# INFORMATIK Oberstufe

# Funktionsweise eines Rechners

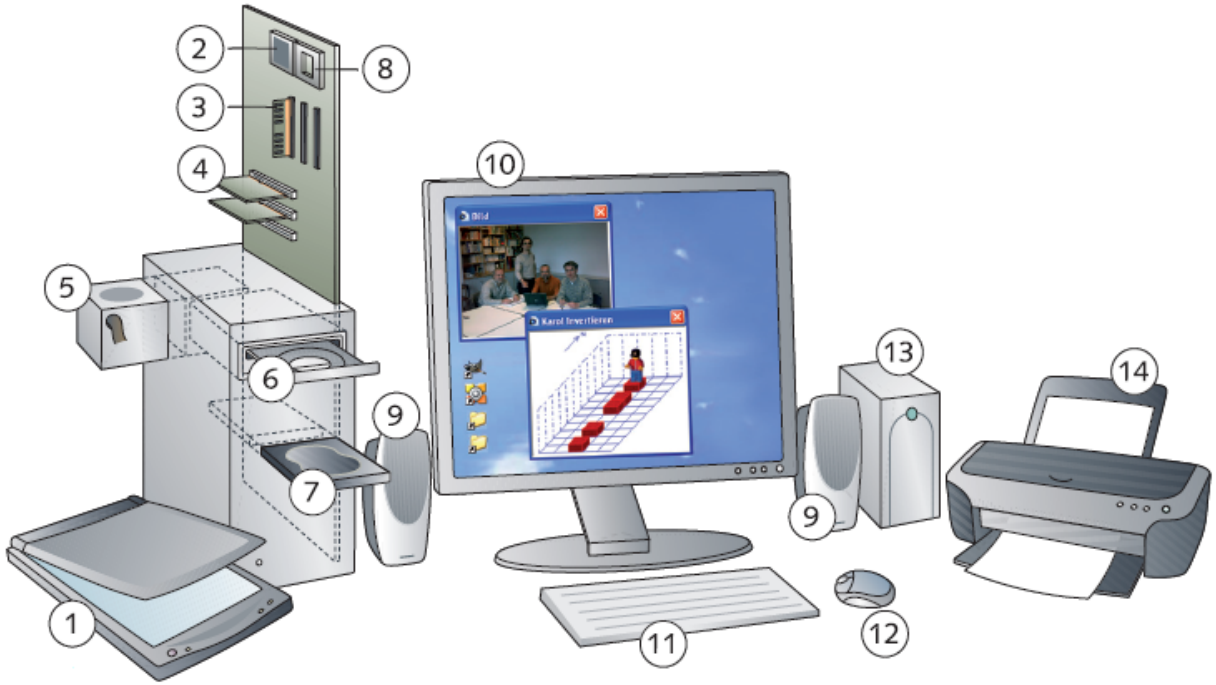
# Lehrplan

Inf 12.3 (ca. 17 Std.): Grundlegende Kenntnisse über den **Aufbau eines Rechners** und seiner **prinzipiellen Funktionsweise** helfen den Schülern, den Kommunikationsvorgang mit einer Maschine besser zu verstehen. Die prinzipielle Automatisierbarkeit des Übersetzungsvorgangs von einer höheren Programmiersprache in eine Maschinensprache wird ihnen bei der **Umsetzung einfacher Algorithmen mit einer maschinennahen Sprache** deutlich.

# Aufbau eines Computersystems

**Aus welchen Geräten besteht ein Computersystem? Was sind die Kernbauteile eines Computers?**

**Praxisorientierter Zugang, Festigung und Einordnung der Begriffe**



# Aufbau eines Computersystems

**Hauptkomponenten eines Computers**

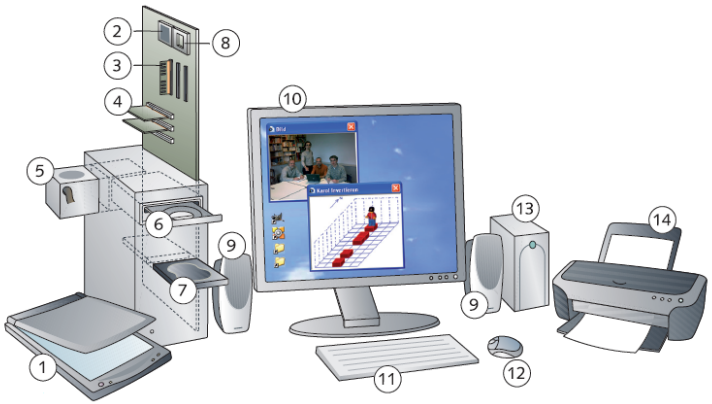
- **Prozessor** (Central Processing Unit, CPU)
- **Arbeitsspeicher** (Hauptspeicher, RAM)
- **Peripheriegeräte**

