#### 1.2.1.5 Die Zukunft durch Zeichenvielfalt

Eine weltweite Anwendbarkeit kann erreicht werden, falls 16Bit für die Codierung zur Verfügung stehen: *UNICODE* , der in neuen Programmiersprachen wie etwa *JAVA* schon benutzt wird.

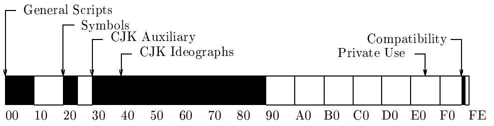



Abbildung 1.10: Verteilung UNICODE

#### Statistik der Version 1.0

Die folgende Tabelle zeigt die Anteile des gesamten im UNICODE zur Verfügung stehenden Platzes, die verschiedenen Schrifttypen in der Version 1.0 bereits zugeteilt wurden:

| | *Allocated* | *Unassigned* | *% Assigend* | |
|---|---|---|---|---|
| General | 2336 | 5856 | 29% | |
| Symbols | 1290 | 2806 | 31% | |
| CJK symbols | 763 | 261 | 75% | |
| Hangul | 2350 | 450 | 84% | |
| Han Compatibility | 268 | 4 | 99% | (Volume 2) |
| Ideographic & other | 20733 | 22275 | 48% | (Volume 2) |
| User Space | 5632 | N/A | N/A | |
| Compatibility Zone | 362 | 133 | 73% | |
| Special | 1 | 13 | | |
| FEFF | 1 | 0 | | |
| FFFE, FFFF | N/A | 2 | | |
| *Totals* | 28706 (assigned) | | | |
| | + 5632 (private use) | | | |
| | = 34338 (allocated) | | | |
| | 52% | | | |

Mit noch über 30000 unbenutzten Character Positionen besitzt der UNICODE auch für die Zukunft noch genug Raum für weitere Expansionen.



#### Die Zukunft des UNICODES

In Zukunft wird der UNICODE Standard um weniger verbreitete und veraltete Schrifttypen erweitert. Schrifttypen dieser Art werden jedoch nicht in ihrer ursprünglichen Form eingebunden, da sich ihr Nutzen schwer einschätzen läßt. So wird bei vielen dieser Schriften eine ausführliche Diskussion nötig sein, bis ein zufriedenstellendes Codierungsschema vorliegt. Die fünf Schriftarten *Ethiopian, Burmese, Khmer, Sinhala* und *Mongolian* werden zum Standard UNICODE hinzugefügt, sobald zuverlässige Informationen über sie vorliegen. Weitere Schriftarten, die für eine mögliche Aufnahme vorgesehen sind, sind

- *Inuktitut/Cree Syllabary:* Das Kommunikationsministerium von Kanada untersucht Standardisierungen von verschiedenen Dialektarten, die von Cree und/oder Inuktitut gesprochen werden und sucht Codierungsschemen.
- *Egyptian Hieroglyphics:* Ein einheitliches Codierungsschema existiert und wird vorangetrieben.
- *Korean Hangul Syllables:* Eventuell werden noch weitere Korean Hangul Dialekte hinzugefügt.

De Unterschied zwischen der logischen Anordung von Zeichen und der Anordnung auf dem Bildschirm zeigt die Abbildung 1.11

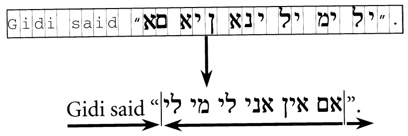



Abbildung 1.11: Bidirectional Ordering

#### General Scripts

Im *General Scripts*-Bereich des UNICODEs sind alle lateinischen und nicht-ideographischen Schriftzeichen codiert:



