Prozessoren

Der klassische Einzelprozessor

- führt zu einer Zeit einen Befehl aus,
- Ausführungsreihenfolge der Befehle bestimmt die Verarbeitungsergebnisse (Steuerflußprinzip),
- der einzelne Befehl:
  - wirkt auf vergleichsweise wenige - und zudem elementare - Operanden,
  - führt vergleichsweise elementare Operationen aus.

v.Neumann-Architektur und Harvard-Architektur

v.Neumann: gemeinsamer Speicher für Daten und Befehle

- Universalität,
- gute Ausnutzung des Speichers sowohl für kurze Programme und viel Daten als auch umgekehrt,
- Leistungsnachteile können durch Caches weitgehend ausgeglichen werden,
- Programme können wie Daten gehandhabt werden,
- nur eine Virtualspeicherorganisation erforderlich,
- selbstmodifizierende Programme möglich.

Harvard: gesonderte Speicher für Daten und Befehle

- unabhängiger, paralleler Zugriff auf Daten und Befehle,
- Befehlsformate und Datenformate voneinander unabhängig (kein Zwang zur Angleichung).

Steuerflußprinzip und Datenflußprinzip

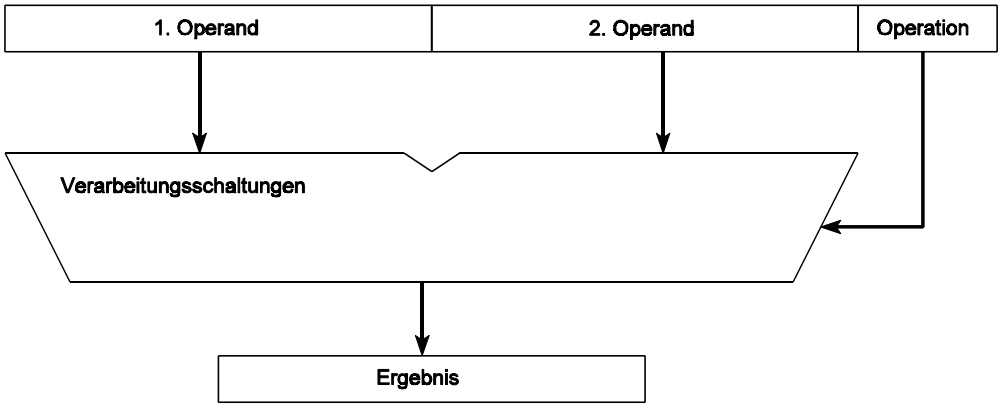
Steuerflußprinzip
Die nacheinander abgerufenen Steuerangaben (Befehle) bestimmen die Verarbeitungsreihenfolge: Befehlslesen - Datenlesen - Befehlsausführung - Entscheidung über den Fortgang.

- intuitiv einsichtig (= automatisierte Steuerung einer Rechenmaschine in Analogie zum numerischen Rechnen von Hand),
- technisch vergleichsweise einfach zu verwirklichen,
- kommt mit adressierbaren Speichern aus,
- Fortschritte der Technologie werden unmittelbar wirksam im Hinblick auf die Verarbeitungsgeschwindigkeit aus (Skalierung).

Datenflußprinzip
Verfügbarkeit der Daten und Datenabhängigkeiten bestimmen die Verarbeitungsreihenfolge.

Einfachlösung (1)
Gemäß dem Datenfluß zusammengeschaltete Operationswerke (Signalprozessoren, Spezialmaschinen).

