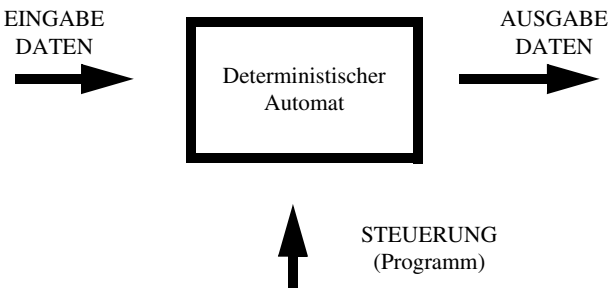
Grundlagen der Informatik für Ingenieure

Background:

3. Grundlagen der Rechnerarchitektur - Teil 1

3.1 Einfaches Rechnermodell

- Blackbox



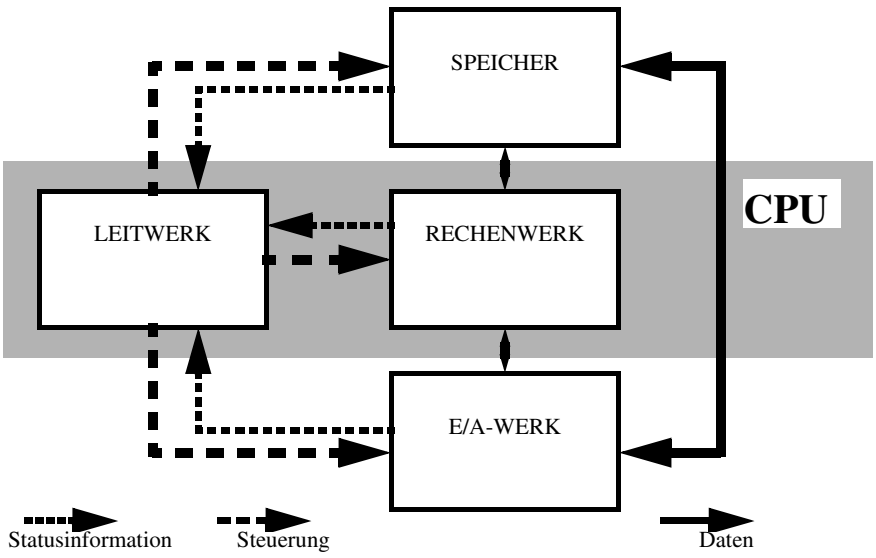
- Ein Eingabedatenstrom wird durch den Automaten in einen Ausgabedatenstrom umgewandelt.
- Der Vorgang wird durch ein Programm gesteuert

3.1 Einfaches Rechnermodell

- Pioniere
  - John (Johann) von Neumann, geb. 1903 in Budapest, vielseitiger Mathematiker in den USA; gest. 1957 in Washington.
    - ENIAC (1945)/EDVAC(1949) - Mauchly, Eckert
    - EDVAC - Theor. Beschreibung “von Neumann - Architektur”
    - IAS-Röhrenrechner 1952 - Realisierung “von Neumann Architektur”
  - Konrad Zuse (1910-1995); 1. programmgesteuerten Rechner Z3 (Relais) 1941; eignenständige Rechnerentwicklung/-produktion bis 1967.

3.1 Einfaches Rechnermodell

- von Neumann-Architektur



3.1 Einfaches Rechnermodell

- Rechnerkomponenten
  - **Leitwerk**:
    - Die Hardware-Steuerlogik zur Steuerung des Rechen-, Speicher- und E/A-Werks.
  - **Rechenwerk**:
    - Hardware zur Verknüpfung von Daten
      - Registersätze
      - **Funktions Einheiten**: +; - ; * ; / ; logische Bitmanipulation; ...
  - Leitwerk und Rechenwerk werden zur **Central Processing Unit (CPU)** zusammengefasst. Die CPU wird auch als **Prozessor** bezeichnet.

