

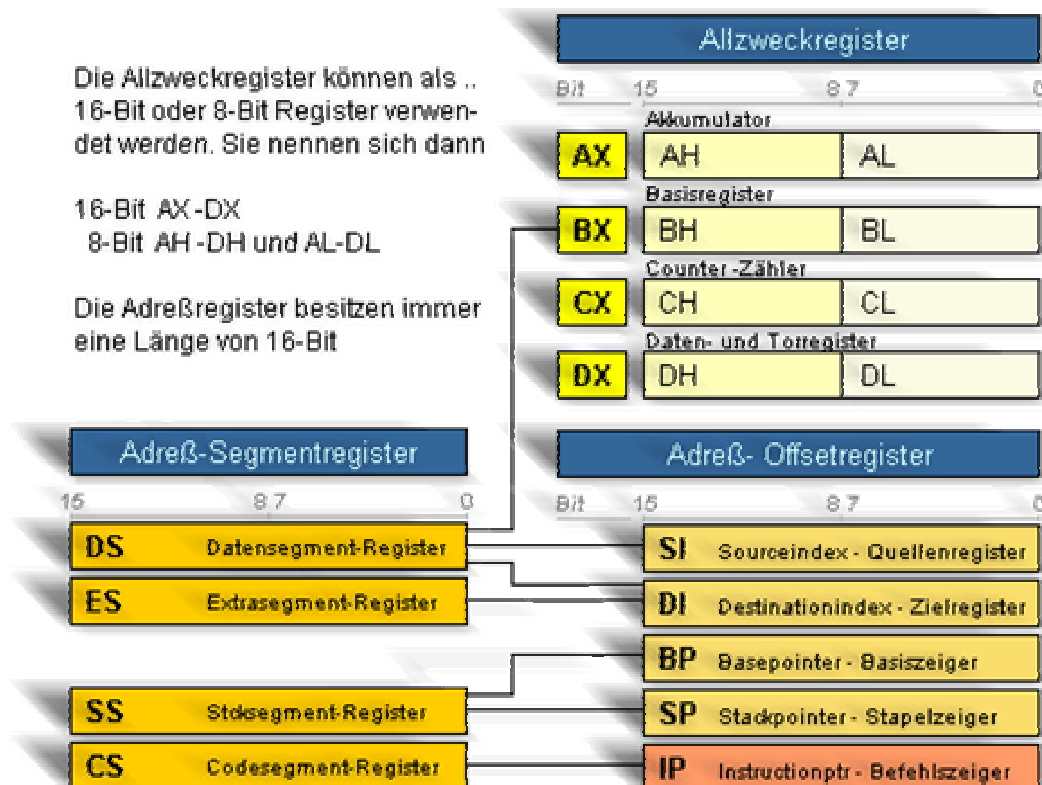
Kapitel 4

Befehlssatzarchitektur (*Instruction Set Architektur ISA*) Die Hardware-Software-Schnittstelle

- Datentypen, Datenformate
- Befehlsformat, Befehlssatz
- Adressierungsarten
- Diskussion: RISC & CISC



Programmiermodell der Intel 80x86



Programmiermodell der Intel 80x86

Der Registersatz ist eine Aufwärts-Entwicklung der Registersätze von 8080 und 8086:

Register des 8080:	8 Bit Register	16 Bit Register
	AL,	SP, IP
	DL, DH (DX)	
	CL, CH (CX)	
	BL, BH (BX)	

Einige 8 Bit Register zusammengefaßt zu 16 Bit Registern (DX, CX, BX)

AL mit dem Statusregister zu einem 16 Bit Register zusammengefaßt (PSW, Processor Status Word)



Programmiermodell der Intel 80x86

Register des 8086:

- Alle Register jetzt 16 Bit Register (AX, DX, CX, BX, SP, IP)
- zusätzliche 16 Bit Register BP, SI, DI
- zusätzliche 16 Bit Register CS, DS, SS

Aus Kompatibilitätsgründen und zur Unterstützung Byte-Orientierter Probleme sind die Register AX, DX, CX und BX weiterhin byteweise ansprechbar

Die zusätzlichen 16 Bit Register CS, DS und SS dienen als Segmentregister zusammen mit SP und IP, sowie CX und DX zur Erweiterung des Adressraums auf 20 Bit (1Mbyte)



