

Grundlagen der Informatik

**für
Hörer aller Fachbereiche
und Mathematiker**

Ulrich Czok

**II. Physikalisches Institut
Justus-Liebig-Universität Gießen**

Grundlagen der Informatik

Czok Stand: January 23, 2004

Inhalt

1	Vorwort	5
2	Datenverarbeitungsanlage	7
2.1	Informationsdarstellung	8
2.1.1	Analoge Informationen	8
2.1.2	Digitale Informationen	9
2.2	Technische Informationsverarbeitung	9
2.3	Elektronik	11
2.3.1	Halbleiter	15
2.4	Zahlensystem	21
2.5	Digitaltechnik	23
2.5.1	digitale Grundbausteine	23
2.5.2	Boole'sche Algebra	24
2.5.3	Schaltnetzwerke	24
2.5.3.1	Schaltungen mit nur einem Eingang	24
2.5.3.2	Verknüpfungen von 2 Eingängen	27
2.5.4	ENCODER	31
2.5.5	DECODER	32
2.5.6	UMCODIERER	33
2.5.7	ADDIERER	34
2.5.8	Subtraktion	36
2.5.8.1	Darstellung negativer Zahlen	37
2.5.8.2	technische Realisierung	39
2.5.9	Inkrementer, Dekrementer	40
3	Speicher	40
3.1	FlipFlop	41
3.1.1	RS-FlipFlop, RS-FF	41
3.1.2	D-FlipFlop, D-FF (Latch, Zwischenspeicher)	42
3.2	Datentransport	42
3.3	Speicherzelle	44
3.3.1	1-Bit-Speicherzelle	44
3.3.2	Register	44
3.4	Memory	44

3.4.1	Zugriff über Adresse	45
3.4.1.1	RAM	46
3.4.1.1.1	SRAM	46
3.4.1.1.2	DRAM	47
3.4.1.2	ROM	49
3.4.1.2.1	ROM, MaskenROM	51
3.4.1.2.2	PROM, Programmierbares ROM	52
3.4.1.2.3	EPROM,EEPROM,EAROM	53
3.4.1.3	dual ported Memory	54
3.4.2	Zugriff über Inhalt, Assoziativspeicher	55
3.4.2.1	Cache	55
4	Input-Output-Port (I/O-Port)	58
4.1	Interface	60
5	Arithmetic and Logic Unit (ALU)	61
6	Ablaufsteuerung	63
6.1	Maschinenprogramm	66
6.2	CISC - RISC	67
6.3	Methoden zur Geschwindigkeitssteigerung	68
6.3.1	Mehrfach-BUS	68
6.3.2	Pipeline	69
6.3.3	Parallelrechner	70
7	Zeitprobleme	70
7.1	Interrupt	73
7.2	Direct Memory Access (DMA, direkter Speicherzugriff)	75
8	Software	77
8.1	Maschinensprache	78
8.2	Höhere Programmiersprachen	80
8.2.1	Erzeugen eines Objektprogrammes	81
9	Massenspeicher	83
9.1	Physikalische Grundlagen	84
9.1.1	Speichermaterial	86
9.1.2	Lesen der Daten	88
9.1.3	FM- und MFM-Modulation	88
9.1.4	Formatierung	90
9.1.5	Kapazität und Speicherformen	91
9.1.6	Zugriffszeiten	91

9.2	optische Speicher	92
9.2.1	ROM, CD	92
9.2.2	beschreibbare CD,PROM	92
9.2.3	wiederbeschreibbare CD	92
10	Index	94

1 Vorwort

Dieses Script soll die Grundlagen der modernen Elektronik näherbringen. Es sollen Grundkenntnisse sowohl der Analog- als auch der Digitaltechnik vermittelt werden. Das wichtigste Ziel ist jedoch, dem Lernenden die inneren Zusammenhänge aufzuzeigen. Dabei soll ein "Gefühl" für die Funktionsweise, die Möglichkeiten und Grenzen entwickelt werden.

Dem Autor ist klar, dass nicht auf Anhieb jeder Leser alles verstehen kann. Auch das Gefühl für die moderne Elektronik kann sich nur allmählich entwickeln. Obwohl nur geringe Vorkenntnisse erwartet werden, wird der Leser mit etwas Erfahrung sich leichter den Stoff aneignen können. Von den Hörern mit weniger Voraussetzungen erhoffe ich mir, dass sie bei weiteren Gelegenheiten, durch das hier Erlernte sich leichter in ähnliche Stoffgebiete einarbeiten können.

Die in diesem Script vorgebrachten Schaltungen und Erläuterungen erheben keinerlei Anspruch auf Funktionalität oder gar Realitätstreue. Sie sind häufig bewusst verändert, um den didaktischen Anforderungen besser gerecht zu werden.

Für den besonders Interessierten finden Sie "zusätzliche Informationen" in einem hervorgehobenen Rahmen.

Gießen, Januar 2004

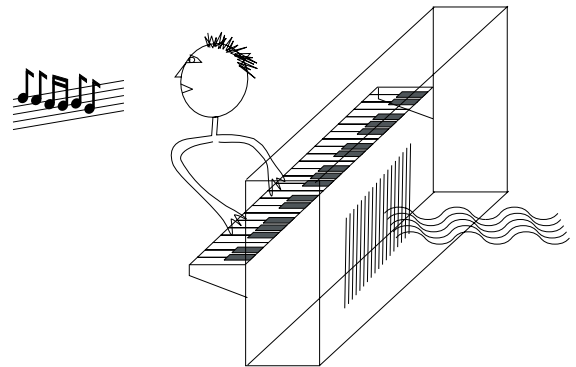
2 Datenverarbeitungsanlage

Definition: Unter dem Begriff "Datenverarbeitungsanlage" (DVA) soll ganz allgemein ein System verstanden werden, in das von Außen über die Eingänge Informationen und Daten eingefüttert werden. Die Daten werden vom System verarbeitet und das Ergebnis (neue Daten) der Verarbeitung über die Ausgänge nach Außen abgegeben.



"Eingang" und "Ausgang" sollen hier als Daten oder Informationen im weitest möglichen Sinn verstanden werden. Prozesssteuerung kann daher auch als Informations- oder Datenverarbeitung bezeichnet werden.

Beispiel: Wir betrachten einen Klavierspieler mit Klavier als Prozesssteuerung. Eine Stimmungslage (Information von Trauer, Liebe, ...) wird dem Zuhörer nahegebracht. Diese Information ist in Form von Noten kodiert. Der Klavierspieler mit dem Sinnesorgan Auge liest die Noten und verarbeitet sie zu Fingerbewegungen. Diese Fingerbewegungen erfasst das System Klavier mit seinen Tasten und bewegt daraufhin Hebel und Hämmerchen, regt Drähte und Saiten zum Vibrieren an. Lufträume und Resonanzkörper beginnen zu schwingen. Am Ausgang erscheinen letztlich Luftdichteschwankungen, die wir als Musik empfinden und so die Stimmungslage nachempfinden.



Wir erkennen also immer das gleiche System. Sinnesorgane (Auge, Klaviertaste) erfassen die kodierten Daten (Noten, Fingerbewegungen) verarbeiten sie und geben das Ergebnis kodiert (Fingerbewegungen, Druckluftschwankungen) heraus. Systeme können zusammengefasst werden (Klavierspieler < – > Klavier) oder komplexe Systeme aufgespalten werden.

Informationen und Daten, die ein System erhält und abgibt, müssen in eine Form gebracht werden, die vom System verstanden werden können. Dazu müssen sie kodiert bzw. mit einem Informationsträger transportiert oder übertragen werden.

Informationsträger, Werkzeuge, Sinnesorgane: Wenn auf dem Klavier ein Lied gespielt werden soll, muss dieses Lied in passender Form an den Eingang des Klaviers gebracht werden. Dazu muss das Lied auf einen Informationsträger (Hand- und Fußbewegungen) aufgeprägt werden. Das Klavier empfängt mit seinen Sinnesorganen (Tasten und Pedale) die Information von den Händen und Füßen des Klavierspielers und verarbeitet diese Informationen mit seinen Werkzeugen (Hebel, Hämmerchen, Saiten, ...). Am Ausgang erzeugt es Musik. Diese Musik prägt das Klavier wiederum auf einen Informationsträger (Druckschwankungen der Luft). Die Information Musik gelangt über den Informationsträger Luftdruckschwankungen zum System Mensch, der mit seinen Sinnesorganen (Ohr) die Information entnehmen kann und sie letztlich zum Gehirn transportiert, wo sie z.B. zu Wohlbefinden oder auch zu Belästigung weiterverarbeitet wird.

2.1 Informationsdarstellung

Wir wollen technische Datenverarbeitung betreiben. Daher benötigen wir eine technische Informationsdarstellung. Wir unterscheiden zwei unterschiedliche Arten von Informationen: Analoge Informationen und Digitale Informationen.

2.1.1 Analoge Informationen

”Analog” heißt ”entsprechend” und meint, dass zu jeder Information eine andere entsprechende zugeordnet werden kann. Im allgemeinen kann einer messbaren Größe eine Zahl zugeordnet werden und zwei messbare Größen können miteinander verglichen werden.

Beispiele:

- Der Temperatur wird die Längenausdehnung einer Quecksilbersäule zugeordnet.
- Der Geschwindigkeit des Autos wird der Winkel eines Zeigers am Tacho zugeordnet.
- Der Lautstärke eines Tons am Klavier wird die Heftigkeit des Anschlags zugeordnet.

Kennzeichen der analogen Information ist ein **kontinuierlicher Wertebereich**: Zwischen zwei Werten kann immer ein weiterer Wert liegen. Wir sagen auch, die Werte liegen dicht.

Auch die reellen Zahlen liegen dicht auf dem Zahlenstrahl, und sind daher auch gut zur Darstellung analoger Informationen geeignet.

Wir Menschen benutzen gerne analoge Informationsträger, wo es um eine schnelle Zuordnung geht. Mit unserem besten Sinnesorgan, dem Auge, können wir sehr schnell eine Länge oder einen Winkel (eventuell sogar mit Skala) erfassen. Für manche Größen besitzen wir nur schlechte (Temperatur, Druck..) oder gar keine Sinnesorgane (elektrische Spannung, Magnetfeld..). Hier benutzen wir vorteilhaft eine entsprechende (analoge) Zuordnung zwischen der Information (Temperatur, Druck, Spannung..) und einer für das Auge passender Information (Längenausdehnung, Winkelstellung).

2.1.2 Digitale Informationen

Die Bezeichnung "digital" leitet sich von "digitus" dem Finger ab. Digitale Informationen sind (mit den Fingern) abzählbar. Kennzeichen der digitalen Informationen sind **diskrete Werte**. Zwischenwerte gibt es nicht.

Beispiele:

- Buchstaben des Alphabets, einen Buchstaben zwischen g und h gibt es nicht
- Tasten am Klavier, einen Ton zwischen zwei Tasten gibt es nicht

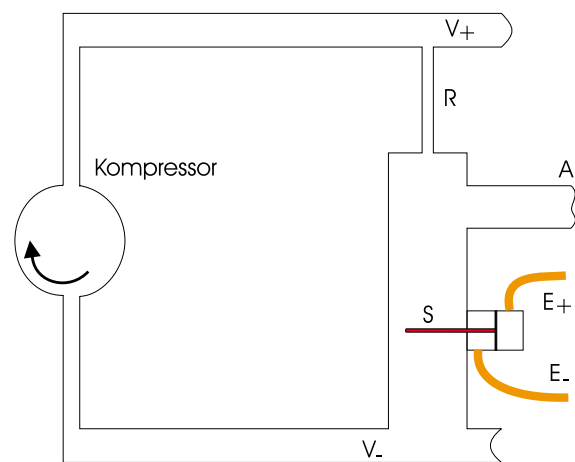
Auch die natürlichen Zahlen bilden diskrete Werte auf dem Zahlenstrahl. Die unterschiedlichen Werte einer digitalen Informationen lassen sich in einer Reihenfolge anordnen und durchnummerieren und so den natürlichen Zahlen zuordnen.

2.2 Technische Informationsverarbeitung

In der technischen Informationsverarbeitung ist es wichtig, dass immer passende Informationsträger und informationsverarbeitende Werkzeuge zur Verfügung stehen. Es muss die Information so aufbereitet und auf einen Informationsträger aufgeprägt werden, daß es passende Werkzeuge und "technische Sinnesorgane" gibt, die in der Lage sind, die Informationen von dem Träger zu entnehmen, sie mit anderen Informationen zu verknüpfen und das Ergebnis wiederum einem technisch lesbaren Informationsträger aufzuprägen.

Das Bild zeigt ein Beispiel zur Realisierung einer Datenverarbeitungsanlage. Das System hat zwei Eingänge E_+ und E_- (Sinnesorgane die auf Druck reagieren) und einen Ausgang A (eine Membran die mehr oder weniger ausgebeult ist).

Ein Kompressor erzeugt einen hohen Versorgungsdruck V_+ , der einen Gas- oder Flüssigkeitsstrom durch ein enges Rohr R presst. Je nach Stellung des Schiebers S wird sich an A ein hoher bzw. niedriger Druck einstellen.

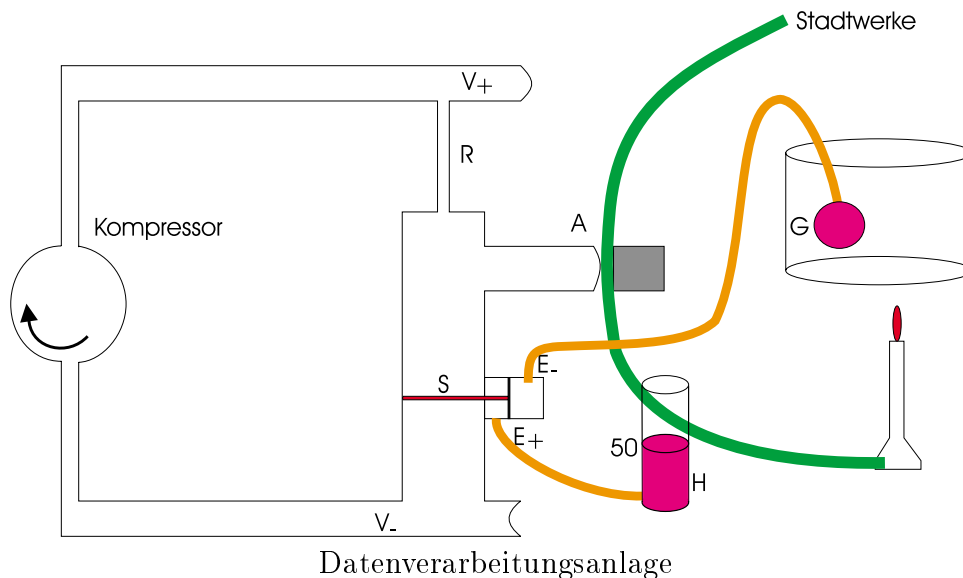


Funktion:

$$\begin{array}{lll}
 E_+ > E_- \implies & A \approx V_+ & A \implies \supset \\
 E_+ < E_- \implies & A \approx V_- & A \implies \subset \\
 I_+ = I_- = 0 & & \text{Eingänge sind dicht} \\
 V_- \leq A \leq V_+ & & \text{Aussteuergrenzen}
 \end{array}$$

In der Elektronik nennt man eine Baugruppe mit diesen Eigenschaften Operationsverstärker (Opamp, operational Amplifier)

Die Wahl von Informationsträger und passendem Werkzeug ist ansonsten nur von den Möglichkeiten oder den Anforderungen abhängig. Im folgenden sollen einige Möglichkeiten angerissen werden.



Das Bild zeigt ein Beispiel aus der Analogtechnik. Aufgabe ist die Temperaturstabilisierung eines Wasserbades. In das Wasserbad reicht ein luftgefüllter Glaskolben G. Der Information "Temperatur des Wasserbades" wird im Glaskolben eine

entsprechende Luftdruck-Information zugeordnet. Der Glaskolben ist ein Sinnesorgan, das die Information Temperatur in die entsprechende Information "Druck" umwandeln kann.

In einem zweiten mit Wasser gefüllten Kolben H erzeugen wir durch Anheben oder Absenken des Wasserstandes einen zweiten Druck, dessen Wert sich nach der Höhe des Wasserstandes richtet. Unsere DVA (Datenverarbeitungsanlage) besteht aus einem Zylinder mit Schieber S. Die Anlage wird von einem Kompressor, der einen Druckunterschied $V_+ \cdots V_-$ erzeugt, gespeist. Bevor die Luft in den Zylinder mit dem Schieber gelangt, muss sie sich durch ein dünnes Röhrchen R quetschen. Die DVA verknüpft die beiden Informationen (Temperatur des Wasserbades und Wasserstand im Kolben H) miteinander: Ist die Temperatur zu niedrig und der Druck im Wasserbadkolben G geringer als der im zweiten Kolben H, so wird der Schieber aufgeschoben und die Luft im Zylinder kann nach unten entweichen. Dadurch geht die Ausbeulung am Ausgang A zurück.

Ist andererseits die Temperatur zu hoch und der Druck im Wasserbadkolben höher als der im zweiten Kolben, so wird der Schieber zugeschoben, und es baut sich über das dünne Röhrchen R in dem Zylinder ein hoher Druck auf und beult die Membran nach außen aus.

Diese DVA erhält am Eingang Informationen in Form von Druckschwankungen, verknüpft diese miteinander und gibt eine Information in Form eines Druckes (oder Ausbeulung einer Membran) nach Außen ab.

Mit der Ausbeulung der Membran lässt sich der Gasstrom in den Brenner steuern und damit die Temperatur des Wasserbades regulieren.

Das Ganze arbeitet als Temperaturregulierung des Wasserbades: Eine Solltemperatur wird durch die Höhe des zweiten Kolbens vorgegeben. Ist die Isttemperatur (dargestellt durch den Luftdruck am Eingang E_+) größer als die Solltemperatur (dargestellt durch die Höhe des zweiten Kolbens und damit des Luftdrucks am Eingang E_-), so wird die Membran am Ausgang A ausgebeult, damit der Gaszufluss abgeschnürt und die Heizung des Wasserbades gedrosselt; Ist umgekehrt die Solltemperatur größer als die Isttemperatur, so verschwindet die Beule am Ausgang, das Gas strömt zum Brenner und das Wasserbad wird aufgeheizt.

2.3 Elektronik

Wie man an dem Beispiel oben sieht, lässt sich Datenverarbeitung mit unterschiedlichen Medien durchführen. Pneumatik bzw. Hydraulik findet sich an vielen Stellen (im Auto, bei Kränen, ...). Das Beispiel soll zum einen die Elektronik anschaulich machen und andererseits demonstrieren, dass Elektronik nicht der einzige Weg ist, bzw. dass in Zukunft auch Datenverarbeitung mit anderen Medien möglich ist.

In der unten folgenden Tabelle sollen Bauelemente und physikalische Größen aus der Hydraulik und der Elektronik verglichen werden. Zuerst möchte ich aber einige Begriffe erläutern.

Widerstand

Ein Widerstand entspricht einem dünnen Rohr, das einen Durchfluss behindert. Die Behinderung (Widerstand R) hängt von der Länge L und der Querschnittsfläche F des Rohres ab. Im elektrischen Fall kommen zusätzliche Materialeigenschaften ρ (Kupfer, Eisen, ...) hinzu.

$$R = \rho \frac{L}{F}$$

Wie ein Druck eine Wassermenge durch ein Rohr presst, drückt eine elektrische Spannung U eine elektrische Ladung Q durch einen Widerstand R . Die pro sehr kurze Zeiteinheit dt fließende sehr kleine Ladungsmenge dQ nennt man den Strom I .

$$I = \frac{dQ}{dt}$$

Strom, Spannung und Widerstand sind durch das Ohm'sche Gesetz verknüpft

$$U = R \cdot I$$

!!! Versuchen Sie sich diese Zusammenhänge anschaulich klar zu machen!!!

Kondensator

Ein Kondensator entspricht einem Behälter mit einem Fassungsvermögen C (Kapazität). Der Ladungsinhalt Q ergibt sich aus der Kapazität und der angelegten Spannung

$$Q = C \cdot U$$

R·C-Zeit

Will man eine Ladung in einem Kondensator aufbewahren, so muss man damit rechnen, dass über unvermeidliche Lecks allmählich Ladung entweicht.

Andererseits dauert es eine gewisse Zeit, bis ein Kondensator über einen Widerstand geladen wird.

Ein Maß für diese Zeit ist $R \cdot C$, die sogenannte $R \cdot C$ -Zeit. Die $R \cdot C$ -Zeit gibt an, wie lange es dauert, bis sich die Ladung im Kondensator bis auf den e -ten Teil ($e \approx 2,7$) dem Endwert angenähert hat.

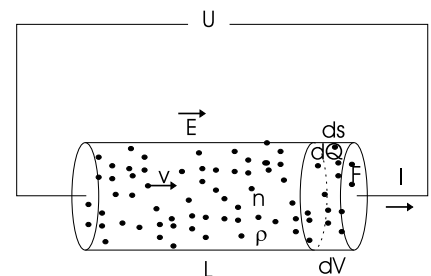
Für große Rechenleistung (hohe Rechengeschwindigkeit) sind kurze RC -Zeiten erforderlich, was leider kleine Widerstände und damit hohen Stromverbrauch mit sich bringt. Benutzt man einen Kondensator als Speicher sind lange RC -Zeiten wünschenswert !

Leiter	meist Metalle	Material mit hoher Dichte von freien Ladungsträgern
Nichtleiter	z.B. Glas, Quartz (SiO ₂)...	Material ohne freie Ladungsträger
Halbleiter	häufig Elemente aus der 4. Gruppe (C,Si,Ge)	Material mit wenigen freien Ladungsträgern, die durch Temperaturbewegungen aus dem Kristall- verband gelöst werden

zusätzliche Informationen

Leitfähigkeit

In einem Draht der Länge L und der Querschnittsfläche F liegt eine Spannung U . Daraus ergibt sich im Draht eine elektrische Feldstärke E
 $E = U/L$



Im Draht sind frei bewegliche Ladungsträger der Ladung q mit einer Dichte n (n = Anzahl der Ladungsträger pro Volumen). Auf diese Ladungsträger wirkt durch das elektrische Feld eine Kraft, die die Ladungsträger auf die Geschwindigkeit \vec{v} beschleunigen. Die Geschwindigkeit ist proportional zur Feldstärke: $v = \mu E$. μ nennt man die Beweglichkeit.

Für den Strom I gilt $I = dQ/dt$. dQ ist die Ladung der Ladungsträger in einer Scheibe (Querschnitt F , Dicke ds), die während der Zeit dt den Draht verlassen.

$$dQ = n \cdot dV = n \cdot (F ds)$$

$$I = dQ/dt = nF ds/dt = nFv \quad v = ds/dt \text{ ist die Geschwindigkeit der Ladungsträger}$$

$$I = nF\mu E = nF\mu U/L$$

Daraus ergibt sich das **Ohm'sche Gesetz** $U = R \cdot I$:

$$U = \frac{1}{n\mu} \cdot \frac{L}{F} \cdot I \quad \text{mit } R = \frac{1}{n\mu} \cdot \frac{L}{F}$$

Daraus ergibt sich für den Widerstand R :

$$R = \frac{1}{n\mu} \cdot \frac{L}{F} = \sigma \frac{L}{F} \quad (\sigma = \text{spezifische Leitfähigkeit; } \rho = 1/\sigma = \text{spez. Widerstand})$$

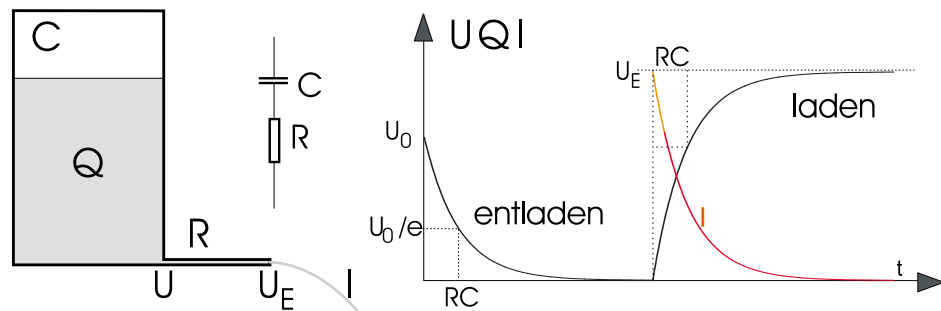
Die **Leitfähigkeit** ergibt sich damit zu $\sigma = n\mu$

Bei **Halbleitern** nimmt die Leitfähigkeit mit Temperaturerhöhung zu, da durch Temperaturbewegungen freie Ladungsträger erzeugt werden. n steigt mit T .

Bei **Metallen** sinkt die Leitfähigkeit mit der Temperatur, da durch Temperaturerhöhung die Zusammenstöße der Ladungsträger mit den Gitteratomen zunehmen und damit die Beweglichkeit μ abnimmt

zusätzliche Informationen

Laden, Entladen und Umladen eines Kondensators



Um den Kondensator C auf den Endwert U_E zu laden, legt man an das Ende des Widerstandes R diese Spannung U_E an. Die Spannung am Kondensator sei U , dann gilt :

Ohm'sches Gesetz (Spannung an $R=U-U_E$) (1) $U - U_E = R \cdot I$

Q =Ladung des Kondensators (2) $Q = C \cdot U$

Der Strom I ändert die Ladung des Kondensators (3) $I = dQ/dt$

(3) in (2) eingesetzt : (4) $I = C \cdot dU/dt$

(4) in (1) eingesetzt liefert die Differentialgleichung : (5) $U_E - U = RC \cdot dU/dt$

die Lösung (wie man durch einsetzen in(5) überprüfen kann) ist : (6) $U = U_E + U_1 \cdot e^{-\frac{t}{RC}}$

U_1 ergibt sich aus den Anfangsbedingungen (U zum Zeitpunkt $t=0$). Z.B.

Spezialfall Entladen (linke Kurve):

$$U(t=0) = U_0 \quad U_E = 0$$

$$U(t=0) = U_0 = U_E + U_1 \cdot e^{-\frac{t}{RC}}$$

$$U_0 = U_1$$

$$U_1 = U_0$$

$$U = U_0 \cdot e^{-\frac{t}{RC}}$$

$$\boxed{U = U_0 \cdot e^{-\frac{t}{RC}}}$$

Spezialfall Aufladen (rechte Kurve):

$$U(t=0) = 0 \quad U_E \neq 0$$

$$U(t=0) = 0 = U_E + U_1 \cdot e^{-\frac{t}{RC}}$$

$$0 = U_E + U_1$$

$$U_1 = -U_E$$

$$U = U_E - U_E \cdot e^{-\frac{t}{RC}}$$

$$\boxed{U = U_E \cdot (1 - e^{-\frac{t}{RC}})}$$

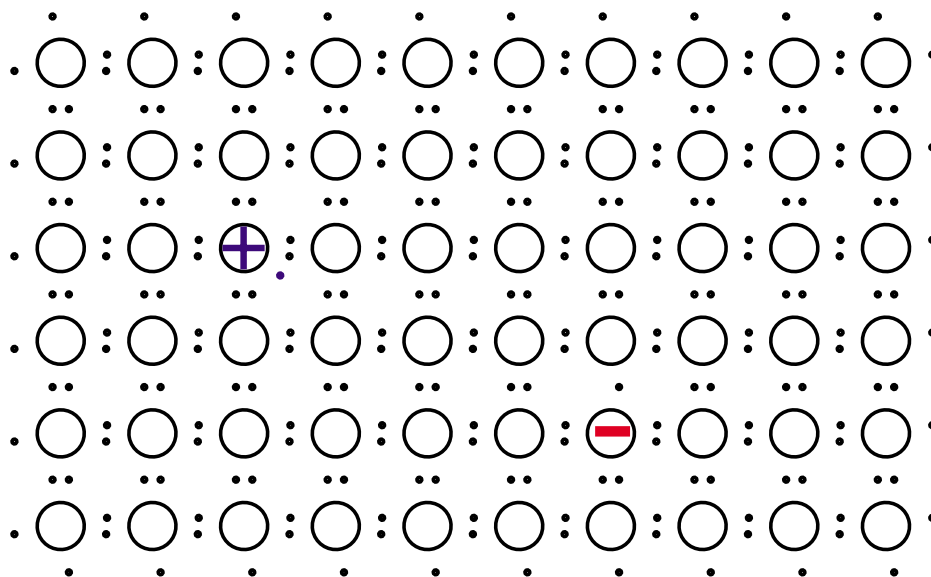
$R \cdot C$ gibt die Zeit an, in der die Spannung sich bis auf $1/e$ ($\approx 1/3$) dem Endwert angenähert hat (RC-Zeit).

Bis zur Annäherung auf $1/10$ (1%) dauert es also $2 \cdot RC$ ($4 \cdot RC$)

Die Gleichungen (1) und (2) zeigen, dass Q und I (I beim Laden, $U_E=0$) den gleichen Verlauf haben, wie U

2.3.1 Halbleiter

Ein Halbleiter besteht meist aus einem Kristallverband von Atomen aus der 4. Reihe des Periodensystems (Kohlenstoff C, **Silizium, Si** oder Germanium). Diese vierwertigen Elemente haben in ihrer äußeren Schale - die insgesamt 8 Elektronen aufnehmen kann - 4 Elektronen. Im Kristallverband teilen sich jeweils 5 Atome eine Schale, so dass ein Atom 4 Elektronen und 4 benachbarte Atome jeweils ein Elektron beitragen. Im Bild unten ist die dreidimensionale Gitterstruktur auf eine Ebene abgebildet.