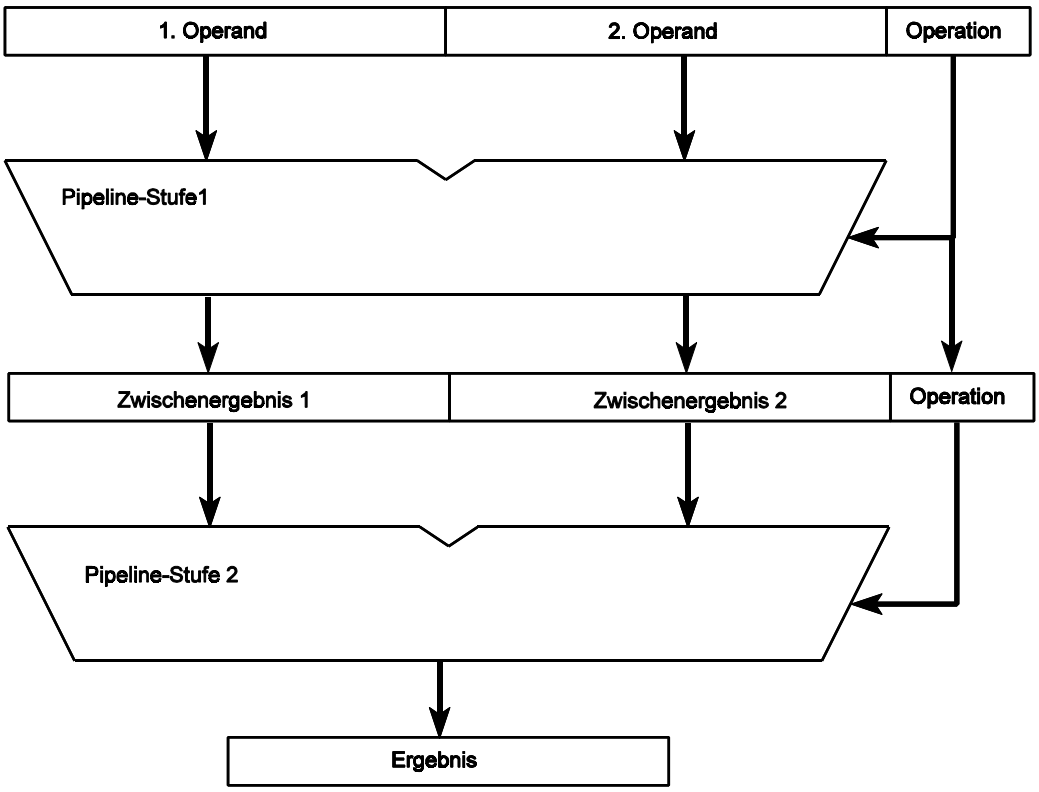
Einfachlösung (2)
Befehlsablaufsteuerung in Hochleistungsrechnern. Zu verarbeitende Daten und Operationssteuerangaben bewegen sich gemeinsam durch die Pipeline.