  **Eingabegeräte**: Tastatur, Maus, Scanner, Digitalkamera, Sensoren

  **Ausgabegeräte**: Monitor, Lautsprecher, Drucker, Beamer

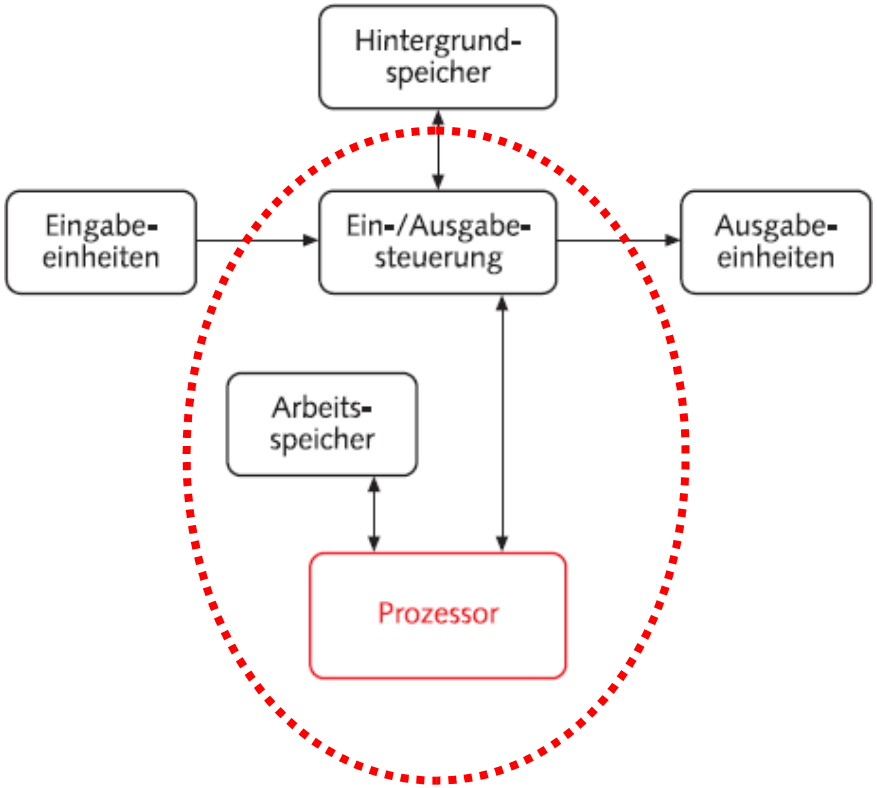
  **Datenspeichergeräte**: Festplatte, CD, DVD, Diskette, Magnetband

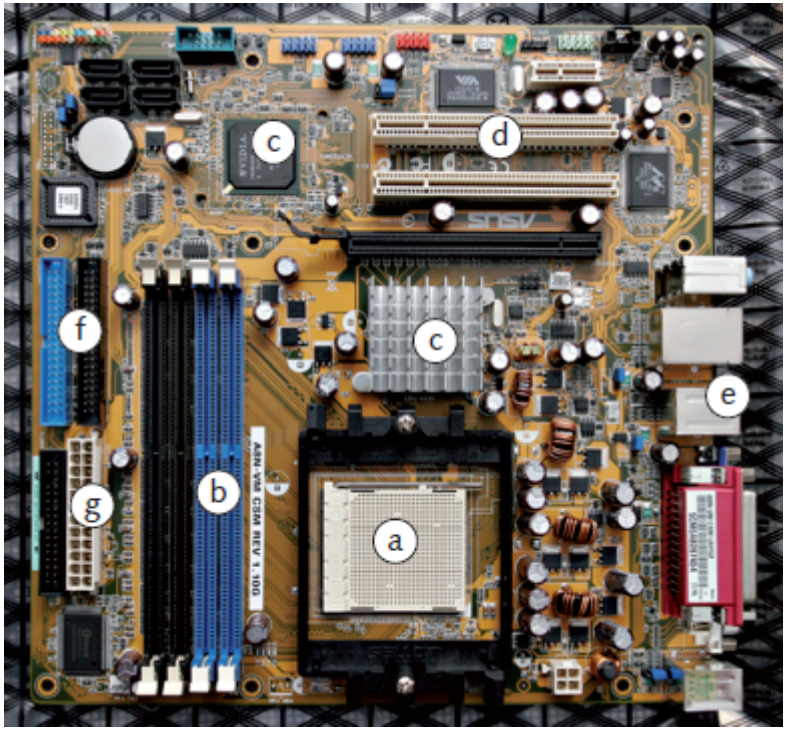
  **Kommunikation**: Netzwerkkarte, WLAN-Karte, DSL, ISDN



# Aufbau eines Computersystems

## Schema eines Computersystems



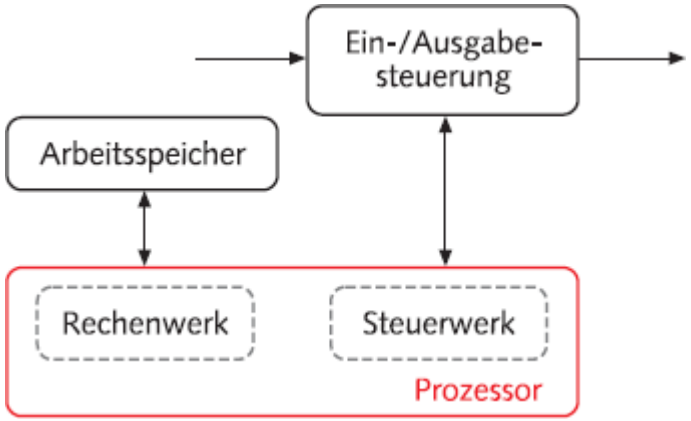



# Registermaschine als Modell

## von-Neumann-Prinzip (1946)

1) Eine Registermaschine besteht aus vier Funktionseinheiten:

- Arbeitsspeicher: Speichert sowohl Programme als auch Daten
- Steuerwerk: Interpretiert Befehle; koordiniert Rechenwerk, Ein-/ Ausgabesteuerung und Arbeits-speicher; regelt die Befehlabfolge



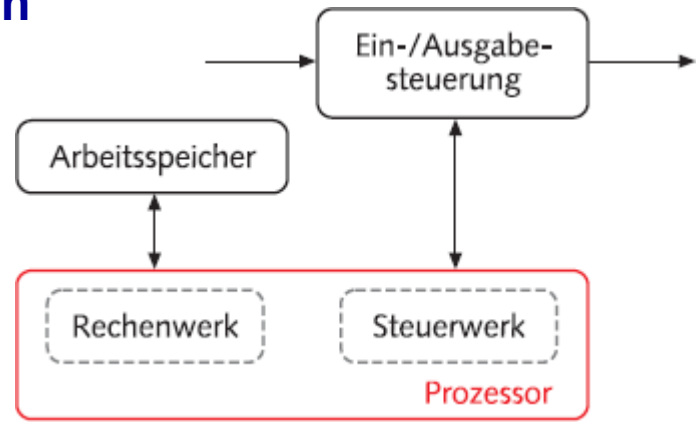

- Rechenwerk (Recheneinheit): Führt einfache arithmetische Rechenoperationen und logische Verknüpfungen durch
- Ein-/Ausgabesteuerung: Kommuniziert mit den Peripheriegeräten und steuert die Ein- und Ausgabe von Daten

# Registermaschine als Modell

## von-Neumann-Prinzip (1946)

2) Die Struktur des Rechners ist unabhängig von der zu bearbeitenden Aufgabenstellung.
Unterschiedliche Aufgaben werden durch entsprechende Programme gelöst (programmgesteuerter Rechner)

3) Programme, Daten, Zwischen- und Endergebnisse werden im selben Speicher, dem Arbeitsspeicher, abgelegt und können vom Rechenwerk verändert werden.




