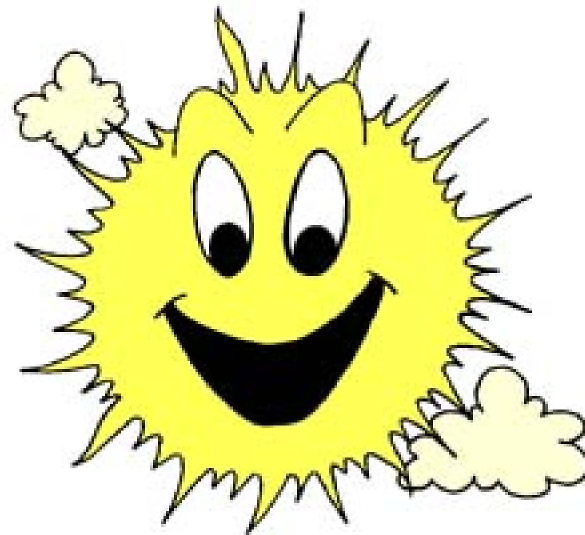
Wiederholung der 8. Vorlesung

Informatik



6.1.6 CPU

FH D Informatik

In realen Rechnern werden die Arithmetisch/Logische Einheit und die Kontrolleinheit häufig in einer physikalischen Einheit (Chip) zusammengefasst.

Diese Kombination wird als **Zentrale Verarbeitungseinheit** (central processing unit, **CPU**) bezeichnet.

Auch ein Teil des Speichers rutscht in die CPU, es werden Operanden für die ALU und allgemeine Register in der CPU gespeichert und nicht jedes Mal aus dem separaten Speicher geholt.

- Stack-Architektur
- Akkumulator-Architektur
- Allgemeinzweck-Register-Architektur (GPR)

6.1.6 von Neumann Instruktionstypen

Informatik

Die wichtigsten Instruktionstypen eines „von Neumann"-Rechners sind:

- **Transferbefehle** (load, store, LDA, STA)
- **Arithmetische Befehle** (addieren, subtrahieren), Logische Befehle (AND, OR,XOR), Shift-Befehle.
- **Sprungbefehle** (bedingt, unbedingt, JMP, JZ, JNZ)

6.1.6 Befehlssatz-Architektur

FH D Informatik

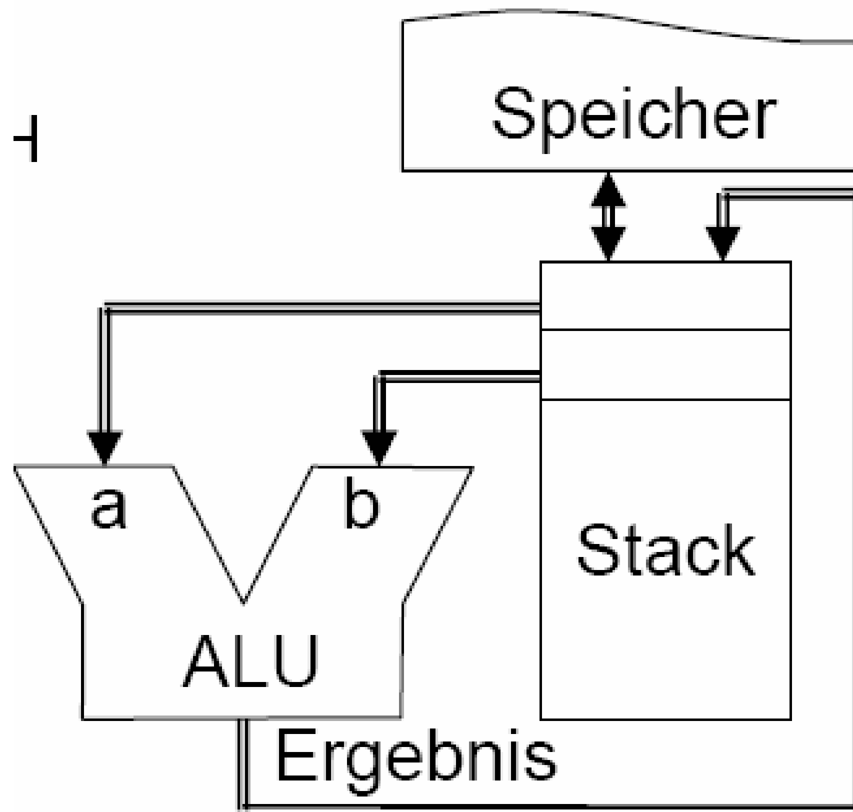
Um ein Maschinenprogramm für einen bestimmten Rechner zu erstellen, muss man natürlich nicht alle Details der Organisation des Rechnerkerns beherrschen.

Es genügt die Kenntnis der ***Instruktionssatz-Architektur*** (ISA).

