

# Automatisierungstechnik AP1

## – Einführung in die Prozessorarchitektur und Maschinenprogrammierung –

### *Lernziele:*

- Grundlagen des Aufbaus und der Wirkungsweise von Prozessoren und Mikrocontrollern (Einführung in die Rechnerarchitektur),
- Grundlagen der Maschinenprogrammierung,
- Grundlagen der Programmentwicklung (Ausdenken – Programmieren – zum Laufen bringen).

### *Weshalb maschinennahe Programmierung?*

- sie ist in der Praxis nach wie vor erforderlich (Nutzung maschinenspezifischer Besonderheiten, maximale Ausnutzung der Hardware (höchstes Leistungsvermögen oder geringster Aufwand), Umgehung von Unzulänglichkeiten (Workarounds)),
- sie vermittelt grundlegendes Erfahrungswissen zum Verstehen, Beurteilen und Auswählen von Prozessorarchitekturen.

### *Grundlagen der Rechnerarchitektur und maschinennahen Programmierung kennenlernen*

Beide Lehrgebiete sind eng miteinander verbunden. Wer beim Programmieren bis auf die blanke Hardware durchgreifen will, muß deren Wirkprinzipien kennen. Die Rechnerarchitektur ist keine exakte Wissenschaft. Es ist nach wie vor üblich, in der Lehre mit dem grundsätzlichen Aufbau des Universalrechners zu beginnen und dann die einzelnen Funktionseinheiten zu diskutieren. Dabei bezieht man sich üblicherweise auf konkrete Beispiele – Rechnerarchitekturen werden zumeist ähnlich beschrieben wie Tier- oder Pflanzenarten in der Biologie. Wir verpassen also nicht viel, wenn wir uns tiefgründige Theorien schenken und sofort beginnen, uns in eine bestimmte Architektur einzuarbeiten.

AP1 betrifft nicht die Rechnerarchitektur im allgemeinen Sinne. Vielmehr beschränken wir uns

- auf Auslegungen, die sich in der Praxis durchgesetzt haben,
- auf Leistungsklassen, wie sie für Embedded Systems typisch sind (ein Befehl zu einer Zeit, keine Parallelverarbeitung usw.).

### *Allgemeine Merkmale einer Rechnerarchitektur:*

- Programmiermodell,
- Datenstrukturen,
- Befehlswirkungen,
- Befehlsformate,
- Registermodell,
- Speicheradressierung,
- Speicherverwaltung,
- Ein- und Ausgabe,
- Unterbrechungssystem,
- Schutzvorkehrungen,
- Maschinenzeit,
- Kaltstart.

*Es gibt so viele Controller und Prozessoren. Wie können wir uns da zurechtfinden?*

Alles halb so schlimm – elementare Datenstrukturen und Befehlswirkungen sind im Grunde gleich. Die Unterschiede betreffen vor allem:

- die Verarbeitungsbreite,
- das Registermodell,
- die Zubringerfunktionen,
- die Speicher- und E-A-Ausstattung.

*Typische Registermodelle:*

- Akkumulatormaschine,
- Universalregistermaschine,
- Stackmaschine.

*Typische Auslegungen der Maschinenbefehle*

Üblich ist eine pauschale Einteilung in “einfache“ und “komplexe“ Befehle, die durch zwei Marketingbegriffe ausgedrückt wird:

- RISC = Reduced Instruction Set Computer. Vergleichsweise einfache Befehlswirkungen. Wenige Befehlsformate. Wichtig: Transport- und Verarbeitungsfunktionen sind voneinander getrennt (Load/Store-Prinzip). RISC-Maschinen sind typischerweise Universalregistermaschinen.
- CISC = Complex Instruction Set Computer. Vergleichsweise komplexe Befehlswirkungen. Wichtig: Transport- und Verarbeitungsfunktionen kommen in manchen Befehlen zusammen vor. Viele Befehlsformate, umfangreicher Befehlsvorrat.

*Unsere Beispiele:*

1. Atmel AVR. Eine RISC-Architektur mit 32 Universalregistern und 8 Bits Verarbeitungsbreite.
2. Zilog Z80 und Intel 8086. Typische CISC-Architekturen.

Um zunächst die typischen Befehlswirkungen kennenzulernen, beginnen wir mit dem Atmel (Überschaubarkeit).