In dem Gitter tauscht man einige sehr wenige Gitterplätze durch **Dotierung**, Zugabe von 3-wertigen Atomen mit nur 3 Außenelektronen (bzw. 5-wertigen mit 5 Außenelektronen) aus. Insgesamt ist der Kristall elektrisch neutral, da jedes Atom gleichviel positive Protonen im Kern wie negative Elektronen in der Hülle besitzt. Jedoch befinden sich in der Hülle des zugegebenen Atoms entweder 9 Elektronen (blau markiert) bzw. 7 Elektronen (rot).

Durch Diffusion (Temperaturbewegungen, denken Sie an die Brownsche Molekularbewegung) kann sich leicht das 9. Elektron auf ein Nachbaratom überspringen und von dort zum nächsten usw. Dadurch entsteht ein **frei bewegliches Elektron**. Das dicke Restatom kann sich nicht von seinem Platz bewegen.

Man erhält : **n-dotiertes Material**

frei bewegliche negative Ladungen und ortsfeste positive Atomrümpfe.

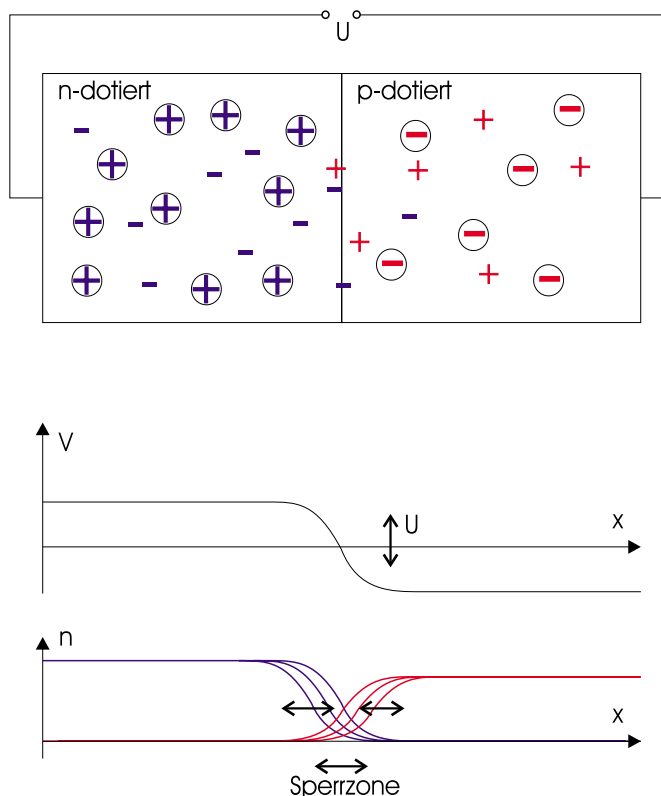
Ähnlich kann auf den freien 8. Platz ein Elektron von einem benachbarten Atom springen, und in den nun entstandenen freien Platz ein Elektron des nächsten Nachbarn usw. Dadurch entsteht eine **frei bewegliche Elektronenfehlstelle**. Wieder kann sich das dicke Restatom nicht von seinem Platz bewegen.

Man erhält : **p-dotiertes Material**

frei bewegliche positive Ladungen und ortsfeste negative Atomrümpfe.

Bringt man p-dotiertes und n-dotiertes Material zusammen, so hat man zunächst auf den beiden Seiten je eine gleichmäßige Verteilung n von positiven bzw. negativen frei beweglichen Ladungsträgern. Die beiden Teile sind elektrisch neutral. Der Potentialunterschied V (Spannung zwischen den beiden Teilen) ist null !

Durch Diffusion gelangen jedoch frei bewegliche negative Elektronen in das p-dotierte und positive Elektronenfehlstellen in das n-dotierte Material. Außerdem können noch Elektronen in Elektronenfehlstellen fallen, wodurch sowohl das Elektron wie auch die Fehlstelle verschwindet. Konsequenz ist, dass im n-dotierten Material insgesamt mehr positive Ladungen vorhanden sind und das Material positiv geladen wird ($V > 0$). Ebenso wird das p-dotierte Material negativ geladen ($V < 0$).



Da sich gleichnamige Ladungen abstoßen und ungleichnamige anziehen, werden die negativen freien Elektronen (blau) nach links und positiven Fehlstellen nach rechts getrieben. Dabei stellt sich ein Gleichgewicht zwischen den elektrostatischen Kräften

(treiben die Ladungen nach außen) und den Kräften, die von thermischen Bewegungen herrühren (treiben die Ladungen in alle Richtungen) ein.

Als Resultat entsteht ein Potentialunterschied zwischen den beiden Bereichen sowie eine an freibeweglichen Ladungsträgern arme Zone im Übergangsbereich zwischen den beiden Gebieten. Da zur Stromleitung aber freibewegliche Ladungsträger vorhanden sein müssen, ergibt sich eine den Strom nicht leitende **Sperrzone**.

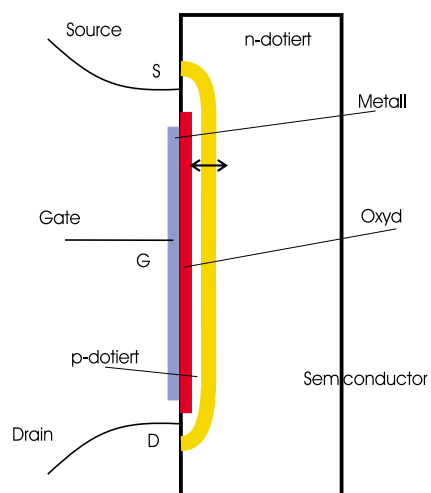
Durch Anlegen einer Spannung U werden – je nach Polung - Ladungsträger in die Sperrzone gedrückt (die Sperrzone wird kleiner) oder weitere aus der Sperrzone herausgezogen (Sperrzone wird breiter).

Die Breite der Sperrzone kann durch die angelegte Spannung U beeinflusst werden !

Da - je nach Polarität der Spannung - freibewegliche Ladungsträger in der Sperrzone vorhanden sind oder nicht, kann auch Strom durch diese Zone fließen oder nicht. Diese Anordnung verhält sich daher wie ein Ventil. In der Elektronik nennt man dieses Bauteil : **Diode**.

Der Feldeffekttransistor, FET

Unser Ziel ist es, ein elektronisches Bauteil zu finden, das, wie der Schieber einen Wasserstrom, einen elektrischen Strom einschnüren kann. Dazu nutzen wir die Möglichkeit, die Breite einer Sperrzone durch eine elektrische Spannung zu variieren. Die (gelb markierte) Sperrzone trennt einen p-dotierten (leitenden) **Kanal** von dem n-dotierten Grundmaterial ab. An den Enden des Kanals befinden sich zwei Anschlüsse S (**Source**, Quelle) und D (**Drain**, Abfluss). Der Widerstand des Kanals hängt von dessen Länge und Breite ab. Die Breite wiederum lässt sich durch die Breite der Sperrzone verstellen. Dazu dient der Anschluss G (**Gate**, Gatter oder Tor). Eine Spannung zwischen Gate und Grundmaterial erzeugt ein elektrisches Feld, das Ladungen in oder aus der Sperrzone treibt, die Breite der Sperrzone und damit gegenläufig die des Kanals verändert.



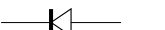


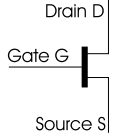
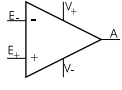
Damit kein Strom vom Gate (Metallfläche, blau markiert) in den Kanal fließen kann, ist das Gate mit einem Isolator (Siliziumdioxid, SiO_2 , Quarz, "Glas", rot markiert) abgetrennt. Das SiO_2 lässt sich leicht durch Oxydation der Siliziumoberfläche realisieren.

Als Resultat erhält man einen Durchgang (Transition) durch einen Kanal von Source nach Drain, dessen Breite durch ein elektrisches Feld (FeldEffekt) gesteuert wird (**FET** FeldEffektTransistor).

Es gibt viele unterschiedliche FETs. Den hier vorgestellten nennt man wegen des Aufbaus Metall-Oxyd-Semiconductor : **MOSFET**.

Natürlich lassen sich die Dotierungen auch vertauschen, und man erhält neben dem hier gezeichneten p-Kanal-MOSFET auch den n-Kanal-MOSFET.

Pneumatik, Hydraulik	Elektronik	Bezeichnung	Einheit, Beziehung
verwendet Luft-, Wasser- oder Ölmoleküle. Viele Moleküle bilden eine Luft- Wasser- oder Ölmenge	verwendet Elektronen. Viele Elektronen bilden eine Ladung	Q	Coulomb [Cb]
Ein Kompressor erzeugt einen Druck	Eine Batterie erzeugt eine Spannung U	U	Volt [V]
Ein Druck kann eine Menge durch, ein Rohr drücken	Eine Spannung kann eine Ladung durch einen Draht drücken		
Eine Wassermenge, die pro Zeiteinheit fließt nennt man Wasserstrom	Eine Ladung, die pro Zeiteinheit t fließt nennt man Strom I	I	Ampere [A] $Q = \int I dt$ $I = dQ/dt$
Bauelement	Widerstand	Symbol R	
Ein enges langes Rohr setzt dem Wasserstrom einen Widerstand entgegen	Ein dünner (Querschnitt q) langer (Länge L) Draht (Material ρ) setzt dem Strom einen Widerstand R entgegen	R	$R = \rho \frac{L}{q}$
Widerstand	Widerstand	R	Ohm [Ω]
Ein Druck presst einen Wasserstrom durch ein Rohr	Eine Spannung drückt einen Strom durch einen Widerstand	Ohmsches Gesetz	$U = RI$
Bauelement	Speicher	Symbol C	
Eine Wassermenge kann man in einem Behälter aufbewahren	Eine Ladung kann man in einem Kondensator C aufbewahren		
Ein Behälter hat ein Fassungsvermögen	Ein Kondensator hat eine Kapazität	C	Farad [F]
Die Menge im Behälter kann durch Druck gesteigert werden	Die Ladung im Kondensator wird durch die Spannung erhöht		$Q = CU$
RC-Zeit : Ein Behälter (Kondensator Kapazität C) wird über ein Rohr (Widerstand R) umgeladen. In der Zeit $R \cdot C$ hat sich die Ladung bis auf den e-ten Teil ($e=2,7$) an den Endwert angenähert. Wichtig: 1. Die Rechengeschwindigkeit wird durch die RC-Zeit begrenzt (langsameres Umladen). 2. gespeicherte Information geht verloren			
Bauelement	Diode, Ventil	Symbol	
Durch ein Ventil kann ein Wasserstrom nur in einer Richtung fließen	Durch eine Diode kann ein Strom nur in einer Richtung fließen		

Pneumatik, Hydraulik	Elektronik	Bezeichnung	Einheit, Beziehung
Bauelement	Feldeffekttransistor (FET)	Symbol	
Durch einen mit Druck gesteuerten Schieber kann die Enge eines Rohres und damit der Widerstand gesteuert werden	In einem Feldeffekttransistor FET kann die Breite eines leitfähigen Kanals (und damit der Widerstand von Source S nach Drain D) durch Spannung am Gate G gesteuert werden		
Bauelement	Operationsverstärker (Opamp)	Symbol	
Der Druckunterschied $E_+ - E_-$ zwischen den beiden Eingängen bestimmt den Ausgangsdruck	Der Spannungsunterschied $E_+ - E_-$ zwischen den beiden Eingängen bestimmt die Ausgangsspannung	$E_+ > E_- \Rightarrow A \approx V_+$ $E_+ < E_- \Rightarrow A \approx V_-$ $I_- = I_+ = 0$ $V_- < A < V_+$	

2.4 Zahlensystem

Die Darstellung digitaler Informationen erfordert die Abbildung der Information auf die natürlichen Zahlen (1, 2, 3, 4...) Wir haben uns von Kind auf an das Dezimalsystem gewöhnt, haben das kleine Einmaleins gelernt und vieles mehr.

Das Zehnersystem ist kein speziell hervorgehobenes System, außer dadurch, dass der Mensch zehn Finger hat. Auch andere Zahlensysteme waren oder sind im Einsatz. In jedem Zahlensystem gibt es ähnliche Regeln wie im Dezimalsystem, z.B. Zählen, Addieren, Multiplizieren, Teilbarkeitsregeln...

An das 12-System erinnert die Einteilung in Dutzend, Gros, Schock oder die Zeiteinheiten:

60 Sekunden	=	1 Minute
60 Minuten	=	1 Stunde
24 Stunden	=	1 Tag
12 Monate	=	1 Jahr

Unsere Erbinformation ist als Folge von Basensequenzen in der DNS codiert. Hier sind vier verschiedene Basen möglich : Adenin (A), Guanin (G), Cytosin (C) und Uracil (U). Die DNS benutzt also ein 4-System. Da aber immer ein Triplet von 3 Basen abgelesen werden (z.B. ACG) entsteht ein 64-System ($64 = 4 \cdot 4 \cdot 4$)

Von den 64 möglichen Kombinationen können jedoch beim Ablesen nur 20 unterschieden werden, so dass sich letztlich ein 20-System ergibt.

Für die Datenverarbeitung ist es sinnvoll, ein möglichst günstiges Zahlensystem auszuwählen. Das 2-System (**Binär- oder Dualsystem**) hat eine Reihe von Vorteilen. Der wichtigste ist, dass zur Realisierung einer Stelle lediglich **einfache** Bauelemente notwendig sind, die nur 2 Zustände annehmen müssen. Die meisten physikalischen Größen lassen sich in 2 Zustände versetzen. Z.B. kann ein Druck, eine Spannung, ein Strom, eine Ladung, ein Widerstand, ... groß oder klein (hoch - niedrig, high - low, H - L) sein.

Ein weiterer Vorteil des Dualsystems ist, daß schon Mitte des 19. Jahrhunderts der englische Mathematiker George Boole eine Algebra entwickelt hat, die sich mit Verknüpfungen von Aussagen beschäftigt (**Boole'sche Algebra**). Aussagen können zwei Zustände annehmen : wahr und falsch (true - false, T - F). Somit lassen sich die Zustände H und L einer physikalischen Größe den Werten T und F der Boole'schen Algebra zuordnen und so die Gesetze und Regeln der Boole'schen Algebra nutzen.

Dezimalsystem 10-System

10 Ziffern (0, 1, 2, 3, ... 9)

Wert der Zahl ergibt sich aus der Summe von Ziffernwert * Wertigkeit der einzelnen Ziffern

Die Wertigkeit ergibt sich aus der Position der Ziffer und Position der Ziffer in der Zahl

Der Wert der rechten Position ist 1 und steigt von Position zu Position nach links um den Faktor **10**

z.B. :

$$4793 = 4 * 1000 + 7 * 100 + 9 * 10 + 3 * 1$$

$$= 4 * 10^3 + 7 * 10^2 + 9 * 10^1 + 3 * 10^0$$

Jede Ziffer bezeichnet man mit **digit**

Anzahl der unterschiedlich möglichen Werte :

1 digit	0..9	10	10 ¹
2 digits	00..99	100	10 ²
3 digits	000..999	1000	10 ³
n digits			10 ⁿ

Zählen im **10**-System : Sobald alle Ziffern in einer Stelle aufgebraucht sind, wird die nächsthöherwertige Stelle um 1 erhöht

- 0
- 1
- 2
- ⋮
- 9
- 10
- ⋮
- 19
- 20
- ⋮
- 99
- 100
- ⋮

Zusammenfassen von digits

2	3	7	6	8	9	5	1
		00					
		99					

Jeder Block aus 2 Ziffern kann 100 Werte (00..99) annehmen
Es ergibt sich also ein 100-System

Faustformel :

$$3 \text{ digits} \Leftrightarrow 10^3 = 1000 \approx 1024 = 2^{10} \Leftrightarrow 10 \text{ bits}$$

Dualsystem 2-System

2 Ziffern (0, 1)

Wert der Zahl ergibt sich aus der Summe von Ziffernwert * Wertigkeit der einzelnen Ziffern

Die Wertigkeit ergibt sich aus der Position der Ziffer in der Zahl

Der Wert der rechten Position ist 1 und steigt von Position zu Position nach links um den Faktor **2**

z.B. :

$$10110 = 1 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 0 * 1$$

$$= 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$

Jede Ziffer ist ein digit im Binärsystem und wird mit binary digit bezeichnet **bit**

Anzahl der unterschiedlich möglichen Werte :

1 bit	0..1	2	2 ¹	0000	0	0
2 bits	00..11	4	2 ²	0001	1	1
3 bits	000..111	8	2 ³	0010	2	2
n bits			2 ⁿ	0011	3	3
				0100	4	4
				0101	5	5
				0110	6	6
				0111	7	7
				1000	8	8
				1001	9	9
				1010	10	A
				1011	11	B
				1100	12	C
				1101	13	D
				1110	14	E
				1111	15	F

Zusammenfassen von bits

1	0	1	1	0	0	1	0	1	0	1	1
5				4				5			3
B				2				B			

Blöcke aus 3 bits (4 bits) werden zu einer 2³ : 8-Stelle (2⁴ : 16-Stelle) zusammengefasst.

Es ergibt sich das Oktal- (Hexadezimal-) System
Als Ziffernsymbole benutzt man A,B,C,D,E,F für die Zahlen 10 bis 15

byte : Unter anderem für Text verwendet man häufig eine Anordnung von 8bit = 1 byte. Mit einem byte sind 2⁸ = 256 unterschiedliche Werte zu beschreiben.

2.5 Digitaltechnik

Wie schon beschrieben lassen sich digitale Informationen auf drei unterschiedliche Weisen beschreiben:

(0 - 1) als bit aus dem Binärsystem

(H - L) als reale physikalische Größe und mit

(T - F) als Zustand einer logischen Aussage

Entsprechend lassen sich auch digitale Funktionen unterschiedlich beschreiben:

- Aus der begrenzten Anzahl von möglichen Zuständen ergibt sich die Möglichkeit alle Zustände tabellarisch aufzulisten. Dies geschieht in Form einer sogenannten **Wahrheitstabelle**
- Die physikalische Realisierung erfolgt mit Hilfe von Anordnungen von Baugruppen, die durch Schaltpläne mit **Schaltsymbolen** dargestellt werden können.
- **logische Gleichungen** erlauben den Einsatz von Regeln aus der Boole'schen Algebra. Dadurch lassen Funktionen beschreiben und Schaltungen verändern.

2.5.1 digitale Grundbausteine

In der Digitaltechnik müssen Baugruppen eingesetzt werden, die in zwei unterscheidbare Zustände gebracht werden können, um die zwei Ziffern 0 und 1 des Binärsystems zu repräsentieren. Weiterhin müssen diese Baugruppen durch die Information steuerbar sein.

Die Baugruppen aus der Analogtechnik erfüllen bereits diesen Zweck. In der Hydraulik lässt sich wieder ein mit einem Schieber abschnürbares Rohr (in der Elektronik ein Feldeffektransistor) verwenden. Damit lassen sich zwei Druckzustände (hoher Druck = High = H; niedriger Druck = Low = L) einstellen. Auch ist der Schieber mit H und L steuerbar.

Den beiden Zuständen H und L können wir beliebig die beiden Ziffern 0 und 1 des Binärsystems zuordnen.

2.5.2 Boole'sche Algebra

Mitte des 19. Jahrhunderts entwickelte der englische Mathematiker George Boole eine Algebra, die sich mit Aussagen beschäftigt und stellte Rechenregeln für logische Aussagen auf.

In der Boole'schen Algebra können Aussagen zwei Werte annehmen:

Aussage ist wahr W TRUE T

Aussage ist falsch F FALSE F

Daher ist die Boole'sche Algebra hervorragend geeignet, die zwei Zustände 0 und 1 eines bits zu behandeln.

Die Zuordnung zwischen den Werten T und F der Boole'schen Algebra, den Werten 0 und 1 eines bits und den Aus- und Eingängen der technischen Geräte L und H ist rein willkürlich und muss gegebenenfalls vereinbart werden.

Wir bezeichnen die Zuordnung	und die Zuordnung
H 1 T als positive Logik	H 0 F als negative Logik
L 0 F	L 1 T

Wahrheitstafel Betrachtet man ein digitales System mit seinen i Eingängen E_i und k Ausgängen A_k , so gibt es eine begrenzte Zahl von Zuständen. Mit i Eingängen können maximal 2^i verschiedene Zustände angelegt werden. Das Verhalten der Ausgänge in Abhängigkeit der Zustände der Eingänge kann man übersichtlich in Form einer Tabelle darstellen. Diese Wahrheitstabelle oder Wahrheitstafel enthält maximal 2^i Zeilen für die einzelnen Zustände, sowie $i+k$ Spalten für die i Ein- und k Ausgänge.

Die Tabelle lässt sich häufig vereinfachen. Durch ein "Don't Care"-Symbol (X) **don't care** können Eingänge gekennzeichnet werden, wenn sie keinen Einfluss auf den Ausgang nehmen. Ein "sonst" symbolisiert, dass die nicht weiter aufgeführten Eingangszustände einen einzigen Ausgangszustand erwirken.

Das klingt alles etwas kompliziert, Sie werden es aber sicher im Folgenden an einigen Beispielen genauer verstehen !

2.5.3 Schaltnetzwerke

Mit Schaltnetzwerken werden Schaltungen zusammengefasst, die aus einer gewissen Anzahl von Eingangszuständen Ausgangszustände erzeugen. Dabei hängt der Ausgangszustand nur vom momentaner Eingangszustand ab. Die Vorgeschichte bleibt unberücksichtigt.

Meist werde ich das Verhalten mit Hilfe der Hydraulik erläutern, das Verhalten kann jedoch - wie gesehen - einfach in Elektronik umgesetzt werden.

2.5.3.1 Schaltungen mit nur einem Eingang

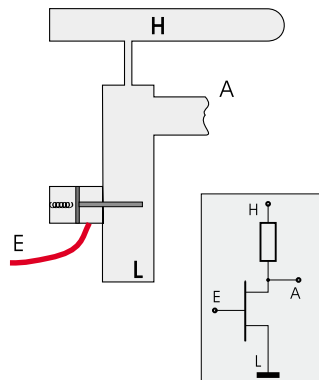
Eine Digitalschaltung mit nur einem Eingang lässt nur 2 Werte (0 und 1) am Eingang zu. Die Wahrheitstafel enthält also nur 2 Zeilen und daher können maximal 4 ($=2^2$) Funktionen erstellt werden :

E	A ₃	A ₂	A ₁	A ₀
0	0	1	0	1
1	0	0	1	1

Zwei davon (A₀ und A₃) davon machen keinen Sinn :

- Alle Ausgänge = 0 bzw. =1 unabhängig vom Eingang.
- A₁ ist unabhängig vom Eingang immer identisch zu E, bringt daher auch nicht viel neues (Könnte als Signalverstärker fungieren)

Bleibt die Funktion A₂: Hier ist der Ausgang – unabhängig von der Zuordnung – immer komplementär zum Eingang. Diese Funktion nennt man **Inverter**, **NOT** oder **NICHT**.



Das erste Beispiel zeigt die Realisierung. Liegt an E ein niedriger Druck (L), so schiebt die Feder den Schieber zu, am Ausgang wird über den Versorgungsdruck und das dünne Rohr R ein hoher Druck (H) aufgebaut : A = H

Liegt an E ein hoher Druck (H), so geht der Schieber auf, der Druck am Ausgang bricht zusammen : A = L

Im Kasten sind die Hydraulik-Komponenten in Elektronik übersetzt

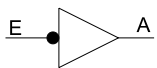
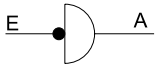
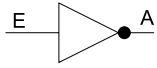
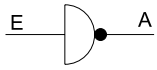
logische Gleichungen

$$A = NOT(E)$$

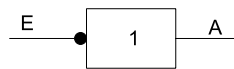
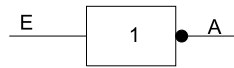
$$A = NICHT E$$

$$A = \overline{E}$$

Im letzten Beispiel wird der Inverter durch den Querstrich markiert.



a.



b.

Schaltsymbole

a. Häufig verwendete Symbole, die auch ich verwenden werde;

Das Symbol erinnert an das des Opamps. Wesentlich ist der Punkt, der das NICHT enthält

b. Symbole nach DIN-Norm, die aber seltener benutzt werden

Beispiel:

E = "Es ist Nacht"

A = "Das Zimmer ist hell"

Ist die Aussage E wahr, so ist die Aussage A falsch

Ist die Aussage E falsch, so ist die Aussage A wahr

Eine doppelte Verneinung führt wieder zur ursprünglichen Aussage.

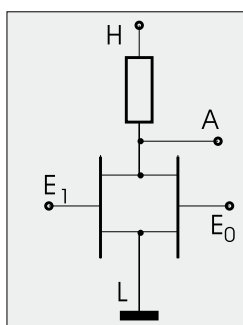
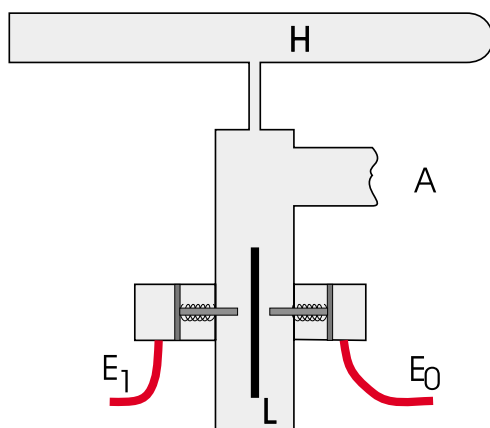
2.5.3.2 Verknüpfungen von 2 Eingängen

Die Verknüpfung $A = A(E_0, E_1)$ von zwei Eingängen E_0, E_1 zu einem Ausgang A kann in eine Wahrheitstabelle mit 4 Zeilen eingetragen werden. Da der Ausgang in 4 Zeilen beschrieben wird, können maximal 16 ($=2^4$) verschiedene Kombinationen des Ausgangs vorkommen.

E_1	E_0	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Auch hier machen viele Funktionen keinen oder wenig Sinn. Die restlichen werden wir in zwei Gruppen aufteilen. Innerhalb einer Gruppe sind die Funktionen nahezu identisch oder zumindest leicht ineinander zu überführen.

Aber zunächst sollten wir ein Beispiel betrachten !



Dieses System enthält zwei parallel geschaltete Schieber: daraus ergeben sich zwei Eingänge und ein Ausgang. Am Ausgang entsteht nur ein hoher Druck H , wenn beide Schieber geschlossen sind, ansonsten bricht der Druck zusammen. Liegt an einem der Eingänge ein niedriger Druck (L), so schiebt die Feder den entsprechenden Schieber auf und der Druck am Ausgang bricht zusammen. Nur wenn an beiden Eingängen ein hoher Druck anliegt, kann nichts entweichen und es wird über den Versorgungsdruck und das dünne Rohr R ein hoher Druck (H) am Ausgang A aufgebaut.

Auch hier sind im Kasten die Hydraulik-Komponenten in Elektronik übersetzt

Anstatt wie hier gezeigt die beiden Schieber parallel anzuordnen, kann man die Schieber auch hintereinander anordnen. Das führt zu sehr ähnlichen Ergebnissen. Wer will, kann selbst die entsprechende Wahrheitstabelle aufstellen

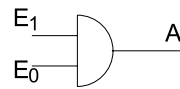
E_1	E_0	A	E_1	E_0	A	E_1	E_0	A
L	L	L	F	F	F	T	T	T
L	H	L	F	T	F	T	F	T
H	L	L	T	F	F	F	T	T
H	H	H	T	T	T	F	F	F

Die Wahrheitstabelle zeigt das Verhalten. Ganz links das physikalische Verhalten; Die anderen beiden Tabellen zeigen die logische Aussage einmal mit positiver und einmal mit negativer Zuordnung

Man erkennt in der mittleren Tabelle, dass sie Ergebnisaussage (A) nur dann wahr ist, falls beide Aussagen (sowohl die Aussage E_0 als auch die Aussage E_1) wahr sind. Also muss E_0 **UND** E_1 wahr sein, damit A wahr ist. Daher nennt man diesen Schaltkreis auch eine UND-Funktion.

Zur Darstellung der UND-Funktion in logischen Gleichungen gibt es eine Reihe von Möglichkeiten. Ich werde hier entweder das Wort "UND" bzw. "AND" oder das Symbol * verwenden. Z.B.
 $A = E_0 \text{ UND } E_1$
 $A = E_0 \text{ AND } E_1$
 $A = E_0 * E_1$

Auch für das Schaltsymbol findet man verschiedene Darstellungen. Ich verwende hier das Symbol

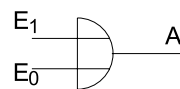


Die Eingänge sind nur bis zur senkrechten Linie gezeichnet

Die rechte Tabelle beschreibt, dass die Ergebnisaussage (A) wahr ist, sobald nur eine der beiden Aussagen wahr ist. Also ist A wahr, wenn E_0 **ODER** E_1 wahr ist. Diesen Schaltkreis nennt man daher auch eine ODER-Funktion.

Auch zur Darstellung der ODER-Funktion in logischen Gleichungen gibt es wieder eine Reihe von Möglichkeiten. Ich werde hier entweder das Wort "ODER" bzw. "OR" oder das Symbol + verwenden. Z.B.
 $A = E_0 \text{ ODER } E_1$
 $A = E_0 \text{ OR } E_1$
 $A = E_0 + E_1$

Auch für das Schaltsymbol findet man wieder verschiedene Darstellungen. Ich verwende hier das Symbol



Die Eingänge sind bis zum Halbkreis gezeichnet

Man erkennt, dass ein und dieselbe physikalische Schaltung - je nach Zuordnung -

sowohl eine UND- als auch eine ODER-Funktion bedeuten kann. UND und ODER sind also sehr ähnlich.