3.1 Einfaches Rechnermodell

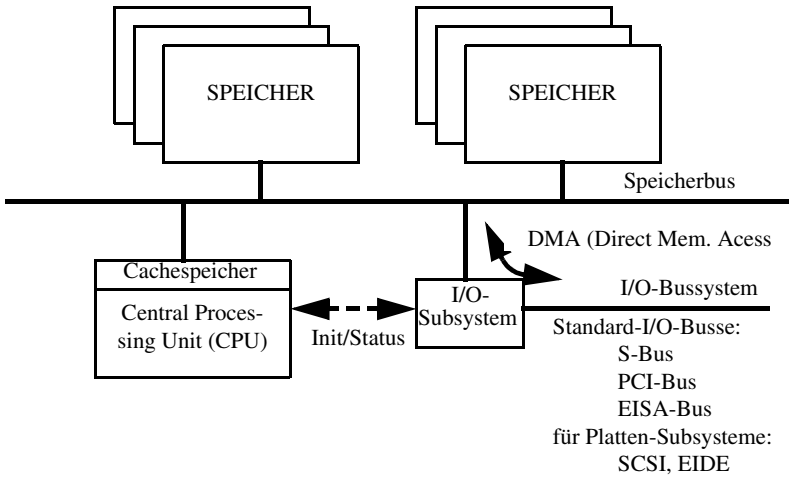
- **Speicher (**Arbeitsspeicher):
  - Lineare Anordnung von (2-wertigen) Speicherzellen zur Speicherung **binär verschlüsselter** Daten für
    - die Steuerung des Systems (Programme; **Code**) und
    - Daten
  - Speicherzellen **(Bits)** werden in Gruppen zusammengefasst:
    - **Byte** (8 Bit)
    - **Worte** (16, 32, 64 Bit)
    - **Seiten** (2k-, 4k-, 8k-Byte)
    - **Segmente** (variabel oder vielfaches einer Seite falls System auch seitenorientiert)
  - Byte-Einheiten werden durch Speicheradressen lokalisiert.

3.1 Einfaches Rechnermodell

◆ **E/A-Werk:**

- Stellt die HW-Schnittstelle zur Außenwelt bereit; z. B. zu
  - Plattenlaufwerken (sekundärer, persistenter Speicher)
  - Bildschirm
  - Tastatur
  - Maus
  - Drucker
  - Scanner
  - Video
  - Audio
  - Schnittstellen zur Steuerung techn. Prozesse
  - ....

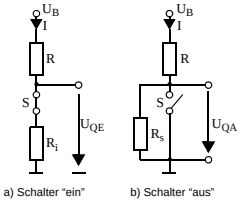
3.2 Beispiel einer Realisierung



3.3 Einige wenige Hardware-Grundlagen

◆ Grundbausteine sind Transistoren

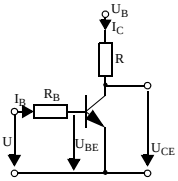
- benötigt werden *Schaltwerke*, die zwei stabile Zustände einnehmen können, also Schaltwerke mit speichernden Eigenschaften.
  - Elektronische Schalter - Ersatzschaltbild



Idealerweise gehen die Widerstandwerte R_i --> 0; R_s --> unendl., was mit elektronischen Schalter nicht "ganz" erreichbar ist!

3.3 Einige wenige Hardware-Grundlagen

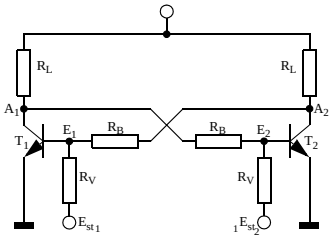
- Bipolartransistor als Schalter



Als Schalter dient die Kollektor-Emitterstrecke:
Ist der Basisstrom $I_B = 0$, so ist die Kollektor-Emitterstrecke gesperrt (es fließt nur noch ein kleiner Reststrom). Der Schalter ist also *offen*.
Ist der Basisstrom $I_B > 0$, ist die Kollektor-Emitterstrecke leitend; es fließt ein hoher Strom. Der Schalter ist *geschlossen*.

3.3 Einige wenige Hardware-Grundlagen

- aus 2 Transistorschaltern kann man mit Rückkopplung ein bistabiles Schaltwerk konstruieren, genannt FlipFlop:



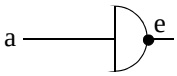
Anfangszustand: Bedingt durch Toleranzwerte der Bauelemente wird beim Anlegen einer Betriebsspannung U_b immer ein Schalter den “Aus-Zustand” und einer den “Ein-Zustand” annehmmen. Durch Ansteuern aller Schaltwerke in gleicher Weise, kann man dafür Sorge tragen, dass alle Schaltwerke den gleichen Zustand annehmen.

Die Rückkopplung bewirkt, dass die Schaltung zwei stabile Zustände einnehmen kann. Durch das Anlegen einer Spannung an E_{St1} oder E_{St2}, wird dann jeweils der zugehörige Transistor leitend, der jeweils andere kommt in den Sperrzustand.

Angenommen wir legen an E_{St1} eine Spannung an (H-Level), dann wird die Kollektor-Emitterstrecke von T1 leitend, die Spannung A_1 nimmt einen niedrigen Wert an (L-Level). Durch die Rückkopplung nach E_2 kann keine Basisstrom mehr am T_2 fließen, die Kollektor-Emitterstrecke von T_2 ist gesperrt. A_2 nimmt also H-Level an! Dieser wiederum rückgekoppelt an E_1 sorgt dafür, dass ein Basisstrom am T_1 fließt. Der Zustand ist stabil!