*Der Inhalt der Lehrveranstaltung:*

1. Einführung in die Maschinenprogrammierung anhand Atmel AVR.
2. Einführung in die Programmierung typischer CISC-Maschinen.
3. Einführung in fortgeschrittene Techniken der Programmierung und Programmentwicklung.
4. Rechnerarchitektur im Überblick (Synopsis).

*Praktikum:*

1. Versuch: Bekanntmachen mit einem neuen Mikrocontroller. Der typische Entwicklungsgang. Elementare Abläufe der Ein- und Ausgabe.
2. Versuch: Elementare Programmierübungen.
3. Versuch: Praktische Anwendungsprogrammierung.

## Themenübersicht Programmierung/Programmentwicklung

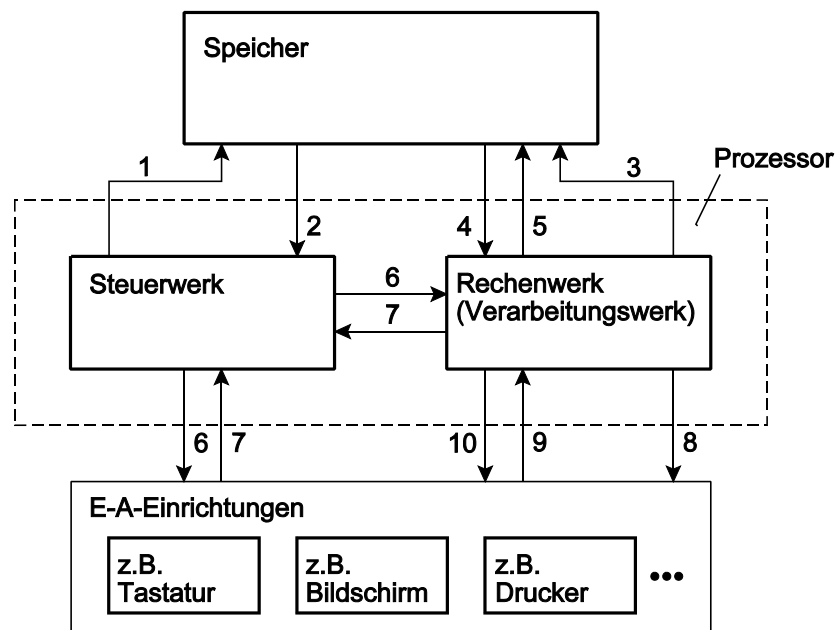
- Was leistet ein Assembler?
- Datenstrukturen
- Informationstransporte
- Ein- und Ausgabe
- arithmetische Verknüpfungen
- kombinatorische Verknüpfungen
- Verschieben und Rotieren
- Einzelbit- und Bitfeldoperationen
- Boolesche Funktionen
- Testen und Vergleichen
- Speicheradressierung und Adreßrechnung
- Verzweigen
- Schleifen
- Unterprogrammrufruf
- Kontrollstrukturen
- mit Sinn und Verstand programmieren
- Entwicklungsgänge
- Programme zum Laufen bringen (Debugging)

### Datenstrukturen:

- Bits, Bytes, Worte
- Binärvektoren
- fest formatierte Zahlendarstellungen
- Bitketten
- Zeichenketten
- Tabellen
- homogene Strukturen (Arrays)
- heterogene Strukturen (Records)
- Stacks
- Konstanten und Variable
- deskriptive Angaben