Diese Eigenschaft wird durch das **Morgan'sche Gesetz** beschrieben. Die beiden folgenden Aussagen beschreiben exakt das Gleiche :

Im Zimmer ist es **HELL** wenn es **TAG** **ODER** wenn die **LAMPE AN** ist
 Im Zimmer ist es **DUNKEL** wenn es **NACHT** **UND** wenn die **LAMPE AUS** ist

Die obere Aussage ergibt sich aus der unteren (und umgekehrt) wenn **jede** der Einzelaussagen negiert ($\text{HELL} \Leftrightarrow \text{DUNKEL}$; $\text{TAG} \Leftrightarrow \text{NACHT}$; $\text{LAMPE AN} \Leftrightarrow \text{LAMPE AUS}$) wird, und wenn die Funktionen UND und ODER vertauscht werden.

Anwendungen von AND, NAND, OR und NOR

Wie die Morgan'schen Gesetze zeigen, lassen sich UND und ODER mit NICHT leicht ineinander überführen. Damit sind auch die Einsatzmöglichkeiten nahezu identisch.

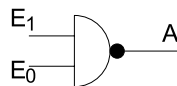
Häufig wird der UND-Funktion noch ein Inverter **nachgeschaltet**. Aus dem AND und NOT ergibt sich daraus das **NAND**.

$$A = \overline{E_0 \text{ UND } E_1}$$

$$A = \overline{E_0 \text{ AND } E_1}$$

$$A = \overline{E_0 * E_1}$$

Im Schaltsymbol wird das NOT durch den Inverter-Punkt markiert hier



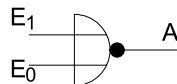
Auch der ODER-Funktion wird häufig noch ein Inverter **nachgeschaltet**. Hier ergibt sich aus OR und NOT das **NOR**.

$$A = \overline{E_0 \text{ ODER } E_1}$$

$$A = \overline{E_0 \text{ OR } E_1}$$

$$A = \overline{E_0 + E_1}$$

Auch hier wird im Schaltsymbol das NOT durch den Inverter-Punkt markiert



E_0	E_1	AND	NAND	OR	NOR
F	X	F	T	E_1	$\overline{E_1}$
T	X	E_1	$\overline{E_1}$	T	F

Das **X** in der Tabelle bedeutet **don't care** (An dieser Stelle darf sowohl ein T oder auch ein F stehen)

Am Ausgang erscheint abhängig von E_0 entweder der Eingangswert E_1 oder nicht

(Bei NAND und NOR der invertierte Eingangswert).

Damit lassen sich diese Funktionen als **Tor für Informationen** einsetzen. Daher kommt auch die Bezeichnung **Gate** oder **Gatter**.

Die Wahrheitstabelle lässt sich noch etwas vereinfachen:

...	E_3	E_2	E_1	E_0	AND	NAND	...	E_3	E_2	E_1	E_0	OR	NOR
T	T	T	T	T	T	F	F	F	F	F	F	F	T
	sonst				F	T		sonst				T	F

Man erkennt zum einen sehr leicht wieder die Morganschen Gesetze (durch Negieren sämtlicher Ein- und Ausgänge wird aus UND ein ODER bzw aus ODER ein UND)

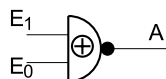
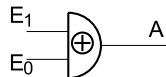
Weiter kann man leicht einsehen, dass die UND- und die ODER-Funktion beliebig erweitert werden kann:

$$A = E_0 * E_1 * E_2 * E_3 * E_4 \dots \quad \text{bzw.} \quad A = E_0 + E_1 + E_2 + E_3 + E_4 \dots$$

Anwendungen von XOR und XNOR

Von den 16 möglichen Funktionen mit 2 Eingängen wurden bisher vier besprochen. Daneben gibt es zwei weitere, die sehr nützlich sind.

E_0	E_1	XOR	XNOR
F	F	F	T
F	T	T	F
T	F	T	F
T	T	F	T



XOR ist die Kurzform von exclusive OR und meint, dass der Ausgang wahr ist, falls E_0 oder E_1 wahr sind aber nicht beide.

Entsprechend ist **XNOR** ein XOR mit nachgeschaltetem Inverter

Die Symbole zeigen das XOR oben sowie unten das XNOR

Betrachten wir das XNOR in der Tabelle, so erkennen wir, dass der Ausgang wahr (T) ist, falls die beiden Eingänge (E_0 und E_1) den gleichen Wert haben, bzw. der Ausgang ist falsch (F), falls die beiden Eingänge (E_0 und E_1) unterschiedliche Werte haben. Aus diesem Grund bezeichnet man das XNOR häufig auch als **Äquivalenz**. Entsprechend ist beim XOR der Ausgang wahr (T), falls die beiden Eingänge (E_0 und E_1) unterschiedliche Werte haben und falsch (F) bei gleichen Werten der beiden Eingänge. Das XOR heißt daher auch **Antivalenz**

XOR und XNOR lassen sich zum **Vergleichen** von Informationen einsetzen

Wenn man die Wahrheitstabelle von XOR bzw. XNOR etwas umschreibt, erkennt man, dass abhängig von einem der Eingänge der zweite Eingang entweder exakt oder invertiert am Ausgang erscheint

E_0	E_1	XOR	XNOR
F	X	E_1	$\overline{E_1}$
T	X	$\overline{E_1}$	E_1

Mit XOR und XNOR lassen sich **wahlweise Informationen negieren !!**

2.5.4 ENCODER

Aufgabe: Verschlüsseln (Zuordnung einer Zahl im Binärcode C_0, C_1, C_2, \dots zu jeder von 2^n Eingangsleitungen E_i). Reduzierung der notwendigen Anzahl der bits von 2^n auf nur n bits.

Wahrheitstafel:

E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	C_2	C_1	C_0
L	L	L	L	L	L	L	H	L	L	L
L	L	L	L	L	L	H	L	L	L	H
L	L	L	L	L	H	L	L	L	H	L
L	L	L	L	H	L	L	L	L	H	H
L	L	L	H	L	L	L	L	H	L	L
L	L	H	L	L	L	L	L	H	L	H
L	H	L	L	L	L	L	L	H	H	L
H	L	L	L	L	L	L	L	H	H	H

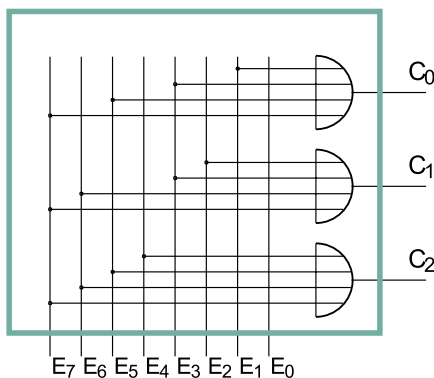
logische Gleichungen

$$C_0 = E_1 + E_3 + E_5 + E_7$$

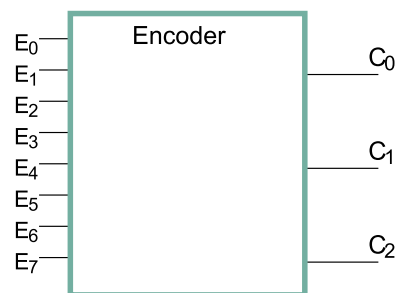
$$C_1 = E_2 + E_3 + E_6 + E_7$$

$$C_2 = E_4 + E_5 + E_6 + E_7$$

Schaltung



Schaltsymbol



Gezeigt ist ein Encoder für 8 Eingänge. Er ordnet jedem der Eingänge einen 3bit-Code zu. Es darf nur eine der Eingangsleitungen aktiviert werden. Für einen n-bit-

Code sind n ODER-Gatter mit je 2^{n-1} Eingängen erforderlich, die n-Ausgangsbits erzeugen.

Zeitverlust bei der Umsetzung durch RC-Zeit und Umladungsdauern in den Halbleitern 1 **Gatterlaufzeit** (GLZ heute \approx 1ns)

2.5.5 DECODER

Aufgabe: Entschlüsseln, eine Codes C_0, C_1, C_2, \dots

Wahrheitstafel:

C_2	C_1	C_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
L	L	L	L	L	L	L	L	L	L	H
L	L	H	L	L	L	L	L	L	H	L
L	H	L	L	L	L	L	L	H	L	L
L	H	H	L	L	L	L	H	L	L	L
H	L	L	L	L	L	H	L	L	L	L
H	L	H	L	L	H	L	L	L	L	L
H	H	L	L	H	L	L	L	L	L	L
H	H	H	H	L	L	L	L	L	L	L

logische Gleichungen

$$A_0 = \overline{C_0} * \overline{C_1} * \overline{C_2}$$

$$A_1 = \overline{C_0} * \overline{C_1} * C_2$$

$$A_2 = \overline{C_0} * C_1 * \overline{C_2}$$

$$A_3 = \overline{C_0} * C_1 * C_2$$

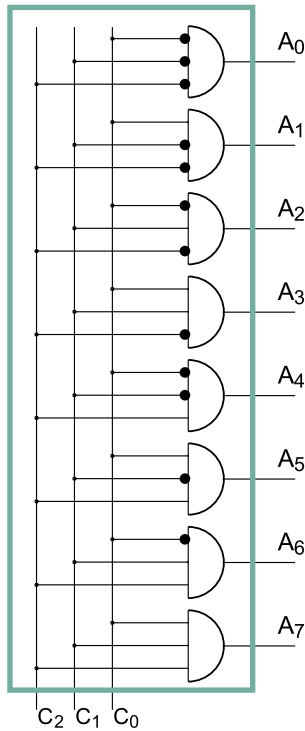
$$A_4 = \overline{C_0} * \overline{C_1} * C_2$$

$$A_5 = C_0 * \overline{C_1} * C_2$$

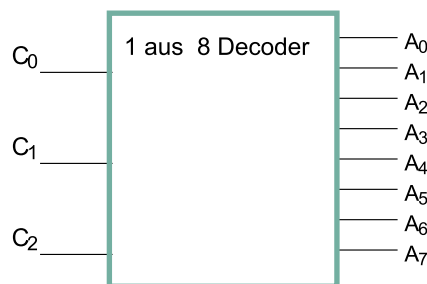
$$A_6 = \overline{C_0} * C_1 * C_2$$

$$A_7 = C_0 * C_1 * C_2$$

Schaltung



Schaltsymbol



Zeitverlust wieder 1 Gatterlaufzeit

Gezeigt ist ein 1 aus 8-Decoder (1-aus-8-Decoder). Er entschlüsselt einen 3bit-Code. Für n bits sind 2^n UND-Gatter mit je n Eingängen erforderlich, die 2^n Ausgangsleitungen bedienen.

Jeder der 2^n möglichen Bitkombinationen wird genau eine Ausgangsleitung zugeordnet.

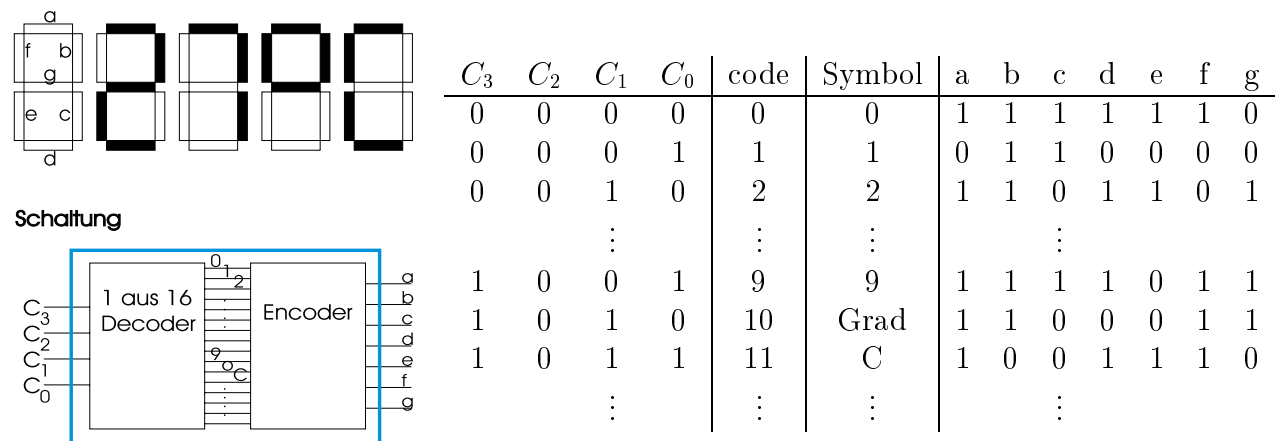
2.5.6 UMCODIERER

Decoder und Encoder haben in der Wahrheitstabelle auf jeweils einer Seite der Tabelle nur ein einziges **T** pro Zeile. Diese Einschränkung wollen wir jetzt aufgeben: Auf jeder der beiden Seiten der Wahrheitstabelle dürfen beliebige Bitkombinationen stehen.

Aufgabe: Zuordnen eines Ausgangscodes $CODE_2$ zu einem Eingangscod $CODE_1$.

Lösung: Möglich ist es immer, den n -bit Eingangscod mit einem 1 aus 2^n -Decoder zu entschlüsseln und anschließend mit dem entschlüsselten Code jedem der 2^n möglichen Eingangskombinationen mit einem Encoder einen neuen Code zuzuordnen. In den vielen Fällen kann jedoch die Schaltung vereinfacht und der Schaltungsaufwand reduziert werden.

Es folgt ein Beispiel : **Siebensegmentanzeige**



Das Ergebnis ist ein Schaltkreis (im blauen Rahmen) mit 4 Ein- und 7 Ausgängen. Die Dauer der Umsetzung beträgt 2 Gatterlaufzeiten !

Da die Form von Symbolen frei wählbar ist, erkennt man an diesem Beispiel, dass man eine beliebige Ausgangs-Bitkombination erzeugen kann.

Auch die Anzahl der bits an Ein- und Ausgang kann verschieden sein.

2.5.7 ADDIERER

Aufgabe: Addition von Binärzahlen

	2^5	2^4	2^3	2^2	2^1	2^0	10^2	10^1	10^0
	32	16	8	4	2	1	100	10	1
A	1	1	0	0	0	1		4	9
B	1	1	0	1	0	1		5	3
Cy	1	1	0	0	0	1	1	1	
Z	1	1	0	0	0	1	1	1	
S	1	1	0	0	1	1	1	0	2

rot markiert ist die Addition innerhalb einer Stelle, die Wertigkeit dieser Stelle ist in den beiden oberen Zeilen angegeben.

Die laufende Nummer der bits (0,1,2,...) entspricht der Hochzahl in der obersten Zeile.

Der Wert der Zahlen A, B und S ist links und rechts identisch !

Die Addition der beiden Zahlen A und B im Binärsystem (links) entspricht der im Zehnersystem (rechts) :

Für jede Stelle i werden die beiden digits (bits) A_i und B_i sowie gegebenenfalls ein Übertrag (Carry) aus der vorhergehenden Stelle Cy_i addiert.

Daraus ergibt sich die Summe der i -ten Stelle ($S_i + 10 \cdot Z_i$) mit Z_i der Anzahl der Zehner

Diese Z_i liefern den Übertrag in die nächste Stelle: $Cy_{i+1} = Z_i$

Im Zweiersystem ergibt sich die Summe des i -ten bits zu $S_i + 2 \cdot Z_i$, mit Z_i der Anzahl der Zweier

Wahrheitstafel für eine Binärstelle:

Die Werte Z und S jeder Stelle i ergeben sich aus A, B und Cy dieser Stelle. Alle Möglichkeiten von A, B und Cy und die entsprechenden Ergebnisse Z und S lassen sich in eine Tabelle eintragen.

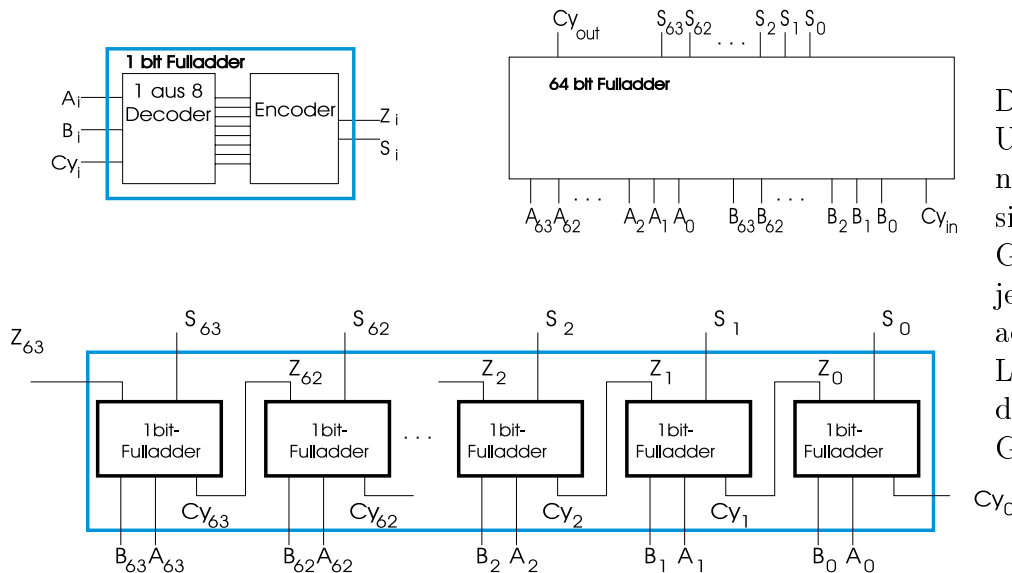
Die nebenstehende Tabelle zeigt das Verhalten im Binärsystem

Die Realisierung als Schaltung kann wie oben mit einem Decoder und nachgeschalteten Encoder erfolgen.

Die Dauer zur Umsetzung von den Summanden (A, B) zur Summe (A+B) beträgt 2 Gatterlaufzeiten. Diesen Schaltkreis nennt man auch **1 bit Volladdierer** oder **1 bit Fulladder**. Voll bedeutet, das bei der Addition ein Übertrag mitberücksichtigt wird.

Cy	B	A	Z	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Zur Addition einer mehrstelligen Binärzahl muss für jede Stelle (bit) ein 1-bit-Volladdierer eingesetzt und der Übertrag (Cy) jeweils von Stelle zu Stelle weitergereicht werden



Die Zeit zur Umsetzung innerhalb eines bits sind zwar nur 2 Gatterlaufzeiten, jedoch bei 64 bits addieren sich die Laufzeiten für die Cys auf $64 \cdot 2$ Gatterlaufzeiten!

Die lange Rechenzeit zur Bestimmung der Summe von zwei 64-bit-Zahlen ($\approx 100\text{ns}$) erzwingt eine andere Lösung. Denkbar wäre es, die beiden 64-bit Summanden A und B auf eine Seite der Wahrheitstabelle einzutragen, und die Summen auf die andere Seite. Jedoch erhält man damit 2 mal 64 bits auf der Tabelleneingangsseite und folglich 2^{128} mögliche Kombinationen. Man sieht sofort ein, dass dies zu einem unvorstellbarem Aufwand führt und daher nicht in Frage kommt.

Auch hier sehen wir wieder die beiden unterschiedlichen Lösungen.

64 einfache (1-bit-fulladder) Baugruppen hintereinander (seriell) schalten; das führt zu einer einfachen und billigen aber langsamen Lösung. Im Falle des 64-bit-Adders meist zu langsam.

Oder eine schnelle Lösung: Viele Baugruppen arbeiten gleichzeitig (parallel). Das erfordert einen höheren (teuren) Schaltungsaufwand. Ein 64-bit-Addierers mit der gleichzeitigen Ausdekodierung sämtlicher Kombinationen ist aber wegen des Aufwandes nicht mehr realisierbar.

Hier bleibt ein Kompromiss zwischen annehmbarer Geschwindigkeit und erträglichem Aufwand.

zusätzliche Informationen

look ahead Carry generator

Gehen wir von der Wahrheitstabelle des Volladdierers aus und betrachten zunächst nur die obere Hälfte (C=0).

Die Ausgänge dieses Ausschnitts bezeichnet man mit p (propagate, vererben) und g (generate, erzeugen):

p und g sind die Summe aus A und B **ohne** Carry (Halbaddierer, halfadder; grüner Kasten)

$$g = A \text{ AND } B \quad p = A \text{ XOR } B$$

$$S = p \quad Z = g$$

Nehmen wir jetzt das Carry hinzu, erkennen wir, dass dann :

grün : $S = \bar{p}$ ist. Also $S = C \oplus p$

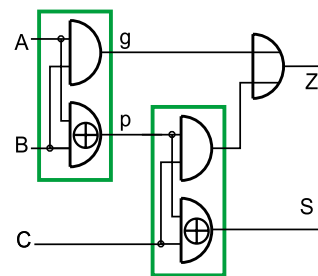
Z besteht aus zwei Teilen:

blau: aus g ODER

rot: bei C=1 aus p

Also : $Z = g \text{ OR } (p \text{ AND } C)$

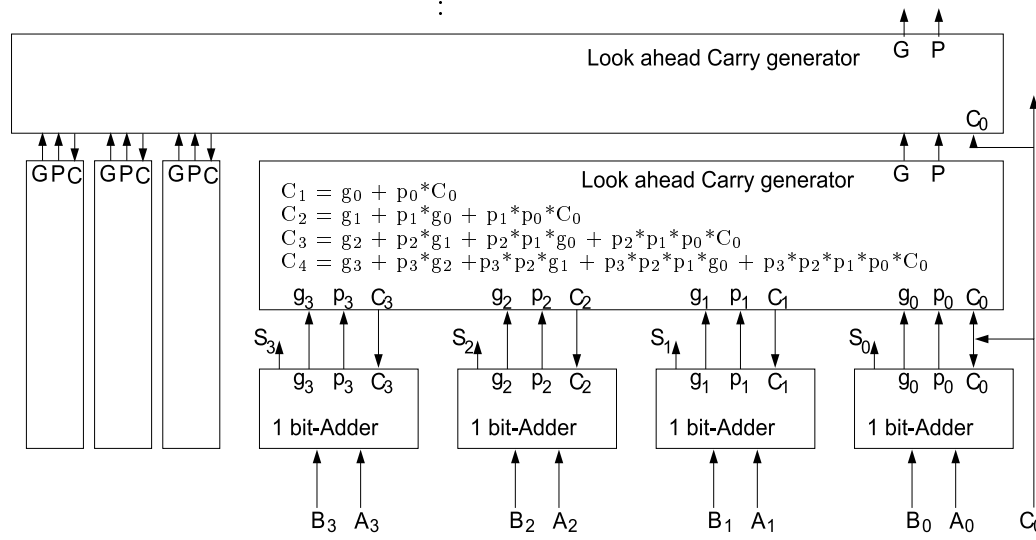
C	B	A	g	p	Z	S
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	0	0	1



Das Hauptproblem war, eine Lösung zu finden, bei der die Zeitverluste durch das Berechnen des Carrys vermieden werden. In der Schaltung oben sieht man, das p und g (ohne C also sehr schnell) vorliegen.

Durch Einsetzen des Carry aus der vorhergehenden Stelle erhält man das aktuelle Carry, das jetzt nur noch von den p_i, q_i und C_0 abhängt, und damit sehr schnell berechenbar ist:

- (1) $C_1 = Z_0 = g_0 + p_0 * C_0$ also $C_1 = g_0 + p_0 * C_0$
- (2) $C_2 = Z_1 = g_1 + p_1 * C_1$ mit (1) $C_2 = g_1 + p_1 * g_0 + p_1 * p_0 * C_0$
- (3) $C_3 = Z_2 = g_2 + p_2 * C_2$ mit (2) $C_3 = g_2 + p_2 * g_1 + p_2 * p_1 * g_0 + p_2 * p_1 * p_0 * C_0$
- (4) $C_4 = Z_3 = g_3 + p_3 * C_3$ mit (3) $C_4 = g_3 + p_3 * g_2 + p_3 * p_2 * g_1 + p_3 * p_2 * p_1 * g_0 + p_3 * p_2 * p_1 * p_0 * C_0$



Die Terme der Gleichung (4) ohne C_0 lassen sich zu G und der andere zu $P * C_0$ zusammenfassen und man erhält rein formal wieder etwas entsprechend der Gleichung (1):

$$C_4 = G + P * C_0$$

Auf diese Weise lassen sich mehrere Blöcke wiederum mit einem look-ahead-carry-generator und auch die dann neu entstandenen Blöcke kaskadieren usw.

Mit einem look-ahead-carry-generator lassen sich 4 bits, einmal kaskadiert 16 bits und ein weiteres mal kaskadiert 64 bits mit mäßigem Aufwand und relativ schnell addieren.

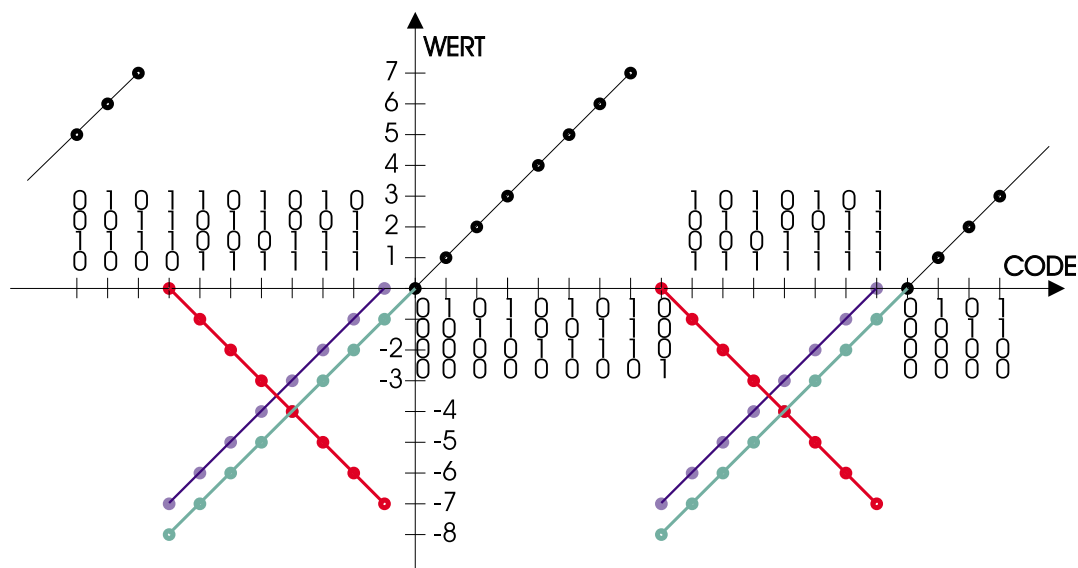
2.5.8 Subtraktion

Bevor wir einen Subtrahierer realisieren, muss man entscheiden, wie negative Zahlen codiert werden sollen.

2.5.8.1 Darstellung negativer Zahlen

Da ein Vorzeichen zwei mögliche Zustände einnehmen kann, kann es einfach auf ein bit abgebildet werden. Es wäre daher sinnvoll ein bit (**sign-bit**) zur Darstellung einzusetzen. In der folgenden Tabelle sind drei Verfahren dargestellt:

				Vorz. * Zahl	Einer-Komplement Ones-Complement	Zweier-Komplement Twos-Complement	
CODE				Vor- Zeichen	$*(-1) : S \mapsto \bar{S}$	$*(-1) : CODE \mapsto \overline{CODE}$	$*(-1) : CODE \mapsto \overline{CODE} + 1$
S	ZAHL				WERT	WERT	WERT
0	0	0	0	+1	0	0	0
0	0	0	1	+1	1	1	1
0	0	1	0	+1	2	2	2
0	0	1	1	+1	3	3	3
0	1	0	0	+1	4	4	4
0	1	0	1	+1	5	5	5
0	1	1	0	+1	6	6	6
0	1	1	1	+1	7	7	7
1	0	0	0	-1	-0	-7	-8
1	0	0	1	-1	-1	-6	-7
1	0	1	0	-1	-2	-5	-6
1	0	1	1	-1	-3	-4	-5
1	1	0	0	-1	-4	-3	-4
1	1	0	1	-1	-5	-2	-3
1	1	1	0	-1	-6	-1	-2
1	1	1	1	-1	-7	-0	-1
0	0	0	0	+1	0	0	0



Daraus ergeben sich die drei oben gezeigten Funktionen. Beim Weiterzählen des CODES wiederholen sich die Werte (letzte Zeile). Daher wiederholt sich auch der Graph nach 16 Codes !

Im positiven Bereich (schwarz) sind die Graphen identisch.