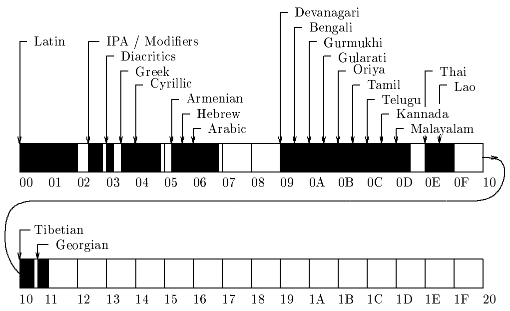



Abbildung 1.12: General Scripts

| Control | | ASCII | | | | | | Control | | Latin 1 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 001 | 002 | 003 | 004 | 005 | 006 | 007 | 008 | 009 | 00A | 00B | 00C | 00D | 00E | 00F |
| NUL | DLE | | 0 | @ | P | ‘ | p | CTRL | CTRL | | ° | À | | à | |
| SOH | DC1 | ! | 1 | A | Q | a | q | CTRL | CTRL | ¡ | ± | Á | Ñ | á | ñ |
| STX | DC2 | ” | 2 | B | R | b | r | CTRL | CTRL | | ² | Â | Ò | â | ò |
| ETX | DC3 | # | 3 | C | S | c | s | CTRL | CTRL | £ | ³ | Ã | Ó | ã | ó |
| EOT | DC4 | $ | 4 | D | T | d | t | CTRL | CTRL | | ’ | Ä | Ô | ä | ô |
| ENQ | NAK | % | 5 | E | U | e | u | CTRL | CTRL | | µ | Å | Õ | å | õ |
| ACK | SYN | & | 6 | F | V | f | v | CTRL | CTRL | \| | ¶ | Æ | Ö | æ | ö |
| BEL | ETB | ’ | 7 | G | W | g | w | CTRL | CTRL | § | · | Ç | × | ç | ÷ |
| BS | CAN | ( | 8 | H | X | h | x | CTRL | CTRL | ¨ | ¸ | È | Ø | è | ø |
| HT | EM | ) | 9 | I | Y | i | y | CTRL | CTRL | © | ¹ | É | Ù | é | ù |
| LF | SUB | * | : | J | Z | j | z | CTRL | CTRL | ª | º | Ê | Ú | ê | ú |
| VT | ESC | + | ; | K | [ | k | { | CTRL | CTRL | « | » | Ë | Û | ë | û |
| FF | FS | , | < | L | \ | l | \| | CTRL | CTRL | ¬ | ¼ | Ì | Ü | ì | ü |
| CR | GS | - | = | M | ] | m | } | CTRL | CTRL | - | ½ | Í | Ý | í | ý |
| SO | RS | . | > | N | ˆ | n | ˜ | CTRL | CTRL | | ¾ | Î | | î | |
| SI | US | / | ? | O | _ | o | DEL | CTRL | CTRL | ¯ | ¿ | Ï | ß | ï | ÿ |

Tabelle 1.17: UNICODE Version 1.0, Character Blocks 0000-00FF



| | |
|---|---|
| 2000..206F | Zeichen für die Zeichensetzung: ,.:;“„ ... |
| 2070..209F | Subscripts und Superscripts: $^2,^3,^4$, ... |
| 20A0..20CF | Währungssymbole: £,$, ... |
| 20D0..20FF | diakretische Zeichen: $\overleftarrow{}, \overrightarrow{}$, ... |
| 2100..214F | buchstabenähnliche Zeichen: $\mathcal{F}$,°F, ... |
| 2150..218F | Zahlen: $\frac{1}{3}$, I, VII, ... |
| 2190..21FF | Pfeile: $\uparrow, \mapsto$, ... |
| 2200..22FF | mathematische Sonderzeichen: $\forall, \exists, \in$, ... |
| 2300..23FF | verschiedene technische Sonderzeichen: #, ... |
| 2400..243F | Symbole für Control-Zeichen: NUL,ESC, ... |
| 2440..245F | OCR-Zeichen |
| 2460..24FF | eingerahmte alphanumerische Zeichen: ©, ... |
| 2500..257F | Formular- und Diagrammzeichen: $\vdash, \dashv, \|$, ... |
| 2580..259F | Blockgraphik-Zeichen |
| 25A0..25FF | graphische Symbole |
| 2600..26FF | verschiedene Dingbats |
| 2700..27BF | Zapf-Dingbats |
| ... | |
| 3000..303F | CJK-Symbole |
| 3040..309F | Hiragana |
| 30A0..30FF | Katakana |
| ... | |

Tabelle 1.18: Weitere Zeichenbereiche

Ein Problem ist die Zeichendarstellung auf dem Bildschirm. Es gibt unter anderem die Methoden des pixelmäßigen Aufbau und des Zeichenaufbaus aus Linien und Kreissegmenten. Ist ein vektorielles Zeichen erst einmal aufgebaut, so ist es ohne Qualitätsverlust beliebig vergrößerbar.

Zeichenaufbau durch Pixel bzw. Linien, Kreissegmente, ...