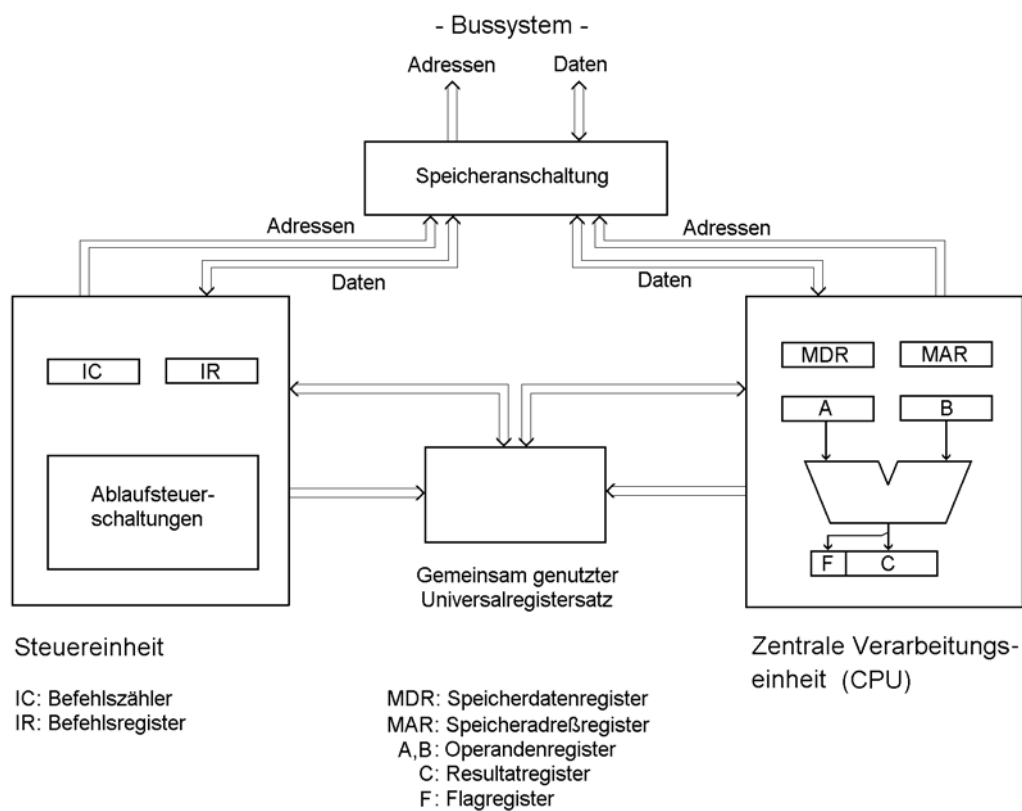
## Der Einzelprozessor

Im klassischen Sinne besteht ein Einzelprozessor (Abb. 1 bis 4) aus der Steuereinheit (Steuerwerk, Control Unit), der zentralen Verarbeitungseinheit (Operationswerk, Central Processing Unit CPU) und der Speicheranschaltung (Memory Port, Storage Adapter, Storage Control Unit SCU). Die Steuereinheit enthält den Befehlszähler, das Befehlsregister sowie alle Schaltmittel zur Ablaufsteuerung. Die CPU enthält die Verknüpfungsschaltungen, die notwendig sind, um die in den Operationsbefehlen angegebenen Operationen auszuführen, die Adressierungsschaltungen für Datenzugriffe sowie die notwendigen Register. Beide Funktionseinheiten liefern Adressen, um Lese- und Schreibzugriffe auszuführen. Die Steuereinheit holt Befehle und Befehlsadressen aus dem Speicher (z. B. bei Rückkehr aus einem Unterprogramm oder beim Einleiten einer Unterbrechungsbehandlung) und schreibt Befehlsadressen zurück (Adreßrettung). Die CPU holt Operanden bzw. Operandenadressen und schreibt Ergebnisse zurück. Der Universalregistersatz wird von Steuereinheit und CPU gemeinsam genutzt. Die Speicheranschaltung verbindet Steuereinheit und CPU mit dem Arbeitsspeicher. Moderne Prozessoren sind um zusätzliche Funktionsblöcke erweitert, beispielsweise um eine Gleitkomma-Verarbeitungseinheit (Floating Point Processing Unit FPU), eine Befehlsvorbereitungseinheit, eine Segmentierungseinheit, eine Seitenverwaltungseinheit, Caches usw.