4) Der Arbeitsspeicher ist in Zellen gleicher Größe aufgeteilt, die fortlaufend nummeriert sind.

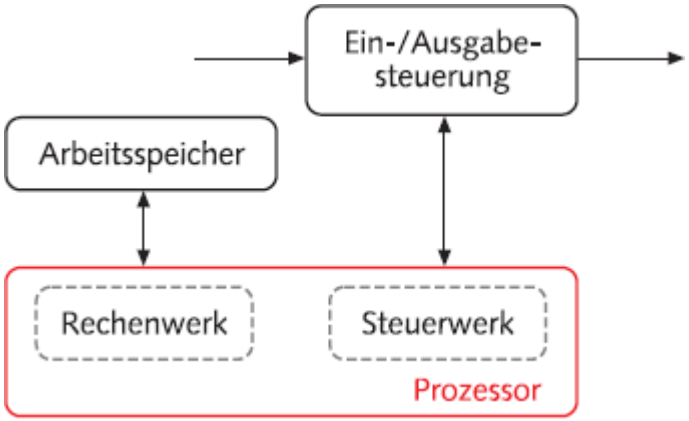
# Registermaschine als Modell

## von-Neumann-Prinzip (1946)

5) Das Programm besteht aus einer Folge von Befehlen, die im Allgemeinen nacheinander ausgeführt werden und in aufeinanderfolgenden Speicherzellen abgelegt sind.

6) Von der sequenziellen Abfolge der Befehlsausführung kann durch Sprungbefehle abgewichen werden.

7) Die Speicherelemente kennen zwei Schaltzustände: ein oder aus.
=> Binäre Codierung der Daten, Befehle und Adressen.



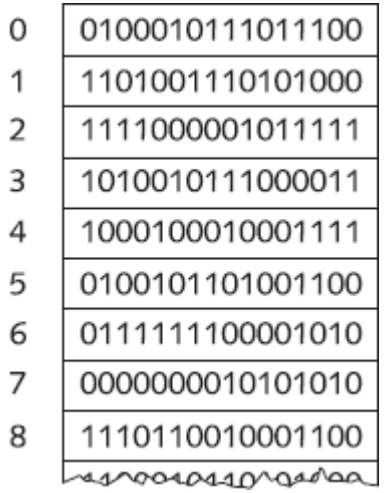
# Registermaschine als Modell

## Arbeitsspeicher

- **Bit: ein Schaltelement, 0 oder 1**

- **Speicherzelle: kleinste adressierbare Einheit;**
  **Anzahl von Bits = Speicherbreite;**
  **8 Bit (Byte), 16 Bit (Word), 32 Bit, 64 Bit;**

- **Speicheradresse:**
  **ermöglicht wahlfreien Zugriff (RAM)**

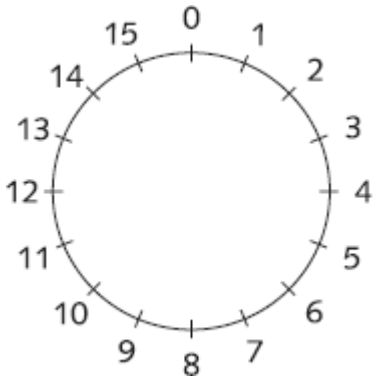



- **Kapazität des Speichers = „maximale Anzahl der Speicheradressen“ * „Größe der Speicherzelle in Bytes“**
  **1 kByte = 1024 Byte; 1 MByte = 1024 kByte;**
  **1 GByte = 1024 MByte**

# Registermaschine als Modell

## Darstellung von Zahlen

- k Bit pro Speicherzelle (Speicherbreite), damit lassen sich Zahlen von 0 bis $2^k$ – 1 darstellen




- vier Bits => 0, 1, 2, 3, …, 14, 15 oder binärer 0000, 0001, 0010, 0011, …, 1110, 1111
- binäre Addition: 0 + 0 gibt 0, 0 + 1 gibt 1, 1 + 1 gibt 0 und Übertrag 1 zur nächsthöheren Stelle

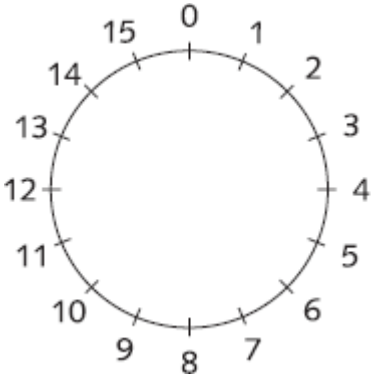
  bei vier Bits: 6 + 1 = 7, aber 15 + 1 = 0 denn 1111 + 1 = 0000 und Übertrag 1 (geht verloren)