- $*(-1) : S \mapsto \overline{S}$: Complement des SIGN-bits (rot) ergibt unterschiedliche Steigungen für positive und negative Werte. Außerdem ist die **0** doppeldeutig.
- $*(-1) : CODE \mapsto \overline{CODE}$ (blau) : Das **Einer-Komplement (one's complement)** ergibt zwar gleiche Steigungen für positive und negative Werte, aber die **0** ist immer noch doppeldeutig.
- $*(-1) : CODE \mapsto \overline{CODE} + 1$ (grün) (**Zweier-Komplement** oder **two's complement**) :
Zum Complement des CODES wird noch eine **1** addiert. Es ergeben sich sowohl gleiche Steigungen für positive und negative Werte und die Doppeldeutigkeit der **0** ist verschwunden.
Der Graph ist eine von -8 bis +7 durchgehende lineare Funktion. Die Zweier-Komplement-Darstellung ist also die zur Anwendung im Computer optimale Form und wird dort fast ausschließlich eingesetzt (**Integer-Darstellung**)

zusätzliche Informationen

Ausnahmeregeln der Integerdarstellung

müssen beachtet werden, wenn das Ergebnis einer Operation den Wertebereich (bei n bits von -2^{n-1} bis $+2^{n-1}-1$) verlässt:

Addition von zwei positiven Zahlen (bzw. falls das Ergebnis negativ wird (Sign-Subtraktion einer negativen Zahl von einer positiven) bit=1) muss 2^n addiert werden

Addition von zwei negativen Zahlen (bzw. falls das Ergebnis positiv wird (Sign-Subtraktion einer positiven Zahl von einer negativen) bit=0) muss 2^n subtrahiert werden

Multiplikation von -2^{n-1} mit (-1) (liefert -2^{n-1}) Addition von 2^n

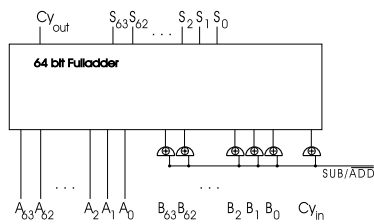
2.5.8.2 technische Realisierung

Addition		Subtraktion	
Summe = $A + B + \text{Carry}$		Differenz = $A - B - \text{Borrow}$	
		Differenz = $A + (-B) - \text{Borrow}$	
		twos-Complement:	
		Differenz = $A + (\overline{B} + 1) - \text{Borrow}$	
$Cy = 0$	Summe = $A + B + 0$	$Bo = 0$	$\overline{Bo} = 1$ Differenz = $A + (\overline{B} + 1) - 0$
$Cy = 1$	Summe = $A + B + 1$	$Bo = 1$	$\overline{Bo} = 1$ Differenz = $A + (\overline{B} + 1) - 1$
$Cy = 0$	Summe = $A + B + 0$	$Bo = 0$	$\overline{Bo} = 1$ Differenz = $A + \overline{B} + 1$
$Cy = 1$	Summe = $A + B + 1$	$Bo = 1$	$\overline{Bo} = 1$ Differenz = $A + \overline{B} + 0$

Die Schaltung für das Integer-Format ergibt sich aus der Tabelle oben: Vor der Subtraktion muss der Minuend und ein eventuelles Borrow komplementiert werden. Dies geschieht mit Hilfe von XOR-Gattern, die eine schaltbare Inversion erlauben (siehe dazu auch Abschnitt "Verknüpfungen mit 2 Eingängen", XOR).

Der Name der Kontrollleitung SUB/\overline{ADD} besagt, dass bei einer **0** auf dieser Leitung die Eingänge B und Cy nicht invertiert werden und dass somit die Baugruppe eine Addition ausführt und bei einer **1** auf dieser Leitung die Eingänge B und Cy invertiert und somit die Baugruppe eine Subtraktion ($A - B - Cy$) ausführt.

Sprich für SUB/\overline{ADD} : "SUB NOT ADD" !



2.5.9 Inkremitter, Dekremitter

Die Funktionen Inkrement (+1) und Dekrement (-1) sind sehr wichtig, z.B. als Schleifenzähler oder als Programmzähler. Diese Funktionen ließen sich natürlich mit einem Addierer oder Subtrahierer realisieren, doch gibt es für diese Aufgabe wesentlich weniger aufwendige Schaltgruppen, den **Inkrementer** und den **Dekremitter**.

Betrachtet man zwei benachbarte Zahlen im Binärkode, erkennt man, dass jedes bit genau dann umschaltet, wenn **alle niederwertigen bits = 1** sind (Increment) bzw. wenn **alle niederwertigen bits = 0** sind (Decrement). Daher lässt sich ein Inkrementer mit einem UND und nachgeschaltetem XOR pro bit und ein Dekremitter mit einem OR und nachgeschaltetem XNOR pro bit realisieren.

Der interessierte Leser mag sich die entsprechende Schaltung überlegen !

3 Speicher

3.1 FlipFlop

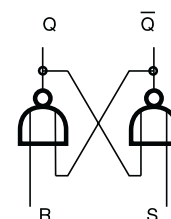
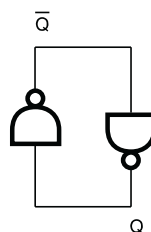
Anwendung : Das Flipflop (FF) ist ein Speicher für ein bit

Das **FlipFlop** besteht aus zwei Invertern, die im Kreis geschaltet sind. Meist werden sie über Kreuz gezeichnet. Dadurch entsteht eine Baugruppe mit zwei Ausgängen Q und \bar{Q} . Das FF kann zwei stabile Zustände annehmen. Die Art der Ansteuererschaltung entscheidet, wie die beiden Zustände eingestellt werden.

3.1.1 RS-FlipFlop, RS-FF

Beim **RS-FF** erweitert man die Inverter zu NOR-Gattern und bezeichnet die beiden freien Eingänge der NOR-Gatter mit R und S (Reset, Rücksetzen und Set, Setzen). Solange R und S auf L liegen, ändert sich nichts an dem stabilen Zustand. Legt man jedoch ein kurzes H-Signal auf einen der Eingänge, so wird das FF einen definierten Zustand ($Q = L$ oder $Q = H$) einnehmen und dort verharren. Werden beide Eingänge gleichzeitig auf H gesetzt, lässt sich der Ausgangszustand nicht eindeutig vorherbestimmen (verboten).

R	S	Q	\bar{Q}	
L	L	Q	\bar{Q}	stabil
H	L	L	H	Reset
L	H	H	L	Set
H	H	unbekannt, verboten		

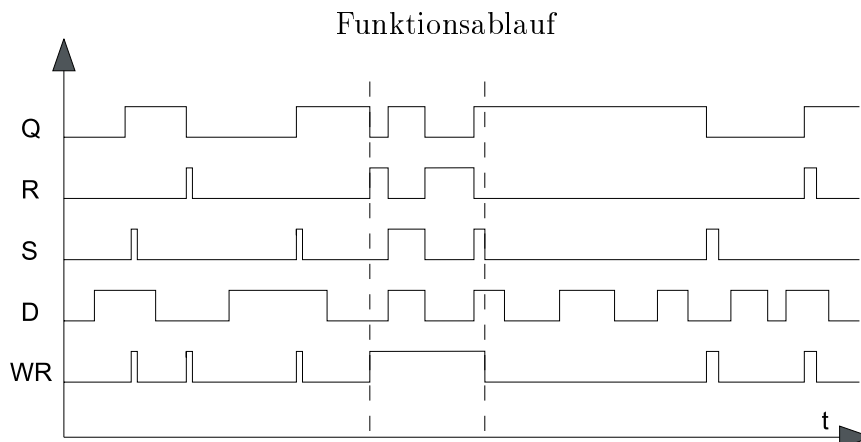
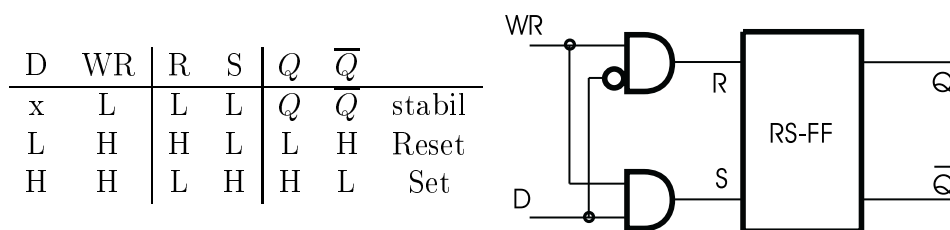


Durch Anlegen von H-Signalen an R und S kann der Zustand des FFs (Q) verändert werden. Durch die endliche Geschwindigkeit der Bauteile schalten die Ausgänge etwas verzögert um.

Wegen der De Morgan'schen Gesetze lässt sich ein FF auch mit NAND-Gattern realisieren. Doch wird eine Aktion hier mit L ausgelöst; die Bezeichnung R muss in \overline{S} und S in \overline{R} umbenannt werden.

3.1.2 D-FlipFlop, D-FF (Latch, Zwischenspeicher)

Das D-FF hat zwei Eingänge D (Daten) und WR (Write, Schreiben), aus denen durch eine einfache Umcodierung R und S erzeugt werden kann. Nach der Umcodierung entfällt der verbotene Zustand.



Während das WR-Signal auf H liegt wird - abhängig vom Zustand von D - entweder ein S- bzw. ein R-Signal erzeugt. Dadurch erscheint während $WR=H$ ist am Ausgang Q das Signal D (Wir sagen, das FF ist durchsichtig (transparent)).

Nach Wegnehmen des WR-Signals bleibt der Zustand an Q erhalten (er wird eingefroren, gespeichert).

3.2 Datentransport

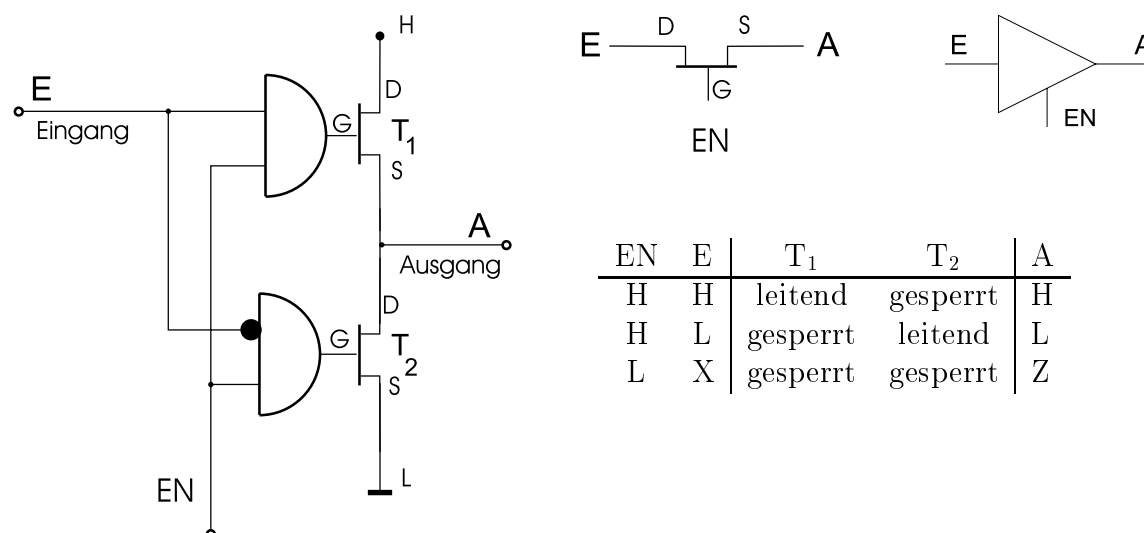
Aufgabe : Transport von Daten (Kopieren) von einer Datenquelle (Source) zu einem Datenziel (Destination)

Um einen geringen Verdrahtungsaufwand zu erreichen, ist es sinnvoll, verschiedene

Datenquellen und Ziele mit einem gemeinsamen Datenkanal zu verbinden. Einen solchen Datenkanal bezeichnet man mit **BUS**, von lat. omnibus = für alle.

Beim Transport von Daten aus verschiedenen Datenquellen über einen gemeinsamen Draht, könnten sowohl H- wie L-Daten anstehen, und es käme zu unerlaubten Querströmen. Zur Vermeidung von Datenkollisionen verwendet man entweder **Open-Kollektor-Ausgänge** (Open-Drain-) oder **Tri-State-Gatter**. Hier soll nur auf die Tri-State-Gatter eingegangen werden. In diesem Fall kann der Ausgang nicht nur die beiden Zustände H und L annehmen, sondern es gibt noch einen dritten Zustand Z, in dem der Ausgang abgeschaltet ist.

Aufgabe : Durchlassen und Sperren von Daten, um Kollisionen zu vermeiden



In dem dritten Zustand sind beide Ausgangstransistoren gesperrt. Alle Verbindungen des Ausgangs mit der übrigen Schaltung sind hochohmig.

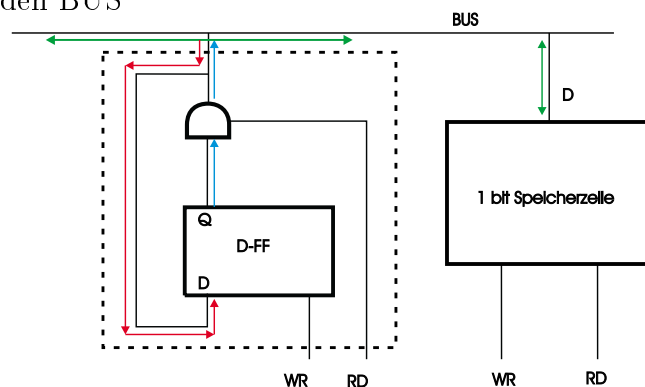
Die Steuerung erfolgt über einen Enable-Eingang (EN). Dabei werden die Gates der beiden Transistoren T₁ und T₂ so angesteuert, dass entweder eine Verbindung von H oder zu L zum Ausgang entsteht, oder beide Transistoren gesperrt sind. So übernimmt der Ausgang bei einem H an EN den Wert des Eingangs, ansonsten hat der Ausgang weder eine Verbindung mit L noch mit H.

Eine einfache Form eines 3-State-Gatters kann durch einen FET gebildet werden. Über den steuerbaren Kanal ist der Ausgang mit dem Eingang verbunden, so dass das Ausgangspotential dem Eingangspotential entspricht. Über das Gate (Enable-Eingang) kann der Kanal gesperrt und so der Ausgang in den Z-Zustand gebracht werden.

3.3 Speicherzelle

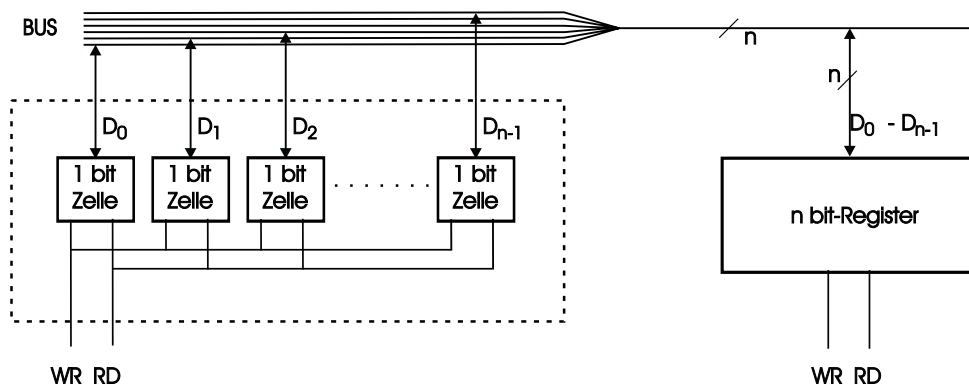
3.3.1 1-Bit-Speicherzelle

Aufgabe : Einschreiben eines bits vom BUS in die Zelle bzw. auslesen eines bits aus der Zelle auf den BUS



Die 1 bit Speicherzelle (gestrichelter Kasten) besteht aus einem D-FF mit 3-State-Gatter. Beim Einschreiben (WR) werden Daten vom BUS in Pfeilrichtung (rot) über den D-Eingang in dem FF gespeichert. Beim Lesen (RD) werden Daten durch das 3-State-Gatter in Pfeilrichtung (blau) auf den BUS gelegt, und können so von anderen Stationen übernommen werden. Im Schaltsymbol (rechts) erkennt man den bidirektionalen (grünen) Datenkanal D und die Kontrollleitungen RD und WR.

3.3.2 Register



Ein n-bit-Register (gestrichelter Kasten) besteht aus n Speicherzellen mit gemeinsamen WR- und RD-Steuerleitungen. Der BUS besteht aus n Leitungen, wobei der Datenanschluss jeder Zelle mit je einem Draht des Busses verbunden ist. Im Schaltsymbol (rechts) sind die n Drähte durch den schrägen Strich markiert.

3.4 Memory

In einem Memory sind viele Daten gespeichert. Eine Übersicht liefert die Tabelle am Ende des Kapitels.

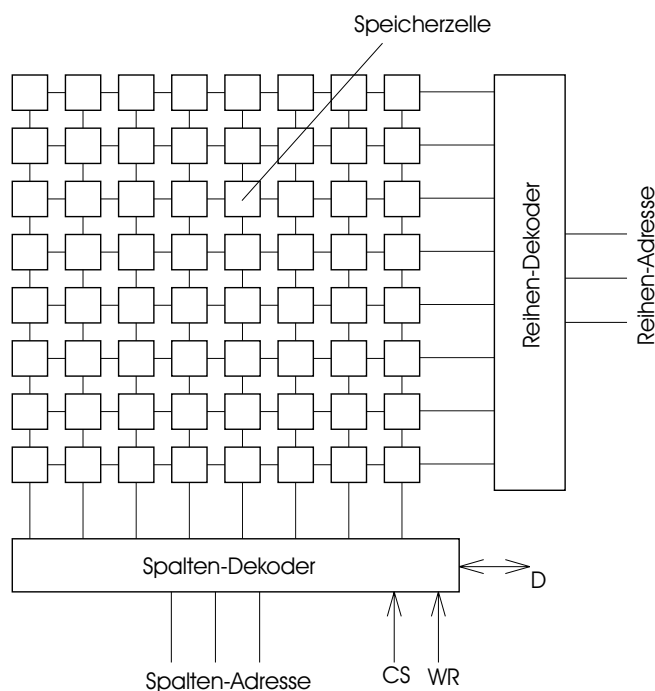
3.4.1 Zugriff über Adresse

Das Memory besteht meist aus matrixartig in Reihen (row) und Spalten (column) angeordneten 1-bit-Speicherzellen. Diese Art des Zugriffs bezeichnet man mit Random Access.

Mit je einem Decoder (Adressdecoder) wird eine Spalte (Bitleitung) und eine Reihe (Wortleitung) selektiert. Die Speicherzelle am Kreuzungspunkt der selektierten Reihe und Spalte ist angewählt, und kann gelesen oder beschrieben werden

Der Zugriff erfolgt über CS (Chip select) und WR, daraus kann man RD und WR erzeugen und auch die gesamte Baugruppe abschalten

CS	WR	RD	WR
1	0	1	0
1	1	0	1
0	X	0	0



Um einen möglichst geringen Aufwand zu erreichen, ist es sinnvoll, den Aufwand für jede einzelne Zelle gering zu halten, auch wenn dazu ein erheblich höherer Aufwand für den Rest der Baugruppe notwendig wird.

Wir unterscheiden zwei Arten von Memories:

RAM : RandomAccessMemory (Memory mit wahlfreiem Zugriff) kann gelesen und beschrieben werden

Vorteil: RAM kann gelesen und beschrieben werden

ROM : ReadOnlyMemory (Nurlese Speicher). Kann nur gelesen werden

Vorteil: ROM kann nicht beschrieben werden, daher können auch die Daten im ROM nicht ungewollt zerstört werden

Bei Netzausfall bleiben die Daten im ROM erhalten

3.4.1.1 RAM

Es gibt zwei Typen von RAMs: Das **statische RAM (SRAM)** verwendet ein FF als Speicherelement und das **dynamische RAM (DRAM)** speichert ein bit in einem Kondensator.

SRAM

Vorteil : kein Refresh, kein Zurückschreiben der Daten notwendig, daher schnell
unterschiedliche Techniken erlauben Speicher mit sehr geringem Stromverbrauch aber auch sehr schnelle Speicher

Nachteil: wegen des höheren Aufwands sind SRAMs teurer und haben eine geringere Kapazität

DRAM

Vorteil : wenig aufwendig, daher große und preiswerte Speicher möglich

Nachteil: nicht zerstörungsfrei lesbar, Refresh notwendig. Beides reduziert die Zugriffsgeschwindigkeit erheblich

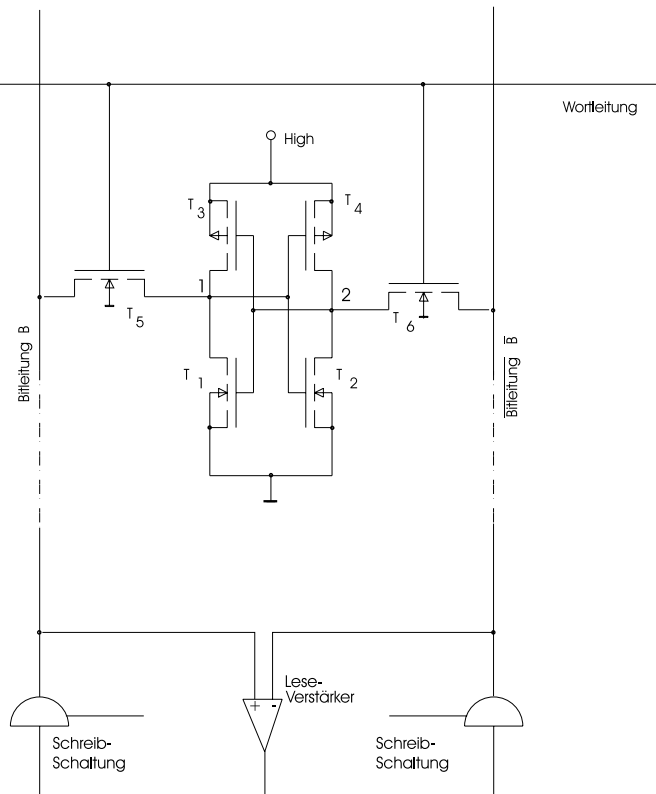
Softerrors lassen einzelne bits kippen (kann durch geeignete Gehäusematerialien reduziert werden) und durch Parity (Prüfsummen) erkannt oder sogar korrigiert werden

wegen der hohen Anforderungen an den Kondensator und den Schalttransistor gibt es nur wenige alternative Techniken. Daher hat sich auch die Zugriffsgeschwindigkeit in den letzten Jahrzehnten wenig verbessert

3.4.1.1.1 SRAM Bei dem statischen RAM besteht die Speicherzelle aus einem FF.

Dabei lassen sich verschiedene Technologien realisieren, bei den man zum Beispiel auf möglichst hohe Geschwindigkeit oder möglichst geringen Stromverbrauch optimieren kann.

In der MOS-Technik besteht das FF aus zwei Transistoren mit zwei Widerständen. In der gezeichneten CMOS-Technik werden die beiden Widerstände durch je einen komplementären Transistor ersetzt. Die statische Speicherzelle CMOS-Technik enthält 6 Transistoren. Zu der Zelle werden 2 Bitleitungen und eine Wortleitung geführt. Die Speicherzelle in MOS-Technik besteht aus den beiden über Kreuz geschalteten Invertern T_1 bzw. T_2 mit je einem Widerstand nach H. In der hier gezeichneten CMOS-Technik sind die beiden Widerstände durch die komplementären Transistoren T_3 bzw. T_4 ersetzt. (komplementär heißt, dass die Transistoren gegensinnig zu T_1 bzw. T_2 sperren und leiten.) Die Transistoren T_5 und T_6 wirken je nach Ansteuerung an der Bitleitungen beim Schreiben als R und S und beim Lesen als 3-State-Gatter



3.4.1.1.2 DRAM Die Speicherzelle des dynamischen RAMs ist ein Kondensator. Da ein Kondensator über nicht vermeidbare Lecks sich wieder entlädt (RC-Zeit !), muss man auf möglichst große Kondensatoren und geringe Leckströme (großes R) achten.

zusätzliche Informationen

Trench-Kondensator

Der Kondensator einer dynamischen Speicherzelle soll eine große Kapazität haben, um die Entladung durch Lecks (besonders des gesperrten FETs) möglichst gering zu halten (große RC-Zeit). Andererseits soll der Platzbedarf für den Kondensator möglichst klein gehalten werden, um eine große Anzahl von bits auf dem Chip unterzubringen.

Die Kapazität eines Kondensators ergibt sich zu :

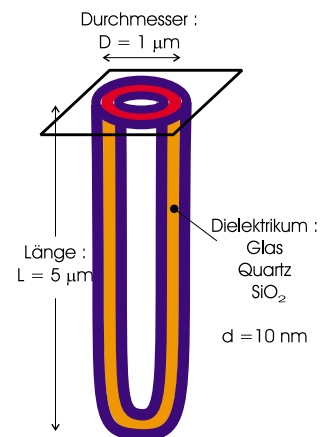
$$C = \epsilon \epsilon_0 F / d$$

Daher wird der Kondensator als aufrechtes dünnes Röhrchen eingegraben (**Trench-Technik**, Grabentechnik), was eine ca. 15mal größere Oberfläche ergibt !

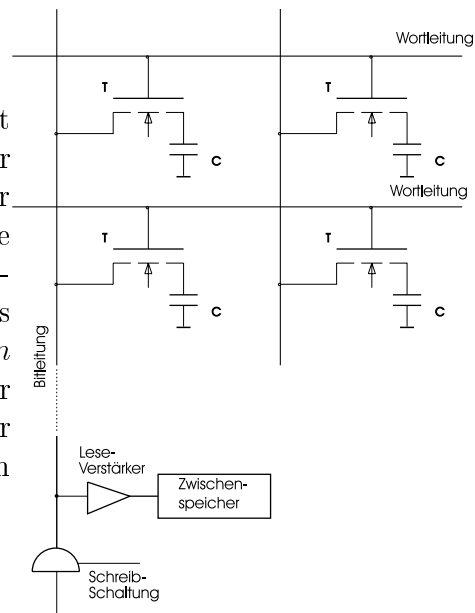
ϵ von SiO_2 :	$\epsilon \approx 5$
relative Dielektrizitätskonstante	$\epsilon_0 \approx 0,885 \cdot 10^{-11} [\text{Vs/Am}]$
Dicke des Dielektrikums ca 100 Atomlagen	$d \approx 100 \text{ \AA} = 10 \text{ nm}$
Fläche des Kondensators	$\pi \cdot D \cdot L \approx 15 (\mu\text{m})^2$
Daraus ergibt sich die Kapazität :	$C \approx 50 \cdot 10^{-15} \text{ F} = 50 \cdot \text{fF}$

Um eine RC-Zeit von 2 ms zu erreichen, ergibt sich für den Mindestperrwiderstand R des FETs :

$$R \text{ aus } R \cdot C : \quad R = 4 \cdot 10^{10} \Omega = 40 \text{ G}\Omega !!$$



Die dynamische Speicherzelle besteht aus einem kleinen Kondensator. Der Platz für diesen Kondensator ist sehr begrenzt - es sollen möglichst viele bits auf einem Chip gespeichert werden - daher verwendet man ein kleines Loch ($\approx 1\mu m$ Durchmesser, $\approx 5\mu m$ tief). Über den Transistor kann der Kondensator je nach Ansteuerung der Bitleitung ausgelesen und beschrieben (geladen) werden



3.4.1.2 ROM

Unter den ROMs gibt es drei unterschiedliche Verfahren

Bei den unterschiedlichen Verfahren ist zu bedenken, dass der ursprüngliche Vorteil des ReadOnly-Speichers mit zunehmender Erleichterung des Schreibens immer mehr verloren geht.

ROM MaskenROM : Die Information wird bei der Herstellung in Form von Masken im Chip gespeichert

Vorteil : einfache Technik, schnell; preisgünstig bei großen Stückzahlen

Nachteil: Herstellung einer speziellen Maske kostet Geld und Zeit

PROM Information wird durch Durchbrennen einer Sicherung (fusible Link) eingespeichert

Vorteil : vor Ort programmierbar, Einzelstücke möglich

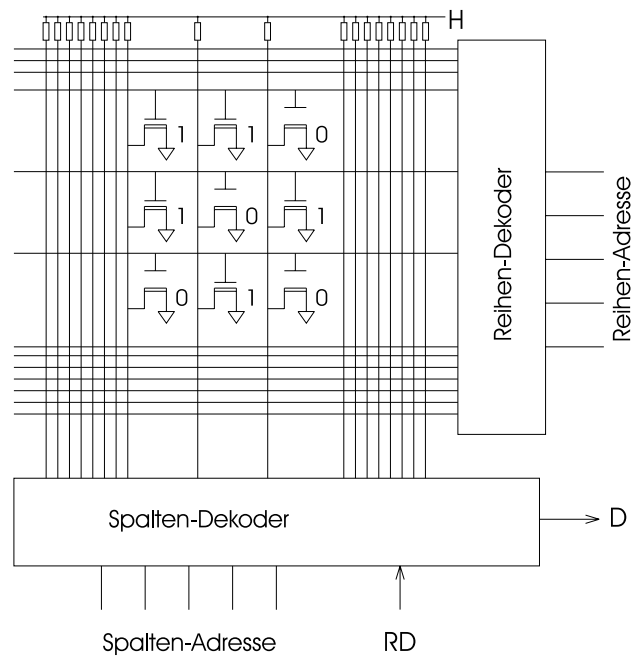
Nachteil: diffiziles Programmieren; geringe Kapazitäten; nur einmal programmierbar

- EPROM ...** Information wird durch die Ladung auf einem "**Floating Gate**" gespeichert und kann meist wieder neu programmiert werden. Die Ladung wird über den **Nordheim-Fowler-Tunneleffekt** durch das im Normalfall isolierende SiO_2 (Glas) auf das floating Gate gebracht. Es gibt unterschiedliche Typen, die sich im Wesentlichen durch die Art der Neuprogrammierung unterscheiden, wobei die Bezeichnungen häufig nicht einheitlich sind
- EPROM Löschen der Information mit Hilfe von UV-Licht durch ein Quartzfenster (ca. 0,5h); preiswerte Alternative ohne Quartzfenster (nicht löschar)
 - EEPROM elektronisches Löschen der Information
 - EAROM : elektronisches Verändern der Information
- Vorteil: leicht und mehrfach programmierbar
 Nachteil: Lesegeschwindigkeit langsam; Information hält sich nur ca. 10-20 Jahre auf dem floating Gate.
- Flash-Memory** Das Flash-Memory (Flash Blitz) verbindet den schnellen Datenzugriff des SRAMs mit der sicheren Speicherung in einem EEPROM. Die Speicherzelle besteht aus einer SRAM-Zelle und einer EEPROM-Zelle. Der Zugriff erfolgt immer über die SRAM-Zelle. Sollen die Daten sicher gespeichert werden, kann der Inhalt des SRAMs in das Floating Gate des EEPROMs geschrieben werden und im Bedarfsfall das SRAM aus dem EEPROM wieder restauriert werden.