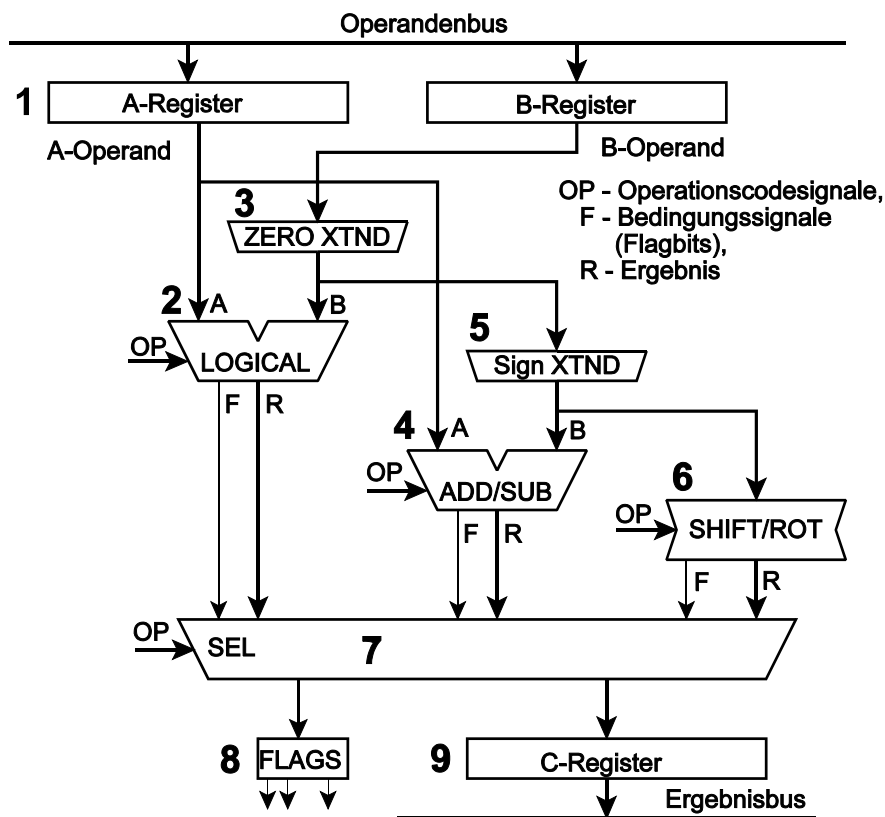


1 - Befehlsadresse; 2 - Lesen der Maschinenbefehle; 3 - Datenadresse; 4 - Lesen von Daten; 5 - Schreiben von Daten; 6 - Steuersignale; 7 - Bedingungs- und Zustandssignale; 8 - E-A-Adressierung; 9 - Eingabe von Daten; 10 - Ausgabe von Daten.

**Abb. 1** Der Einzelprozessor im Blockschaltbild



**Abb. 2** Dieses Blockschaltbild zeigt weitere Einzelheiten



1 - Operandenregister; 2 - bitweise Verknüpfungen (AND, OR, XOR, CMP usw.); 3 - Nullerweiterung kürzerer Operanden; 4 - binäre Addition/Subtraktion (Zweierkomplement-Arithmetik; Näheres s. Abb. 4); 5 - Vorzeichenerweiterung kürzerer Operanden; 6 - Verschiebe- und Rotationsnetzwerk; 7- Ergebnisauswahl gemäß Operationscode; 8 Bedingungssignale; 9 - Ergebnisregister

**Abb. 3** Eine typische Arithmetik-Logik-Einheit. Alle gängigen Prozessoren beruhen auf den hier dargestellten elementaren Operationen (bitweise Verknüpfungen, Addition/Subtraktion von Binärzahlen in Zweierkomplementarithmetik, Verschieben/Rotieren)

#### v. Neumann-Architektur und Harvard-Architektur

Die Begriffe bezeichnen allgemeine Architekturkonzepte, die sich darin unterscheiden, wieviele Speicheradreßräume bzw. Speicherzugriffswege grundsätzlich vorgesehen sind (Abb. 4).

##### *Gemeinsamkeiten:*

Beide Architekturen haben einige wesentliche Prinzipien gemeinsam:

- es gibt einen einzigen Befehlsstrom, wobei Befehl für Befehl nacheinander ausgeführt wird,
- die Befehlsadressierung erfolgt vorzugsweise durch Weiterzählen der Befehlsadresse (Ausnahmen davon sind Verzweigungen, Unterprogrammrufe, Unterbrechungen usw.).

Architekturen, die diese Merkmale nicht aufweisen, sind – beim aktuellen Stand der Technik – eher in den Bereich der akademischen Forschung einzuordnen (man spricht hier von rekursiven Architekturen, funktionalen Architekturen, Datenflußarchitekturen usw.).