Die ISA beinhaltet:

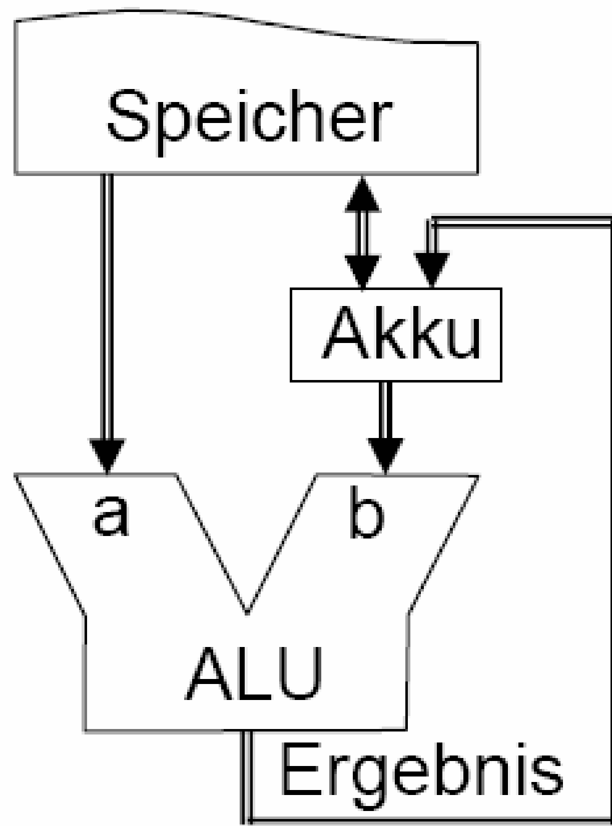
- den Maschinenbefehlssatz (Art und Format der Befehle)
- die adressierbaren Register des Rechnerkerns
- die Darstellungsmöglichkeiten für Daten (Maschinen-Datentypen: Byte, Integer-Zahlen, Fixpunktzahlen etc.)
- die Adressierungsmöglichkeiten des Speichers (Adressierungsmodi)
- die Art der Daten-Ein- und Ausgabe.

6.1.6 Stack-Architektur



Stacks werden unabhängig von der jeweiligen Architektur bei Unterprogrammaufrufen und Parameterübergaben verwendet. Ein Call Befehl pusht implizit die Rücksprungadresse und ggf. Registerstati, ein Return popt diese Werte wieder.

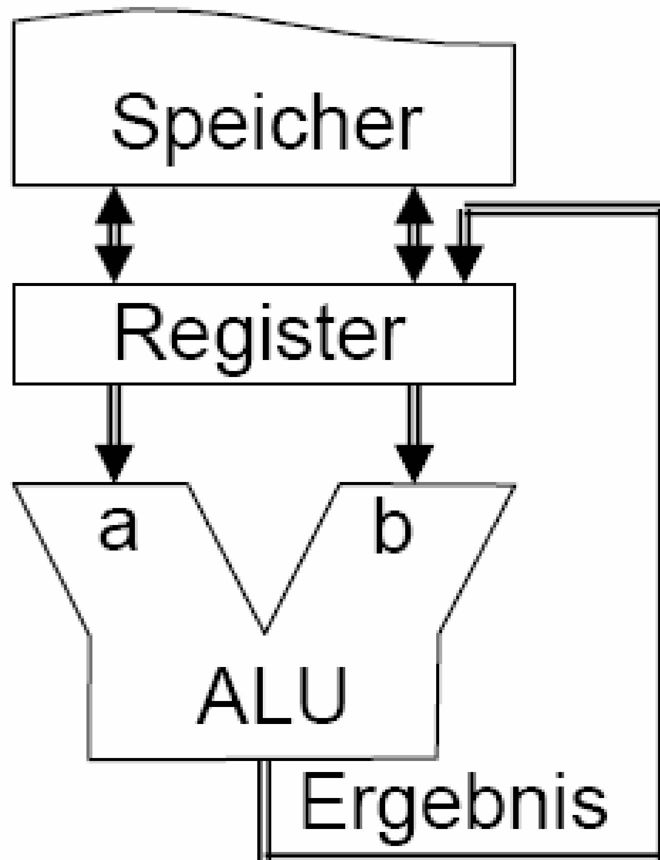
6.1.6 Akkumulator-Architektur



- Ausgezeichnetes Register: Akku
- LOAD und STORE wirken nur auf Akku. Er ist als expliziter Operand an jeder Operation beteiligt. Jede Operation braucht nur eine Adresse
- Sehr kompaktes Befehlsformat

LDA Op_A ;	Op_A von Speicher -> Akku
ADD Op_B ;	Akku = Akku + Op_B
STA Op_C ;	Op_C (= Op_A + Op_B) Akku -> Sp