3.4.1.2.1 ROM, MaskenROM

Die Speicherzelle eines Masken-ROMs besteht aus einem einzigen Transistor (FET). Die Information steckt in der Art des Gates: Ein Signal am Gate (Wortleitung) kann entweder den Transistor leitend schalten oder auch nicht. Dadurch wird beim Lesen die Bitleitung mit L verbunden oder nicht.

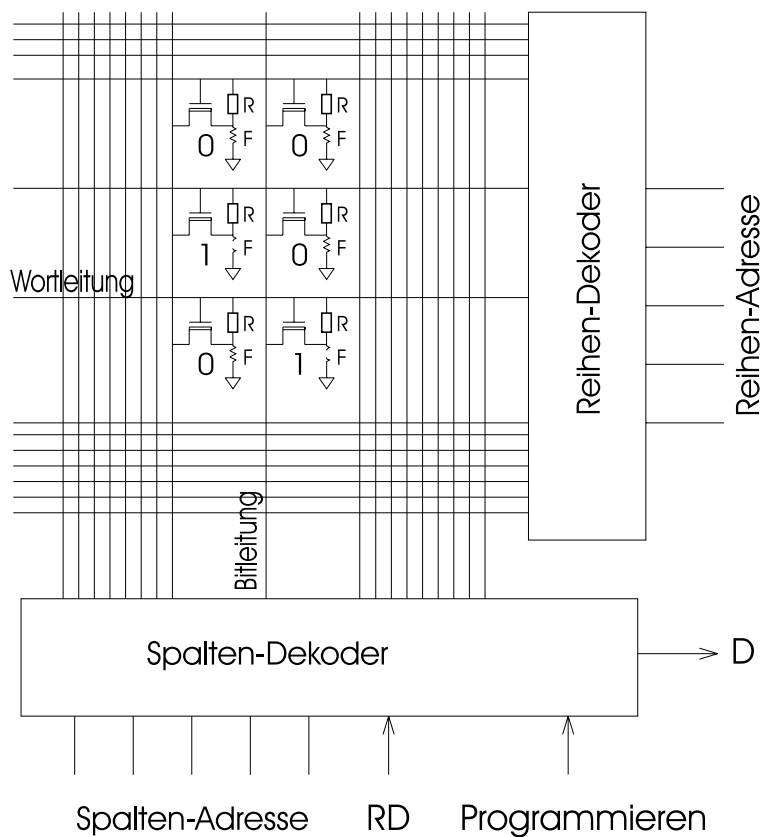
Auf diese Weise erhält man sehr einfache, preisgünstige und schnelle Speicher.



Da die Information bei der Herstellung der Maske bereits gespeichert werden muss, treten hohe einmalige Kosten auf, die sich aber bei hohen Stückzahlen schnell auf viele Chips verteilen und dann nicht mehr ins Gewicht fallen. Nachteilig ist aber, dass wegen der Herstellung der Maske lange Lieferzeit für die erste Serie anfallen, die sich bei einer Änderung des Inhaltes weiter verlängern.

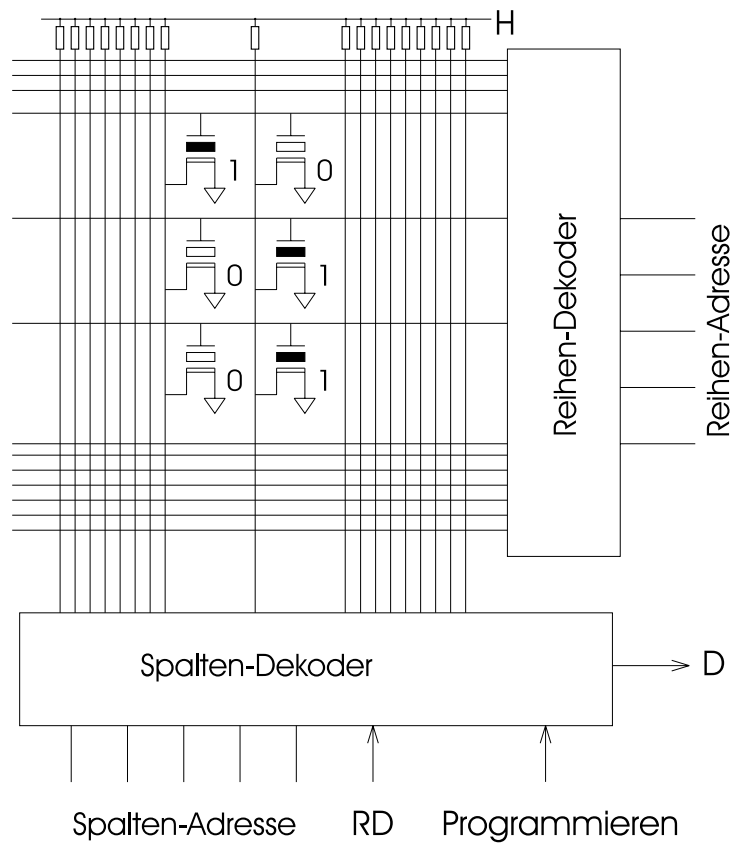
3.4.1.2.2 PROM, Programmierbares ROM

Die Speicherzelle besteht aus einem Spannungsteiler aus Widerstand (R) und durchbrennbarer Verbindung (Fusible Link). Beim Programmieren (Schreiben) wird über die Wortleitung der Transistor geöffnet und durch einen starken Stromimpuls auf der Bitleitung die Sicherung (F), die später eine 1 enthalten soll, durchgebrannt. Beim Lesen wird wieder der Transistor mit einem Signal auf der Wortleitung geöffnet, dadurch kann man auf der Bitleitung entweder eine 0 Lesen (Sicherung ist intakt) oder eine 1 (Sicherung ist durchgebrannt und die Verbindung nach L ist unterbrochen)



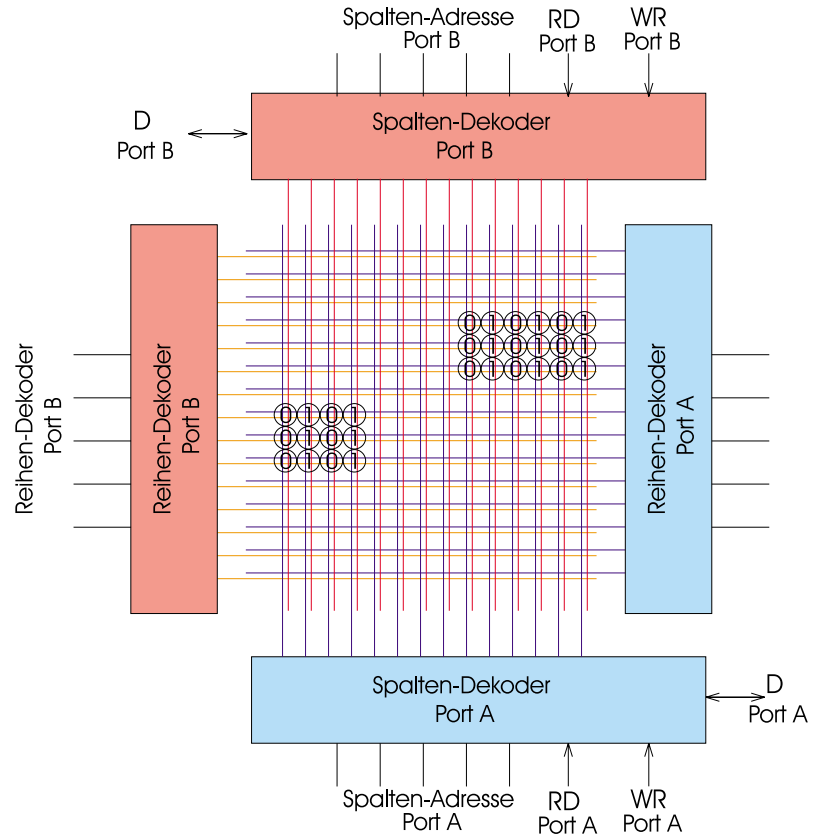
3.4.1.2.3 EPROM,EEPROM,EAROM

Die Speicherzellen nach dem Nordheim-Fowler-Effekt arbeiten ähnlich wie die Zellen des Masken ROMs. Die Durchschaltfähigkeit wird hier jedoch nicht bei der Herstellung erzeugt, sondern bei der Programmierung durch Aufbringen einer Ladung auf das Floating Gate. Das Floating Gate ist vom Rest der Schaltung rundum durch eine hochisolierende SiO_2 -Schicht getrennt (RC-Zeit ca 10-20 Jahre !). Diese Schicht kann aber durch ein starkes elektrisches Feld etwas durchlässig gemacht werden und damit das Floating Gate geladen oder entladen werden.



3.4.1.3 dual ported Memory

Da ein Memory normalerweise nur ein Tor (Port) hat, über den der Datenverkehr gesteuert wird (Anlegen von Adresse, Daten und Steuersignale WR und RD), kann immer nur eine einzige Speicherzelle angesprochen werden. Ein Verschieben oder Kopieren von Daten von einem Memoryplatz zu einem anderen ist daher nur über Lesen der Daten, Zwischenspeichern in einem Register, Anwählen des Zielbereichs und Schreiben des Zwischenspeichers möglich - ein erheblicher Zeitverlust !



Auch wenn zwei Benutzer unabhängig von einander auf das Memory zugreifen wollen, müssen sie den Zugang zum Port regeln. Dieser Fall tritt zum Beispiel beim Video-Controller auf. Hier muss das Programm das darzustellende Bild im Speicher ablegen und unabhängig davon muss der Video-Controller das Bild vom Memory zum Monitor transportieren.

In beiden Fällen hilft ein dual-ported-Memory. Diese Baugruppe enthält zwei Ports für Adresse, Daten und Kontrolleitungen RD und WR mit Adressdekodern und Row- und Column-Leitungen. Über jedes Port können unabhängig und gleichzeitig zwei verschiedene bits über getrennte Reihen und Spalten gelesen und geschrieben werden. Ausnahme: Über beide Ports soll gleichzeitig auf ein und dieselbe Zelle geschrieben werden.

3.4.2 Zugriff über Inhalt, Assoziativspeicher

Die bits des Speichers sind in einzelne Bereiche untergliedert. Zu jedem bit des Speichers ist ein Vergleicher (Komparator, XOR) zugeordnet. Der Zugriff erfolgt über ein Suchwort. Dieses Suchwort wird mit Hilfe der Komparatoren mit jedem der Speicherinhalte verglichen. Findet sich das Suchwort irgendwo im Memory, wird ein Treffer (Hit) gemeldet. Mit dem Hit-Signal wird der Bereich mit dem Treffer zugänglich. Der Zugriff erfolgt also über den Speicherinhalt (Content-Adressed-Memory CAM)

Vorteile bringt der Assoziativspeicher wenn ein Speicherbereich nach irgendwelchen Daten durchsucht werden soll (z.B. in Datenbankanwendungen). Bei einem Zugriff über Adresse muss Speicherplatz für Speicherplatz nacheinander (sequentiell) angesprochen, gelesen und mit dem Suchbegriff verglichen werden. Das ist ein hoher Zeitaufwand, wenn große Datenmengen durchforstet werden müssen. Bei einem Assoziativspeicher erfolgt der Vergleich aller Daten gleichzeitig !

Leider ist der Aufwand für den Assoziativ-Speicher so hoch, dass heute noch nicht genügend große und preiswerte Bausteine zur Verfügung stehen. Eine Ausnahme bildet das Cache-Memory. Das ist vom Prinzip her auch ein Assoziativ-Speicher, jedoch speziell für die Anwendung als schneller Zusatzspeicher ausgelegt.

3.4.2.1 Cache

In jedem System setzt man gerne Baugruppen ein, die entweder wenig Strom verbrauchen, schnellen Zugriff erlauben oder sonst einen Vorteil bieten. Leider sind solche Baugruppen meist teurer, aufwendiger oder haben einen anderen Nachteil, der einen Einsatz im großen Umfang verbieten.

Z.B. finden Sie im PC den großen preiswerten **Hauptspeicher** aus DRAM-Bausteinen, würden aber gerne die wesentlich schnelleren aber leider teuren SRAM-Bausteine verwenden. Jetzt könnte man einen Teil des Speichers mit schnellen Bausteinen ausrüsten und darauf hoffen, dass die häufigsten Zugriffe auf den schnellen Bereich erfolgen. Das ist aber leider nicht zu gewährleisten.

Man kann aber meist davon ausgehen, dass einmal benutzte Daten demnächst wieder gebraucht werden (z.B. in Programmschleifen) oder dass im Memory benachbarte Daten (Datenblöcke) demnächst auch gebraucht werden. In dieser Hoffnung kann man also solche Daten im schnellen SRAM ablegen.

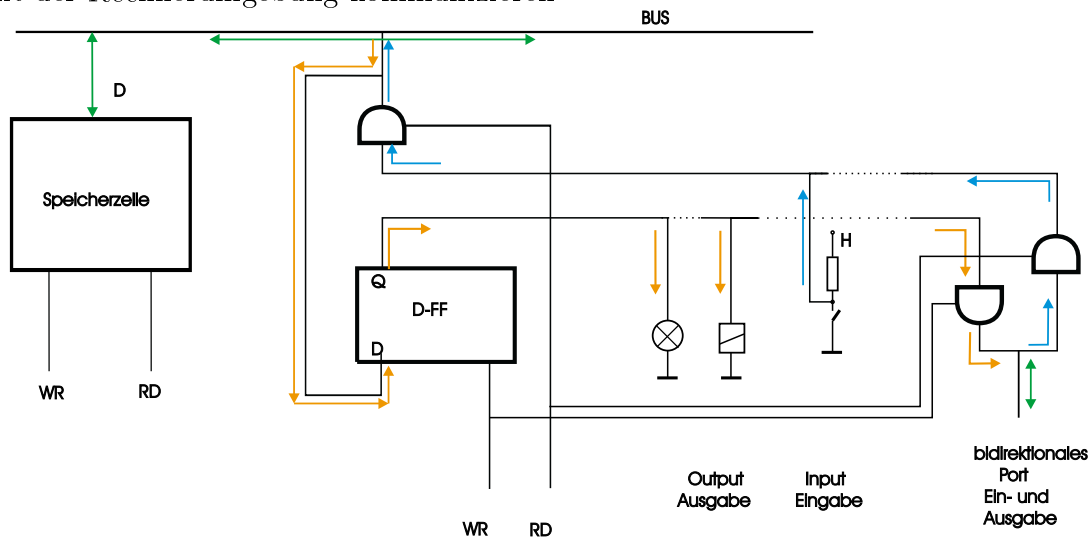
Hierzu dient das **Cache-Memory**. Dieses enthält ein schnelles SRAM sowie einige Vergleicher. Im SRAM legt man nicht nur die vermutlich benötigten Datenblöcke, sondern zu jedem Block auch die entsprechende Adresse der Daten im Hauptspeichers ab. Diesen abgelegten Adressen wird ein Komperator zugesellt. Zur Anforderung von Daten aus dem Speicher wird die entsprechende Speicheradresse angegeben. Diese

Überblick Memory (Speicher)

Zugriff über Adresse		Zugriff über Inhalt
<p>RandomAccess (RAM) : Eine Adresse wählt eine Reihe (ROW-Adresse) und eine Spalte (Column-Adresse) an. Am Kreuzungspunkt von Reihe und Spalte sitzt eine Speicherzelle.</p>		
Schreib-Lese-Speicher (RAM)		Assoziativ-Speicher Contents-Adressed-Memory (CAM)
<p><u>Vorteil:</u> Schreiben und Lesen möglich</p>	<p><u>Nur-Lese-Speicher (ROM)</u> <u>Vorteil:</u> Kein Datenverlust bei Netzausfall kein ungewolltes Überschreiben</p>	<p>Viele Komperatoren vergleichen Suchbegriff mit Speichereinhalten Bei Treffer (Hit) wird passender Bereich ausgegeben</p>
Typenspektrum		
SRAM	DRAM	EPROM EEPROM EAROM
FF	C (Kondensator)	Nordheim-Fowler
<p><u>Vorteil:</u> große Typenvielfalt z.B.: sehr schnell; oder wenig Strom</p>	<p><u>Vorteil:</u> preisgünstig</p>	<p><u>Vorteil:</u> schnell und leicht zu programmieren</p>
<p><u>Nachteil:</u> teuer</p>	<p><u>Nachteil:</u> lange Lieferzeiter große Stückzahlen</p>	<p><u>Nachteil:</u> Datenverlust nach ca. 10 Jahren; relativ langsam</p>
Sonderformen :		
<p><u>Flash-Memory</u> Kombination aus SRAM und EAROM schneller Zugriff über SRAM speichern in EAROM-Zelle</p>	<p><u>Dual-Ported-Memory</u> gleichzeitiger Zugriff auf zwei Speicherzellen möglich</p>	<p><u>Anwendung:</u> Cache Datenbank</p>

4 Input-Output-Port (I/O-Port)

Aufgabe : Tor zur Außenwelt (Peripherie) Über die I/O-Ports kann der Rechner mit der Rechnerumgebung kommunizieren



Das Bild zeigt drei Möglichkeiten, Daten von außen mit dem BUS zu verbinden und entweder in einer Speicherzelle eines Registers abzulegen oder aus der Speicherzelle nach außen zu legen. Die Funktionsweise ist ähnlich der der Speicherzellen. Beim Ausgabe-Port gelangt ein bit vom BUS zum D-Eingang eines D-FFs und kann mit einem WR-Steuersignal im DD-FF gespeichert werden. Am Ausgang des FFs kann dann jedes beliebige Gerät angeschlossen werden, das mit einem H-L-Pegel gesteuert werden kann. Eingezeichnet ist ein Lämpchen und ein Relais.

Ein H-L-Pegel, welches eine Signalquelle erzeugt, hier eine Schalterstellung, wird vom Eingabe-Port mit einem RD-Signal durch das Tristate-Gatter auf den BUS gelegt und kann von dort in einem Register abgelegt werden.

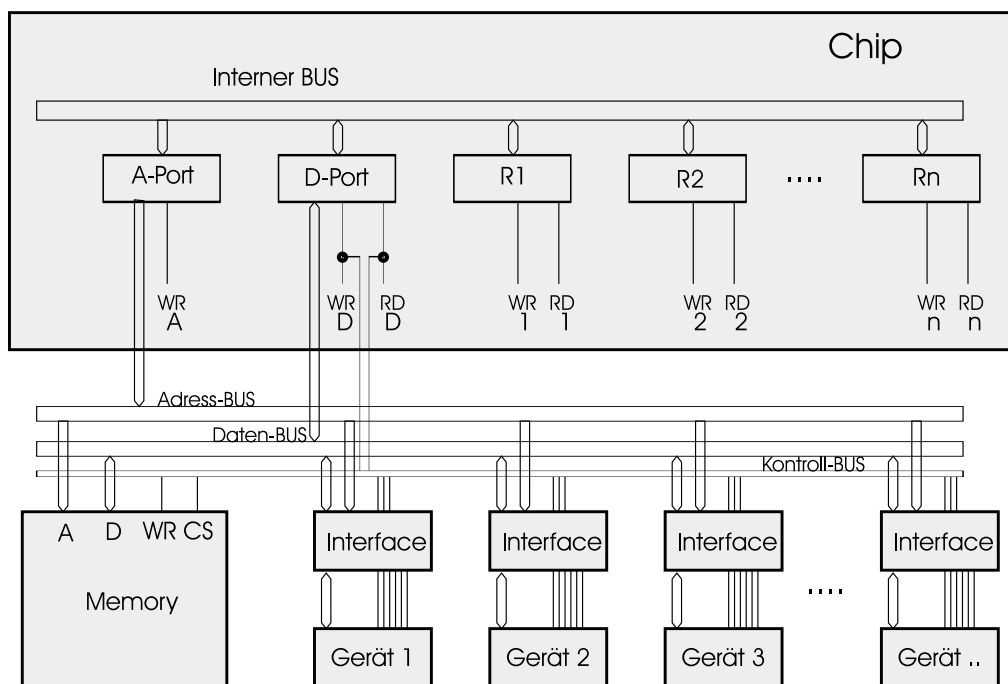
Das bidirektionale Port erlaubt den Anschluss von Geräten, die sowohl Daten empfangen als auch ausgeben können. So können von einem Plattenlaufwerk Daten auf eine Platte geschrieben als auch Daten von einer Platte gelesen werden. Hier wird der Datenfluss nach außen durch zwei Tri-State-Gatter geregelt, die die Datenrichtung entsprechend der RD- WR-Signale steuern. Auch das außen angeschlossene Gerät muss entsprechend der Steuersignale seinen Datenstrom regeln.

Wie schon beim Register beschrieben können gleichzeitig mehrere bits ausgegeben oder eingelesen werden. Natürlich lassen sich auch mehrere Ports anschließen.

In der Praxis findet man häufig zwei spezielle Ports, das **Adressport** (A-Port) und das **Datenport** (D-Port). Das A-Port ist ein reines Ausgabe-Port und das D-Port ein bidirektionales Port.

Diese beiden Ports sind häufig die einzigen Zugänge zu dem BUS, und alle Geräte aber auch das Memory müssen über diese beiden Ports bedient werden. Der BUS ist innerhalb eines Chips (z.B. dem Mikroprozessor oder Mikrocontroller), man nennt ihn daher auch den **internen BUS**. Alle angeschlossenen Geräte hängen über einen gemeinsamen Datenkanal am A- und D-Port. Diesen gemeinsamen Datenkanal nennt man daher den BUS. Wenn man vom BUS spricht meint man daher fast immer den externen Datenkanal von Adress- und DatenPort. Zusätzlich zu den Adress- und Datenleitungen gibt es noch einige Kontrollleitungen, die den Datenverkehr steuern, die wichtigsten sind die RD- und WR-Leitung.

4.1 Interface



Ein Interface verbindet ein Gerät oder das Memory mit dem (externen) BUS. Aus der Adresse muss das Interface ermitteln, ob das ihm zugeordnete Gerät angesprochen werden soll. Dies erledigt ein **Adressdekoder**. Manche Geräte belegen mehrere Adressen. Z.B. muss ein Plattenspeicher Daten lesen und schreiben können, aber hier muss auch der Schreiblesekopf positioniert werden. Die unterschiedlichen Funktionen müssen auch vom Interface bereitgestellt werden. Auch die Daten müssen gegebenenfalls angepasst werden; Zwischenspeicherung, Verstärkung, Pegelanpassung, ..

Adressraum

n bits des Adressbusses spannen einen Adressraum von 2^n Adressen auf. Wird eine Adresse angewählt, die sowohl ein Gerät als auch ein Platz im Memory ansprechen könnte, muss man dafür sorgen, dass es nicht zu unerwünschten Datenzuweisungen kommt. Dazu haben die Rechner meist eine Art zusätzliche Adressleitung (MEM/\overline{IO}). Diese Leitung wird vom Mikroprozessor mit 0 aktiviert, wenn ein Gerät und mit 1, wenn das Memory angesprochen werden soll. Mit dieser Leitung kann direkt das Memory über den CS-Eingang aktiviert (enable) oder deaktiviert (disable) werden.

Ebenso müssen die Interface der Geräte die Leitung MEM/\overline{IO} beachten und nur Steuersignale an das Gerät weitergeben, wenn diese Leitung auf 0 liegt.

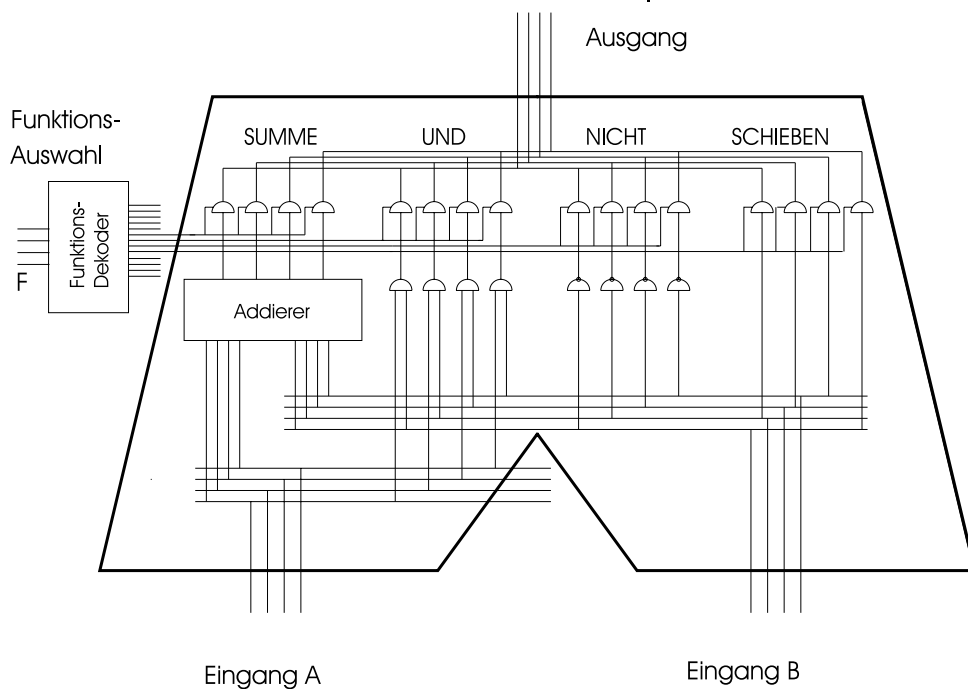
Memory mapped IO

Da der Adressraum, des Memories sehr groß ist, ist es keine große Einschränkung, wenn ein Bruchteil des Memories für Geräte benutzt wird. Im Plan (map) des Memories wird daher ein Teil ausgeblendet, und die freigewordenen Adressen können für die Geräte verwendet werden.

5 Arithmetic and Logic Unit (ALU)

Zur Verknüpfung und zur Modifikation der Daten dient die **Arithmetic and Logic Unit, (ALU)**. Wir unterscheiden arithmetische und logische Operationen. Eine Auswahl zeigt die Tabelle :

arithmetische Operationen	logische Operationen
addieren	bitweise UND
subtrahieren	bitweise ODER
Multiplikation mit 2 = schieben eins nach links	bitweise NICHT
Division durch 2 = schieben eins nach rechts	bitweise XOR
schieben n Positionen nach rechts	
schieben n Positionen nach links	
Inkrement (+1)	
Dekrement (-1)	



Daten an den Eingängen A und B werden miteinander verknüpft. Alle Ergebnisse der verschiedenen Verknüpfungen stehen bereit. Eine ALU-Funktion (F) wählt über den Dekoder ein Tristate-Gatter aus, das dann das Ergebnis (E) der ausgewählten Funktion zum Ausgang leitet. (Im Bild sind nur 4 Funktionen eingezeichnet).

$$E = A \text{ (F) } B$$

Das Ergebnis einer Operation kann zusätzlich unterschiedliche Bedingungen erfüllen: So kann die Summe (Differenz) positiv, negativ oder null sein; es kann ein Übertrag

entstehen oder beim Schieben links (rechts) ein bit (Cy-bit) herausfallen. Diese Bedingungen bezeichnet man als **Condition-Flag** und lassen sich zur Steuerung des Programmablaufs einsetzen. In der ALU können die Flags zur Berechnung neuer Daten benutzt werden, z.B. $A+B+Cy$ oder beim Schieben wieder eingefüttert werden.

Die folgende Tabelle zeigt einige Flags .

equal, zero	gleich,ungleich
sign	positiv,negativ
carry	Cy, Übertrag
parity	gerade-ungerade Parität
halfcarry	

6 Ablaufsteuerung

Wie schon beschrieben, besteht eine Aktion in einem Rechner aus Datentransporten von einer Datenquelle (Source) zu einem Datenziel (Destination) teilweise mit Umweg über eine Einheit zur Modifikation bzw. Verknüpfung von Daten, z.B. der ALU, Inkrementer usw. Die Aktion wird gesteuert durch Signale (z.B. RD und WR), die Datentore öffnen.

Es sei nochmal daraufhingewiesen, dass Schaltungen und Lösungswege, wie sie hier vorgestellt werden, mehr "didaktische" Schaltungen sind, und dem Studenten ein Gefühl für die Funktionsweise geben soll. Die tatsächlich realisierten Verfahren können erheblich abweichen.

Nehmen wir an, im Rechner sei die folgende Aufgabe durchzuführen :

Addiere den Inhalt von Register R3 zum Inhalt von Register R4 und speichere das Ergebnis im Memory auf Speicherplatz 230

oder in Kurzform: ADD R3,R4,230

Da der Rechner nur Digitalzahlen versteht, enthält die

die Aufgabe	ADD	R3	R4	230
die Zahlen	..5..	..3..	..4..	..230..