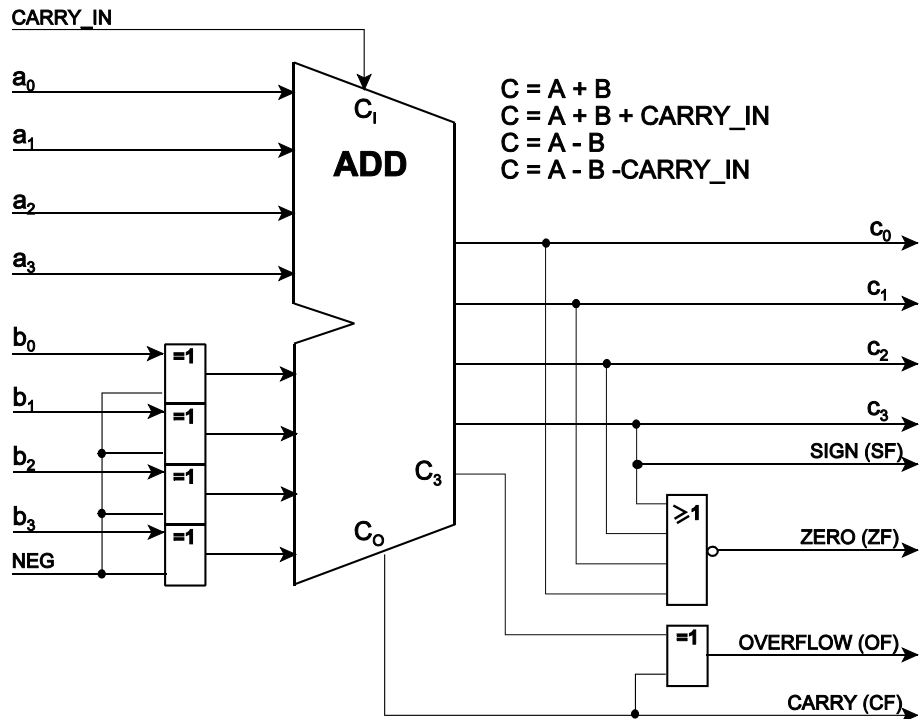
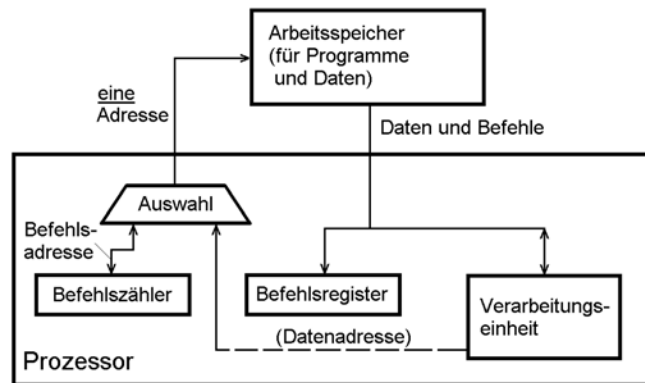
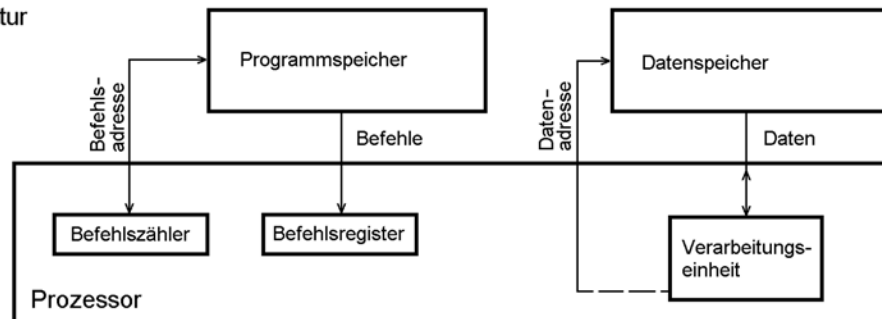


Abb. 4 Die

a) v. Neumann-Architektur



b) Harvard-Architektur



Gr

dlage des eigentliche Rechnens in der Maschine – das Zweierkomplemet-Addierwerk

un

Abb. 5 v. Neumann.- und Harvard-Architektur

*Zu den Namen:*

- v. Neumann-Architektur: nach dem Mathematiker John v. Neumann,
- Harvard-Architektur: nach der Harvard-Universität (Cambridge, Massachusetts)<sup>1</sup>.