6.1.6 Register-Register-Architektur

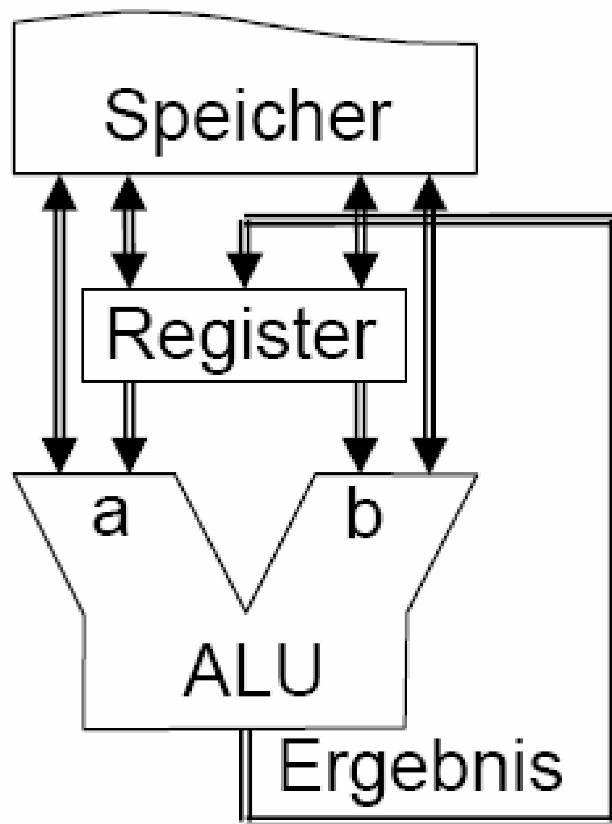


RISC (Load-Store-Architektur)

- alle Operationen greifen nur auf Register zu,
- nur LOAD und STORE greifen auf Speicher zu
- 32 – 512 Register verfügbar
- einfaches Befehlsformat fester Länge
- alle Instruktionen brauchen in etwa gleich lange

LOAD R1, Op_A ; lade Op_A aus Speicher in R1
LOAD R2, Op_B ; lade Op_B aus Speicher in R2
ADD R3, R1, R2 ; addiere R1 und R2, Ergbn. -> R3
STORE Op_C, R3 ;speicher R3 -> Op_C (=Op_A + Op_B)

6.1.6 Register-Speicher-Architektur



CISC (Mischung von Akkumulator- und Load-Store-Architektur)

- Operationen greifen auf Register und/oder Speicher zu
- Befehlsformat variabler Länge
- mächtige Befehle
- stark unterschiedliche Zeiten für Instruktionsausführung

MOV AX, Op_A ; Op_A von Speicher -> Register AX
ADD AX, Op_B ; AX = AX + Op_B
MOV Op_C, AX ; Op_C = AX (= Op_A + Op_B)

6.1.7 Bussysteme

Informatik

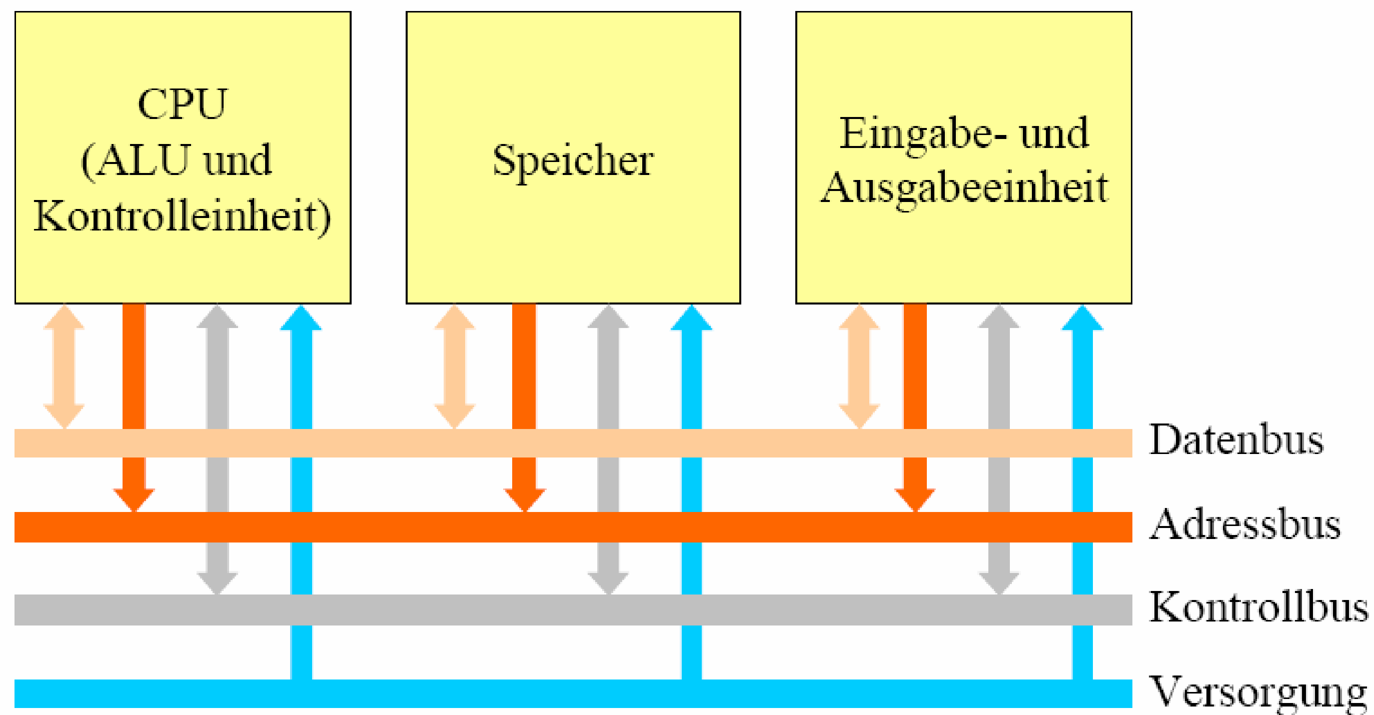
- Systembus
- Adress- und Datenbus
- Kontrollbus
- Bus zur elektrischen Versorgung
- Buszyklen
  - Synchroner Schreibzyklus
  - Asynchroner Lesezyklus