Programmiermodell der Intel 80x86

Erweiterung aller Register auf 32 Bit

□ **Allgemeine Register:** Daten- und Adressregister, aber teilweise mit Spezialfunktionen

- Akkumulator (EAX)
- Datenregister für Ein-/Ausgabe (EDX)
- Zählregister für Schleifen (ECX)
- Basisregister (EBX, EBP)
- Indexregister (ESI, EDI)
- Stackregister (ESP)



Programmiermodell der Intel 80x86

□ **Spezialregister:**

- **Segment-Register** zur virtuellen Adressverwaltung (logische → physikalische Adresse) sowie aus Kompatibilitätsgründen zu 8086
- Code-Segment Register (CS)
- Stack-Segment Register (SS)
- Daten-Segment Register (DS - GS)
- **Programmzähler** (EIP)
32 Bit (4 Gbyte Adressraum)



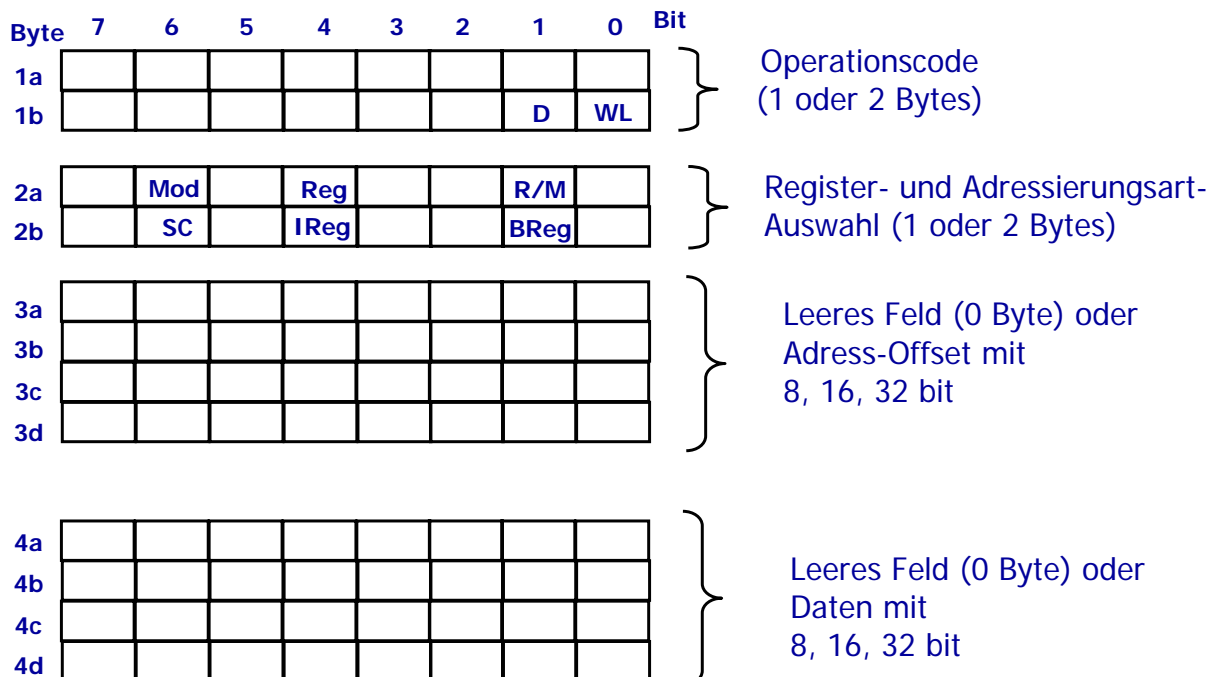
Programmiermodell der Intel 80x86

□ Spezialregister:

- **Statusregister** (EFR): Flags, wie CF, PF, AF und DF
- **Steuerregister** (CR0, Control Register 0): enthält Flags, die den Prozessorzustand beschreiben
 - PG: Paging Enable (Speicherverwaltung mit Seitenwechsel)
 - ET: Processor Extension (Art der Coprozessorsteuerung)
 - TS: Task Switch (Prozesswechsel)
 - EM: Emulate Coprozessor (Coprozessor Emulation)
 - MP: Monitor Coprozessor (Coprozessor Überwachung)
 - PE: Protection Enable (virtuelle (protected) oder reale (real) Speicherverwaltung)