### **v. Neumann-Architektur**

Es gibt nur einen einzigen Speicheradreibraum und einen einzigen Zugriffsweg. Mit anderen Worten: es gibt aus der Sicht des Programmierers nur einen einzigen, von Adresse 0 an fortlaufend adressierbaren Speicher, der alle Programme, Daten, deskriptiven Angaben usw. aufnimmt, die für die laufende Arbeit benötigt werden.

### **Harvard-Architektur**

Es gibt zwei Speicheradreibräume – einen für Daten und einen für Befehle. Gemäß der technischen Auslegung unterscheiden wir:

#### *1. echte Harvard-Maschinen*

Es gibt eine gesonderte Speicheranordnung je Adreibraum (also einen Programmspeicher und einen Datenspeicher) und unabhängige Zugriffswege zu beiden Speicheranordnungen (Abb. 4b). Architekturbeispiel: Atmel AVR.

#### *2. Maschinen mit Harvard-Architektur*

Diese haben zwar die beiden getrennten Adreibräume, aber nur einen einzigen gemeinsamen Speicherzugriffsweg (Speicherbus). Architekturbeispiel: 8051.

### **Der große Vorteil der v. Neumann-Architektur: der einheitliche lineare Adreibraum**

Wir können mit der Speicherkapazität anstellen, was wir wollen. Vor allem können wir die gesamte installierte Speicherkapazität voll ausnutzen– ob wir vorwiegend Programme oder vorwiegend Daten speichern, ist gleichgültig. Auch lassen sich Programme ohne weiteres als Daten behandeln, also mit den üblichen Maschinenbefehlen transportieren, ändern usw.

### **Die Vorteile der Harvard-Architektur**

#### *1. Höhere Leistung*

Eine v. Neumann-Maschine holt Befehle und Daten nacheinander aus dem selben Speicher. Dies beeinträchtigt naturgemäß das Leistungsvermögen<sup>2</sup>). Eine echte Harvard-Maschine kann hingegen gleichzeitig (parallel) auf Daten und Befehle zugreifen.

#### *2. Aufwandsoptimierung*

Beide Speicher einer echten Harvard-Maschine kann man unabhängig voneinander hinsichtlich der Zugriffsbreite, der Technologie<sup>3</sup>) usw. optimieren. Ist beispielsweise das Byte die elementare Datenstruktur, so müssen bei einer v. Neumann-Maschine auch die Befehlsformate in Bytestrukturen gezwängt werden (auch Befehle sind Aneinanderreihungen von Bytes). Eine echte Harvard-Maschine kann man hingegen so auslegen, daß der Befehlsspeicher eine jeweils genau passende Zugriffsbreite hat – auch wenn es ein “krummer” Wert ist. U. a. sind Prozessoren mit Befehlen von 12, 14, 24 Bits usw. gebaut worden. Beispiel: PIC 16x und 24x.

---

1): die ersten funktionsfähigen Maschinen gemäß dieser Architektur wurden übrigens von Konrad Zuse gebaut (in Berlin) ...

2): man spricht bildhaft vom “v. Neumann-Flaschenhals” (v. Neumann Bottleneck).

3): z. B. liegt es nahe, den Programmspeicher eines Mikrocontrollers als ROM auszulegen und den Datenspeicher als RAM.

### 3. Höheres Adressierungsvermögen

Da es zwei Adreßräume gibt, wird das Adressierungsvermögen der Hardware praktisch verdoppelt. Beispiel: ein Prozessor sei für 16-Bit-Adressen ausgelegt (Byteadressierung). In einer v. Neumann-Architektur beträgt das Adressierungsvermögen insgesamt  $2^{16} = 64$  kBytes. Eine Harvard-Maschine könnte hingegen mit einem Befehls- und einem Datenspeicher von jeweils 64 kBytes bestückt werden (insgesamt 128 kBytes)<sup>1)</sup>. Die schlechte Nachricht: wenn wir z. B. 20 kBytes für die Programme, aber 100 kBytes für die Daten brauchen, so paßt es nicht (Abhilfe: tricksen ...).