- Zahlenkreis: Addition im Uhrzeigersinn, Subtraktion im Gegenuhrzeigersinn

# Registermaschine als Modell

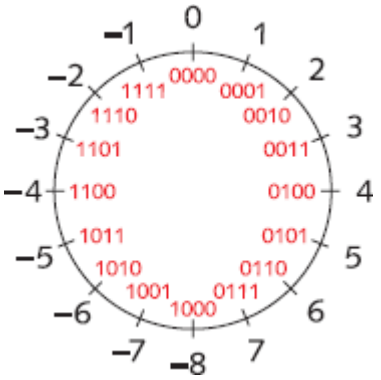
## Darstellung von Zahlen

- **Zahlenkreis: Addition im Uhrzeigersinn, Subtraktion im Gegenuhrzeigersinn**



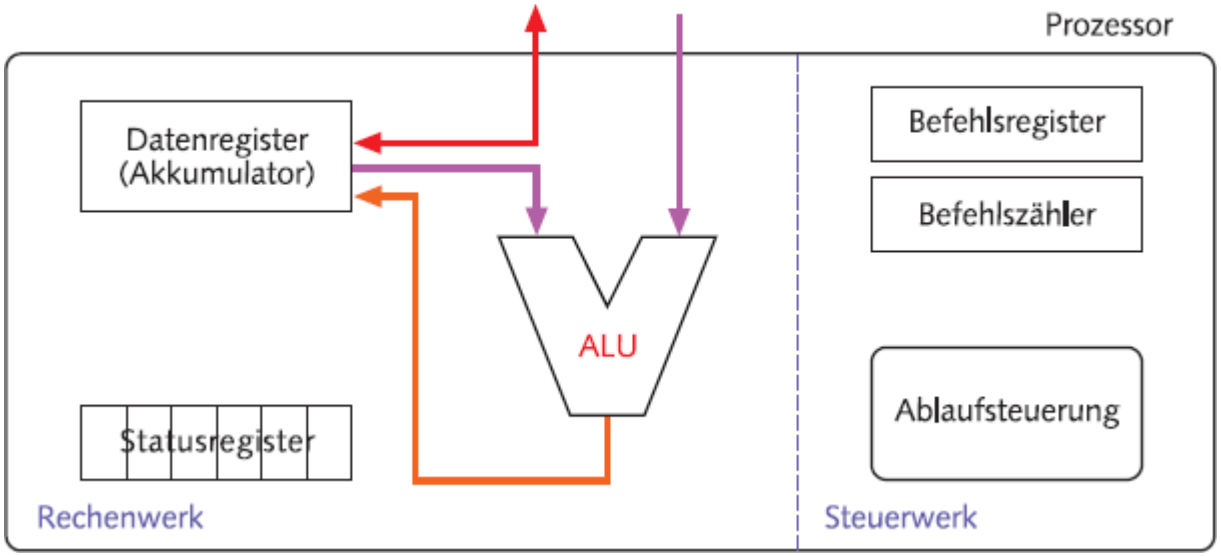

- **Darstellung negativer Zahlen: „links von Null“ negative Zahlen, „rechts von Null“ positive. Negative Zahlen haben in der Binärschreibweise an der linken Position eine 1.**

- **mit 4 Bit von -8 bis +7**
  **mit 16 Bit von -32768 bis +32767**
  **mit n Bit von $-2^{n-1}$ bis $+2^{n-1} - 1$**



# Registermaschine als Modell

## Vereinfachtes Modell eines Prozessors

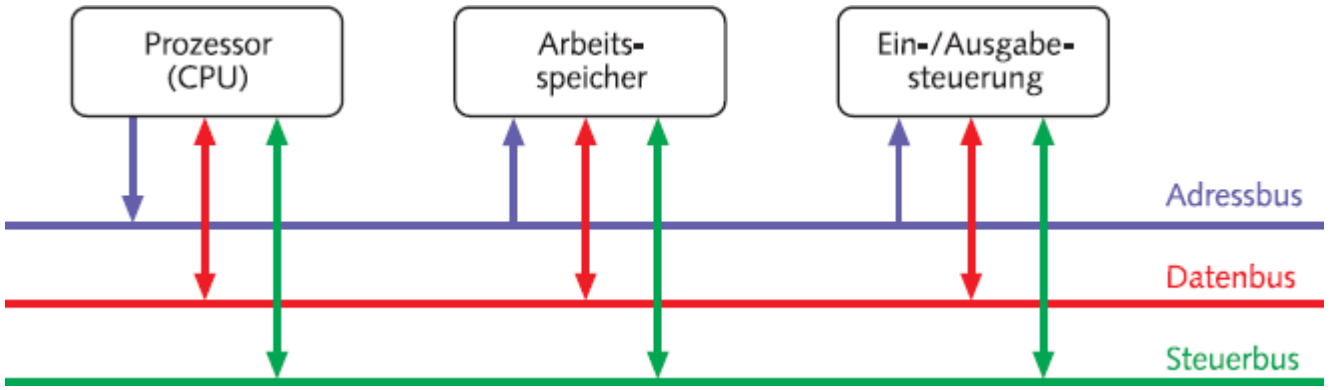



- **Befehlsregister (IR)**
- **Befehlszähler (IP)**
- **Datenregister Akkumulator (AC)**

- **arithmetisch-logischen Einheit (ALU)**
- **Statusregister (SR)**
  **Flags: Z = zero, N = negativ, V = overflow**

# Registermaschine als Modell

## Kommunikation Prozessor und Arbeitsspeicher

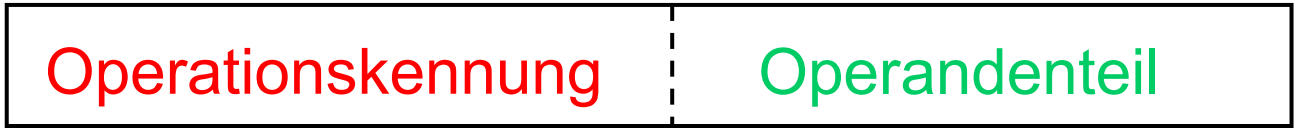



- **Adressbus** Adressierung jeder einzelne Speicherzelle
- **Datenbus** Übertragung der Daten (Umfang Speicherzelle)
- **Steuerbus** koordiniert Benutzung von Adress- und Datenbus (Protokoll)

# Operationsprinzip einer Registermaschine

## Maschinensprache

- **Maschinensprache** — **Menge aller Befehle, die ein Prozessor beherrscht**
- **Maschinenprogramm** — **Aneinanderreihung von Befehlen**
- **Maschinenbefehl**

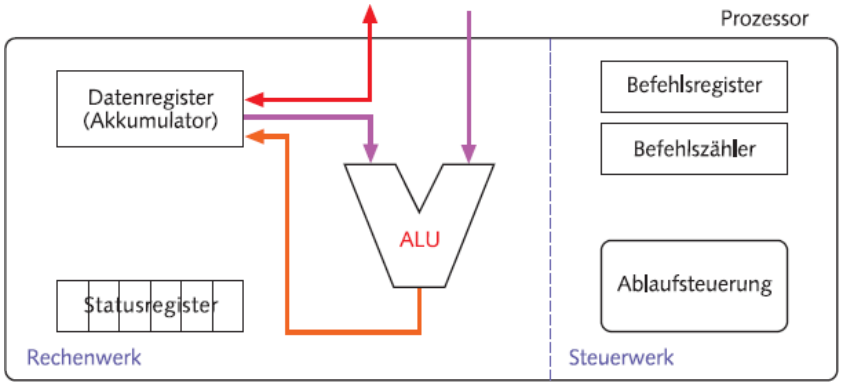



- **Operationskennung** — **bestimmt die Operation, die vom Rechenwerk bzw. Steuerwerk ausgeführt werden soll**
- **Operandenteil** — **zusätzliche Angaben: Daten, Speicher-adresse woher/wohin die Daten kommen**