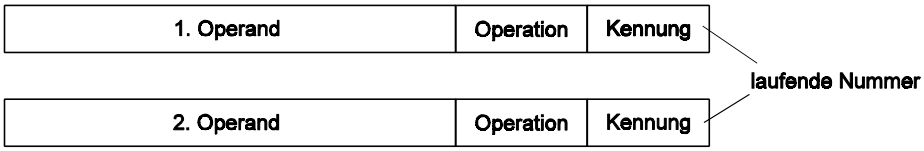


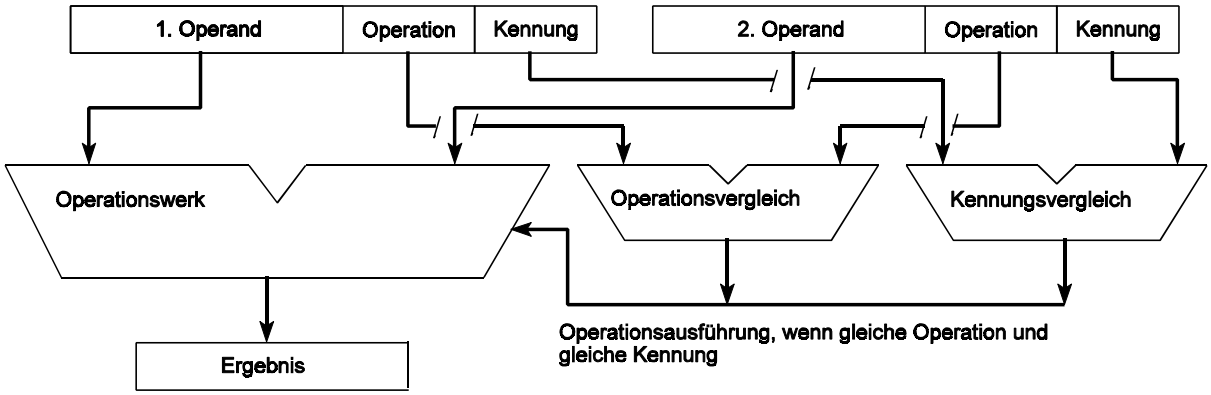
Einfachlösung (3)
Befehlsablaufsteuerung in Hochleistungsprozessoren über assoziativ adressierten Steuerspeicher (Reordering Buffer).

Komplexlösung (fiktiv)
In der akademischen Welt hängen Datenflußmaschinen und Programmierphilosophien eng zusammen. Hier soll ein universelles Datenflußschema ohne Berücksichtigung von Programmierphilosophien erläutert werden.

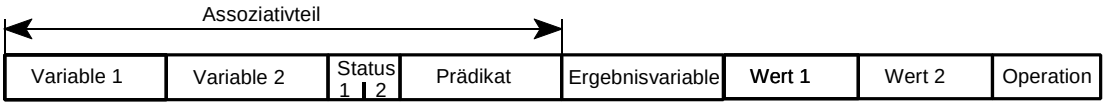
Operationen finden dann statt, wenn die benötigten Daten verfügbar sind. Hierzu brauchen wir anstelle eines adressierbaren eine assoziativen Steuerspeicher (der in unserem Entwurf auch gleich die Variablenwerte mit enthält). Eine offensichtlich aufwendige Angelegenheit...


1. Operand
2. Operand
Operation
Pipeline-Stufe1
Zwischenergebnis 1
Zwischenergebnis 2
Operation
Pipeline-Stufe 2
Ergebnis