#### Welche dieser Architekturen bestimmt den Stand der Technik?

Alle modernen Hochleistungsprozessoren<sup>2)</sup> sind ausnahmslos v. Neumann-Maschinen. Das heißt, sie verhalten sich aus Sicht des Programmierers als solche. Intern gibt man sich aber Mühe, die Vorteile beider Architekturen zu vereinigen. Demgegenüber sind die meisten Signalprozessoren und auch viele der kleinen Mikrocontroller als Harvard-Maschinen ausgelegt (Tabelle 1).

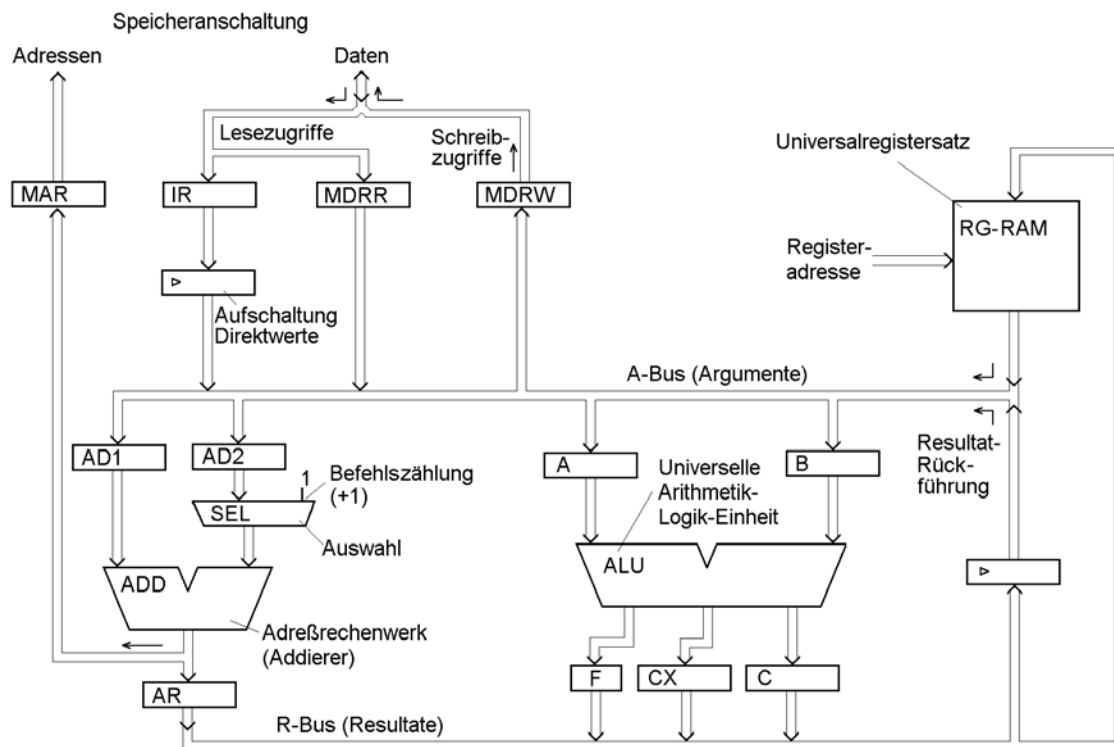
Vorteil	Erläuterung	Beispiele
höheres Adressierungsvermögen	weil es 2 unabhängige Adreßräume gibt	Intel 8051
technologische Trennung zwischen Daten- und Programmspeicher	Datenspeicher: SRAM, Programmspeicher: ROM (auch EPROM oder Flash)	Microchip PIC, Atmel AVR
interne Optimierung	Befehle werden so breit ausgelegt wie es jeweils zweckmäßig ist (also nicht in Bytestrukturen gepreßt)	Microchip PIC16x, 24x
Leistungssteigerung	durch parallelen Zugriff auf Befehle und Daten	die meisten Signalprozessoren