Nehmen wir an, R3 enthielte eine 17 und R4 eine 31, dann ergibt sich der folgende Ablauf (zunächst beachten wir nur die Zeilen 3 bis 6):

Zeile	Steuersignale		interner BUS	Daten BUS	Adress BUS
1	RD1	WRAD	537		
2	RDD	WR2	.5.3.4.230.		
3	RD3	WRA	17		
4	RD4	WRB	31		
5		WRAD	230		230
6	ADD	WRD	48	48	

Jede Zeile der obigen Tabelle beschreibt eine minimale Aktion. Diese minimale Aktion nennt man eine **Mikroinstruction** (μI). Zur Ausführung der gesamten Aufgabe, muss eine Folge von μI (die Zeilen 3 bis 6 der Tabelle oben) ausgeführt werden. Diese Folge nennt man ein **Mikroprogramm** (μP) oder auch **Maschinenbefehl**. Die Ablaufsteuerung sorgt für die korrekte Abfolge der Steuersignale. Ähnlich wie bei einer Waschmaschine die Signale zum Wassereinlauf, zum Heizen oder zur Zugabe der Waschmittel werden in der Ablaufsteuerung unter anderem die RD- WR-Steuersignale erzeugt. Jeder Zeile in der Tabelle (μI) ist ein Speicherplatz in einem Speicher, dem **Mikroprogramm Speicher** zugeordnet (siehe auch Bild unten). Dabei entspricht jedem RD- und WR-Steuersignal ein bit im Speicherplatz. Gegebenenfalls können auf dem Speicherplatz weitere zusätzliche Daten,

Adressen oder sonstiges untergebracht sein. Ist ein Speicherplatz angesprochen, wird durch eine 1 des bits das entsprechende Steuersignal aktiviert und die entsprechende Mikroinstruction ausgelöst. Die Auswahl der Speicherzelle geschieht durch eine Adresse im **Mikroprogramm-Adressregister**. Anstatt jedem Steuersignal ein bit zuzuordnen kann platzsparender von jeder Datenquelle und jedem Datenziel eine Adresse gespeichert werden. Gibt es beispielsweise 32 Datenquellen können die 32 RD-Leitungen aus einem 5-bit-Code ($2^5 = 32$) durch einen Dekoder erzeugt werden.

Um ein Mikroprogramm auszuführen, müssen nacheinander mehrere Plätze im Mikroprogramm Speicher angesprochen werden. Ein Oszillator erzeugt im Bereich von MHz bis zu einigen GHz Taktimpulse. Mit jedem **Taktimpuls** wird das Mikroprogramm-Adressregister weitergeschaltet und damit eine neue Mikroinstruction ausgelöst. Dieses Weiterschalten kann auf unterschiedliche Weise erfolgen. Einfach kann das Adressregister als Zähler ausgelegt und mit jedem Taktimpuls um 1 erhöht werden. Damit entsteht eine sequentielle Abfolge der Mikroinstructionen. Einen Zähler erhält man auch, wenn man bei jedem Taktimpuls den Inhalt des Adressregisters über einen Inkrementer um 1 erhöht und wieder in das Adressregister einspeichert. Universeller lässt sich der Ablauf steuern, wenn in jedem Speicherplatz zusätzlich zu den Steuersignalen auch die Adresse für die nächste Mikroinstruction abgelegt ist, die dann mit dem Taktimpuls in das Adressregister gespeichert wird. Zusätzlich können, z.B. von Conditionflags gesteuert, mit einem Multiplexer unterschiedliche Adressen geladen werden. Auf diese Weise sind neben dem sequentiellen Ablauf auch Sprünge und Schleifen möglich.

Der Speicher mit den Mikroprogrammen ist im Normalfall ein ROM. Damit sind die Maschinenbefehle fest. Die Gesamtheit aller realisierten Maschinenbefehle nennt man den **Befehlssatz** einer Anlage. In Ausnahmefällen besteht der Befehlsspeicher aus einem RAM. In diesem Fall können die Maschinenbefehle an ein Problem angepasst, neue Befehle kreiert oder Befehle einer anderen Anlage nachgebildet werden.

Die gesamte Steuerung befindet sich in der **Zentraleinheit (CentralProcessingUnit; CPU)**. In die CPU werden die Taktimpulse eingespeist und am Ausgang kommen der Adress-, der DatenBUS sowie einige Steuersignale (nicht eingezeichnet) heraus.

6.1 Maschinenprogramm

Die Zentraleinheit stellt viele unterschiedliche Maschinenbefehle zur Verfügung. Zur Durchführung einer universellen Aufgabe aber, ist eine Abfolge vieler Maschinenbefehle notwendig. Eine solche Abfolge nennt man das **Maschinenprogramm**. Die Maschinenbefehle stehen im Normalfall im Hauptspeicher und müssen erst von dort gelesen werden. Diese Holen eines Maschinenbefehls erfolgt im μ -Programm in den Zeilen 1 und 2 der Tabelle oben: Dazu nehmen wir an, dass der Maschinenbefehl "ADD R3,R4,230" (.5.3.4.230.) im Hauptspeicher auf dem Platz 537 steht. Dann muss zunächst die Adresse dieses Speicherplatzes (537) dem Hauptspeicher mitgeteilt werden (Zeile 1) und anschließend in der 2. Zeile der Inhalt des Hauptspeichers (.5.3.4.230.) gelesen werden. Diesen Vorgang bezeichnet man mit **Instruction fetch**. In der CPU braucht man zwei weitere Register: Ein Register enthält die Adresse (in unserem Fall haben wir dafür R1 verwendet) und in einem zweiten Register (hier R2) wird der Maschinenbefehl gespeichert.

Hier erkennt man, dass die Parameter für den Befehl ADD R3,R4,230 nach dem Instruction fetch im Befehlsregister R1 stehen. Daher können diese Daten im Mikroprogramm Speicher entfallen, denn das Mikroprogramm kann nun die Adressen für R3, R4 und dem Speicherplatz 230 sowie die ALU-Funktion ADD (=5) aus dem Befehlsregister holen.

Nach Ablauf der Zeilen 1-6 ist der Maschinenbefehl abgeschlossen, jedoch muss danach der nächste Maschinenbefehl geholt werden. Dazu muss das Register R1 (**Programmzähler (Programcounter; PC)**) auf den nächsten Maschinenbefehl zeigen. Das μ P muss daher noch ergänzt werden (nicht in der Tabelle eingetragen) um den PC (Register R1) zu inkrementieren (um 1 erhöhen). Damit erhält man einen sequentiellen Ablauf der Maschinenbefehle. Zur Ausführung von Programmsprüngen muss lediglich Register R1 auf die Sprungadresse geladen werden.

Achtung : Unterscheiden Sie sorgfältig zwischen der μ I, die im Mikroprogramm Speicher steht, und die in einem Taktzyklus ausgeführt wird und einem Maschinenbefehl, der im Memory steht und viele Taktzyklen benötigt, da er erst vom Mikroprogramm geholt und ausgeführt werden muss.

6.2 CISC - RISC

In einer CPU, wie sie oben vorgestellt wurde, lassen sich beliebige Maschinenbefehle realisieren, da sie aus einem Mikroprogramm bestehen. Teilweise kann das Mikroprogramm auch Schleifen enthalten, so dass ein solcher Maschinenbefehl sehr komplex sein kann. Rechner dieser Art bezeichnet als Rechner mit einer **CISC-Struktur (Complex Instruction Set Computer)**. Leider benötigt ein solcher Maschinenbefehl entsprechend viele Taktzyklen und ist daher langsam.

Bei der Analyse von Programmen hat sich gezeigt, dass die meisten komplexen Befehle nur wenig oder sogar überhaupt nicht benutzt wurden. Das hatte seine Ursache darin, dass Programmierer nicht alle Befehle kannten oder beherrschten, bzw. dass für Compiler die komplexe Vielfalt der Maschinenbefehle zu aufwendig war. Aus diesem Grund hat man überlegt, die Anzahl der Maschinenbefehle auf einige wenige zu beschränken, diese aber so zu realisieren, dass sie in ganz wenigen oder sogar nur in einem einzigen Taktzyklus ausführbar sind. Dieses erfordert allerdings einen erhöhten technologischen Aufwand. Diese sehr aufwendige Rechnerstruktur bezeichnet als **RISC-Struktur (Reduced Instruction Set Computer)**.

Lange Zeit gab es einen Wettkampf zwischen beiden Strukturen. Heute finden wir meist CISC-Rechner die einige wenige Befehle haben, die in einem Zyklus abgearbeitet werden.

6.3 Methoden zur Geschwindigkeitssteigerung

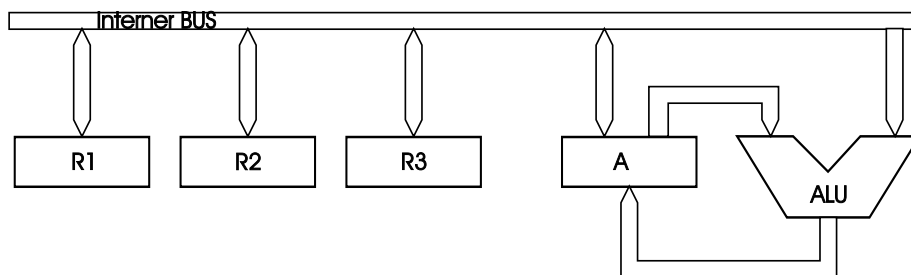
Prinzipiell gibt es zwei Verfahren, die die Verarbeitungsgeschwindigkeit steigern. Zum einen lässt sich die **Taktfrequenz erhöhen**, jedoch sind hier meist die Möglichkeiten stark beschränkt. Zum einen steigt der Energieverbrauch mit allen Nebenwirkungen (Hitze, Wärmeabfuhr, Ladung der Batterie ...), aber vor allem begrenzen physikalische Effekte eine weitgehende Steigerung der Taktfrequenz. Die zweite Möglichkeit ist das **Parallelisieren**, viele Aktionen werden gleichzeitig durchgeführt. Im folgenden sollen einige Verfahren zur Parallelisierung vorgestellt werden.

6.3.1 Mehrfach-BUS

Ein Engpass ist, wenn alle Register an einem einzigen internen BUS angeschlossen sind. Unterschiedliche Daten müssen daher zu unterschiedlichen Zeiten transportiert werden. Abhilfe bringt ein Mehrfach-BUS. Jedes Register kann an mehrere Datenkanäle angeschlossen werden, wenn nur für jeden Datenkanal ein eigenes RD- bzw. WR-Signal Daten vom entsprechenden Kanal holt bzw auf den Kanal legen soll.

Das kann zu einem großen Aufwand führen, da über den internen BUS - wie aus der Tabelle ersichtlich - viele unterschiedliche Daten transportiert geleitet werden. Um den Aufwand in Grenzen zu halten, gibt man den Vorteil eines sogenannten **orthogonalen Befehlssatz** auf. Von einem orthogonalen Befehlssatz spricht man, wenn alle Register gleichberechtigt und jeder Befehl mit jedem Register ausgeführt werden kann. Man lässt vielmehr für jedes Register nur eine begrenzte Möglichkeit der Anwendung zu : So verknüpft beispielsweise die ALU ein Register immer nur mit einem einzigen speziellen Register, das man dann **Akkumulator** nennt. Der Befehl "ADD R3" würde dann den Inhalt von R3 zum Inhalt von dem Akkumulator addieren und das Ergebnis wieder im Akkumulator abspeichern. Diese Addition wird nur ein Taktzyklus dauern, da über den internen BUS zur ALU nur der Inhalt von

R3 transportiert werden muss. Der zweite Eingang und der Ausgang der ALU sind direkt mit dem Akkumulator verbunden. Der Akkumulator hat daher 3 Datenkanäle (einer zum internen BUS, einer zum zweiten Eingang und einen vom Ausgang der ALU).



Meist ist auch für den Programmcounter und das Befehlsregister nur ein einziges Register zu verwenden, die dann auch **PC** und **InstructionRegister** heißen.

Will man Daten aus dem Memory holen, kann man die Adresse des Speicherplatzes in einem Register ablegen und dieses Register als Zeiger verwenden. Diese Art der Adressierung bezeichnet man auch als indirekte Adressierung. Mit einem orthogonalen Befehlssatz ließe sich jedes Register als Zeiger einsetzen. Häufig werden auch hier Datenkanäle eingespart und nur wenige ausgewählte Register sind als Zeiger möglich.

Allgemein : Viele Datenkanäle erlauben parallelen (schnellen) Datentransport. Spezialregister reduzieren den Aufwand. Da nun nicht jeder Befehl mit jedem Register durchführbar ist, muss ein Programmierer bzw. der Compiler planen, für was er welches Register verwenden will.

6.3.2 Pipeline

Eine Pipeline entspricht einem Fließband. In einer Autofabrik dauert die Montage eines Autos ca. einen Tag, trotzdem läuft alle 5 Minuten ein fertiges Auto vom Band. Auch hier wird der Geschwindigkeitsgewinn durch Parallelarbeiten erreicht. Entlang dem Fließband befinden sich viele Stationen mit je einem Auto in unterschiedlichem Fertigungszustand. Am Anfang des Bandes ist das Auto noch in einem sehr unfertigen Zustand, während das Auto von Station zu Station wandert, wird es immer mehr ergänzt, bis es am Ende des Bandes praktisch komplett zusammengebaut ist. Hier wird also parallel an mehreren Autos gleichzeitig gearbeitet. Dabei kann jedes Auto durchaus verschieden sein, es müssen nur zu jeder Station das passende Bauteil zugeliefert werden.

Entsprechend arbeitet eine **Pipeline** in einem Computer. Auch hier arbeitet das Mikroprogramm μI nach μI ab bis der Maschinenbefehl nach vielen Taktzyklen beendet ist. Da jedoch an mehreren Stationen gleichzeitig gearbeitet wird, werden gleichzeitig

(parallel !) mehrere μ -Programme ausgeführt, die sich natürlich in unterschiedlichen Fertigungsstufen befinden. Teilweise kann pro Taktzyklus ein Maschinenbefehl beendet werden (bei RISC-Strukturen) sogar mehrere).

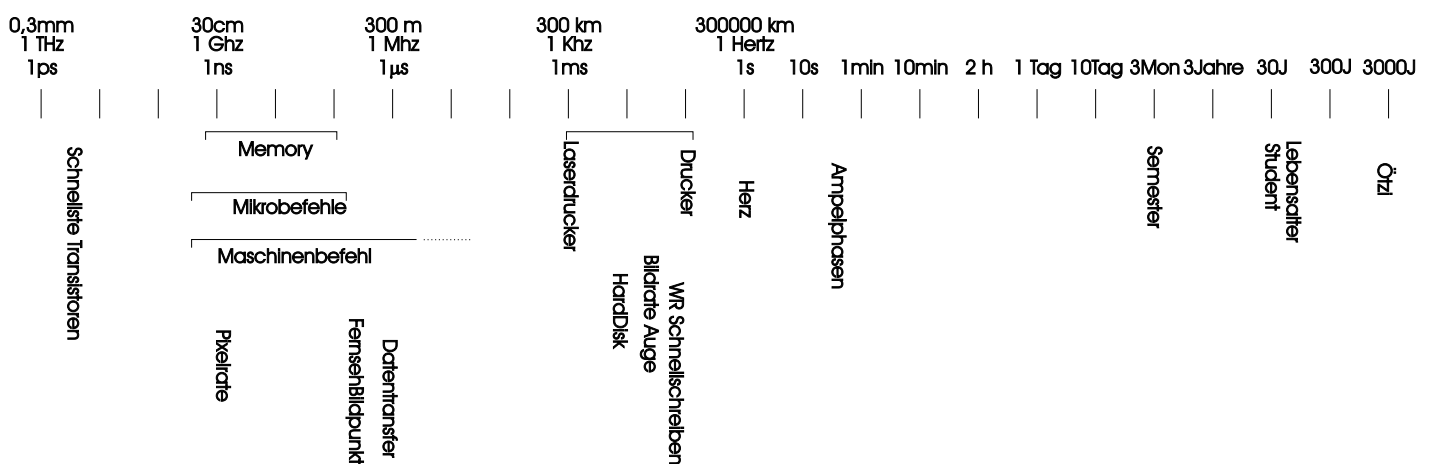
6.3.3 Parallelrechner

In einem Parallel-Rechner arbeiten mehrere Rechner gleichzeitig an einem Problem. Die Parallelschaltung erfolgt entweder, dadurch dass mehrere Rechner sich gemeinsame Datenbereiche teilen, wodurch ein Datenaustausch erfolgen kann. Auch können die Rechner über Datenkanäle miteinander verbunden sein. Solche **RechnerCluster** haben meist das Problem, dass die gemeinsam zu lösende Aufgabe auch auf Rechner verteilbar sein muss.

Anwendungen finden sich dort, wo sich ein Problem auf viele kleine Bereiche aufteilen lässt. Programmschleifen können gegebenenfalls von verschiedenen Rechnern einzelne Schleifendurchläufe übernommen werden. Bei der Wettervorhersage kann man unterschiedlichen Rechnern kleinere lokale Gebiete zuordnen. Bei der Analyse von Welt- raumdaten versucht man weltweit am Internet angeschlossene Rechner zu verknüpfen, und Pausenzeiten zu nutzen.

7 Zeitprobleme

In der Zeitskala sind von Teilstrich zu Teilstrich der Faktor 10 an Zeit, Frequenz und Weg des Lichts eingetragen. Die **Transportgeschwindigkeit** von Informationen auf Kabeln ist in etwa die halbe Lichtgeschwindigkeit, so dass eine Information innerhalb einer Nanosekunde ($1\text{ns}=1/1000.000.000\text{s}$) nur etwa 15cm weit kommt. Bei Hochgeschwindigkeitsrechnern muss daher neben den elektronischen Problemen mit HF (Hochfrequenztechnik) auch auf sehr kurze Drahtverbindungen geachtet werden.



Gehen wir bei einem Menschen von einer Taktfrequenz von 1 Hertz (etwa die Herzschlagfrequenz) aus, so haben wir eine mittlere Lebenserwartung von ca 9 Größenordnungen (10^9 Taktzyklen). Ein Computer mit einer Lebenserwartung von 10^9 Taktzyklen würde nur wenige Sekunden überleben.

Ein Mensch denkt meistens in einem Zeitbereich von weniger als 3 Größenordnungen: Unter 1 Sekunde kann ein Mensch kaum noch reagieren (Schrecksekunde), alles über 1 Minute kommt ihm unheimlich lang vor (Die Rotphase an einer Ampel dauert ca. 0,5 Minuten, das ist bereits für viele viel zu lang, um bei Gelb anzuhalten).

Die Anforderungen an die Rechengeschwindigkeit richten sich nach dem Zeitgefühl des Anwenders: Für den Menschen ist ein Ergebnis im Bereich seiner Reaktionszeit (0,2 bis 0,5 s) angenehm. Ergebnisse schneller als in 0,1s sind nicht erforderlich. Es ist dagegen lästig, wenn Ergebnisse erst nach 1min eintreffen. Eine Programmlaufzeit von 1 Stunde bis 8 Stunden kann man höchstens einmal am Tag verkraften (etwa während der Mittagspause oder über Nacht).

Die Leistungsbandbreite bei der Rechengeschwindigkeit von Computern reicht von wenigen MHz bis zu mehreren GHz (Faktor 1000 !!). Bei der Befehlsrate finden wir noch einen größeren Faktor, da hier RISC, CISC, Pipelining usw eine große Rolle spielen. Gehen wir insgesamt von einem Faktor 10000 aus, entspricht dieser

Leistungsunterschied dem Verhältnis von einem Spaziergänger (4km/h) zu einer Interstellar-Rakete (40000km/h). Dabei sind noch zusätzliche Faktoren wie etwa höhere Geschwindigkeit durch größere Wortbreite unberücksichtigt.

Die erforderliche Rechnerleistung sollte sich nach dem Problem richten. Soll ein vorgegebenes Problem in einer Sekunde gelöst sein, dann kann eine CPU, die mit 100MHz fährt, 100.000.000 Mikroinstruktionen ausführen. In der Praxis ergibt sich jedoch meistens das folgende Naturgesetz:

Die Problemstellung ergibt sich aus der Fähigkeit des Werkzeugs (Rechners)

Problemkreise :

Anhand der Zeitskala lassen sich einige Probleme aufzeigen:

1. Rechenzeit ist schnell gegenüber Anforderungszeit;
Zum Beispiel bei der Ausgabe von Text auf dem Drucker muss der Rechner nach jedem Zeichen warten, bis der Drucker für ein neues Zeichen bereit ist.
2. Es können seltene aber dringende Ereignisse eintreten:
Übernimmt ein Rechner Überwachungsaufgaben, so muß im Falle eines Fehlers innerhalb einer systembedingten Zeit reagiert werden (Reaktionszeit). Die Häufigkeit der Abfrage des zu überwachenden Zustandes richtet sich nach notwendigen Reaktionszeit. Tritt ein Ereignis selten oder fast nie auf (z.B. ein Netzausfall), muss aber im Falle des Auftretens sofort reagiert werden (Abspeichern wichtiger Daten), so muss der Rechner ständig den Zustand der Netzspannung beobachten und würde viel Zeit für andere Aufgaben verlieren. Auch dürfen andere Programme nicht länger dauern als die Reaktionszeit, oder sie müssen laufend unterbrochen werden
3. Rechenzeit ist langsam oder vergleichbar zur Anforderungszeit:
Beim Einlesen von Daten von der Magnetplatte ist häufig die CPU überfordert, die Daten schnell genug abzuholen und abzuspeichern.

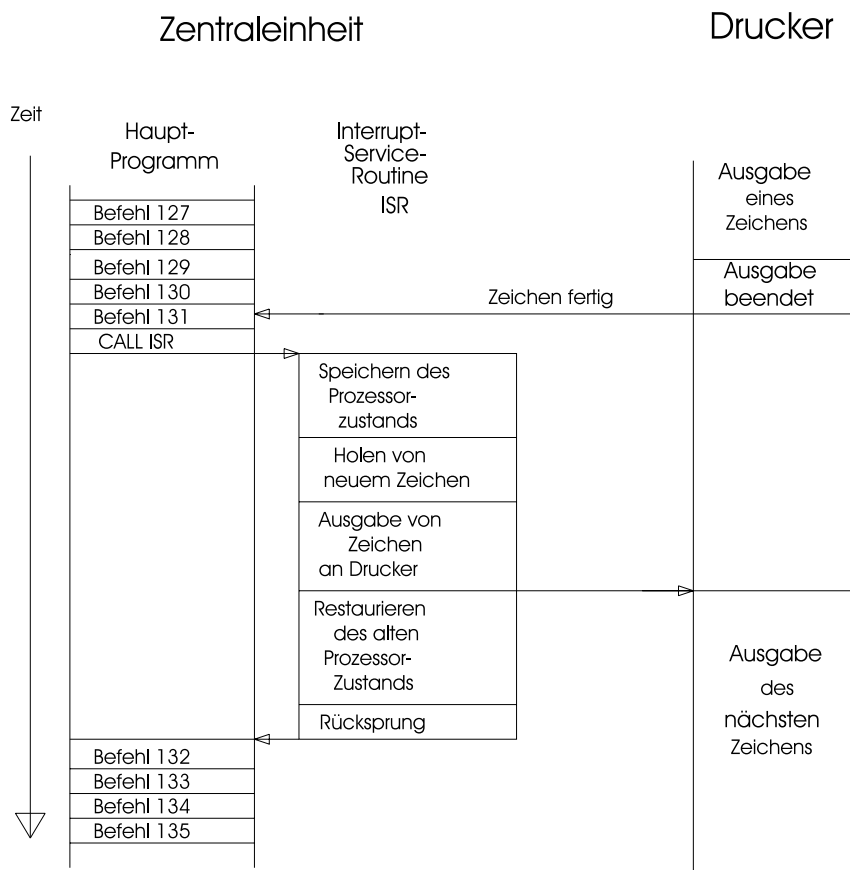
Folgende Lösungsmöglichkeiten bieten sich an :

1. (Rechenzeit schnell gegenüber Anforderungszeit):
In diesem Fall sollte die Wartezeit des Rechners ausgenutzt werden.
Hier ist die Interrupt-Einrichtung der CPU von Vorteil. Eine Interruptfähigkeit entspricht einer Klingel: Immer wenn ein Gerät bedient werden soll oder immer wenn ein wichtiges Ereignis eintritt, wird die Klingel betätigt, die CPU unterbricht ihre Arbeit und bedient das Gerät oder das Ereignis
2. (seltene aber dringende Ereignisse)
Auch hier ist die Interrupt-Einrichtung der CPU von Vorteil. Wie oben erwähnt,

wird, wenn ein wichtiges Ereignis die Klingel betätigt (Interrupt ausgelöst). Die CPU unterbricht ihre Arbeit und bedient das Ereignis.

3. (Rechenzeit langsam gegenüber Anforderungszeit):
 Hier muß eine Spezialeinheit die CPU unterstützen.
 Spezialeinheiten gibt es für viele zeitintensive oder rechenintensive Aufgaben: Bildschirmbearbeitung mit dem Videocontroller, ein Floatingpoint-Prozessor übernimmt Rechenarbeit und ein DMA-Controller tranferiert schnell Daten von und zum Memory

7.1 Interrupt



Die Interrupt-Einrichtung an der CPU erlaubt die Unterbrechung eines gerade ablaufenden Jobs. Durch ein Signal an dem Interrupteingang wird an geeigneter Stelle ein Programm unterbrochen. Dies geschieht am Ende eines Maschinenbefehls, oder bei langen Befehlen sogar innerhalb des Mikroprogramms. Dazu wird ein Unterprogrammaufruf (CALL ISR) eingeschoben und die CPU führt ein Unterprogramm aus, um den unterbrechenden Anlass zu bedienen (**Interrupt-Service-**

Routine, ISR).

Da nach der ISR die CPU ihre unterbrochene Arbeit wieder fortsetzen soll, muss der Status der CPU zu Beginn der ISR gespeichert werden. Am Ende der ISR muss der alte Status der CPU wieder restauriert werden, damit die CPU ihre alte Aufgabe weiterführen kann.

Polling : Meistens können mehrere unterschiedliche Geräte ein Programm unterbrechen. Jedes der Geräte benötigt eine eigene Interrupt-Service-Routine. Manchmal werden sogar verschiedene ISR für das gleiche Gerät verwendet.

Da erkannt werden muss, welches der angeschlossenen Geräte eine Bedienung anfordert, sollte jedes der angeschlossenen Geräte auch einen eigenen Interrupt-Eingang an der CPU haben. Hat die CPU nur einen einzigen Interrupt-Eingang, so gibt es fertige Interruptbausteine (z.B. **Interrupt-Controller**), die den Zugriff zur CPU regeln. Bei diesem Verfahren wird durch den Interrupt, direkt die entsprechende ISR angesprochen.

Ist nur ein einziger Interruptkanal vorhanden, so müssen alle unterbrechenden Geräte ihr Interrupt-Signal mit einem ODER verknüpfen. Ein Interrupt entsteht, wenn irgendein Gerät eine Interruptanforderung meldet. In diesem Fall gibt es nur einen gemeinsamen Interrupt. In der ISR muss die CPU nun jedes der angeschlossenen Geräte nacheinander befragen, welches von ihnen den Interrupt angefordert hat. Daraufhin wird der spezielle Teil der ISR gestartet, der zu dem erkannten Gerät gehört. Einen solchen seriellen Abfragevorgang bezeichnet man mit **Polling**. Ein Polling mit vielen angeschlossenen Geräten ist natürlich zeitaufwendiger als das schaltungstechnisch aufwendigere direkte Verfahren.

Enable Interrupt/Disable Interrupt Ein Interrupt ist nicht zu jeder Zeit sinnvoll. Besonders, wenn gerade eine ISR abläuft ist die Unterbrechung durch eine andere ISR - schlimmer noch durch die gleiche - meist sehr lästig. In solchen Fällen bietet die CPU die Möglichkeit, den Interrupt-Eingang abzuschalten.

Prioritäten Es kann jedoch vorkommen, dass es besonders wichtige oder besonders eilige Ereignisse gibt, die auf jeden Fall vorrangig bearbeitet werden müssen. Für solche Fälle erlaubt eine Vorrangschaltung, dass ein wichtigeres Ereignis auch die ISR eines weniger wichtigen Ereignisses unterbrechen kann (Prioritätslogik).

Anwendungen :

Eine CPU führt **mehrere Aufgaben parallel** aus - aber **nicht gleichzeitig** ! Z.B. wird im Vordergrund (Foreground) ein Datensatz ausgedruckt und im Hintergrund (Background) wird eine langwierige Rechenoperation ausgeführt. Der Drucker bemerkt nichts von der Nebentätigkeit der CPU, denn immer wenn er ein neues Zeichen ausdrucken möchte, sendet er ein Interrupt an die CPU, die die Nebentätigkeit kurz unterbricht, mal eben ein Zeichen zum Drucker sendet und anschließend ihre

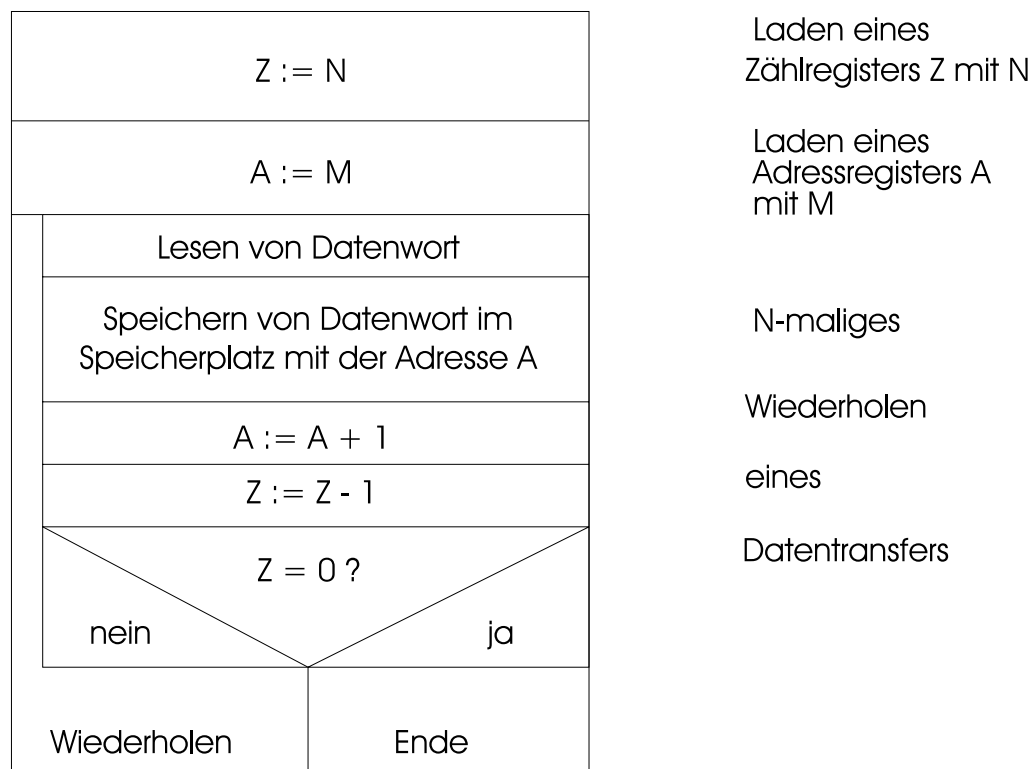
Nebentätigkeit fortsetzt.