6.1.7 Bussysteme

Informatik

Ein **Systembus** ist aus einem **Datenbus**, **Adressbus** und **Kontrollbus** sowie einem Bus zur elektrischen **Versorgung** der Komponenten aufgebaut.

Einige Architekturen beinhalten zusätzlich noch einen I/O-Bus.



6.1.7 Bussysteme

Informatik

Physikalisch ist jeder Bus aus einer Anzahl von Leitungen aufgebaut.

Der Systembus setzt sich aus mehreren Bussen zusammen, wobei jeder Bus eine individuelle Funktion besitzt.

Zum **Transfer** von Daten zwischen Einheiten werden die Signale der einzelnen Busse in einer spezifizierten Reihenfolge auf den Bussen angelegt.

Für jede Busleitung darf es zu einem Zeitpunkt nur eine Einheit geben, welche die Busleitung treibt und damit einen gültigen Logikpegel auf dem Bus anlegt.

Ansonsten entstehen Kurzschlüsse, die zu Konflikten bei Buszyklen oder schlimmer noch zur Zerstörung von Bustreibern führen können.

6.1.7.1 Adress- und Datenbus

Informatik

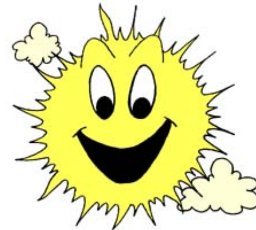
Daten werden von „Quelle“ zur „Senke“ übertragen.

Adress- und **Datenbus** sind homogene Busse, die Signale gleicher Funktion zusammenfassen.

Der **Datenbus** ist ein bidirektionaler Bus:

- Beim Schreibens werden Daten von der CPU (Quelle) zu Speicher oder Ausgabeeinheit (Senke) übertragen.
- Im Falle des Lesens werden Daten in der umgekehrten Richtung von Speicher oder Eingabeeinheit (Quelle) zur CPU (Senke) übertragen.
- Die Anzahl der Datenbusleitungen ergibt sich aus der **Datenbusbreite** eines Rechners, sie stimmt normalerweise mit der Speicherwortbreite überein.

Ende der Wiederholung



6.1.7.1 Adressbus

Informatik

- Der **Adressbus** ist normalerweise ein unidirektionaler Bus. Er transferiert Adressen von der CPU (Quelle) zu Speicher und Ein-/Ausgabeeinheiten (Senke).
- Er spezifiziert die Speicheradresse, unter der ein Speicherwort angesprochen wird. In den allermeisten Rechnern wird durch die Adresse ein Byte im Speicher adressiert.
- Bei Datenbusbreiten von 16 Bit (2 Bytes) oder größer, werden beim Wortzugriff auf den Speicher die unteren Adressbits nicht benötigt.
- **Gemultiplexter Adress- und Datenbus** zur Einsparung von Signalleitungen und Anschluss-Pins am Chip.
- Adresse und Daten werden zeitlich versetzt auf den gleichen Leitungen angelegt.
- Die Zeitpunkte, wann Adressen und wann Daten gültig sind, wird durch Signale des Kontrollbusses festgelegt.

6.1.7.1 Kontrollbus

Informatik

Der **Kontrollbus** ist ein inhomogener Bus, er fasst Signale unterschiedlicher Funktion zusammen.