Befehlsaufbau der Intel-x-86-Prozessoren



Befehlsaufbau der Intel-x-86-Prozessoren

- ❑ Ein Befehl besteht aus maximal 12 Bytes.
- ❑ Der Opcode belegt je nach Befehlstyp 1 oder 2 Bytes
 - Das WL (Word Length) bestimmt die Länge eines im Befehl „unmittelbar“ angegebenen Datum oder eines Offsets
- ❑ Die folgenden beiden Bytes dienen zur Auswahl der Adressierungsart und der dabei benutzten Register
- ❑ Falls ein Offset für die Adressberechnung erforderlich ist, wird dieser in den folgenden 1 bis 4 Bytes angegeben
- ❑ In den nächsten Bytes kann ein „unmittelbar adressiertes“ Datum mit einer Länge von 1, 2, 4 byte auftreten.



CISC & RISC

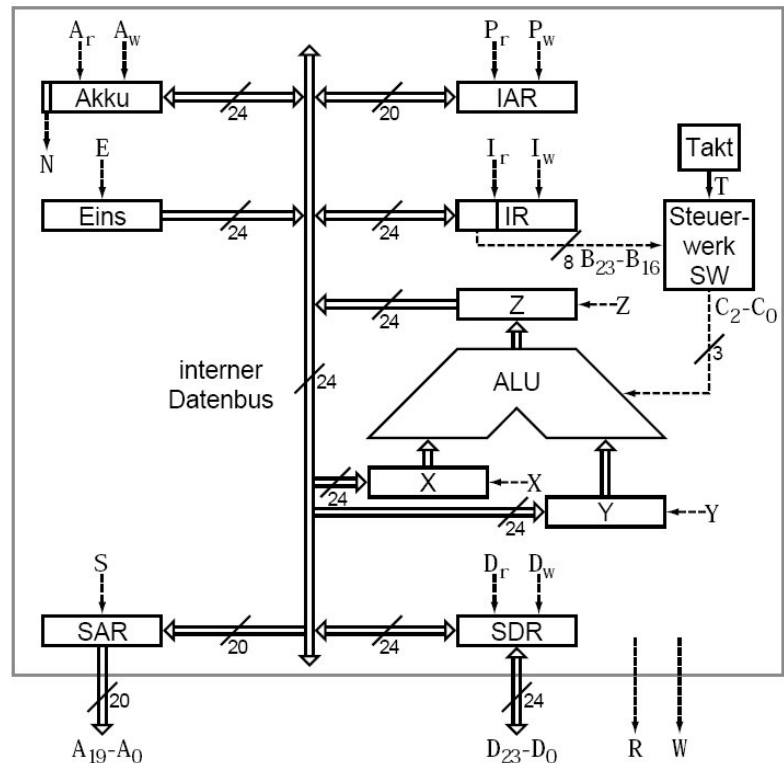
Zwei Techniken zur Implementierung von Befehlen im Rechner

- **Direkt durch Hardware**
 - aufwendige Schaltnetze und Schaltwerke bei großer Anzahl von Befehlen (200 300)
- **Mikroprogramme als Befehlsinterpreter**
 - Mikroprogrammspeicher im Steuerwerk, der neu geladen werden kann
 - verschiedene Befehlssätze können implementiert werden



Mima Architektur (Übungsblatt 3)

- Mikroprogrammierte Minimalmaschine (von Neumann Prinzip)
- SW mit 10 Meldesignale, 18 Steuersignale und Mikroprogrammspeicher für maximal 256 Mikrobefehle
- Befehlsabarbeitung:
 - Lese-Phase
 - Dekodierphase
 - Ausführungsphase
- 3 Taktzyklen für Lese und Schreibzugriffe



Mikroprogrammsteuerwerke

Mikrobefehlsformat:

A_r	A_w	X	Y	Z	E	P_r	P_w	I_r	I_w	D_r	D_w	S	C_2	C_1	C_0	R	W	0	0	Folgeadresse F
27	24					20				16				12				9	8	0

Jedes Bit des Mikrobefehls entspricht einem Steuersignal

- **Horizontale Mikroprogrammierung:**
Steuerungsfeld im Mikrobefehl für jedes Steuersignal im Rechner → Kein Dekoder
- **Vertikale Mikroprogrammierung:**
Zusammenfassung von Steuersignalen zu Feldern.



