugriffsmuster abschätzen oder Cache-Hit Benchmark
- **Floating-Point-Performance**: → logscale y-Achse
  - Eigenschaft des Prozessors / der Mikroarchitektur
  - Verbesserung: TLP / MultiCore, ILP, Loop unrolling, Vektorinstr., FLOP-Balance (meist 1 ADD & 1 MUL Unit → ADD- / MUL-bound)
  - Bestimmung: #Kerne, Mikroarchitektur & verwendete Instruktionen
- **Bandwidth**: → logscale(B/W · AI) y-Achse (linear in AI / x-Werten)
  - Eigenschaft des Speichersystems
  - Verbesserung: prefetching, alignment, mehr Speicherbänke, process affinity (sonst durch QPI B/W limitiert)
  - Bestimmung: Per Benchmark messen (STREAM)

Allgemein (Formel resultierende Perf.)
- Bound and Bottleneck Modell
- Betrachtet eig. nur die Hauptspeicherbandbreite, auf Caches übertragbar
- hier: FLOPS = floating point operations (nicht ~~pro Sekunde~~!)
- benötigte Speicherbandbreite: $\frac{\text{Peak F-Perf.}}{\text{AI}} \rightarrow \frac{\text{FLOPS/s}}{\text{FLOPS/Byte}} = \frac{\text{Byte}}{\text{s}}$
- Erreichbare Rechenleistung = $\min\{\text{Peak F-Perf.}, \text{Memory Bandwidth} \cdot \text{AI}\}$ → bandwidth limited / compute bound, je nach kleinerem Wert

3 Eingebettete Systeme

Eingebettete Systeme
- "Computersystem, was in System eingebettet ist, aber nicht als PC erscheint"
- "device that includes programmable computer [...] not a general-purpose pc"
- geringe Flexibilität (für spezielle Anwendung entworfen)
- bei speziellen Aufgaben schneller / effizienter / ...
- in analoge Umgebung eingebettet

Einordnung (4 "Gruppen" / Beispiele, 5 Eigenschaften ↓↑, Unterschiede)
- Universalprozessoren (**CISC**, **RISC**) *kein* ES
  - Befehle werden prinzipiell einzeln & sequentiell abgearbeitet
- (Mikrocontroller, **DSP**s, app. spec. instruction-set proc. ASIPs) ES
- Rekonfigurierbare Hardware (Field Programmable Gate Array **FPGA**s) ES
  - Blöcke aus LookUpTables und FlipFlops
  - Funktionalität und Verbindungen "umprogrammierbar"
  - Verhalten nach Konfigurierung ähnlich ASIC
- (Application Specific Integrated Circuits **ASIC**s) ES?
  - "Leitungen", keine große ALU sondern flussorientiert angeordnete Gatter

↓ zunehmende Stückzahlen, Leistung

↑ zunehmende Flexibilität, Kosten, Leistungsverbrauch

Mikrocontroller
- Kontrollfluss-dominanter Code → geringe Wortbreiten
- einfach, günstig & effizient
- geringer Datendurchsatz → wenig arithmetische Operatione
- priphere Einheiten integriert (A/D, D/A, Timer, etc.)

Hochleistungs-Mikrocontroller
- Normale Wortbreiten
- MultiCores
- hohe Datenraten / hohe Berechnungsanforderungen

Digitale Signalprozessoren (DSPs)
- Sensoren
- A/D bzw. D/A Wandler
- Signal bleibt stabil, ohne Verzerrung speichern / übertragen
- Analoge Filter nicht für sehr niedrige Freq. geeignet, digitale schon

Beispiel TSP: RC-Tiefpassfilter
- Harte Realzeit-Anforderung, jeder Takt ein Ergebnis
- Analog: $RC\frac{\partial V_{\text{out}}}{\partial t}(t) + V_{\text{out}}(t) = V_{\text{in}}(t)$
- Digital:
  - $V_{\text{in}}(t) \Rightarrow X_1, X_2, ...$
  - $V_{\text{out}}(t) \Rightarrow Y_1, Y_2, ...$
  - $Y_n = \alpha Y_{n-1} + \beta X_n$
    $\rightarrow$ 2 MUL, 1 ADD, 1 Register fürs Delay

4 Parallelrechenarchitekturen

(5 topics)
- Normaler Rechner $\rightarrow$ Funktionales Trennen / Pipelining / Datenparallelität
- **Vektorrechner** $\rightarrow$ spezialisierte Einheiten, z.B. Pipelinestufen Vektor-Pipeline
  - am besten eine ALU / Vektorelement (Feldr.?!), tatsächlich Pipelines
- **Feldrechner** $\rightarrow$ gleichartige Rechenwerke
  - gleiche Recheneinheiten mit zentraler Steuereinheit (vgl. Vektorinstr.)
  - Netzwerk, z.B. Gitter
  - durch Maskierung einzelne Recheneinheiten ausschließen (vgl. GPUs)

  $\rightarrow$ Signal- / Bildverarbeitung (stencil), datenparallele Algos.
- **Multiprozessoren** $\rightarrow$ Standardprozessoren / Mehrkern-Prozessor (ein Rechner)
  - **Speichergekoppelte Multiprozessoren**, gemeinsamer Adressraum
  - Symmetric Multiprocessor (**SMP**): ein glob. Speicher, gleiche Zugriffszeit
  - Distributed-shared-memory-systen (**DSM**): mehrere Speicher, versch. Zugrifszeit
  - Uniform-memory-access (**UMA**): ein Speicher, gleiche Zugriffszeit
  - Nonuniform-memory-access (**NUMA**): Speicher auf Prozessoren aufgeteilt
    ccNUMA: cache-coherent NUMA
- **Cluster-Computing** $\rightarrow$ vollständige Rechner
  - **Nachrichtengekoppelte Multiprozessoren**
  - Uniform-communication-architecture (**UCA**):
    - all-pairs einheitliche Übertragungszeit
  - Non-uniform-communication-architecture (**NUCA**):
    - verschiedene Übertragungszeiten, von Netzwerkstruktur abhängig
  - Cluster-Nodes enthalten Multiprozessoren $\rightarrow$ **Kombination, MPI + X**
  - Nodes über Netzwerk verbunden (InfiniBand direkt an PCIe-Slot)

Parallelrechentechnik in Mikroprozessoren (4)
- Superskalarität $\rightarrow$ funktionales Trennen / spezielle ALUs $\rightarrow$ ILP
- VLIW (heute nicht mehr)
- Synchronous Multithreading (HT)
- MultiCore

Flynn (4 Kategorien & Beispiele)
- SISD: klassischer von-Neumann-Rechner
- SIMD: Pipelining, ILP, Vektorinstruktionen, Feldrechner, Vektorrechner
- MISD: klingt fast wie MIST $\rightarrow$ leer (ggf. Sicherheit $\rightarrow$ Flugzeug / AKW)
- MIMD: MultiCores, Cluster (MPI: SPMD = Single program multiple data)