# Operationsprinzip einer Registermaschine

## Befehlszyklus

- **FETCH-Phase, 1.Teil**
  **Befehl (OP-Code) aus Speicherzelle holen deren Adresse im Befehlszähler steht; in Befehlsregister ablegen; IP + 1**

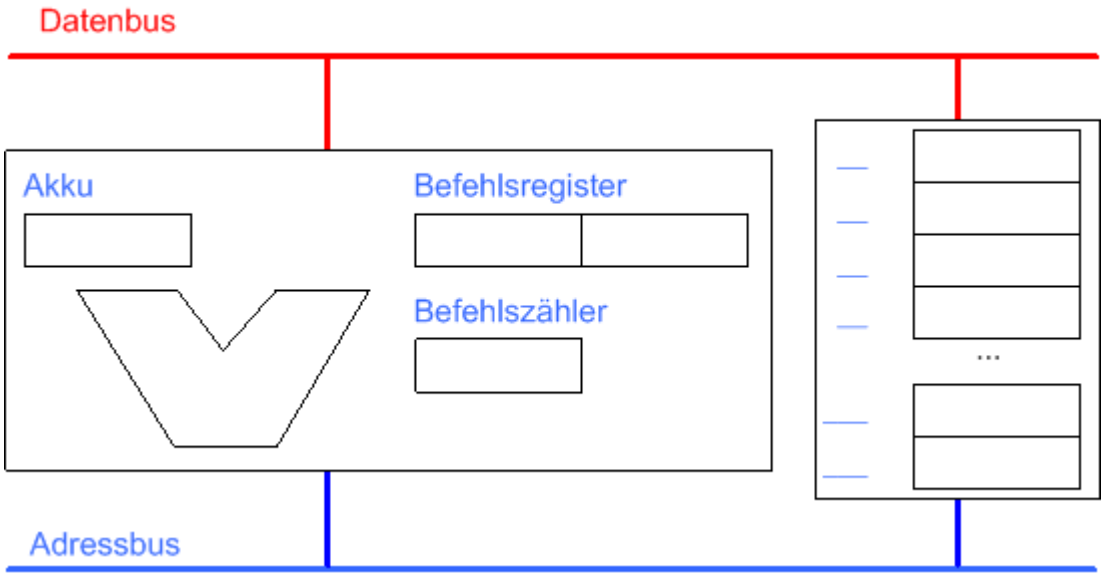



- **DECODE-Phase**
  **Befehl dekodieren**
- **FETCH-Phase, 2.Teil**
  **Befehl vervollständigen; im Befehlsregister ablegen; IP + k**
- **EXECUTE-Phase**
  **Operanden holen, an ALU bereitstellen; Befehl ausführen; Ergebnis rückschreiben; ggf. IP bei Sprungbefehlen anpassen**

# Operationsprinzip einer Registermaschine

## Befehlszyklus

- **Animation für eine einfache Maschine**
  **Speicherbreite 1 Word, Maschinenbefehlslänge 2 Word**
- **Anzeige von Prozessor und Speicherausschnitten**
- **Speicherinhalte und Adressen in Dezimalschreibweise**




Animation

# Operationsprinzip einer Registermaschine

## Befehlssatz

| Opcode | zusätzliche Daten | PC nachher | SR beeinflusst? | Beschreibung |
|---|---|---|---|---|
| **Transportbefehle** | | | | |
| LOAD | *Adresse* | +2 | x | Lade den Akku mit Daten aus dem Hauptspeicher an der *Adresse*. |
| LOADI | *Wert* | +2 | x | Lade den Akku mit dem *Wert*, der unmittelbar der Operationskennung folgt (I für immidiate; engl. unmittelbar). |
| STORE | *Adresse* | +2 | | Speichere den Akkuinhalt im Hauptspeicher an der *Adresse*. |
| **arithmetische Befehle** | | | | |
| ADD | *Adresse* | +2 | x | Addiere zum Akkuinhalt den Inhalt der Speicherzelle an der *Adresse* und speichere das Ergebnis im Akku. |
| ADDI | *Wert* | +2 | x | Addiere zum Akkuinhalt den *Wert* und speichere das Ergebnis im Akku. |
| SUB | *Adresse* | +2 | x | Subtrahiere vom Akkuinhalt den Inhalt der Speicherzelle an der *Adresse* und speichere das Ergebnis im Akku. |
| SUBI | *Wert* | +2 | x | Subtrahiere vom Akkuinhalt den *Wert* und speichere das Ergebnis im Akku. |
| MUL | *Adresse* | +2 | x | Multipliziere den Akkuinhalt mit dem Inhalt der Speicherzelle an der *Adresse* und speichere das Ergebnis im Akku. |
| MULI | *Wert* | +2 | x | Multipliziere den Akkuinhalt mit dem *Wert* und speichere das Ergebnis im Akku. |