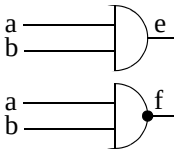
3.3 Einige wenige Hardware-Grundlagen

- ..und “Verknüpfungsschaltwerke” wie
  - NOT



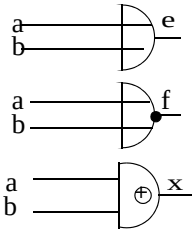
a	e
0	1
1	0
0	1
1	0

  - AND; NAND



a	b	e	f
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

  - OR; NOR, XOR



a	b	e	f	x
0	0	0	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	0	0

3.3 Einige wenige Hardware-Grundlagen

- Speicher sind aneinandergereite FlipFlops mit einer Auslese/Schreiblogik. Da nach dem Lesen der Inhalt zerstört ist, muss das ausgelesene Datum wieder zurückgeschrieben werden - das ist der sog. Lese/Schreibzyklus.
- Register sind aneinandergereite FlipFlops jeweils mit Verknüpfungsschaltungen (Func. Units), um die gewünschten Operationen auszuführen.

3.4 Binäre Zahlensysteme

- Dualzahlensystem: Basis 2: 2^n .. 2^4 2^3 2^2 2^1 2^0
- Oktalsystem (3er-Gruppen aus dem Dualsystem)
- Hexadezimalsystem (4er-Gruppen aus Dualssystem)

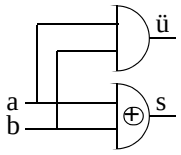
2^3	2^2	2^1	2^0	Dez	O	X		2^4	2^3	2^2	2^1	2^0	Dez	O	X
0	0	0	0	0	0	0		0	1	0	0	0	8	10	8
0	0	0	1	1	1	1		0	1	0	0	1	9	11	9
0	0	1	0	2	2	2		0	1	0	1	0	10	12	A
0	0	1	1	3	3	3		0	1	0	1	1	11	13	B
0	1	0	0	4	4	4		0	1	1	0	0	12	14	C
0	1	0	1	5	5	5		0	1	1	0	1	13	15	D
0	1	1	0	6	6	6		0	1	1	1	0	14	16	E
0	1	1	1	7	7	7		0	1	1	1	1	15	17	F
								1	0	0	0		16	20	10
								usw					17	21	11

3.4 Binäre Zahlensysteme

◆ Addieren dualer Zahlen:

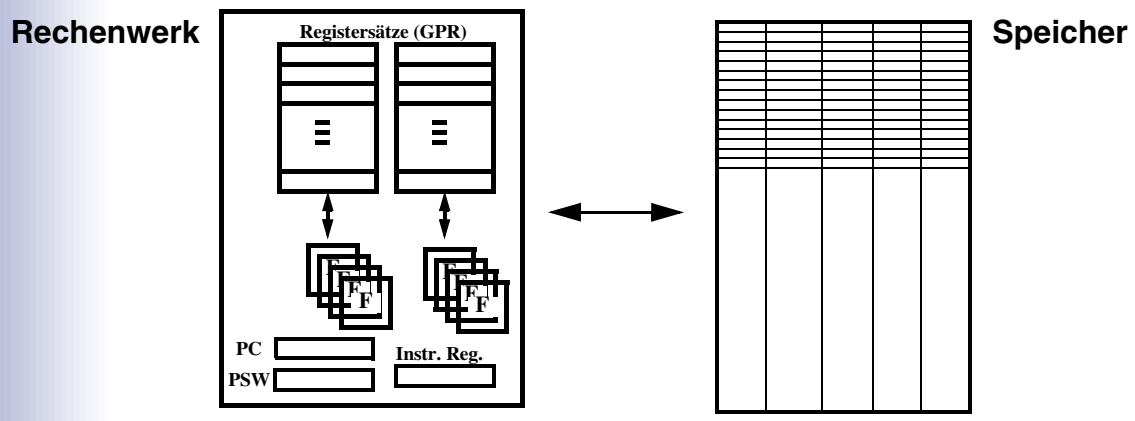
Dual	Dez.	Oktal	Hexa
010110101	= 181	= 265	= 0B5
111001100	= 460	= 714	= 1CC
1010000001	= 641	=1201	= 281

◆ Ein einfacher Addierer (Halbaddierer) ist durch folgende Schaltung realisiert



a	b	s	ü
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

3.5 CPU/Speichermodell



- **PC**: Programcounter (Befehlsadressregister, Befehlszähler);
- **PSW:** Programmstatuswort
- **Instr. Reg.**: Instruction (Befehls-) Register
- **Speichereinheit**: 1 Byte = 8 Bit (in der Regel kleinste adressierbare Einheit)
- **Speicherwort**: in der Regel Standardregisterbreite (16, 32, 64 Bit)

3.6 Programmsteuerung

◆ Wie gewinnt ein Programm Einfluss auf die Steuerung der Hardware?