Rechner besitzen zum Transfer zwischen der CPU und Speicher oder Ein-/Ausgabe einen synchronen oder asynchronen Systembus:

- **synchroner Systembus**: das zeitliche Verhalten der Signale auf dem Bus wird allein durch die CPU gesteuert.
- **asynchroner Systembus**: dort können langsame Speicher oder Ein-/Ausgabeeinheiten das zeitliche Verhalten der Bussignale bei Bedarf verlangsamen.

Hauptaufgaben der Signale:

- Markieren einer gültigen Adresse (Beginn Transfer).
- Auswahl eines Schreib- oder Lesetransfers.
- Abschluss des Transfers.
- Quittung für Datentransfer (nur asynchr. Systembusse)

6.1.7.1 Bus-Zyklen

Informatik

Die zeitliche Abfolge von Signalen auf den Bussen zum Transfer eines Datenwortes zwischen den Einheiten des Rechners wird als **Buszyklus** bezeichnet.

Für jeden Buszyklus gibt es immer genau einen **Bus-Master**, der die logische und zeitliche Abfolge der Signale beim Transfer steuert. Üblicherweise ist die CPU der Bus-Master.

Lesezyklus:

- CPU (Master): → Kontr.Signale + Adresse
- Speicher: Daten von Adresse → Datenbus
- CPU: deakt. Kontr.Sign.

Speicher (Quelle) → CPU (Senke)

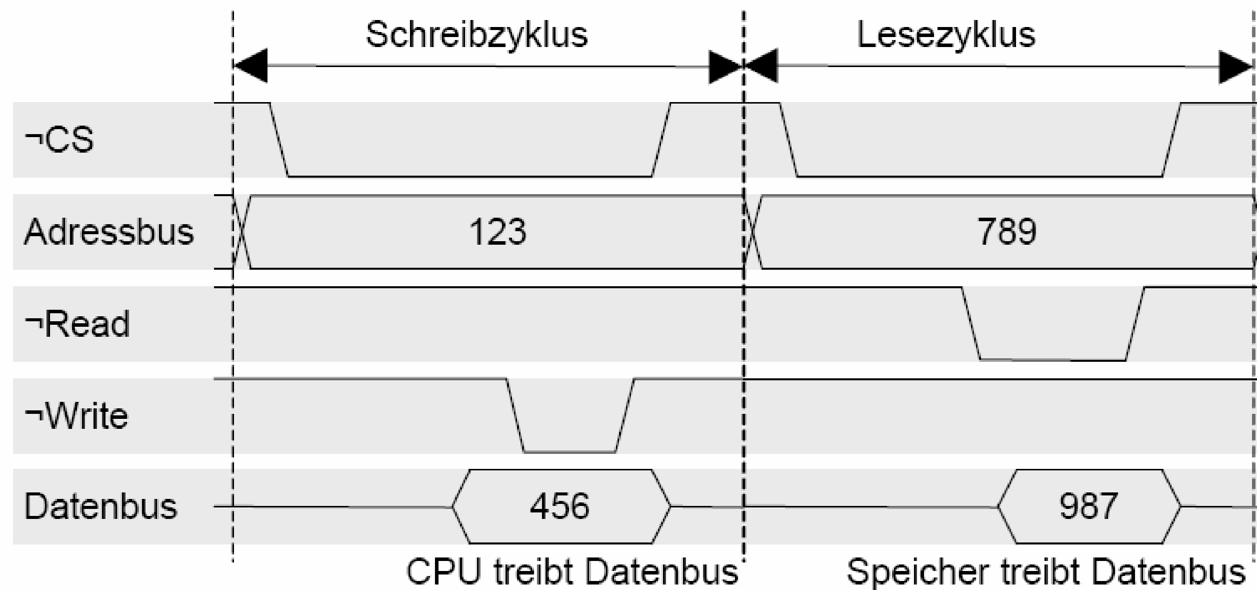
Schreibzyklus:

- CPU (Master): → Kontr.Signale + Adresse
- CPU: Daten → Datenbus
- Speicher: Daten ← DB
- CPU: deakt. Kontr.Sign.

CPU (Quelle) → Speicher (Senke)