Die Geschwindigkeit der Nebentätigkeit wird zwar durch die Unterbrechungen etwas verringert. Ohne die Interruptmöglichkeit könnte die CPU aber gar nichts nebenher arbeiten.

Besonders deutlich tritt dies bei Überwachungsaufgaben hervor, hier muss die CPU nur ganz selten eingreifen und nur bei einem Fehlerfall etwas unternehmen.

7.2 Direct Memory Access (DMA, direkter Speicherzugriff)

Beim schnellen Transport von Daten von und zum Memory ist die CPU sehr stark belastet. Soll ein Datenblock (N Datenwörter) von einer Platte zum Memory ab Memoryplatz M transportiert werden, oder ein Datenblock von einem Memorybereich in einen anderen Memorybereich kopiert werden, so muss etwa das folgende Programm ablaufen:

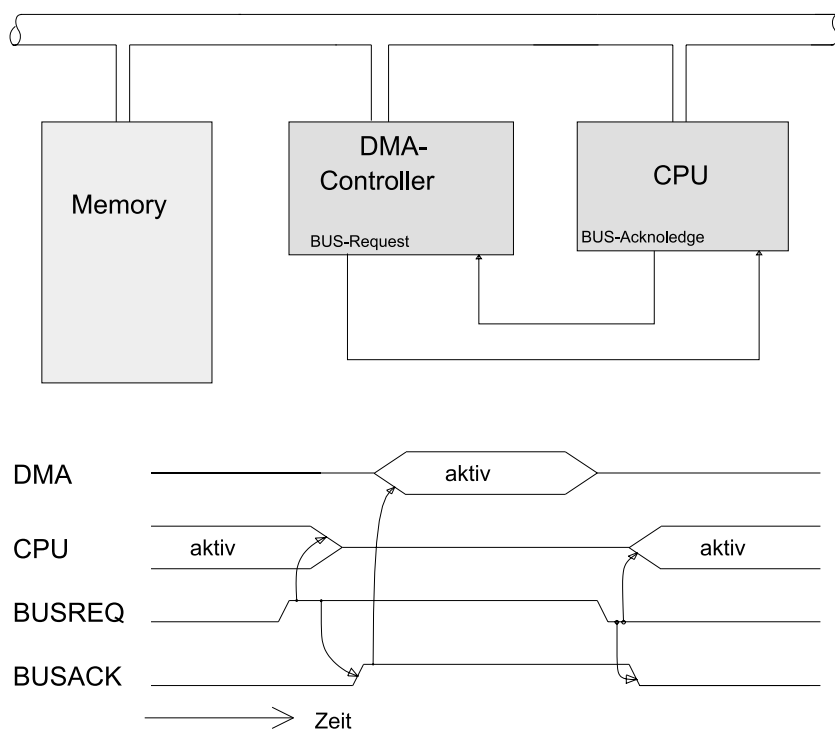


Für eine universelle CPU, die ausgelegt ist, um viele unterschiedliche Probleme zu lösen, ist eine solche spezielle Aufgabe sehr zeitintensiv. Für jeden einzelnen Da-

transfer ist ein kleines Maschinenprogramm mit Maschinenbefehlen aus mehreren Mikroinstruktionen (Taktzyklen) notwendig. Im Prinzip könnte diese Problemstellung von einer Spezial-„CPU“, die nur für diese Aufgabe konzipiert ist, viel besser gelöst werden. Eine solche Spezialeinheit für den Datentransfer nennt man **DMA-Controller**.

Ein DMA-Controller benötigt wenige Register und eine einfachere ALU (im Wesentlichen zwei Inkremente und einen Dekrementer). Dafür muss aber der oben angegebene Ablauf in möglichst wenigen Taktzyklen abgearbeitet werden.

CPU und DMA-Controller sind zwei Einheiten, die beide auf das Memory und damit auch auf den BUS zugreifen. Da beide unabhängig von einander arbeiten und Daten und Adressen auf den BUS legen, könnte es zu Datenkollisionen auf dem BUS kommen. Um dies zu vermeiden, ist eine Abstimmung der beiden Einheiten notwendig. Der DMA-Controller (oder wer auch immer den BUS braucht,) sendet ein Signal **BUS-Request** (BUS-Anforderung) an die CPU. Daraufhin zieht sich die CPU - nach angemessener Zeit - vom BUS zurück, und erlaubt anderen Einheiten, den BUS zu benutzen.



Um diesen BUS-Zugriff zu ermöglichen, können die Ausgänge von CPU und DMA-Controller vom BUS abgetrennt werden (3-State-Gatter). Im Normalfall arbeitet die CPU mit dem BUS und der DMA-Controller ist abgetrennt. Muss der DMA-Controller nun ein Datentransfer von oder zum Memory ausführen, so benötigt er

den BUS. Er sendet an die CPU die BUS-Anforderung BUSREQ. Die CPU erkennt an diesem Signal, dass irgendeine Einheit den BUS benötigt. Sie wartet das Ende des momentan ablaufenden BUS-Zugriffs ab und trennt danach ihre Verbindung zum BUS. Dem anfordernden Gerät - in diesem Fall dem DMA-Controller - gibt die CPU mit dem Signal BUSACK Bescheid, dass nun der BUS frei ist. Der DMA-Controller weiss nun, dass der BUS frei gegeben ist, und kann den Datentransfer durchführen.

Sobald der DMA-Controller den BUS nicht mehr braucht, gibt er ihn wieder frei. Dazu schaltet er zunächst seine BUS-Anschlüsse in den 3-State-Zustand, und gibt über das Signal BUS-REQ der CPU Bescheid. Sobald die CPU erkennt, dass der BUS freigegeben ist, kann sie wieder den BUS für eigene Aufgaben benutzen.

Während eines BUS-Zugriffs könnte der DMA-Controller einen kompletten Datenblock übertragen. Ist aber genügend Zeit zwischen einzelnen Daten, ist es effektiver, zwischendurch den BUS wieder der CPU zur Verfügung zu stellen. Da in der CPU ein internes Mikroprogramm abläuft, bei dem zum Teil der externe BUS nicht verwendet wird, kann die CPU auch teilweise ohne BUS-Zugriff weiterarbeiten. In diesem Fall arbeiten also CPU und DMA-Controller echt parallel nebeneinander, und es geht keine Zeit verloren.

CPU und DMA-Controller können gleichzeitig arbeiten !

Wie schon angedeutet gilt das oben gesagte für beliebige Baugruppen, die den BUS benutzen wollen. Der DMA-Controller war nur ein Beispiel.

8 Software

Die Bezeichnungen Software (weiche Ware) und Hardware (feste Ware) deuten einerseits auf die veränderlichen (weichen) Komponenten eines Systems und andererseits auf die unveränderlichen (harten) Komponenten hin. In diesem Sinne versteht man unter der Software meist die Programme und unter der Hardware die elektronischen, mechanischen und sonstigen greifbaren Baugruppen. Der Übergang zwischen Software und Hardware ist jedoch sehr fließend.

Software Firmware

		Befehlssatz	
Programme Im RAM	Programme Im ROM	MikroPrg Im RAM	MikroPrg Im ROM

Elektronische Schaltung

Schaltplan	Zusammen- gebaut
Im CAD- System	auf Papier

Hardware

		Bauteile
Integrierte Schaltung	Transistor, Diode, Widerstand	
In der CAD- Anlage	In SI Kondensator Stecker Kabel...	

PLD,PLA,...

Hier soll es jetzt ausschließlich um die Probleme der Programme gehen.

8.1 Maschinensprache

Die **Maschinensprache** besteht aus HL-Pegel, die an die CPU gesandt werden und dort die Steuersignale auslösen. Die HL-Pegel können auch benutzergerechter ausgedrückt werden. Die erste Stufe ist der

Binärcode: Die HL-Pegel werden durch 1 oder 0 ersetzt, was jedoch zu langen Zahlen führt. Durch Zusammenfassen von mehreren Ziffern kommt man zum

Octalsystem: 3 bits werden zu einer Ziffer mit $2^3=8$ Werten zusammengefasst. Im Octalcode gibt es die 8 Ziffern 0..7. Gebräuchlich ist heute mehr das

Hexadezimalsystem manchmal auch Sedezimalsystem genannt. Hier werden 4 bits zu einer Ziffer mit $2^4=16$ Werten zusammengefasst. 4 bits passen besser zu der Einheit byte und zu den Rechner-Wortbreiten mit 8, 16, 32 oder 64 bits. Zur Darstellung der 16 Ziffern reichen die 10 Zahlensymbole 0..9 nicht aus. Man verwendet daher die ersten sechs Buchstaben des Alphabets (A..F) als zusätzliche Symbole für die Ziffern 10,11,12,13,14 und 15.

Zum Teil werden Programme im Hexadezimalcode verbreitet, doch ist ein Programm in dieser Form für einen Benutzer nahezu unverständlich. Besser ist eine Schreibweise, die der menschlichen Sprache angepasst ist:

Mnemonic Ein Mnemonic ist eine Gedächtnisstütze für uns Menschen. Jedem Be-

fehl wird eine Buchstabenkombination zugeordnet, aus der man leicht die Funktion des Befehls ablesen kann (z.B. ADD für Addiere, SUB für Subtrahiere oder JMP für Springe, Jump). Ein in Mnemonics ausgedrücktes Programm können wir leichter verstehen. Da die CPU jedoch nur H und L versteht und mit den Buchstaben der Mnemonics nichts anzufangen weiß, müssen die HL-Bitkombinationen aus den anderen Schreibweisen erst zusammengestellt werden. Diese Aufgabe übernimmt der

Assembler Der Assembler ist ein Programm, das einen Text aus Mnemonics (Quell- oder Sourceprogramm) in ein in der CPU lauffähiges Programm (**Objektprogramm**) umsetzt. Dabei versteht der Assembler nicht nur die Mnemonics, er kann z.B. auch Sprungadressen berechnen, Text in Binärcode umsetzen, mathematische Berechnungen ausführen und mit vielem mehr die Arbeit des Programmierers erleichtern.

Ein-Pass-Assembler Manchmal erzeugt ein Assembler in einem Durchlauf (Pass) direkt das Objektprogramm. Die meisten Assembler sind jedoch

Mehrpass-Assembler Aus dem Quellcode wird zunächst ein Zwischencode erzeugt. Ein Gesamtprogramm besteht meist aus mehreren getrennten Teilen (Module). Die einzelnen Module sollen später zusammenarbeiten, daher müssen Adressen, Daten und Referenzen gegenseitig bekannt sein. Deren Werte hängen aber von der Größe der übrigen Module ab.

Der **Linker**, Binder hat die Aufgabe, die einzelnen Module zu einem gemeinsamen Programm zusammenzubinden. Dabei müssen die einzelnen Module verschoben (relokiert) werden können und Adressen und Referenzen zu anderen Modulen eingetragen werden.

Debugger In einem Source-Programm können leicht Syntaxfehler entstehen. Z.B. ist das Format oder die Schreibweise eines Mnemonics fehlerhaft, oder andere Teile des Textes sind inkorrekt. Solche Fehler fallen dem Assembler auf und werden gemeldet, sie sind daher leicht auszumerzen. Schwieriger wird es, wenn das Quellprogramm frei von **Syntaxfehlern** ist und ein lauffähiges Programm erstellt werden kann; dennoch können in dem Programm **logische Fehler** (bugs) enthalten sein. Logische Fehler sind z.B. falsche Abfragen ($=0$ anstatt <0), falsche Schleifenlängen (1000 statt 100), Überschreiben eines bereits verwendeten Registers und vieles, vieles mehr!!! Logische Fehler sind meist schwierig zu finden. Hier hilft ein Debugger-Programm. Damit kann man den Ablauf eines Programms schrittweise verfolgen, die Auswirkungen des Programms beobachten und den tatsächlichen Ablauf mit dem geplanten vergleichen. Hat man logische Fehler aufgespürt, muss das Quellprogramm korrigiert, anschließend assembliert und gelinkt, und erneut geprüft werden. Dieser Vorgang muss solange wiederholt werden, bis das Ergebnis zufriedenstellend ist.

Vorteile der Maschinensprache : Da die CPU ausschließlich Maschinenbefehle versteht, liegt letztlich jede Aufgabe, die von der CPU abgearbeitet wird, in Maschi-

nensprache vor. Das bedeutet, wenn ein Problem überhaupt mit einer Anlage lösbar ist; dann ist es auf jeden Fall in Maschinensprache lösbar. Dem Benutzer stehen alle Fähigkeiten der CPU zur Verfügung und er kann die Problemlösung optimal an die Fähigkeiten der CPU, der an die CPU angeschlossenen Baugruppen und an die Problemstellung anpassen. Der Programmierer kann Rechengenauigkeit, Rechengeschwindigkeit, Speicherbedarf,... optimal anpassen.

8.2 Höhere Programmiersprachen

Leider hat die Maschinensprache zwei wesentliche Nachteile. Erstens sind die Mnemonics zwar für den Menschen verständlich, doch meist nur dem, der sich mit Rechner auskennt. Außerdem ist meist die spezielle Registerkonfiguration und der Befehlssatz nicht bekannt, zumindest nicht geläufig. Viele wollen einen Computer nur benutzen und mit ihm in einer dem Problem des Benutzers angepassten Form kommunizieren. Der Mathematiker will in mathematischen Formeln reden und der Kaufmann in Bilanzen, aber nur wenige interessieren die CPU-Register.

Das zweite Problem ist, dass ein Maschinenprogramm an den Befehlssatz der CPU angepasst ist. Ein Objektprogramm der einen CPU kann von einer CPU mit einem anderen Befehlssatz nicht ausgeführt werden. Auch in einer Anlage mit der gleichen CPU treten leicht Probleme auf, da das Maschinenprogramm auch an die Rechenumgebung und Memory angepasst ist.

Aus diesen Problemen resultieren zwei wichtige Forderungen an ein Programm:

1. Das Quellprogramm soll **problemorientiert** sein
2. Das Quellprogramm soll **übertragbar** (portabel) sein

Um diese beiden Wünsche der Benutzer zu befriedigen wurden Programmiersprachen entwickelt. Je nach Problemkreis des Benutzers eine unterschiedliche Sprache:

FORTRAN	Formula Translation	mathematische Probleme
COBOL	Common Business orientated Language	kaufmännische Probleme
PL/I	Program Language I	IBM für Mathematiker und Kaufleute
PEARL	Prozess and Experiment Realtime Language	Prozeßsteuersprache
BASIC	Beginners All purpose Symbolic Instruction Code	Allzwecksprache für Anfänger
PASCAL		Lernsprache
MODULA		Modulare Sprache
C		Maschinennahe Sprache
LISP	List Processing	Datenbanken und künstliche Intelligenz

8.2.1 Erzeugen eines Objektprogrammes

Das Quellprogramm einer höheren Programmiersprache enthält Sprachelemente aus dem Problemkreis der Benutzer und sollte unabhängig von der CPU... sein. Zur Ausführung des Quellprogramms mit Elementen des Befehlssatzes kann man zwei Verfahren (Compiler und Interpreter) einsetzen.

Compiler: Ein Compiler erzeugt - ähnlich wie der Assembler - aus dem Quellprogramm einen Zwischencode, der mit einem Linker zu einem Maschinenprogramm zusammengebunden wird. Dabei werden die Sprachelemente der Programmiersprache teils direkt in Maschinenbefehle umgewandelt teils werden Unterprogrammaufrufe aus einer Bibliothek eingesetzt. Beim Linken werden dann aus der Bibliothek die benötigten Maschinenprogramme mit eingebunden. Das Objektprogramm ist danach ein komplettes Maschinenprogramm, das recht schnell ablaufen kann (runtime).

Das Debuggen des vom Compiler erzeugten Objektprogramms ist recht langwierig, da während des Programmablaufs die Beziehung zum ursprünglichen Quellprogramm und zu den Programmdateien verlorengegangen ist. Das Quellprogramm, und die Daten sind nicht griffbereit.

Interpreter: Das Quellprogramm bleibt während der Programmausführung im Memory und wird nicht geändert. Der Programmablauf wird vom Interpreter gesteuert. Der Interpreter liest Anweisung für Anweisung den Quelltext und startet entsprechend der gelesenen Anweisung ein Unterprogramm (der Quelltext wird interpretiert).

Der Vorteil ist, dass jederzeit der Quelltext zur Verfügung steht und daher jederzeit der Ablauf des Programmes und das Verändern der Daten beobachtet werden kann.

Dadurch ist eine effektive und schnelle Fehlersuche möglich.

Nachteilig ist die langsame Programmausführungsgeschwindigkeit. Sie rührt daher, dass vor der Ausführung einer Anweisung, diese jedesmal neu interpretiert werden muss. Besonders bei Programmschleifen ist das sehr zeitaufwendig.

Anwendungsbereiche für Interpreter und Compiler: Aus den vorgenannten Vor- und Nachteilen ergeben sich direkt die Anwendungsgebiete:

Compiler (langwierige Programmerstellung, schnelle Programmlaufzeit) wird dort eingesetzt, wo immer wieder das gleiche Programm mit neuen Daten ablaufen soll. Das Programm wird nur einmal erstellt und danach häufig benutzt. Typischer Vertreter eines Compilers ist FORTRAN.

Interpreter (schnelle Programmerstellung, langsame Programmlaufzeit) wird eingesetzt, wenn möglichst schnell ein Erfolgserlebnis da sein muss. Also besonders bei Anfängern und Benutzern die gerne mit einem Computer spielen wollen. Daher ist BASIC im allgemeinen ein Interpreter. Doch sollte man nicht Interpreter mit Anfängersprache abtun. Gerade der Vorteil, schnell ein lauffähiges Programm zu erstellen, erlaubt den sinnvollen Einsatz bei häufig wechselnden Problemstellungen, z.B. im Labor. Hier ist es wichtig, dass ein lauffähiges Programm in kurzer Zeit vorliegt (ca 1/2 Stunde); dabei ist es unwichtig, ob das Programm anschließend innerhalb einer zehntel Sekunde oder einer Minute abläuft.

Um die Vorteile von Interpreter und Compiler miteinander zu verbinden, kann man zwei Wege einschlagen.

Die logischen Fehler im Quelltext können mit einem Interpreter schnell entfernt werden. Anschließend wird der fehlerfreie Quelltext mit einem Compiler in ein schnelles Maschinenprogramm übersetzt. Leider können sich jedoch Interpreterablauf und Maschinenprogrammablauf unterscheiden, so dass das identische Quellprogramm bei den unterschiedlichen Verfahren unterschiedliche Ergebnisse erstellen.

Für einen Compiler gibt es häufig Debug-Möglichkeiten. So gibt es die Möglichkeit, durch zusätzliche Debug-Routinen auf den Quelltext und auf die Daten zugreifen zu können.

Dialekte: Einige Programmiersprachen sind bereits sehr alt. Ihr Wortschatz wurde entwickelt, als die Rechner noch wenig leistungsfähig waren. Auch unter den modernen Rechnern findet man eine große Vielfalt von Rechnern mit sehr unterschiedlicher Leistung. Es macht wenig Sinn, von einem kleinen einfachen Rechner den gleichen Wortschatz zu fordern, den ein Hochleistungsrechner problemlos verstehen könnte. Auch haben viele Rechner, besonders im Hobby- und Spielbereich, viele Zusatzkomponenten, die andere Rechner nicht oder anders haben. Fordert man einen Sprachumfang, den alle Rechner verstehen und benutzen können, so muss man den Umfang stark einschränken. Die Folge ist, dass sich die ursprünglichen Sprache weiterent-

wickelt hat und in verschiedenen Versionen neue Sprachelemente eingebaut wurden.

Solche Sprachweiterentwicklungen und Spracherweiterungen bezeichnet man als Dialekt. Der Wunsch auf Übertragbarkeit eines Programms kann nur erfüllt werden, wenn der verwendete Dialekt auch vom Zielrechner verstanden wird. Im Zweifelsfall darf nur ein Bruchteil des Wortschatzes benutzt werden.

Implementierung: Compiler und Interpreter sind Programme, die aus dem Quelltext letztlich Maschinenbefehle erzeugen. Jede Aufgabe - so auch die des Compilers, das Umsetzen von Quelltext in Maschinencode - kann auf sehr unterschiedliche Weise gelöst werden. Unter Implementierung versteht man die Art und Weise der Umsetzung des Quellcodes in Maschinensprache. Dabei können sehr unterschiedliche Anforderungen gestellt werden. Neben der selbstverständlichen Forderung nach Fehlerfreiheit - leider zu oft nicht erfüllt - können viele unterschiedliche Gesichtspunkte eine Rolle spielen:

- Rechengenauigkeit
- Rechengeschwindigkeit
- Compiliergeschwindigkeit
- Speicherbedarf
- Fehlerprüfung
- Verhalten im Fehlerfall
- Debugging-Möglichkeiten

um nur einige zu nennen.

Benchmark: Ein herstellerneutraler Test (Benchmark) versucht die Leistungsfähigkeit unterschiedlicher Rechner zu vergleichen. Dabei ist ein Vergleich insofern schwierig, da neben absoluten Zahlen, wie Taktfrequenz, vor allem die Leistungsfähigkeit getestet werden soll. Diese hängt jedoch sehr stark von der Implementierung der Software ab.

Weiterentwicklung der Hardware: Eine neue Hardware kann nur sinnvoll mit Software gemeinsam benutzt werden. Soll ein neuer Rechner auf dem Markt Erfolg haben, muss daher gleichzeitig die entsprechende Software mitgeliefert werden. Das ist nur schnell möglich, wenn bereits vorhandene Software auf die neue Hardware übertragen werden kann. Daher sollte die vorhandene Software als Quellcode in einer höheren Programmiersprache vorliegen. Eine optimale Anpassung an eine CPU ist aber am besten mit Maschinensprache möglich. Bleibt die Frage zu stellen:

”Werden überhaupt die vielen neuen Features einer besseren CPU genutzt ???”

9 Massenspeicher

Wie der Name sagt, ist die Aufgabe eines Massenspeichers, große Datenmengen zu speichern. In den Anfängen der Datenverarbeitung benutzte man Papier als Speichermedium, in das Löcher (ca. 1mm x 1mm) gestanzt wurden. Diese Löcher wurden teils mechanisch teils optisch abgetastet. Diese Speicher gab es in Endlosform als Lochstreifen oder als Lochkarten mit ca. 80 Zeichen.

Heute benutzt man diese Form der Datenspeicherung nur noch sehr selten. Hauptnachteil ist die geringe Kapazität durch den großen Platzbedarf für ein bit. Zur Zeit haben sich zwei Formen des Massenspeichers durchgesetzt, die magnetische und die optische Speicherung.

9.1 Physikalische Grundlagen

Die **Maxwellgleichungen** beschreiben den Zusammenhang von elektrischen und magnetischen Größen. Die für die Datenspeicherung wichtigen Beziehungen sind in folgender Tabelle zusammengefasst:

$\oint H ds = nI$	n Drähte mit dem Strom I erzeugen ein magnetisches Feld der Feldstärke H
$B = \mu\mu_0 H$	das magnetische Feld bewirkt einen in sich geschlossenen magnetischen Fluss der Flussdichte B, die Permeabilität $\mu\mu_0$ beschreibt so etwas wie die Leitfähigkeit für den Fluss. Im Vakuum ist die Leitfähigkeit μ_0 , in Materialien ändert sie sich um den Faktor μ .
$B = \mu_0(H + M)$	die Flussdichte lässt sich auch mit der Magnetisierung M ausdrücken. M ist dann so etwas wie eine zusätzliche Feldstärke, die durch das Material hervorgerufen wird. Im Vakuum ist M=0 Wir unterscheiden diamagnetische Stoffe (μ etwas kleiner als 1, M < 0), paramagnetische Stoffe (μ etwas größer als 1, M > 0) und ferromagnetische Stoffe (μ viel größer als 1, M ist sehr groß)
$\Phi = \int B df$	Die Flussdichte B ist der Fluss Φ pro Flächeneinheit
$n d\Phi/dt = U$	Induktionsgesetz: Die Flussänderung durch eine Drahtschleife mit n Windungen erzeugt eine elektrische Spannung U. Das Induktionsgesetz wird im Transformator zur Umsetzung von Wechselströmen ausgenutzt. Ein Wechselstrom erzeugt einen wechselnden magnetischen Fluss, der eine Wechselspannung induziert. In einem Generator (Dynamo) wird durch Drehung die Fläche verändert und so eine Spannung induziert.

Die Leitung des magnetischen Flusses lässt sich ganz analog wie die Leitung von Flüssigkeiten in Rohren oder die Leitung von Strom in Drähten beschreiben:

	Pneumatik	Elektronik	Magnetfeld
treibende Größe	Druck P	Spannung U	magnetische Spannung nI
es fließt	Wasserstrom I	elektrischer Strom I	magnetischer Fluss Φ
durch	Rohr	Draht	Medium
Widerstand	$R = L/q$	$R = \rho L/q$	$R = 1/(\mu\mu_0) L/q$
Ohm'sches Gesetz	$I = P/R$	$I = U/R$	$\Phi = nI \mu\mu_0 q/L$
oder			$B \approx \mu\mu_0/L n I$
			$M \approx (\mu - 1)/L n I$

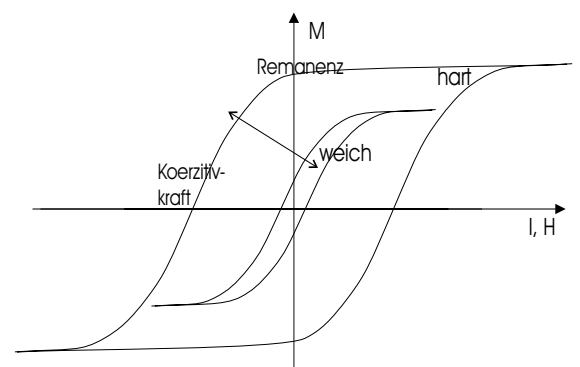
Die vorletzte Reihe zeigt, dass die magnetische Flussdichte mit dem erregenden Strom ansteigt. $1/\mu\mu_0$ gibt so etwas wie den spezifischen Widerstand eines Materials gegen den magnetischen Fluss an ($\mu\mu_0$ die Leitfähigkeit). μ_0 ist die magnetische Leitfähigkeit des Vakuums.

Die Magnetisierung (letzte Reihe) beschreibt, dass die Materie zu einem zusätzlichen Magneten wird, dessen Feld sich dem erzeugenden überlagert. Dabei wird das ursprüngliche Feld verstärkt ($\mu > 1$) bzw. abgeschwächt ($\mu < 1$).

Die meisten Stoffe haben etwa die gleiche Leitfähigkeit ($\mu \approx 1$). Diamagnetische Stoffe haben ein μ etwas kleiner und paramagnetische μ etwas größer als 1.

Bei den ferromagnetischen Materialien (Eisen, Nickel Kobalt und deren Verbindungen) ist μ wesentlich größer als 1 (50 ... 10000) aber vor allem zeigt sich, dass μ keine Konstante ist, M also nicht linear mit I oder H ansteigt, sondern dass bei höherer Erregung die Kurve flacher wird. Man spricht von **Sättigung**. Zusätzlich kommt hinzu, dass bei Reduzierung der Erregung die Magnetisierung M nicht in gleichem Maße abfällt, sondern wesentlich langsamer. Diese Abhängigkeit wird in der sogenannten **Hysteresis-Kurve** dargestellt. Das Material hat also eine Merkfähigkeit (Anwendung als Speicher) !!

Aufgetragen ist die Magnetisierung M in Abhängigkeit von der erregenden magnetischen Feldstärke H (bzw. dem erregenden Strom I). Wenn die erregende Feldstärke ansteigt, steigt auch M bis zu einem Endwert an. Danach ist das Material gesättigt, M steigt kaum noch. Wird aber nun die erregende Feldstärke reduziert, so sinkt zwar M ab, aber deutlich geringer. Erreicht die Feldstärke den Wert 0, so bleibt ein Rest Magnetisierung erhalten (**Remanenz**). die Feldstärke weit genug in den negativen Bereich gebracht wird (**Koerzitivkraft**), ist das Feld wieder abgebaut.