- das Register “PC” (**Programcounter = Befehlsadressregister**) enthält die Speicheradresse des als nächstes auszuführenden **Maschinenbefehls.**
- Dieser wird aus dem Speicher in des **Befehlsregister (Instruction Register)** geladen.
- Ein “Bitmuster” (Befehlscode) steuert das **Leitwerk** und das **Rechenwerk**.
- Außerdem enthält das Register die notwendigen Informationen über die “**Orte**” (Register- oder Speicheradressen) der zu bearbeitenden Daten.

3.6 Programmsteuerung

◆ Die Java-Code-Zeile *“**a = b + c;**”*

soll von einem Prozessor abgearbeitet werden:

- Die Operation “**+**” wird auf die Inhalte der Speicherzellen “**b**” und “**c**” ausgeführt.
- Das Ergebnis der Operation “**+**” wird in dem Speicherplatz mit der symbolischen Bezeichnung “**a**” abgespeichert.

3.6 Programmsteuerung

◆ Was ist zu tun, damit der Prozessor genau diese Operation ausführt?

- Abbildung der symbolischen Operandenadressen in **phys. Adressen** (Datensegment):
  - Compiler erzeugt ein (relatives) **Maschinenprogramm**
  - Zur Laufzeit lädt die Speicherverwaltung des Betriebssystems das **Maschinenprogramm** in einen **freien Speicherbereich.**
  - Die **Anfangsadresse** des freien Speicherbereichs stellt ein spezielles Register das sog. **Segmentierungsregister** (mit dem Inhalt X2) zur Verfügung.
  - Die **physikalische Adresse** erhält man durch Addition des Registerinhalts (X2) mit der relativen Adresse des Maschinenbefehls bzw. Datums (D2).

3.6 Programmsteuerung

- Das **Segmentierungsregister** hat neben der Bereitstellung der Anfangsadresse des Segments auch noch die Aufgabe den **Zugriffsschutz** und die zulässigen **Zugriffsmodi** sicherzustellen:
  - **Zugriffsmodi**:
    - Lesen
    - Lesen/Schreiben,
    - Ausführen (execute)
  - **Zugriffsschutz**:
    - Daten können nur durch den Programmcode des (der) “zugehörigen” Prozesse(s) (Prozessbegriff siehe Background 1) verändert werden.

3.6 Programmsteuerung

◆ Was ist zu tun, damit der Prozessor genau diese Operation ausführt? (cont)

• Die Abbildung der Java-Anweisung “a = b + c;” in einzelne Maschinenbefehle (Assemblersprache) könnte z. B. wie folgt aussehen:

```
(relativ einfach strukturierte klass. IBM-Assemblersprache)
Operanden vom Typ INTEGER (ganzzahlig):
        ......
        LD      R1,D2(X2)
        LD      R2,D2+4(X2)
        ADD     R1,R2 (Ergebnis in R1)
        BFC     PSW, ERR (z. B. Overflow)
        ST      R1, D2+8(X2)
        ....
ERR:    ....
```

– immer noch symbolische Form, aber prozessorspezifisch

– **Assemblierer**/Codegenerator erzeugt **Binärcode**

– **Ladeobjekt enthält Binärcode und Binärdaten**

3.7 Struktur eines Maschinenbefehls

◆ Beispiel:

```
LD      R1,D2(X2)
```

– Befehlscode: 0...7

– Registeradressen: 8...11; 12...15

– Displacement: 16...31 (ggf.: 32...64)

◆ Betriebssystem lädt Programmcode in Codesegment im Arbeitsspeicher

• Zugriffsmodus: nur “ausführen” (*execute*) erlaubt.

3.8 Speicherwortbreite/Registerbreite

- Einfluss der Speicherwortbreite/Registerbreite:
  - Wertebereiche der darstellbaren Zahlen
  - Genauigkeit der realen (und komplexen) Zahlen
  - Größe des adressierbaren (physikalischen/virtuellen) Speichers
  - Größe der Struktureinheiten der Speicher (Segmente/Pages(Seiten))
  - Größe von Dateisystemen und Länge von Dateien
- Einfluss der Bezeichnung von Datentypen
  - Bestimmt die Auswahl von Registertypen und "Functional Units".