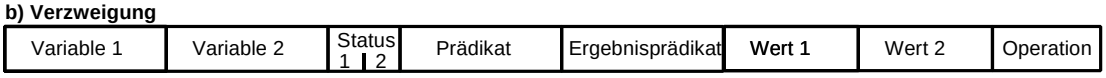

1. Operand
Operation
Kennung
laufende Nummer
2. Operand
Operation
Kennung

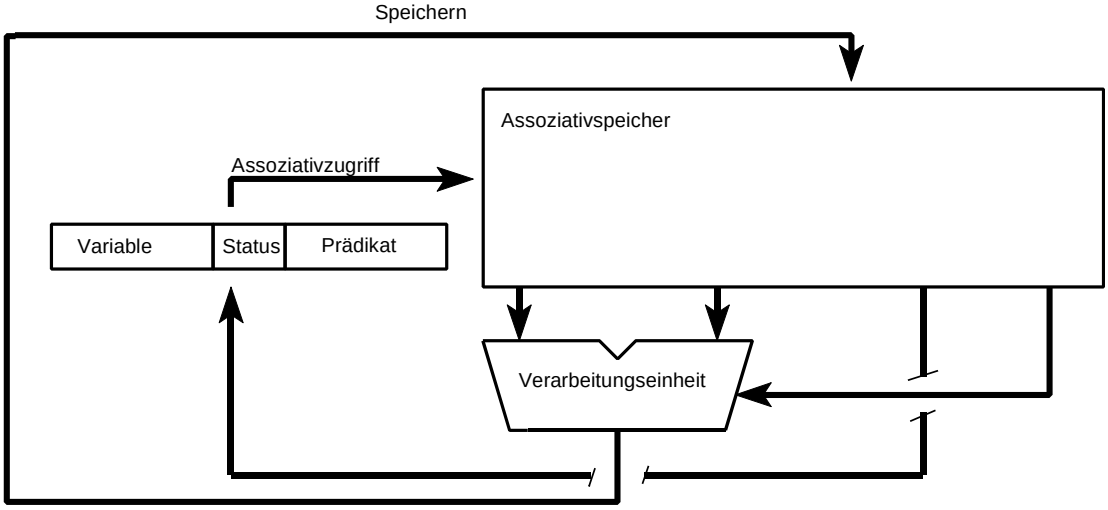

1. Operand
Operation
Kennung
2. Operand
Operation
Kennung
Operationswerk
Operationsvergleich
Kennungsvergleich
Ergebnis
Operationsausführung, wenn gleiche Operation und
gleiche Kennung

a) Verknüpfungsoperation



b) Verzweigung





a) Steuerwortformat

Variable 1	Variable 2	Status	Prädikat	Ergebnisvariable	Wert 1	Wert 2	Operation

b) Berechnung des Funktionswertes X := (A + B) · (C + D)

Umformung:

HILF1 := A + B — HILF1 wird berechnet, sobald die Werte von A und B verfügbar sind

HILF2 := C + D — HILF2 wird berechnet, sobald die Werte von C und D verfügbar sind

X := HILF1 · HILF2 — X wird berechnet, sobald HILF1 und HILF2 verfügbar sind

A	B	STATUS 1 | 2	Prädikat	HILF1	Wert 1	Wert 2	ADD
C	D	STATUS 1 | 2	Prädikat	HILF2	Wert 1	Wert 2	ADD
HILF1	HILF2	STATUS 1 | 2	Prädikat	X	Wert A+B	Wert C+D	MUL

Anfänglich: alle Variablenwerte ungültig.

Variablenwerte werden eingetragen. Der Werteintrag macht die betreffende Variable gültig.

Sind beide Variable im Steuerwort gültig, wird die betreffende Operation ausgelöst. Daraufhin werden die Variablen im betreffenden Steuerwort ungültig (manche Operationen lassen eine der beiden Variablen gültig; zudem gibt es den Sonderfall Variable OP Direktwert).

Das Ergebnis wird der jeweiligen Ergebnisvariablen als Wert zugewiesen. Der Werteintrag macht die betreffende Ergebnisvariable gültig.

Steuerung der Verarbeitungsreihenfolge:

Es werden nur jene Steuerworte ausgeführt, die dem aktuellen Prädikat entsprechen.

Verzweigungsoperationen liefern als Ergebnis ein neues Prädikat.

IF A > B THEN x

Ist die Relation A > B erfüllt, wird das neue Prädikat x wirksam. Die Umschaltung triggert den Abruf all jener Steuerworte, deren Prädikat = x ist und die gültige Variablenpaare enthalten.