Erst wenn bei noch negativerer Feldstärke wiederholt sich der Vorgang im negativen Bereich. Man erkennt, dass bei der Feldstärke 0 (abgeschaltetes Feld) zwei Zustände des Materials entstehen, positive bzw. negative Magnetisierung, die den bits 0 und 1 zugeordnet werden können.

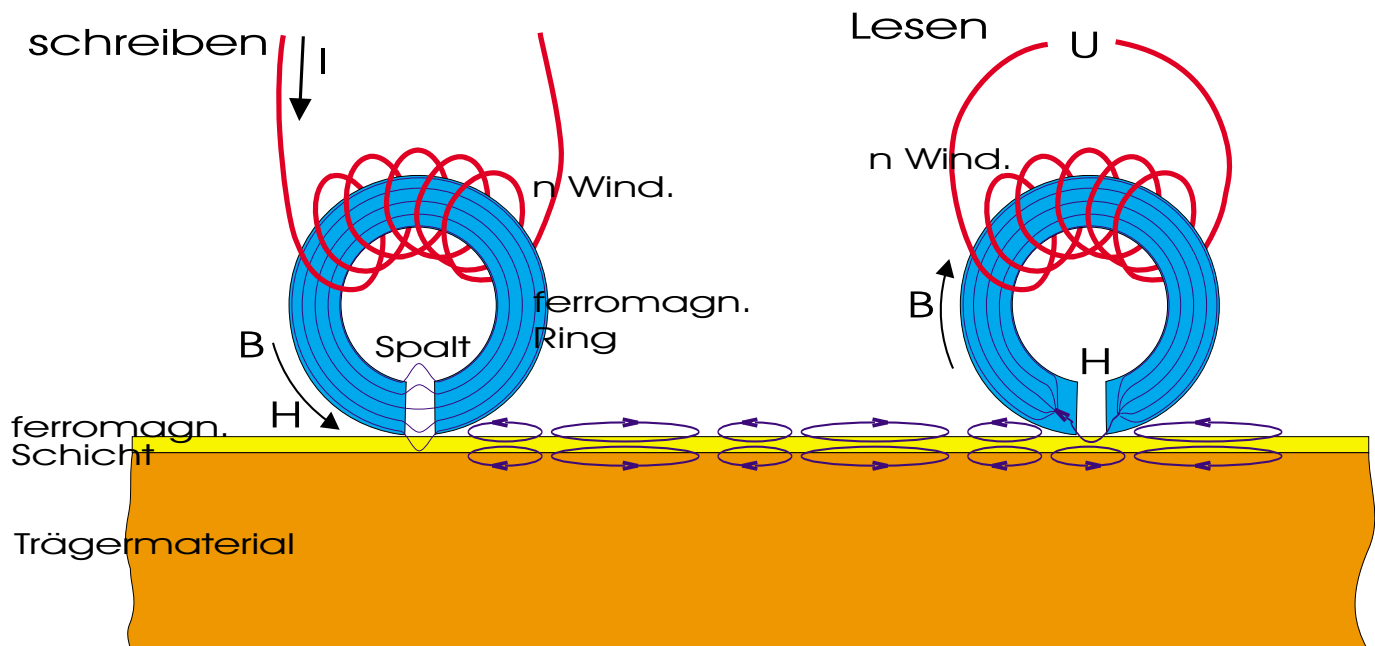
Die Form der Hysterese-Kurve kann durch Bearbeiten verändert werden. Material mit einer weit geöffneten Kurve (große Hysterese) nennt man magnetisch hart, Material mit einer engen Hysterese-Kurve ist magnetisch weich. Für die Anwendung als Datenspeicher ist auch wichtig anzumerken, dass beim Erwärmen ab einer gewissen Temperatur (Curie-Temperatur) Material magnetisch sehr weich wird.

Um den komplementären Zustand zu erreichen, muss die erregende Feldstärke so weit verändert werden, bis das Material die gegengesetzte Sättigung erreicht, um dann nach Abschaltung der Erregung in den entsprechenden Zustand (positive bzw. negative Remanenz) zu fallen. Bei geringen Änderungen der erregenden Feldstärke verbleibt das Material im alten Zustand.

9.1.1 Speichermaterial

Als Speichermaterial für magnetische Speicher ist jedes Material mit einer gewissen magnetischen Härte zu verwenden. Dazu beschichtet man sehr dünn ein Trägermaterial (Plastikfolien in Form von Bändern oder Scheiben, oder Metallscheiben) mit Ferrit oder ähnlichem. Zu Beschreiben muss man möglichst kleine begrenzte Flächen einem starken magnetischen Feld aussetzen. Das Feld muss so groß sein, dass das Speichermaterial gesättigt wird, aber trotzdem so räumlich begrenzt bleibt, dass benachbarte Bereiche nicht beeinflusst werden.

Zu diesem Zweck muss ein starkes aber sehr lokales magnetisches Feld erzeugt werden. Dies geschieht in einem **Schreibkopf**. Der Schreibkopf enthält eine Spule aus n Drahtwindungen die bei Stromdurchfluss ein magnetisches Feld H erzeugen. Durch die Spule führt ein magnetisch leitender Ring, durch den das magnetische Feld den magnetischen Fluss zu der Oberfläche des Speichermediums leitet. Um ein starkes sehr eng begrenztes Feld zu erzielen, enthält der Ring einen extrem dünnen Spalt. Der Fluss muss diesen Spalt (geringe magnetische Leitfähigkeit) überwinden und konzentriert fast komplett die erzeugte Feldstärke am Spalt. Ein Teil des Flusses quillt seitlich aus dem Spalt heraus. Es ergibt sich, dass dieser herausquellende Fluss sich nur in sehr naher Umgebung (ca. die Spaltbreite) des Spaltes ausbreitet. Zum Schreiben von Daten muss nun die Oberfläche des Speichers bis unter die Spaltbreite an den Spalt des Schreibkopfes herangebracht werden.



Je enger die Informationen geschrieben werden soll, desto enger muss der Spalt und desto dichter der Kopf an das Medium geführt werden. (Früher waren Spaltweiten von ca. $35\mu m$ gebräuchlich, heute erreicht man Werte um $1\mu m$. Der geringe Abstand zum Medium verlangt sorgfältiges Arbeiten, wenig Staub, Rauch usw. Erst mit der **Winchester-Technik**, staubsicher gekapselte Platten, war die Steigerung der Plattenkapazität möglich.

9.1.2 Lesen der Daten

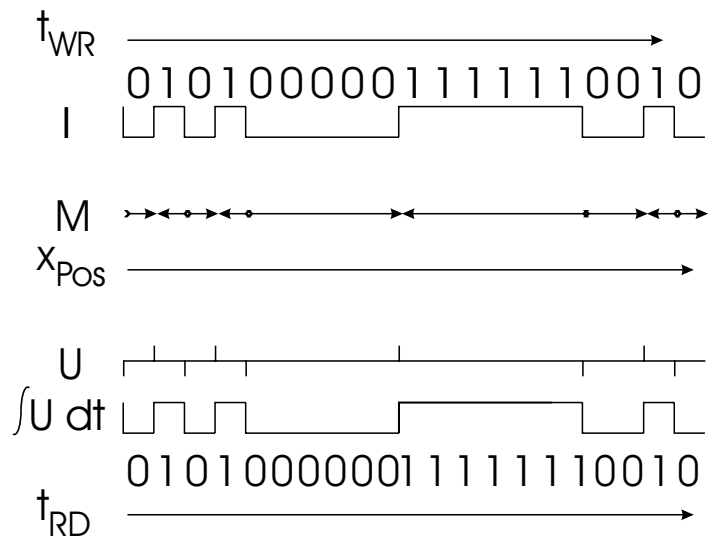
Wie aus der Tabelle ganz oben ersichtlich, kann mit einer Flussänderung eine Spannung induziert werden. Wieder verwendet man einen Kopf, den **Lesekopf**, der identisch wie der Schreibkopf aufgebaut ist. Man kann auch denselben Kopf als Lese- und als Schreibkopf einsetzen.

Führt man den Lesekopf an dem Speichermaterial vorbei, das nach dem Beschreiben kleine Bereiche mit Magneten enthält, so wird der Fluss dieser Magnete durch den Ring des Kopfes und somit durch die Spule mit den Drahtwindungen geführt. Immer, wenn der Spalt am Übergang zweier verschieden gepolter Magnete vorbeizieht, wechselt die Richtung des Flusses durch den Ring und damit durch die Spule. Durch diesen Flusswechsel entsteht ein positiver bzw. negativer Spannungsimpuls.

Beim Schreiben (WR) wird je nach Information zeitlich nacheinander (t_{WR}) ein positiver oder negativer Strom erzeugt, der je nach Geschwindigkeit eine Position (x_{Pos}) des Speichermediums magnetisiert.

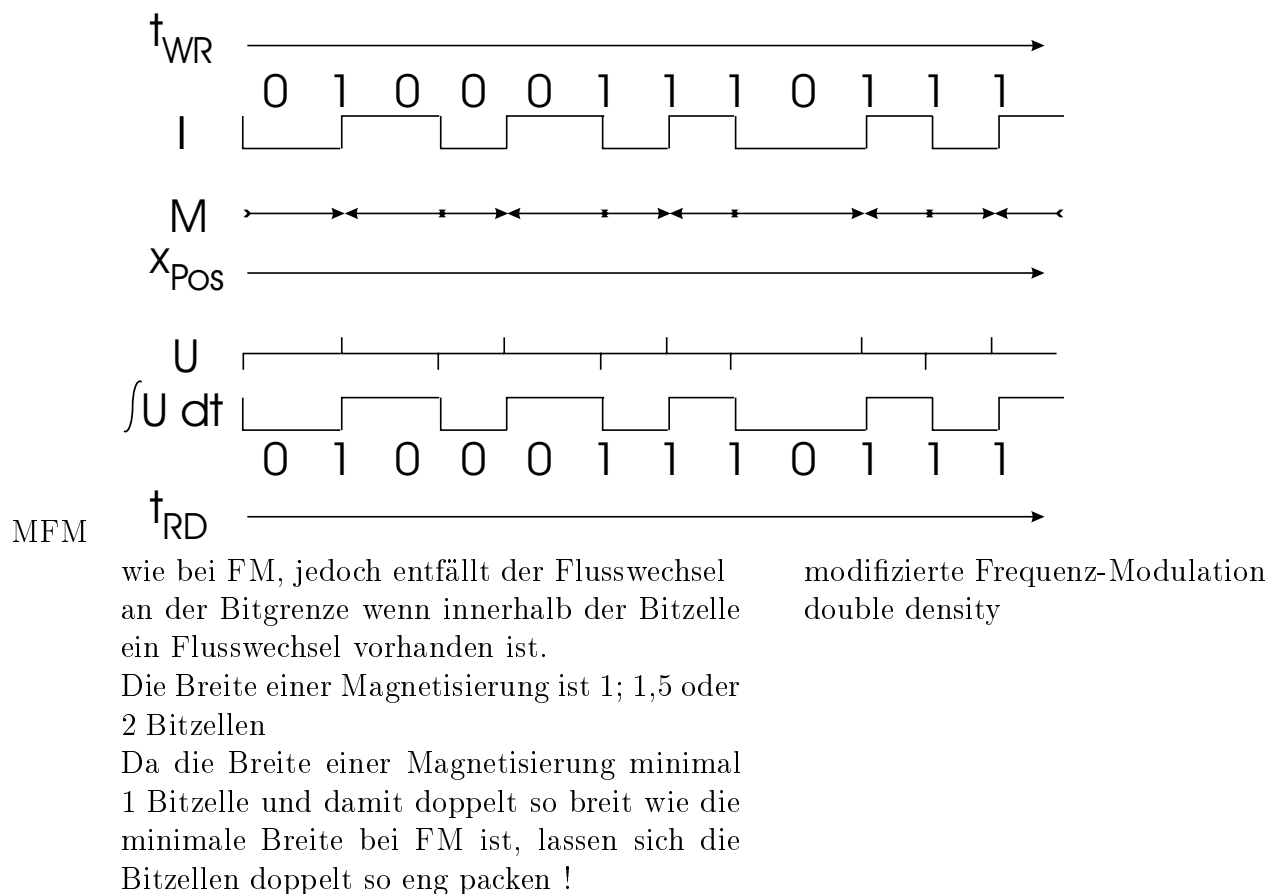
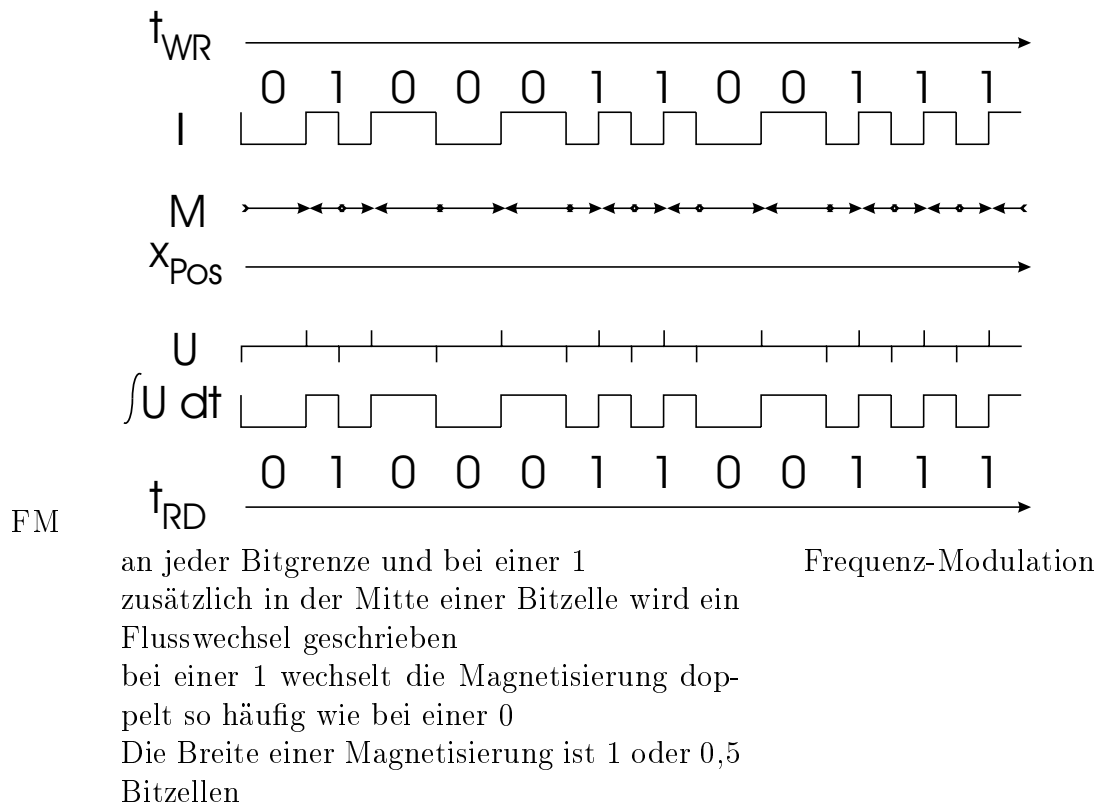
Der beim Lesen bei jedem Wechsel der Magnetisierung induzierte Spannungsimpuls wird elektronisch integriert und man erhält wieder HL-Pegel, aus denen je nach Lesegeschwindigkeit (t_{RD}) 0 und 1 gewonnen werden.

Schreibt man nur Daten auf, so entsteht bei langen Folgen von 0 bzw. 1 nach dem Lesen eine kontinuierliche H- bzw. L-Spannung. Um daraus wieder die Folge der Nullen und Einsen zu restaurieren, ist es notwendig, die einzelnen bits abzutasten. Der Abtastrhythmus muss daher mit dem Schreibrhythmus übereinstimmen. Das ist kaum zu erzielen, da Schreiben und Lesen zu unterschiedlichen Zeiten und unterschiedlichen Bedingungen erfolgen.



9.1.3 FM- und MFM-Modulation

Wie im Diagramm oben ersichtlich sind geschriebene und gelesene Daten unterschiedlich ! Eine Lösung ist, zusätzlich zu den Daten noch ein Synchronisierungssignal mitzuschreiben:



Erkennen der Bitgrenzen : Nach dem Lesen müssen aus dem Lesesignal die Bitgrenzen erkannt und der Inhalt zugeordnet werden (1 = Flusswechsel; 0 kein Flusswechsel innerhalb der Zelle). In einer **PLL-Schaltung** (phased locked loop) wird ein freilaufender Oszillator mit dem gelesenen Signal synchronisiert. In einer PLL werden die Signale eines freilaufenden Oszillators und eines Referenzoszillators verglichen. Bei einer Abweichung in Frequenz oder Phase wird die Frequenz des freilaufenden Oszillators soweit verändert, dass beide Oszillatoren übereinstimmen. Als freilaufenden Oszillator verwendet man einen sogenannten **VCO** (voltage controlled oscillator, spannungsgesteuerter Oszillator).

Das Lesesignal und das Signal des VCO werden so aufbereitet und dann der PLL zugeführt. Man erreicht dadurch, dass der VCO immer fest an das Lesesignal angebunden ist und Abweichungen durch unterschiedliche Schreib- und Lesegeschwindigkeit ausgeglichen werden.

Jeder gespeicherte Datenblock enthält einen Vorspann (Header). Fährt der Kopf über diesen Vorspann, rastet die PLL ein, der Oszillator schwingt synchron in Frequenz und Phase mit der gelesenen Bitfolge. Jetzt können die Bitgrenzen erkannt und die Abtastung synchron mit den Bitzellen erfolgen.

9.1.4 Formatierung

Der Datenträger enthält nicht nur die eigentlichen Daten, sondern die Daten sind auf einzelne Datenblöcke aufgeteilt. Jeder Datenblock wird als ganzes gelesen bzw. geschrieben. Der Zugriff auf Teildaten erfolgt durch Lesen des Blocks, Modifizieren der Teildaten und Zurückschreiben des Blocks. Jeder Datenblock enthält außer den Daten noch einen Vorspann (**Preamble,Header**) und einen Nachspann (**Postamble**). Der Vorspann enthält unter Anderem eine Adressinformation des Datenblocks, um den Datenblock zu identifizieren; der Nachspann eine Prüfsumme, um Lesefehler zu erkennen und zu korrigieren. Zwischen den Datenblöcken muss eine Lücke (**Gap**) gelassen werden, da die Schreibdichte gering variieren kann und daher die Blöcke unterschiedlich lang sein können.

Die Größe der Datenblöcke bestimmt die effektive Kapazität, da durch Gap, Preamble und Postamble Plattenplatz verbraucht wird (overhead). Kleine Datenblöcke verschlingen Kapazität durch Overhead. Große Datenblöcke sind oft nur teilweise gefüllt, da ein zu speichernder Datensatz einen Block nur zum Teil ausfüllt oder gerade nicht ganz in einen Block passt und ein zweiter Block angefangen werden muss. Die optimale Blockgröße hängt daher von der Anwendung ab.

9.1.5 Kapazität und Speicherformen

Die Kapazität eines Speichers ergibt sich aus der nutzbaren Oberfläche und dem Platzbedarf für ein bit (Abstand von bit zu bit und von Spur zu Spur). Ein Band mit einer Länge von mehreren Hundertmetern und einer Breite im Bereich von cm hat eine Oberfläche von einigen m^2 . Von der Oberfläche einer Platte lassen sich nur die äußeren Bereiche verwenden. Man kommt je nach Plattendurchmesser auf wenige cm^2 bis maximal wenige hundert cm^2 . Die Kapazität der Platten lässt sich durch stapeln steigern (Plattenstapel).

Die Bitgröße hängt wesentlich von dem Abstand des Kopfes von der magnetischen Beschichtung ab. Kleine Speicherzellen verlangen einen sehr geringen Abstand und damit auch eine sehr hohe Staubfreiheit. Die Einhaltung dieser Bedingungen ist bei Wechselmedien nur schwer zu erzielen.

9.1.6 Zugriffszeiten

Die Zugriffsgeschwindigkeit hängt von mehreren Faktoren ab. Dabei unterscheiden wir die Positioniergeschwindigkeit und die reine Lesegeschwindigkeit. Letztere ist meist sehr hoch und hängt nur von der Bitdichte und der Geschwindigkeit des Mediums (Umdrehungsgeschwindigkeit bei einer Platte bzw. Transportgeschwindigkeit des Bandes) ab. Entscheidend beeinflusst wird die Zugriffszeit durch das Positionieren.

Da die Platte sich dreht und dabei die Daten am Kopf vorbeiführt, ist der günstigste Fall gerade dann, wenn der Block sich gerade vor dem Kopf befindet. Im ungünstigsten Fall hat sich der Anfang des Blocks gerade am Kopf vorbeibewegt. In diesem Fall muss eine ganze Umdrehung abgewartet werden. Im Mittel muss man daher eine halbe Plattenumdrehung warten. Bei einer Geschwindigkeit von 9600 Umdrehungen/min ergibt das eine mittlere Wartezeit von 3ms.

Da die Daten auf unterschiedlichen Spuren (Ringe) stehen, muss der Kopf gegebenenfalls auf einer anderen Spur positioniert werden. Auch dazu vergeht eine Zeit, die **Head-Settling-Time** (ca. 5msec).

9.2 optische Speicher

Der Vorteil der optischen Speicher ist darin begründet, dass der Kopf einen größeren Abstand vom Speichermedium haben darf. Dadurch kann man die Oberfläche des Speichermediums versiegeln, das Medium wird unempfindlich gegen Staub. Wechselbare Speichermedien sind verfügbar.

9.2.1 ROM, CD

Ein sehr preiswertes und durch die Konsumindustrie weit verbreitetes Medium ist die **CD**. Hier werden bei der Herstellung die bits in Form von kleinen Löchern (**pitch**) in das Medium geprägt und die Oberfläche unter einer Glasschicht versiegelt. Eine nachfolgende Veränderung ist nicht möglich (ROM). Beim Lesen wird ein Lichtstrahl über die Pitches geführt, reflektiert und in einem Sensor empfangen. Je nach Beschaffenheit des Pitches wird das Licht unterschiedlich reflektiert und so die Helligkeit am Sensor beeinflusst.

9.2.2 beschreibbare CD,PROM

Bei den beschreibbaren CD lässt sich durch einen auf dem Medium fokussierten intensiven Lichtstrahl die Oberfläche lokal erhitzen und verändern. Auch auf diese Weise lassen sich lokale Stellen mit unterschiedlichem Reflexionsverhalten erstellen (Brennen). Eine einmal veränderte Oberfläche lässt sich nicht mehr löschen (PROM). Die Abtastung erfolgt wie bei der CD.

9.2.3 wiederbeschreibbare CD

Bei diesem Verfahren nutzt man die Eigenschaft, dass ein magnetisierbares Medium durch Erhitzen magnetisch weich und nach dem Abkühlen wieder hart wird. Beim Schreiben wird breitflächig ein relativ schwaches magnetisches Feld angelegt, damit magnetisch harte Gebiete nicht verändert werden. Dann wird lokal eng begrenzt das Speichermedium erhitzt (magnetisch weich) und damit magnetisiert. Nach dem Abkühlen sind diese Bereiche wieder magnetisch hart und damit die Information eingefroren.

Da das Feld breitflächig angelegt ist, haben alle so beschriebenen bits den gleichen Wert. Bits mit komplementärer Information müssen anschließend geschrieben werden. Dazu polt man das magnetische Feld um und wiederholt die Prozedur mit den komplementären bits.

Das Schreiben erfolgt also in zwei Phasen: Zuerst werden alle bits einer Polarität (z.B. alle 1) eingespeichert, danach das Feld umgeschaltet und die anderen bits geschrieben.

Zum Lesen nutzt man den **Faraday-Effekt** : Substanzen ohne natürliche optische Aktivität werden optisch aktiv, wenn sie sich in einem Magnetfeld befinden. **Optisch aktiv** heißt, dass die Richtung in der Licht schwingt verdreht wird. Da die Stärke des Drehwinkels von der Magnetisierung abhängt, kann man das magnetisierte Medium optisch auslesen. Licht wird polarisiert: von dem eingestrahltten Licht, das normalerweise in allen Richtungen schwingt, wird nur das Licht einer Schwingungsrichtung durchgelassen. Das polarisierte Licht wird durch das magnetisierte und somit optisch aktive Medium geführt. Dort verdreht sich abhängig von der Magnetisierung (0 oder 1) die Schwingungsrichtung. Die Richtung eines weiteren Polfilters wird so justiert, dass Licht einer der beiden Schwingungsrichtungen durchscheint und in einem Sensor nachgewiesen wird.

Index

- 1 bit Fulladder 34
- 1 bit Volladdierer 34

- A**dressdekoder 60
- Adressport 59
- Adressraum 61
- Akkumulator 68
- ALU 62
- Analog 8
- AND 28
- Antivalenz 30
- Äquivalenz 30
- Arithmetic and Logic Unit 62
- Assembler 79
- Assoziativspeicher 55
- Ausnahmeregeln 39

- B**efehlssatz 65
- Benchmark 83
- Binärcode 78
- bit 22
- bug 79
- BUS 43
- BUS-Request 76

- C**ache-Memory 55
- Cache-Zugriff 56
- CD 92
- Central Processing Unit 65
- CISC 67
- Compiler 81
- Complex Instruction Set Computer 67
- Condition-Flag 63
- CPU 65

- D**atenport 59
- Debugger 79
- Decoder 32

- Decrementer 40
- Destination 42
- D-FF 42
- Dialekte 82
- digital 9
- Diode 17
- Disable Interrupt 74
- DMA-Controller 76
- don't care 24, 29
- Dotierung 15
- Drain 18
- DRAM 46, 47
- dual ported Memory 54
- Dualsystem 22
- dynamische RAM 46

- E**AROM 50, 53
- EEPROM 50, 53
- Einer-Komplement 38
- Ein-Pass-Assembler 79
- Enable Interrupt 74
- Encoder 31
- EPROM 50, 53
- Erkennen der Bitgrenzen 90

- F**araday-Effekt 93
- Feldeffekttransistor 18
- ferromagnetische Stoffe 84
- FET 18
- Flash-Memory 50
- FlipFlop 41
- Floating Gate 50
- Flussdichte 84
- frei bewegliche Elektronenfehlstelle 16
- frei bewegliches Elektron 15
- fusible Link 52

- G**ap 90

- Gate 18, 30
- Gatter 30
- Gatterlaufzeit 32

- H**albaddierer 36
- Halbleiter 13, 15
- halfadder 36
- Hauptspeicher 55
- Header 90
- Head-Settling-Time 91
- Hexadezimalsystem 22, 78
- Hysterese-Kurve 85

- I**mplementierung 83
- Incrementer 40
- Induktionsgesetz 84
- Informationsdarstellung 8
- Informationsträger 8
- Instruction fetch 67
- InstructionRegister 69
- Integer-Darstellung 38
- Interface 60
- interner BUS 60
- Interpreter 81
- Interrupt-Controller 74
- Interrupt-Service-Routine 74
- Inverter 25
- IO-Port 59
- ISR 74

- K**anal 18
- Kapazität 19
- Koerzitivkraft 85
- Kondensator 12, 19

- L**eitfähigkeit 13
- Lesekopf 88
- Linker 79
- logische Fehler 79
- logische Gleichungen 23
- look ahead Carry generator 36

- magnetischen Fluss 84
- magnetisches Feld 84
- Magnetisierung 84
- Maschinensprache 78
- Maschinenbefehl 64
- Maschinenprogramm 67
- MaskenROM 49, 51
- Maxwellgleichungen 84
- Mehrpass-Assembler 79
- Memory mapped IO 61
- Memory 45
- Mikroinstruction 64
- Mikroprogrammspeicher 64
- Mikroprogramm 64
- Mikroprogramm-Adressregister 65
- Mnemonic 78
- Morgan'sche Gesetz 29
- MOSFET 18

- N**AND 29
- n-dotiert 15
- NICHT 25
- NOR 29
- Nordheim-Fowler-Tunneleffekt 50
- NOT 25

- O**bjektprogramm 79
- Octalsystem 78
- ODER 28
- Oktalsystem 22
- one's complement 38
- Opamp 10
- Open-Kollektor 43
- Operationsverstärker (Opamp) 20
- Optisch aktiv 93
- OR 28
- orthogonalen Befehlssatz 68

- P**arallelrechner 70
- PC 67, 69

- p-dotiert 16
- Permeabilität 84
- Pipeline 69
- pitch 92
- PLL-Schaltung 90
- Polling 74
- portabel 80
- Postamble 90
- Preamble 90
- Prioritäten 74
- problemorientiert 80
- Programmzähler 67
- PROM 49, 52

- R**·C-Zeit 12
- RAM 46
- RechnerCluster 70
- Reduced Instruction Set Computer 68
- Remanenz 85
- RISC 68
- ROM 46, 49, 51
- RS-FF 41

- Sättigung 85
- Schaltsymbolen 23
- Schreibkopf 86
- Sedezimalsystem 78
- sign-bit 37
- Silizium, Si 15
- Source 18, 42
- Sperrzone 17
- SRAM 46
- statische RAM 46
- Syntaxfehlern 79

- T**aktimpuls 65
- Tor für Informationen 30
- Transportgeschwindigkeit 71
- Trench-Kondensator 48
- Trench-Technik 48
- Tri-State-Gatter 43

- two's complement 38

- U**mcodierer 33
- UND 28

- V**CO 90

- W**ahrheitstabelle 23
- Wahrheitstafel 24
- Weiterentwicklung der Hardware 83
- Widerstand 12
- Winchester-Technik 87

- X**NOR 30
- XOR 30

- Z**ahlensysteme 21
- Zentraleinheit 65
- Zugriff über Inhalt 55
- Zweier-Komplement 38