# Operationsprinzip einer Registermaschine

## Befehlssatz

| Opcode | zusätzliche Daten | PC nachher | SR beeinflusst? | Beschreibung |
|---|---|---|---|---|
| **arithmetische Befehle** | | | | |
| DIV | *Adresse* | +2 | x | Dividiere den Akkuinhalt durch den Inhalt der Speicherzelle an der *Adresse* und speichere das Ergebnis der Ganzzahl-division im Akku (z. B.: 38 : 3 → 12). |
| DIVI | *Wert* | +2 | x | Dividiere den Akkuinhalt durch den *Wert* und speichere das Ergebnis der Ganzzahldivision im Akku. |
| MOD | *Adresse* | +2 | x | Dividiere den Akkuinhalt durch den Inhalt der Speicherzelle an der *Adresse* und speichere den Rest der Ganzzahldivision im Akku (z. B.: 38 : 3 → 2). |
| MODI | *Wert* | +2 | x | Dividiere den Akkuinhalt durch den *Wert* und speichere den Rest der Ganzzahldivision im Akku. |
| CMP | *Adresse* | +2 | x | Vergleiche den Akkuinhalt mit dem Inhalt der Speicherzelle an der *Adresse* und setze das Statusregister entsprechend. Dies ist gleichwertig zu einer Subtraktion „Akku – Speicher-zelleninhalt“, nur wird das Ergebnis nicht im Akku abgelegt.<br>Akkuinhalt > Speicherzelleninhalt ⇒ kein Flag<br>Akkuinhalt = Speicherzelleninhalt ⇒ Zero-Flag<br>Akkuinhalt < Speicherzelleninhalt ⇒ Negativ-Flag |
| CMPI | *Wert* | +2 | x | Vergleiche den Akkuinhalt mit dem *Wert* und setze das Statusregister entsprechend (siehe CMP). |

# Operationsprinzip einer Registermaschine

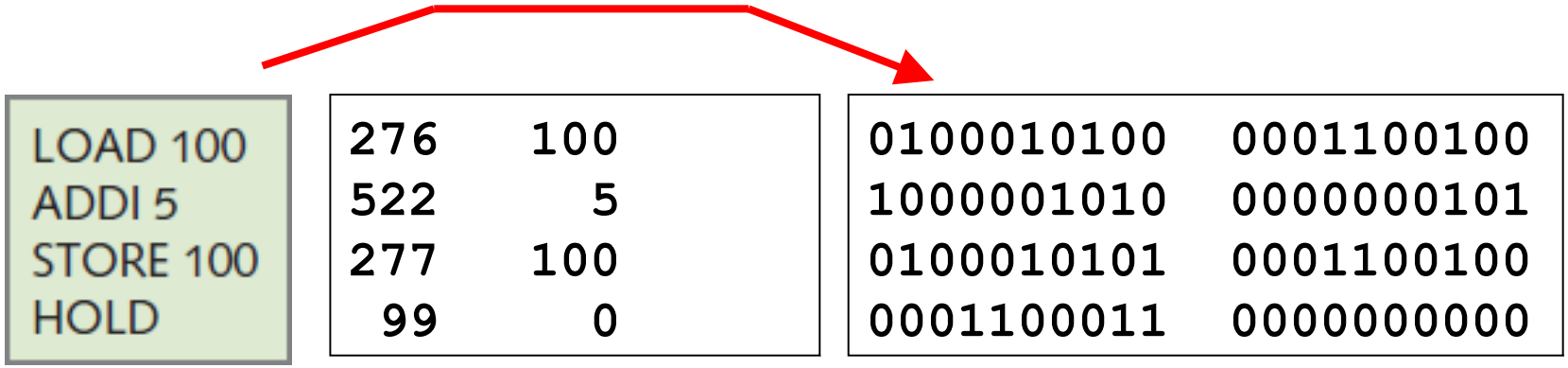
## Befehlssatz

| Opcode | zusätzliche Daten | PC nachher | SR beeinflusst? | Beschreibung |
|---|---|---|---|---|
| **Steuerbefehle** | | | | |
| JMP | *Adresse* | *Adresse* | | Springe zur *Adresse* (jump), durch Schreiben von *Adresse* in den Befehlszähler (PC). |
| JMPZ | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Zero-Flag gesetzt ist (jump on zero). |
| JMPNZ | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Zero-Flag nicht gesetzt ist (jump on not zero). |
| JMPN | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag gesetzt ist (jump on negativ). |
| JMPNN | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag nicht gesetzt ist (jump on not negativ). |
| JMPP | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag und das Zero-Flag nicht gesetzt sind (jump on plus). |
| JMPNP | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag oder das Zero-Flag gesetzt sind (jump on not plus). |
| | | | | |
| HOLD | | | | Halte den Prozessor an. |

# Systemnahe Programmierung

## Assembler

**Addiere zu der Zahl, die an der Speicheradresse 100 abgelegt ist, die Zahl 5. In Kurzschreibweise: [100] = [100] + 5**



| Assembler-sprache | Maschinen-programm (Dezimalschreibw.) | Maschinen-programm (Binärschreibw.) |
|---|---|---|
| LOAD 100 | 276 100 | 0100010100 0001100100 |
| ADDI 5 | 522 5 | 1000001010 0000000101 |
| STORE 100 | 277 100 | 0100010101 0001100100 |
| HOLD | 99 0 | 0001100011 0000000000 |

# Systemnahe Programmierung

## MiniMaschine

**Die Minimaschine ist ein CPU-Simulator. Kern ist ein typischer Prozessor einer Registermaschine, der im Einzelschrittmodus, Mikroschrittmodus und Abarbeitungsmodus arbeiten kann. (Autor Albert Wiedemann)**

```
Programm   = {Zeile} .
Zeile      = [Markenvereinbarung] [Befehl] .
Markenvereinbarung  = Marke ":" .
Marke      = "A" .. "Z" | "a" .. "z" { "A" .. "Z" | "a" .. "z" | "0" .. "9" | "_" | "$" } .
Befehl     = Operation Adresse .
Operation  = "HOLD" | "RESET" | "NOOP" | "ADD" | "SUB" | "MUL" | "DIV"" |
             "MOD" | "CMP" | "LOAD" | "STORE" | "JMPP" | "JMPNN" | "JMPN" |
             "JMPNP" | "JMPZ" | "JMPNZ" | "JMPV" | "JMP" | "ADDI" | "SUBI" |
             "MULI" | "DIVI" | "MODI" | "CMPI" | "LOADI" | "WORD" .
Adresse    = Marke | Zahl .
Zahl       = "0" .. "9" { "0" .. "9" } .
```

MiniMaschine

# Systemnahe Programmierung

## MiniMaschine

### Verwendung von Marken; spaltenorientierte Notation

```
LOADI 32
STORE 100

LOAD 100
ADDI 5
STORE 100
HOLD
```

Beispiel1.mia

```
Vorbeleg:  LOADI   32
           STORE   100

Start:     LOAD    100
           ADDI    5
           STORE   100
Ende:      HOLD
```