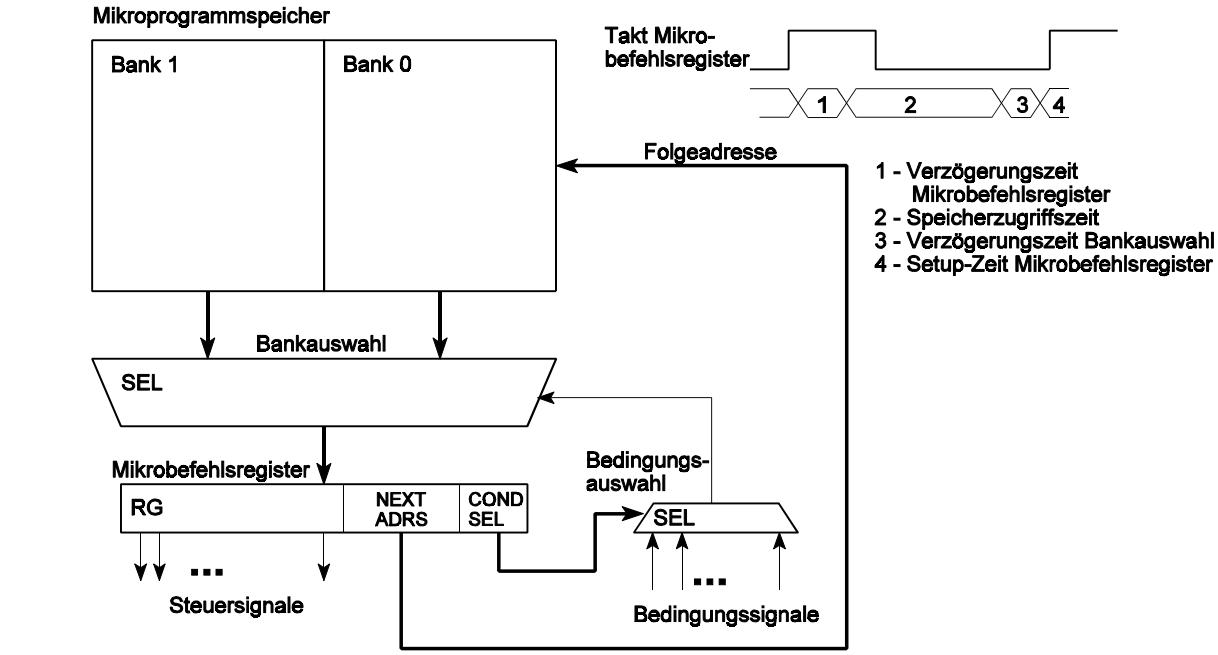
Ist die Relation nicht erfüllt, bleibt das bisherige Prädikat erhalten (ELSE-Zweig).

Unterprogrammruf: durch Retten/Rückspeichern von Prädikaten. Unterprogramm übergibt Werte an Variable mit dem “alten” Prädikat, so daß nach der Rückkehr das Rechnen weitergehen kann.

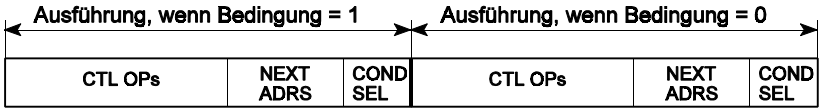
Spezialrechner (Datenstrukturmaschinen)