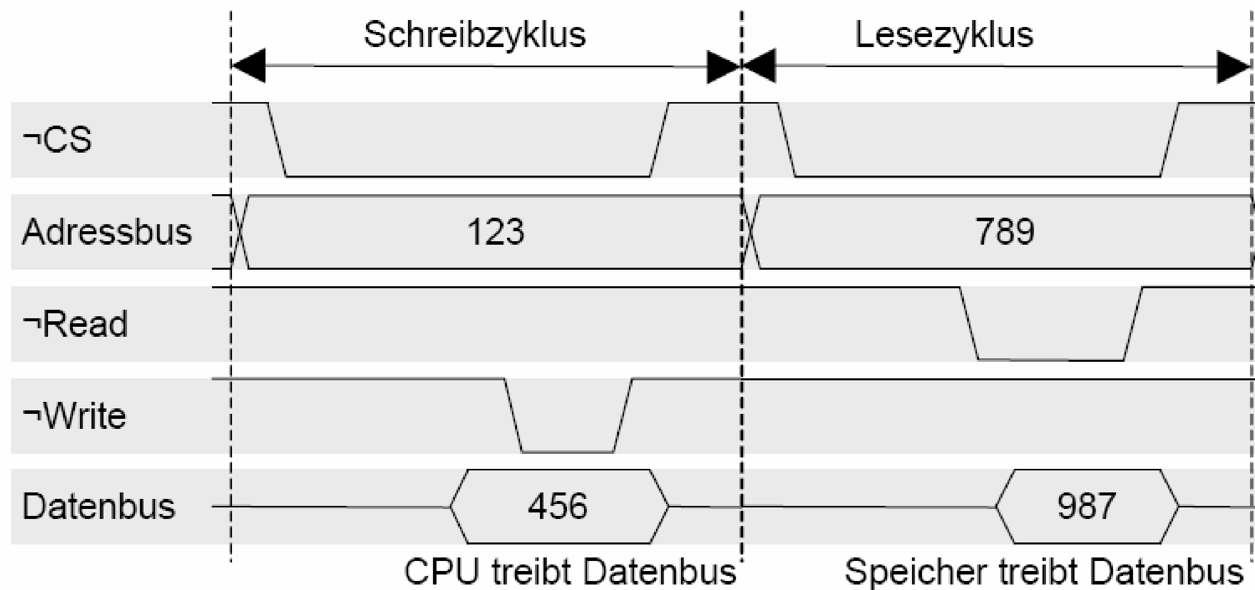
6.1.7.1 Buszyklen: Synchroner Schreib-Zyklus

FH D Informatik



1. CPU legt Adresse an
2. CPU aktiviert ¬CS, Speicher wertet Adresse aus, überprüft Zuständigkeit.
3. CPU legt Daten an
4. CPU aktiviert ¬Write,
5. Speicher übernimmt Daten in Adresse
6. CPU deaktiviert ¬Write, Speicher muss Daten übernommen haben.
7. CPU deaktiviert Datenbus, ¬CS

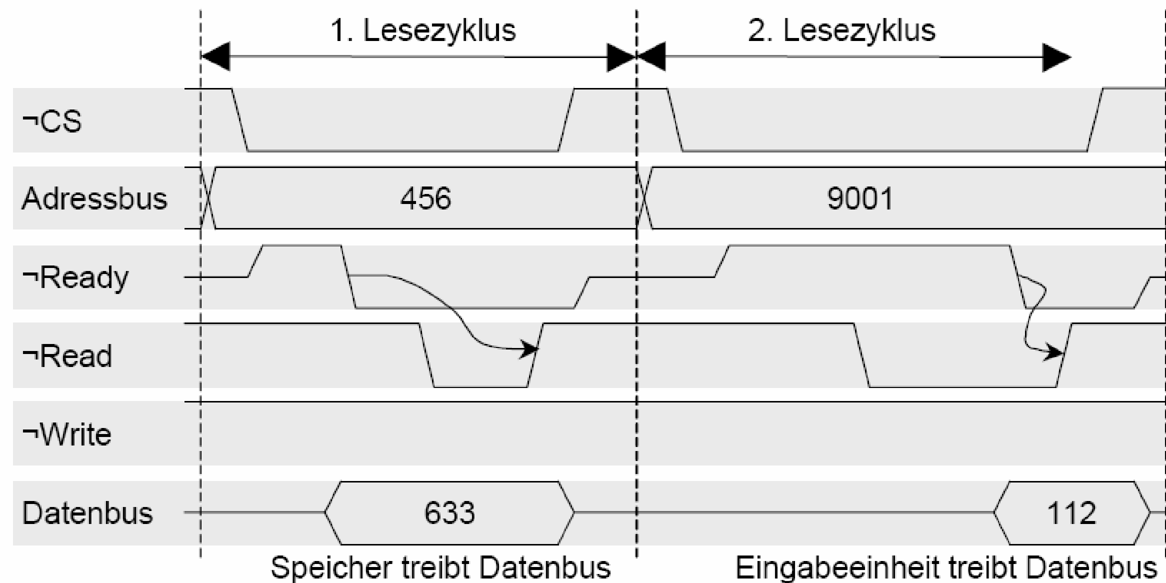
6.1.7.1 Buszyklen: Synchroner Lese-Zyklus



1. CPU legt Adresse an
2. CPU aktiviert ¬CS, Speicher wertet Adresse aus, überprüft Zuständigkeit.
3. CPU aktiviert ¬Read
4. Speicher legt Wort an
5. CPU übernimmt Daten-Wort vom Bus
6. CPU deaktiviert ¬Read, Speicher muss Daten vom Bus nehmen.
7. CPU deaktiviert Datenbus, ¬CS