**Tabelle 1** Weshalb Mikrocontroller und Signalprozessoren oft als Harvard-Maschinen ausgelegt werden

---

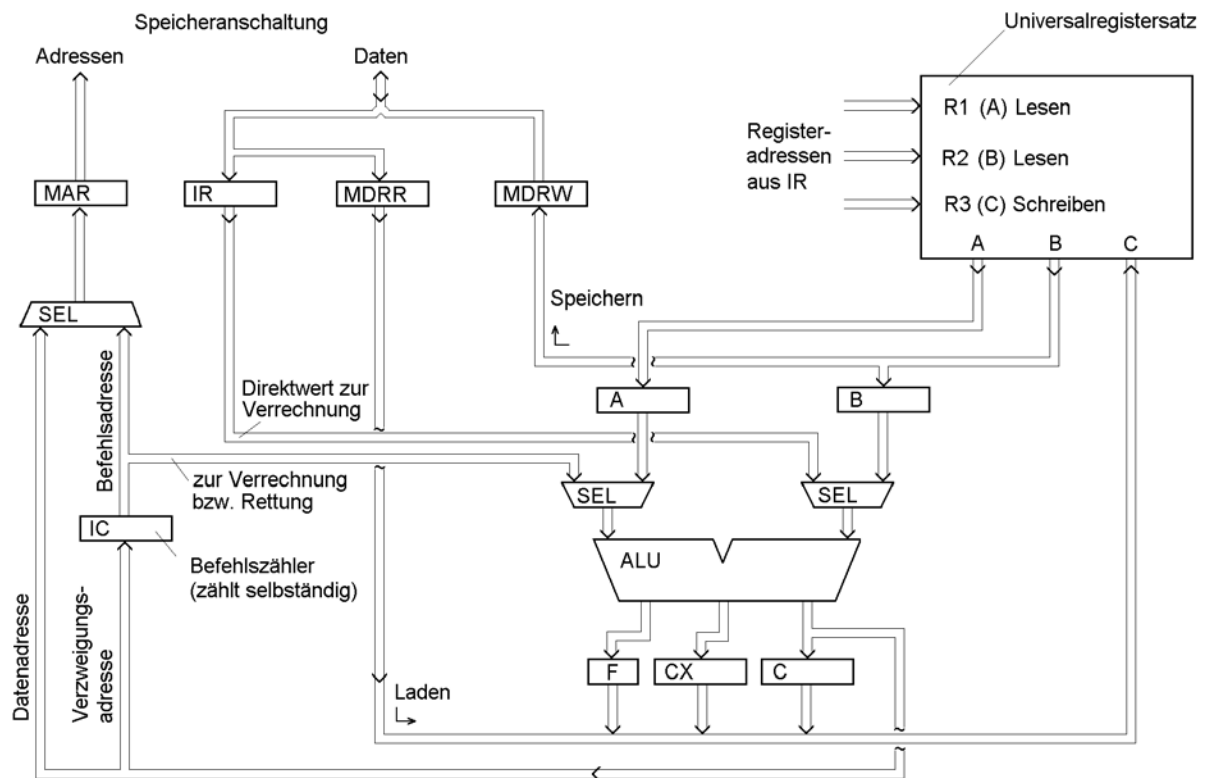
1): deshalb hat man viele Mikrocontroller, obwohl sie nur einen einzigen Speicherbus haben, als Harvard-Maschinen ausgelegt. Beispiel: 8051.

2): und auch die meisten Prozessoren und Mikrocontroller der mittleren Leistungsklassen.



- MAR: Speicheradreßregister
- IR: Befehlsregister
- MDDR: Speicherdatenregister für Lesen
- MDRW: Speicherdatenregister für Schreiben
- AD1, 2: Argumentregister für Adreßrechnung
- AR: Resultatregister der Adreßrechnung
- A,B: Argumentregister für Operationen
- C: Resultatregister
- CX: Erweitertes Resultat (z. B. bei Verschiebungen)
- F: Flagregister

**Abb. 6** Ein typischer CISC-Prozessor im Blockschaltbild



Operationsbefehle

OP	R3	R1	R2	OP
----	----	----	----	----

1) : Basisadresse

Transportbefehle (Laden, Speichern)

OP	R3/R2 2)	R1 1)	DISPLACEMENT
----	----------	-------	--------------

2) : R3: Ziel; R2: Quelle

Verzweigen

OP	COND 4)	R1 1)	DISPLACEMENT
----	---------	-------	--------------

3) : Rettung

4) : Verzweigungsbedingungen

Unterprogrammaufruf

OP	R3 3)	R1 1)	DISPLACEMENT
----	-------	-------	--------------

**Abb. 7** Ein typischer RISC-Prozessor im Blockschnittbild