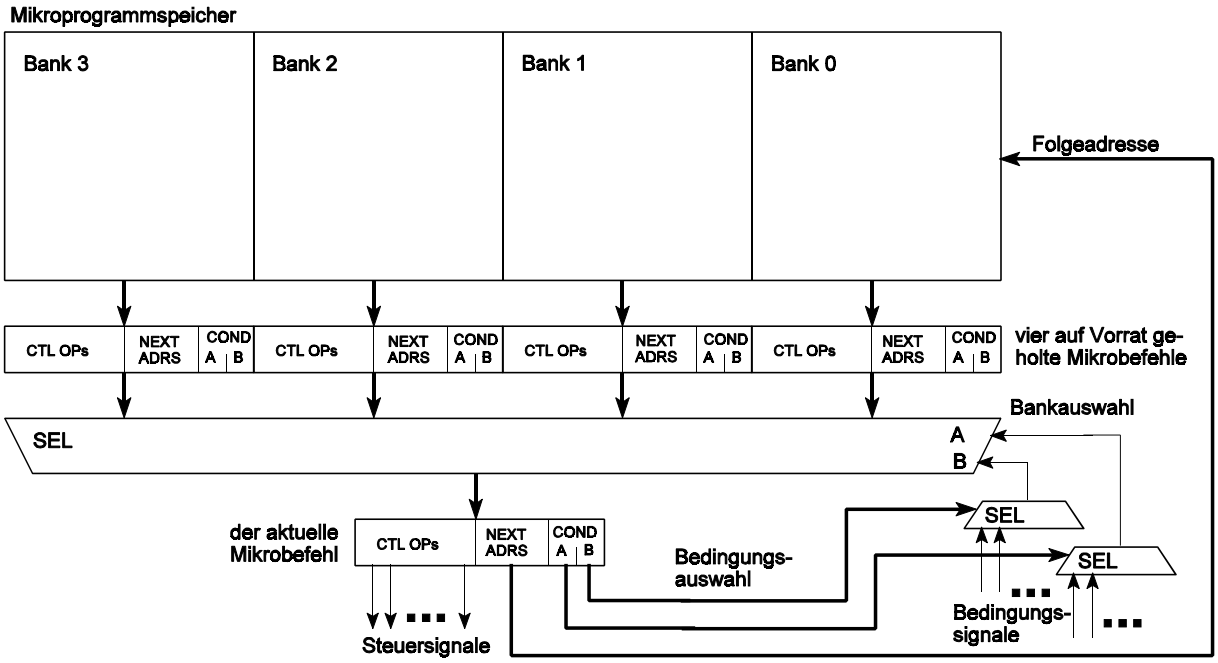
- Verarbeitungswerke an die jeweiligen Datenstrukturen angepaßt,
- elementares Datenflußprinzip druch zweckgerechtes Zusammenschalten von Verarbeitungswerken,
- Verarbeitung, Adressierung und Schleifensteuerung überlappen sich (jeweils in speziellen Werken ausgeführt),
- mehrere Speicher, die an die jeweiligen Datenstrukturen angepaßt sind (über Harvard-Prinzip hinausgehend),
- “Befehle” können komplexe Operationen über komplexe Datenstrukturen betreffen. (Kann sein, daß der einzelne Befehl eine länger dauernde Operation anwirft. MIPS-Angaben gegenstandslos.)

Entwurfsziel: nicht wenigstens ein *Befehl* je Taktzyklus, sondern eine *Nutz*operation (die zur angestrebten Problemlösung beiträgt) je Taktzyklus.



Mikrobefehle werden stets paarweise gelesen:





CISC	***RISC***
komplexe Befehle	***einfache Befehle***
Was ist hier komplex? ▪ Operationsbefehle mit Speicherzugriff ▪ verschiedene Arten der Adreßrechnung ▪ Unterstützung aller 4 Grundrechenarten ▪ Unterstützung variabel langer Datentypen (Dezimalzahlen, Zeichenketten usw.) ▪ Blockoperationen (Transportieren, Wandeln, vergleichen) ▪ Unterstützung komplizierter Organisationsabläufe (Funktionsaufruf, Taskumschaltung usw.) So großartig komplex ist CISC aber auch nicht - die meisten Befehle haben nur einen Speicher- und einen Registeroperanden: <R> := <R> **OP** <MEM>	Was ist hier einfach? ▪ Operationsbefehle wirken nur auf Registerinhalte ▪ Trennung von Speicherzugriffs- und Operationsbefehlen (LOAD-STORE) ▪ nur einfache Adreßrechnung in den Speicherzugriffsbefehlen (Basis + Displacement) ▪ es werden nur Datentypen fester Länge unterstützt ▪ nur Operationen, die in einem Maschinenzyklus ablaufen können ▪ nur elementarer Unterprogrammruf ▪ alles, was komplizierter ist, ist auszuprogrammieren
Befehle variabler Länge. Kompakt, aber aufwendig zu decodieren (ggf. mehrere Pipeline-Stufen).	Befehle fester Länge. Teils Platzverschwendung, aber einfach zu decodieren (oft genügt eine einzige Pipeline-Stufe).
Mikroprogrammsteuerung	sequentielle Steuerung
Leistungsvermögen des einzelnen Befehls entscheidend (darf länger dauern, wenn er entsprechend viel leistet). Anwendungs-seitige Eleganz (Assembler-programmierung).	Ausführungsgeschwindigkeit entscheidend. Alle Befehlsabläufe müssen in ein Pipeline-Schema passen (es wird nur unterstützt, was in so ein Schema paßt - ohne Rücksicht auf Eleganz und Komfort). Ohne Compiler nur schwer zu programmieren.
Anwendungsproblem wird mit vergleichsweise wenigen, leistungsfähigen, langsam ablaufenden Befehlen gelöst. Speicherbandbreite vergleichsweise unkritisch. Wegen des kompakten Codes sind die Cache-Trefferraten hoch. Befriedigende Leistung auch auch mit vergleichsweise kleinen Caches. Steuerung kompliziert.	Anwendungsproblem wird mit elementaren, schnell ablaufenden Befehlen gelöst. Deshalb werden mehr Befehle benötigt. Speicherbandbreite leistungsentscheidend. Befriedigende Leistung erfordert bisweilen große Caches. Steuerung vergleichsweise einfach.
Befehlsliste (API) isoliert Hardware von Architektur (eine Architektur, viele verschiedenen kompatible Implementierungen). Shrinkwrapped Software praktikabel.	Architektur und Hardware hängen eng zusammen. Im (akademischen) Extremfall schlägt Struktur der Pipeline auf die API durch. Beeinträchtigt Narrenfreiheit beim Implementieren. Shrinkwrapped Software nicht immer praktikabel (statt dessen Neucompilierung für jede neue Hardware)