Mikroprogrammsteuerwerke

Fetch-Phase bei der MIMA:

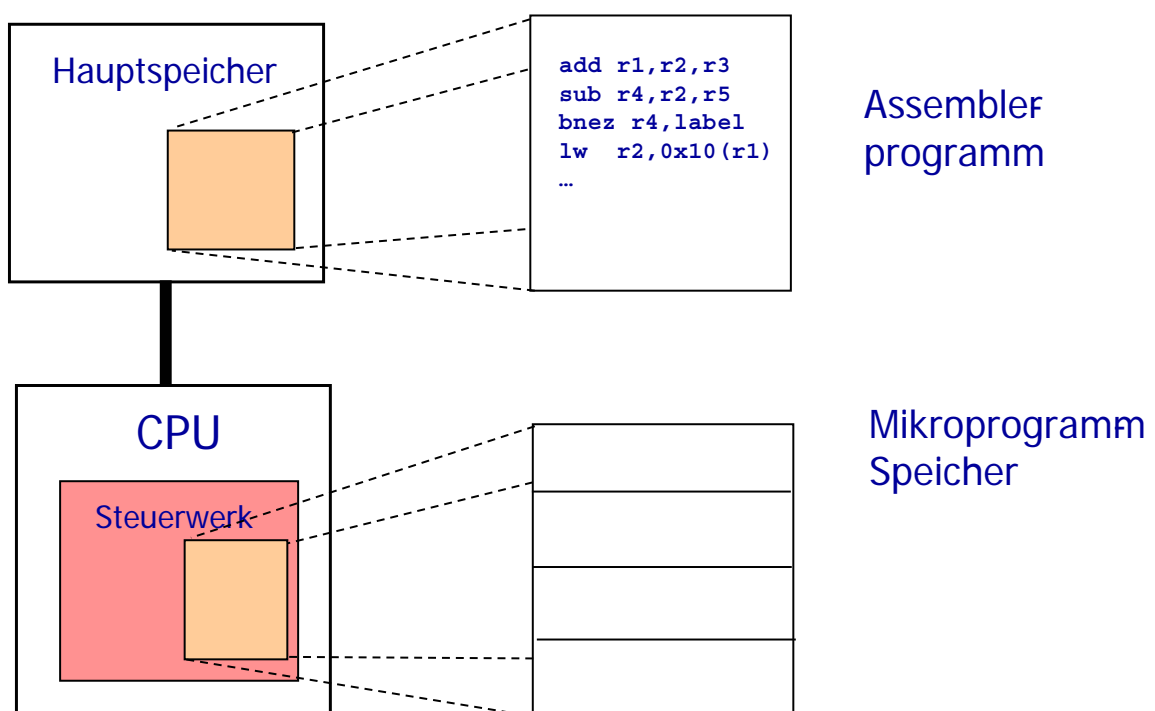
1. Takt: IAR -> SAR; IAR -> X; R = 1
2. Takt: Eins -> Y; ALU auf addieren; R = 1
3. Takt: ALU auf addieren; R = 1
4. Takt: Z -> IAR
5. Takt: SDR -> IR