6.1.7.1 Buszyklen: Asynchroner Lesezyklus

Informatik



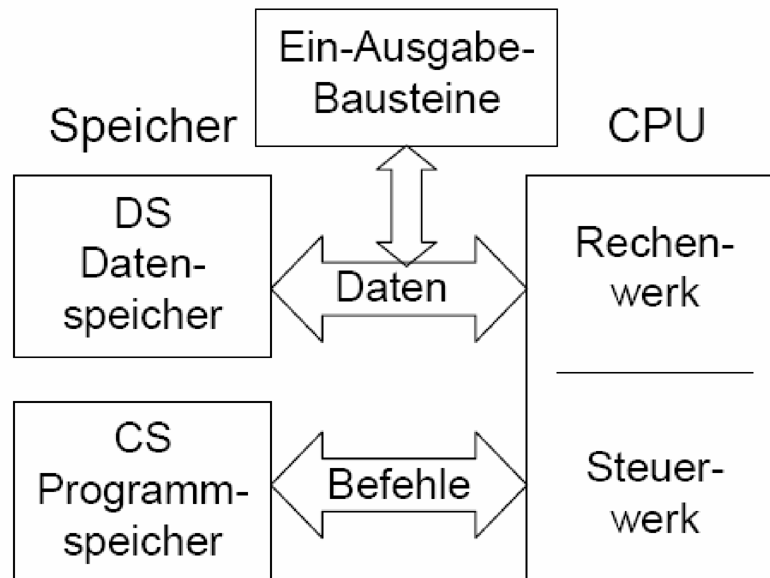
1. CPU legt Adresse an
2. CPU aktiviert ¬CS, Speicher wertet Adresse aus, überprüft Zuständigkeit.
3. Speicher ¬Ready inaktiv
4. Speicher legt Wort auf Datenbus (Bus ‚x' → aktiv)
5. Speicher ¬Ready aktiv
6. CPU aktiviert ¬Read
7. CPU überprüft ¬Ready, liest und deaktiviert ¬Read. Speicher muss Daten vom Bus nehmen
8. Speicher deakt Datenbus, CPU deaktiviert ¬CS
9. Speicher deaktiviert ¬Ready

6.2 Harvard-Architektur

Informatik

Harvard - Architektur

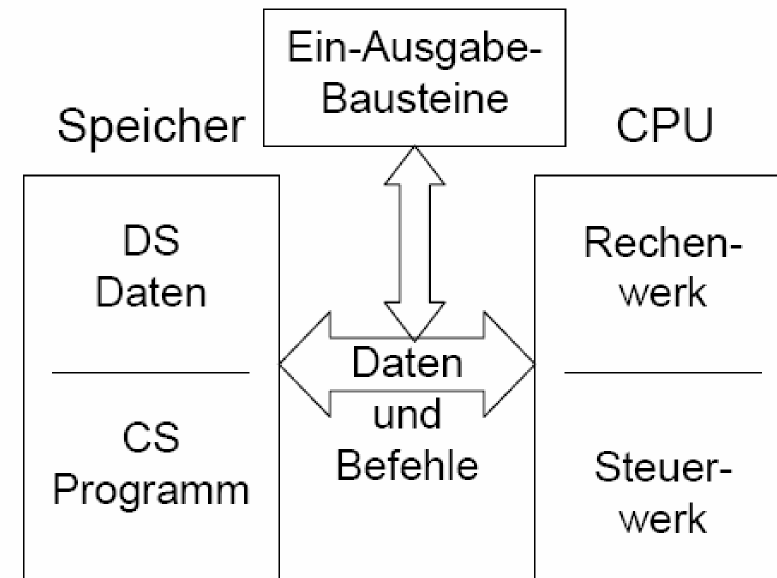
Befehle und Daten in getrennten Speichern



- Je ein Befehls- und Datenbus
- Schneller gleichzeitiger Zugriff auf Code und Daten
- Hauptanwendung: Signalprozessoren

Von Neumann – Architektur (Princeton Architektur)

Befehle und Daten im gemeinsamen Speicher



- Nur ein Bus für Befehle und Daten
- Flexible Aufteilung zwischen Code und Daten
- Hauptanwendung: Allgemeine Computer

➢ Gemeinsames Kennzeichen: Sequentielle Abarbeitung des Programms / der Befehle

➢ Heute sind häufig Mischformen anzutreffen

6.2 Digitale Signalprozessoren (DSP)

Informatik

Die **Harvard-Architektur** findest sich z.B. in **DSP**'s:

DSP weisen zwar immer noch Kontrollfluss-Befehle auf, haben jedoch einige Spezialbefehle (z.B. Filterung), die nach dem Datenfluss-Prinzip arbeiten.