RISC-Leistung mit einer CISC-Architektur

Bedeutet: in jedem Maschinenzyklus einen Befehl auszuführen.

Ist das möglich? - Ja, wenngleich mit einigem Aufwand:

- Befehlspipeline
- Register-Transfer-Struktur gemäß den zu unterstützenden Wirkungen (= gesonderte Schaltmittel in jeweils voller Verarbeitungsbreite für jede Befehlswirkung)
- horizontale Mikroprogrammsteuerung, wobei jedem Befehl ein einziger Mikrobefehl entspricht

Anmerkungen:

- das gelingt nicht für alle Befehle,
- wer mit den so unterstützten Befehlen programmiert, wird (meistens) durch höhere Leistung belohnt. Das führt auf eine RISC-Philosophie beim Programmieren (bzw. beim Erzeugen von Maschinencode (= beim Compilieren)) hinaus. Die wirklich komplexen befehle weden kaum genutzt.

RISC für Praktiker

- Extreme akademische Auffassungen werden aufgegeben -

- komplexere Befehlswirkungen (alle 4 Grundrechenarten und mehr),
- hardwareseitige Steuerung der Pipeline (API von Pipeline-Struktur unabhängig)

Vom akademischen RISC-Ansatz verbleiben:

- Operationsbefehle der Art Register - Register - Register (3 Registeradressen)
- große Universalregistersätze (32 Register und mehr)
- Speicherzugriffe nur mit besonderen Befehlen (LOAD-STORE)
- einfacher Unterpogrammruf
- keine Unterstützung variabel langer Datentypen

Auch RISC-Prozessoren können mehr als 200 verschiedene Befehle haben (z. B. PowerPC)...

RISC und Mikroprogrammierung

Befehls- und Datencaches mit parallelen, unabhängigen Zugriffswegen machen aus der v. Neumann-Maschine eine Art Harvard-Maschine. Befehls-Cache entspricht Mikroprogrammspeicher. RISC-Befehle entsprechen näherungsweise vertikalen Mikrobefehlen*), sind aber ein entschieden besseres Ziel für einen Compiler.

*): VLIW-Befehle entsprechen näherungsweise horizontalen Mikrobefehlen

Leistungsschwächen (von RISC im Vergleich zum Mikroprogramm):

- es werden weniger Befehlswirkungen parallel erbracht
- Verzweigungen kosten Zeit (sind weniger effektiv als clevere Mikrobefehls-verzweigungsverfahren)

Ausgleich:

- Parallelausführung mehrerer Befehle (Superskalarität)
- Sprungvorhersage