Beispiel2.mia

```
Start:     LOAD    Zahl
           ADDI    5
           STORE   Zahl
Ende:      HOLD

Zahl:      WORD    32
```

Beispiel3.mia

# Systemnahe Programmierung

## Zustand der Registermaschine

**Der Zustand einer Registermaschine wird durch die aktuellen Inhalte aller Register beschrieben, das sind die internen Register in der CPU (Befehlszähler, Datenregister, Statusregister) sowie die Speicherzellen des Hauptspeichers.**
**Tupel (IP, AC, SR, AS0, AS1, AS2, AS3, …)**

```
LOAD 100
ADDI 5
STORE 100
HOLD
```

| | | | | Arbeitsspeicher | | | |
|---|---|---|---|---|---|---|---|
| IP | AC | SR | | AS100 | AS101 | AS102 | |
| 0 | ? | | | 32 | ? | ? | |

# Systemnahe Programmierung

## Sprungbefehle

**Sprungbefehle ermöglichen, je nach Wert der Statusregister-flags, eine Steuerung des Programmablaufs.**

| | | | | |
|---|---|---|---|---|
| JMP | *Adresse* | *Adresse* | | Springe zur *Adresse* (jump), durch Schreiben von *Adresse* in den Befehlszähler (PC). |
| JMPZ | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Zero-Flag gesetzt ist (jump on zero). |
| JMPNZ | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Zero-Flag nicht gesetzt ist (jump on not zero). |
| JMPN | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag gesetzt ist (jump on negativ). |
| JMPNN | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag nicht gesetzt ist (jump on not negativ). |
| JMPP | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag und das Zero-Flag nicht gesetzt sind (jump on plus). |
| JMPNP | *Adresse* | +2 oder *Adresse* | | Springe zur *Adresse*, wenn das Negativ-Flag oder das Zero-Flag gesetzt sind (jump on not plus). |

# Systemnahe Programmierung

## Bedingte Anweisung (einseitig)

```
x = Preis
wenn x >= 150
dann
    x = x – 10
endewenn
```

| | | |
|---|---|---|
| 0 | LOAD 100 | |
| 2 | CMPI 150 | Berechnet Akku – 150 |
| 4 | JMPN 8 | Wenn aufgrund des letzten Befehls das Negativ-Flag gesetzt wurde, darf kein Rabatt abgezogen werden. Es erfolgt der Sprung zur Speicheradresse 8. |
| 6 | SUBI 10 | Sonst wird 10 abgezogen. |
| 8 | STORE 101 | Im Akku ist das Ergebnis; es wird in 101 abgelegt. |
| 10 | HOLD | |

| | | | | Arbeitsspeicher | | | |
|---|---|---|---|---|---|---|---|
| IP | AC | SR | | AS100 | AS101 | AS102 | |
| 0 | ? | | | 50 | ? | ? | |
| 2 | 50 | | | 50 | ? | ? | |
| 4 | 50 | N | | 50 | ? | ? | |
| 8 | 50 | | | 50 | ? | ? | |
| 10 | 50 | | | 50 | 50 | ? | |
| 10 | 190 | | | 200 | 190 | ? | |

# Systemnahe Programmierung

## Bedingte Anweisung (einseitig)

### Verwendung von Marken für die Sprungziele

```
LOAD 100
CMPI 150
JMPN 8
SUBI 10
STORE 101
HOLD
```

```
        LOAD 100
        CMPI 150
        JMPN Ziel1
        SUBI 10
Ziel1:  STORE 101
        HOLD
```

| allgemeine Form | Label | Befehle |
|---|---|---|
| | | *Bedingung* in Flag-Status umsetzen |
| wenn *Bedingung* dann | | wenn Flag-Bedingung nicht zutrifft, dann Sprung zu weiter |
| dann-Teil | | Befehle des dann-Teils |
| endewenn | | |
| weiterer Programmverlauf | weiter: | weitere Befehle |

# Systemnahe Programmierung

## Bedingte Anweisung (zweiseitig)

Von zwei Zahlen soll die kleinere in der Speicherzelle Zelle1 und die größere in der Speicherzelle Zelle2 abgelegt werden.

```
x1 = Zahl1
x2 = Zahl2
wenn x1 >= x2
dann
    Zelle1 = x2, Zelle2 = x1
sonst
    Zelle1 = x1, Zelle2 = x2
endewenn
```

| allgemeine Form | Label | Befehle |
|---|---|---|
| | | *Bedingung* in Flag-Status umsetzen |
| wenn *Bedingung* dann | | wenn Flag-Bedingung nicht zutrifft, dann Sprung zu sonst |
| dann-Teil | | Befehle des dann-Teils |
| sonst | | immer Sprung zu weiter |
| sonst-Teil | sonst: | Befehle des sonst-Teils |
| endewenn | | |
| weiterer Programmverlauf | weiter: | weitere Befehle |

# Systemnahe Programmierung

## Wiederholung mit Anfangsbedingung

Die Fakultät einer Zahl soll berechnet werden, mit
$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$
und $0! = 1$

```
n = Zahl
Ergebnis = 1
wiederhole solange n > 0
    Ergebnis = Ergebnis * n
    n = n – 1
endewiederhole
```

| allgemeine Form | Label | Befehle |
|---|---|---|
| | | Vorbelegung von Variablen, die in der Wiederholung bearbeitet werden |
| wiederhole solange | wiederh: | *Bedingung* in Flag-Status umsetzen |
| | | wenn Flag-Bedingung nicht zutrifft, dann Sprung zu weiter |
| *Bedingung* wahr | | |
| wiederhol-Teil | | Befehle des wiederhol-Teils |
| endewiederhole | | immer Sprung zu wiederh |
| weiterer Programmverlauf | weiter: | weitere Befehle |

# Systemnahe Programmierung

## Wiederholung mit Anfangsbedingung

Die Zahl, deren Fakultät berechnet wird, steht in der Zelle 100, das Ergebnis in 101.