Kontrollfluss:

- Beschreibt die Reihenfolge, wie einzelne Schritte ausgeführt werden, oder Bedingungen zur Ausführung.
- Nicht linear, Sprünge (Verzweigungen) möglich

Datenfluss:

- Beschreibt, wie Daten von einem Schritt zum nächsten kommen, d.h. von den Eingabewerten über die Operationen zum Ergebnis.
- Im Datenfluss gibt es keinen expliziten Kontrollfluss, sondern der Datenfluss enthält einen impliziten, dem Datenfluss gleichgerichteten Kontrollfluss.

6.2.1 Kontrollfluss

Informatik

Beim **Kontrollfluss** unterscheidet man:

- **Deklarative** Semantik:
  Formuliert die Bedingungen. Reihenfolge wird nicht spezifiziert.
  - *Beispiel*: „Es ist Schwimmbadwetter *falls* die Sonne scheint *und* es warm ist“.
- **Prozedurale** Semantik:
  Definiert die Reihenfolge der auszuführenden Schritte.
  - Beispiel: „Um herauszufinden, ob Schwimmbadwetter ist, schaue *zuerst* auf das Thermometer und vergleiche die angezeigte Temperatur mit 25°C, *dann* sieh‘ hoch, ob die Sonne scheint“.

6.2.1 Datenfluss

Informatik

Der **Datenfluss** beschrieben werden:

- durch **Eingabeparameter**,
  wobei die Position der einzelnen Parameter in der Liste ebenfalls zu beachten ist.
- durch **Ausgabeparameter**:
  - Rückgabewert einer Funktion
  - Parameter in der Übergabe-Liste, falls er entsprechend spezifiziert ist (z.B. Übergabe *per reference*, Übergabe einer *Adresse*)

6.2 Digitale Signalprozessoren (DSP)

Informatik

Weitere Besonderheiten von **DSP**'s:

- Sättigungsarithmetik:
  Bei Über- oder Unterlauf kein Vorzeichenwechsel. Bei Überlauf wird Größtmögliche, bei Unterlauf kleinstmögliche Zahl dargestellt.
  → keine extreme Verzerrung von Signalen.
- Indirekte Adressierung über Hilfsregister
  → kurze Befehle, schneller Zugriff
- Ausführung der meisten Befehle in einem Zyklus
- Hartverdrahtetes Steuerwerk, keine Mikroprogrammierung
- Spezialbefehle für Filterung, FFT, modulare Adressierung, Sättigungsarithmetik
- Sehr performante ALU → Multiplikation in wenigen oder nur einem Taktzyklus

6. Rechnerarchitektur: Zwischenbetrachtung

Informatik

- Datenwegzyklus (data path cycle):
  Zeit, in der zwei Operanden durch CPU geschleust und Ergebnis gespeichert wird.
  → je schneller, desto schneller Maschine.
- Programm muss nicht unbedingt von einer Hardware-CPU ausgeführt werden. Kann auch von einem Hilfsprogramm in HW-Befehle umgesetzt werden.
  → Interpreter
- Streben nach immer umfangreicheren und komplexen Befehlssätzen
  - IBM /360: Kompatible Familie von Rechnern. Nur die teuersten Modelle hatten direkte Hardware-Implementierung.
  - DEC VAX-Familie: reine Interpreter-Implementierung mit mehreren hundert Instruktionen mit ca. 200 Optionen für die Operanden.
- Vorteil Interpreter: einfacher Prozessor, Komplexität wird in den Speicher verlagert.
- Interpreter in schnellen Steuerspeichern (control store), die Nur-Lese-Speicher (Read Only Memory) sind, ermöglichen schnelle Ausführung. Zugriff µROM ca. 5 mal schneller als auf Hauptspeicher.
  - Motorola 68000, interpretierender Befehlssatz → großer Erfolg
  - Zilog Z8000, vergleichbarer Instruktionssatz in reiner HW → Misserfolg

6.3 CISC (Complex Instruction Set)

FH D Informatik

Im Laufe der Jahre waren den ursprünglich einfachen Maschinebefehlssätzen immer mehr spezialisierte (komplexe) Befehle hinzugefügt worden (CISC).

Man wollte Hochsprachenkonstrukte besser unterstützen:

Befehlsbreite nimmt zu → Befehls-Phase braucht mehr Zyklen → Prozessor aufwendiger und langsamer.

Schließen der „semantischen Lücke“: Fähigkeit Hochsprachen ←→ Maschinen

- Komplexe Assembler → komplexe Compiler
- Hochsprachenbefehl passt u.U nicht zu Assembler

→ Erkenntnis der 70er: Hochsprachen benutzen überwiegend nur sehr einfache Befehle, komplexe nur sehr selten.
→ **RISC**-Prozessoren