Mikroprogramm der Fetch Phase

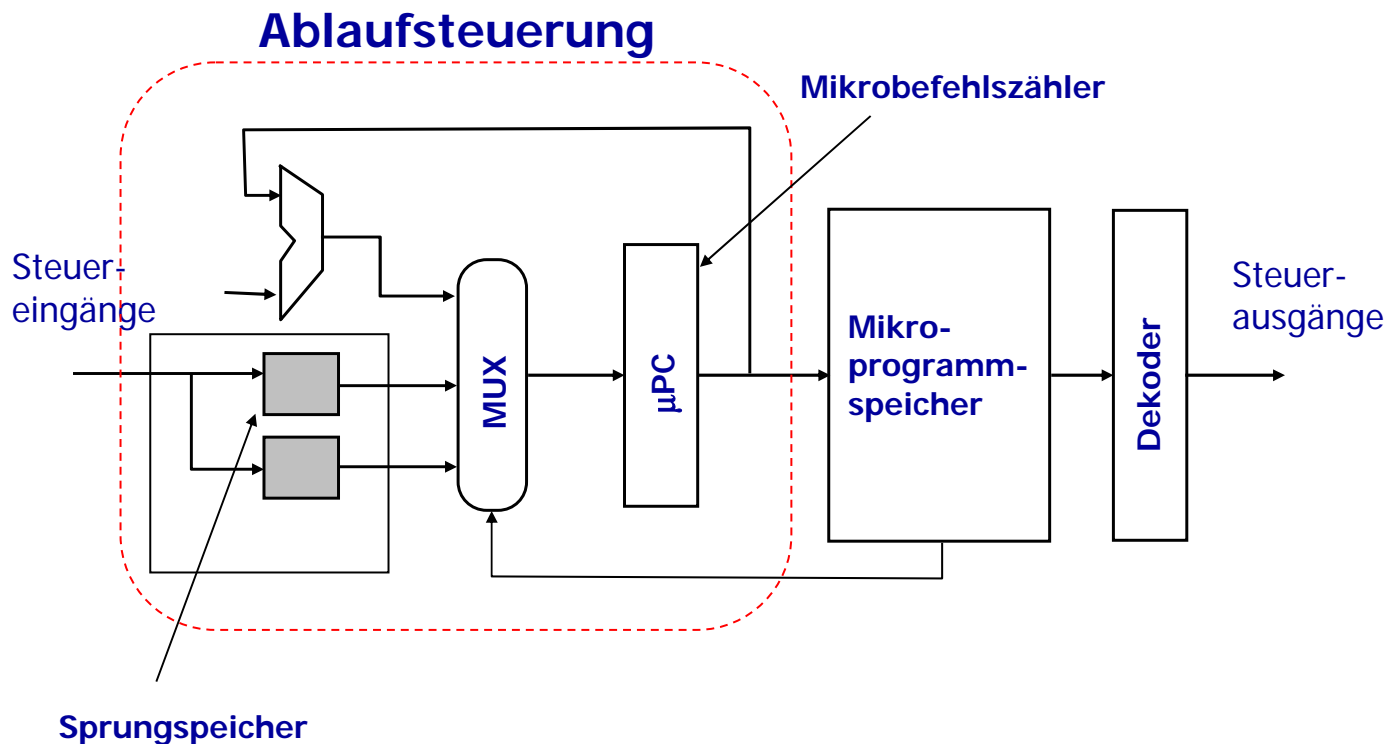
```
0010 0001 0000 1000 1000 0000 0001
0001 0100 0000 0000 1000 0000 0010
0000 0000 0000 0001 1000 0000 0011
0000 1010 0000 0000 0000 0000 0100
0000 0000 1001 0000 0000 0000 0101
```



Prinzip der Mikroprogrammierung



Implementierung des Steuerwerks



Vorteile und Nachteile

➤ Vorteile der Mikroprogrammierung:

- Mehrere Befehlssätze auf einem Rechner → Anpassung des Befehlssatz an der Anwendung
- Mehrere Rechnertypen mit dem gleichen Befehlssatz

➤ Nachteile der Mikroprogrammierung:

- Aufwendig
- Langsam



CISC (complex instruction set computers)

Gründe dafür:

- Ausführung komplizierter Befehle ist immer noch schneller als die Ausführung von Programmen gleicher Funktion
- Mikroprogrammierung begünstigt komplizierte Befehle
- komplizierte Befehle führen zu kurzen Programmen
- Umfang des Befehlssatzes wird oft als Werbeargument verwendet
- Unterstützung höherer Programmiersprachen durch komplizierte Befehle (Direkte Abbildung: *Sprachkonstrukt* → *Befehl*)



CISC (complex instruction set computers)

Gründe dafür:

- Unterstützung von Compilern durch entsprechende Befehle
- Unterstützung spezieller Einsatzgebiete

Fazit:

**Entwicklung von Hardware,
Programmiersprachen und Einsatzgebieten
begünstigt „komplizierte“ Befehle.**



CISC (complex instruction set computers)

Gründe dagegen:

- Schnellere Hauptspeicher und die Verwendung von Cache-Speichern beschleunigen die Programmausführung
- Mikroprogramme wurden immer umfangreicher; Verlängerte Entwurfszeit, Komplexe Steuerwerke (> 50% der Chipfläche)
- Nur relativ kleine Teile des großen Befehlssatzes werden häufig benutzt
- Größere Fehlerhäufigkeit auf der Mikroprogrammebene
- Schwieriger Compilerbau



CISC (complex instruction set computers)

• Systemprogramme in XPL auf IBM/360:

90 % aller ausgeführten Befehle: 10 verschiedene Befehle

95 % aller ausgeführten Befehle: 21 verschiedene Befehle

99 % aller ausgeführten Befehle: 30 verschiedene Befehle

• COBOL-Programme auf IBM/370:

90,28 % aller ausgeführten Befehle: 26 verschiedene Befehle

99,08 % aller ausgeführten Befehle: 48 verschiedene Befehle

(nur 84 verschiedene Befehle wurden überhaupt benutzt)



Limitationen der CISC Architekturen

➤ Befehlsausnutzung (80/20 Regel):

viele mächtige Befehle, komplexes Befehlsformat, Mikroprogrammierung, nur 20 % der Befehle werden überwiegend benutzt

➤ Kritisches Problem: Anzahl der Zyklen pro Instruktion (CPI)

bei allen heutigen CISC Architekturen ist $CPI \gg 2$



Prozentualer Anteil von Anweisungen in Hochsprachenprogrammen

Großteil der in Hochsprachen verwendeten Anweisungen ist sehr einfach:

Anweisung	Mittlerer zeitlicher Anteil
Zuweisung	47 %
if	23%
call	15 %
loop	6 %
goto	3 %
Andere	7 %



RISC (reduced instruction set computers)

Grundprinzipien:

- Viel benutzte einfache Befehle so schnell wie möglich machen (Ausführung möglichst in einer Taktphase. Keine Mikroprogrammierung mehr, Befehls-Pipeline)
- Der größte Teil der Arbeit soll durch optimierende Compiler zur Übersetzungszeit erledigt werden
- Operanden werden nach Möglichkeit in großen Registersätzen gehalten → schneller Zugriff → schnelle Verarbeitung
- Einheitliche Befehlsformate → schnelle Decodierung
- Pipelining anwenden, so gut es geht



RISC (reduced instruction set computers)

Entwurfsziele:

- Ausführung jedes Befehls in einem Taktzyklus
(*Befehl ≈ bisheriger Mikrobefehl bei CISC*)
- Alle Befehle gleich lang:
Decodierschaltung wird einfacher
Programme länger, aber Ausführungszeit kürzer
- Nur Load-Store und Register-Register-Befehle:
weniger Adressierungsarten → schnelle Ausführung
- Koprozessorarchitektur für komplexe Befehle



RISC (reduced instruction set computers)

Keine Entwurfsziele sind z. B.:

- Unterstützung von Gleitkomma-Arithmetik
- Unterstützung von Betriebssystemfunktionen



Zielvorstellungen für RISC Rechner

- Ein-Zyklus-Befehle
- Einheitliches Befehlsformat
- Wenige Maschinenbefehle
- Load/Store-Architektur
- Großer Registersatz
- Verzicht auf Mikroprogrammierung
- 32-Bit-Architektur
- Pipeline-gerechter Maschinenbefehlssatz (gleiche Befehlsausführzeiten)
- Keine Unterstützung für Betriebssystem und Gleitkomma-Arithmetik



Forderungen an RISC-Systeme

- Mindestens 75% aller Befehle sind Ein-Zyklus-Befehle
- Einheitliche Länge aller Befehle entsprechend der Datenbusbreite
- Nicht mehr als 128 Befehle
- Nicht mehr als 4 Befehlsformate
- Nicht mehr als 4 Adressierungsarten
- Load/Store-Architektur
- Festverdrahtete Steuereinheit, keine Mikroprogrammierung
- Mindestens 32 allgemein verwendbare Register



RISC Rechner aus heutiger Sicht

Geblieben ist von der RISC-Idee im wesentlichen:

- das Befehlspipelining
- die Load/Store-Architektur
- ein großer Registersatz: 32 allgemeine und 32 Gleitpunkt-Register
- ein einheitliches Befehlsformat
- die Verwendung weniger Adressierungsarten
- der Verzicht auf Mikroprogrammierung



RISC & CISC

CISC	RISC
Komplexe Befehle, Ausführung in mehreren Taktzyklen	Einfache Befehle, Ausführung in einem Taktzyklen
Jeder Befehl kann auf den Speicher zugreifen	Nur Lade- und Speicherbefehle greifen auf den Speicher zu
Wenig Pipelining	Intensives Pipelining
Befehle werden von einem Mikroprogramm interpretiert	Befehle werden durch festverdrahtete Hardware ausgeführt
Befehlsformat variabler Länge	Alle Befehle mit fester Länge
Die Komplexität liegt im Mikroprogramm	Die Komplexität liegt im Compiler
Einfacher Registersatz	Mehrere Registersätze