| | | | |
|---|---|---|---|
| 0 | | LOADI 1 | |
| 2 | | STORE 101 | Vorbelegung: Ergebnis = 1 |
| 4 | wiederh: | LOAD 100 | Lädt die Zahl n in den Akku, setzt das Zero-Flag, wenn die Zahl 0 ist. |
| 6 | | JMPZ weiter | Springt bei gesetztem Zero-Flag, d. h. bei Ende der Wiederholung mit Anfangsbedingung. |
| 8 | | MUL 101 | Akku enthält n multipliziert mit dem Ergebnis. |
| 10 | | STORE 101 | (Zwischen-)Ergebnis ablegen. |
| 12 | | LOAD 100 | Lädt wieder n. |
| 14 | | SUBI 1 | n verringern. |
| 16 | | STORE 100 | Verringertes n speichern; Akku enthält n. |
| 18 | | JMP wiederh | Ende des Wiederholungsteils |
| 20 | weiter: | HOLD | |

# Systemnahe Programmierung

## Wiederholung mit Anfangsbedingung

```
Vorbeleg:  LOADI   5
           STORE   100

Start:     LOADI   1
           STORE   101
wiederh:   LOAD    100
           JMPZ
Ende
           MUL     101
           STORE   101
           LOAD    100
           SUBI    1
           STORE   100
           JMP
wiederh
Ende:      HOLD
```

Die Zahl, deren Fakultät berechnet wird, steht in der Zelle Zahl1, das Ergebnis in Ergeb.

```
Start:     LOADI   1
           STORE   Ergeb
wiederh:   LOAD    Zahl1
           JMPZ    Ende
           MUL     Ergeb
           STORE   Ergeb
           LOAD    Zahl1
           SUBI    1
           STORE   Zahl1
           JMP     wiederh
Ende:      HOLD

Zahl1:     WORD    5
Ergeb:     WORD    0
```

# Systemnahe Programmierung

## Wiederholung mit fester Anzahl

Es soll die Summe der natürlichen Zahlen, ausgehend von einer gegebenen Untergrenze bis zu einer gegebenen Obergrenze, berechnet werden.

```
Summe = 0
wiederhole mit n von Startwert bis Endwert
    Summe = Summe + n
endewiederhole
```

| allgemeine Form | Label | Befehle |
|---|---|---|
| | | Vorbelegung von Variablen, die in der Wiederholung bearbeitet werden |
| | | Zählspeicherzelle mit Startwert belegen. |
| wiederhole mit i von Startwert bis Endwert | wiederh: | Ist der Inhalt der Zählspeicherzelle kleiner oder gleich Endwert? Flag-Status setzen. |
| | | wenn Flag-Status nicht zutrifft, dann Sprung zu weiter |
| wiederhol-Teil | | Befehle des wiederhol-Teils |
| | | Inhalt der Zählspeicherzelle um 1 erhöhen. |
| endewiederhole | | immer Sprung zu wiederh |
| weiterer Programmverlauf | weiter: | weitere Befehle |

# Systemnahe Programmierung

## Wiederholung mit fester Anzahl

Es soll die Summe der natürlichen Zahlen bestimmt werden, deren Startwert sich in Zelle 100 und deren Endwert sich in Zelle 101 befindet; die Summe ist in Zelle 102 abzulegen.

| | | | |
|---|---|---|---|
| 0 | | LOADI 0 | |
| 2 | | STORE 102 | Vorbelegung: Summe = 0 |
| 4 | | LOAD 100 | Zähler initialisieren mit Startwert. |
| 6 | | STORE 103 | Zählerspeicherzelle 103 |
| 8 | wiederh: | CMP 101 | Zähler vergleichen mit Endwert; falls Zähler kleiner oder gleich Endwert (d. h. Zähler – Endwert <= 0), wird Negativ-Flag bzw. Zero-Flag gesetzt. |
| 10 | | JMPP weiter | Springt, wenn das Negativ-Flag und das Zero-Flag nicht gesetzt sind, d. h. bei Ende der Wiederholung. |
| 12 | | ADD 102 | Akku enthält Zähler; Zwischensumme dazuaddieren. |
| 14 | | STORE 102 | (Zwischen-)Summe ablegen. |
| 16 | | LOAD 103 | Lädt wieder Zähler. |
| 18 | | ADDI 1 | Zähler erhöhen. |
| 20 | | STORE 103 | Erhöhten Zähler speichern; Akku enthält Zähler. |
| 22 | | JMP wiederh | Ende des Wiederholungsteils |
| 24 | weiter: | HOLD | |

# Systemnahe Programmierung

## Von der Hochsprache zur Maschinensprache

```
PROGRAM Summe;

VAR Summe, Von, Bis, i;

BEGIN
Von := 5;
Bis := 10;
Summe := 0;

FOR i := Von TO Bis
DO
     Summe := Summe + i;
END

END Summe.
```

übersetzen

```
        LOADI   5
        STORE   Von
        LOADI   10
        STORE   Bis
        LOADI   0
        STORE   Summe
        LOAD    Von
        STORE   i
        LOAD    Bis
        STORE   hi$1
M$1:
        LOAD    i
        CMP     hi$1
        JMPP    M$2
        LOAD    Summe
        ADD     i
        STORE   Summe
```

```
        LOAD    i
        ADDI    1
        JMPV    M$2
        STORE   i
        JMP     M$1
M$2:
        HOLD
Summe:  WORD    0
Bis:    WORD    0
Von:    WORD    0
i:      WORD    0
hi$1:   WORD    0
```

# Systemnahe Programmierung

## Von der Hochsprache zur Maschinensprache

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 532 | 5 | 277 | 46 | 532 | 10 | 277 | 45 | 532 | 0 |
| **10** | 277 | 44 | 276 | 46 | 277 | 47 | 276 | 45 | 277 | 48 |
| **20** | 276 | 47 | 271 | 48 | 286 | 42 | 276 | 44 | 266 | 47 |
| **30** | 277 | 44 | 276 | 47 | 522 | 1 | 293 | 42 | 277 | 47 |
| **40** | 292 | 20 | 99 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **50** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Funktionsweise eines Rechners

**Aufgaben siehe Arbeitsblatt.**
**Verwendung der MiniMaschine.**

**Viel Erfolg!**