Der UNICODE stellt weitgehende Kompatibilität zu bestehenden Codes durch (verschobenes) Einfügen oder Bereitstellen von Code-Umwandlungstabellen her: 0000..007F entspricht ASCII. Für andere Codes werden UNICODE-Übersetzungstabellen bereitgestellt, z.B. für UNICODE zu SGML (Tabelle 1.20), UNICODE zu Postscript (Tabelle 1.19) oder UNICODE zu MacIntosh (Tabelle 1.21). Analoge Tabellen gibt es zur Übersetzung von UNICODE zu Microsoft Windows, zu

PC Code Page Mappings (Latin, Greek,...), zu EBCDIC Code Page Mappings und weiteren.

| UNIC | StdEnc | ISO Latin1 | ZapfDB Symbol | Adobe glyph name | Unicode character name |
|---|---|---|---|---|---|
| 0020 | 20 | 20 | 20 | space | SPACE |
| 0021 | 21 | 21 | 21 | exclam | ECLAMATION MARK |
| 0022 | 22 | 22 | | quotedbl | QUOTATION MARK |
| 0023 | 23 | 23 | 23 | numbersign | NUMBERSIGN |
| 0024 | 24 | 24 | | dollar | DOLLAR SIGN |
| 0025 | 25 | 25 | 25 | percent | PERCENT SIGN |
| 0026 | 26 | 26 | 26 | ampersand | AMPERSAND |
| 0027 | A9 | | | quotesingle | APOSTROPHE-QUOTE |
| 0028 | 28 | 28 | 28 | parenleft | OPENING PARENTHESIS |
| 0029 | 29 | 29 | 29 | parenright | CLOSING PARENTHESIS |
| 002A | 2A | 2A | | asterisk | ASTERISK |
| 002B | 2B | 2B | 2B | plus | PLUS SIGN |
| 002C | 2C | 2C | 2C | comma | COMMA |
| 002D | 2D | AD | | hyphen | HYPHEN-MINUS |
| 002D | | 2D | | minus | HYPHEN-MINUS |
| 002E | 2E | 2E | 2E | period | PERIOD |
| 002F | 2F | 2F | 2F | slash | SLASH |
| 0030 | 30 | 30 | 30 | zero | DIGIT ZERO |
| 0031 | 31 | 31 | 31 | one | DIGIT ONE |
| 0032 | 32 | 32 | 32 | two | DIGIT TWO |
| 0033 | 33 | 33 | 33 | three | DIGIT THREE |
| 0034 | 34 | 34 | 34 | four | DIGIT FOUR |
| 0035 | 35 | 35 | 35 | five | DIGIT FIVE |
| 0036 | 36 | 36 | 36 | six | DIGIT SIX |
| 0037 | 37 | 37 | 37 | seven | DIGIT SEVEN |
| 0038 | 38 | 38 | 38 | eight | DIGIT EIGHT |
| 0039 | 39 | 39 | 39 | nine | DIGIT NINE |
| 003A | 3A | 3A | 3A | colon | COLON |
| 003B | 3B | 3B | 3B | semicolon | SEMIKOLON |

Tabelle 1.19: UNICODE to Adobe Standard Mappings



| UNIC | 6862.2 | SGML | Unicode character name |
|---|---|---|---|
| 0021 | | excl | EXCLAMATION MARK |
| 0023 | | num | NUMBER SIGN |
| 0024 | | dollar | DOLLAR SIGN |
| 0025 | | percnt | PERCENT SIGN |
| 0026 | | amp | AMPERSAND |
| 0027 | | quot | APOSTROPHE-QUOTE |
| 0028 | | lpar | OPENING PARENTHESIS |
| 0029 | | rpar | CLOSING PARENTHESIS |
| 002A | | ast | ASTERISK |
| 002B | 05.00 | plus | PLUS SIGN |
| 002C | | comma | COMMA |
| 002D | | hyphen | HYPHEN-MINUS |
| 002E | | period | PERIOD |
| 002F | | sol | SLASH |
| 003A | | colon | COLON |
| 003B | | semi | SEMICOLON |
| 003C | | lt | LESS-THAN SIGN |
| 003D | | equals | EQUALS SIGN |
| 003E | | gt | GREATER-THAN SIGN |
| 003F | | quest | QUESTION MARK |
| 0040 | | commat | COMMERCIAL AT |
| 005B | | lsqb | OPENING SQUARE BRACKET |
| 005C | | bsol | BACKSLASH |
| 005D | | rsqb | CLOSING SQUARE BRACKET |
| 005E | | circ | SPACING CIRCUMFLEX |
| 005F | | lowbar | SPACING UNDERSCORE |
| 0060 | | grave | SPACING GRAVE |
| 007B | | lcub | OPENING CURLY BRACKET |
| 007C | | verbar | VERTICAL BAR |
| 007D | | rcub | CLOSING CURLY BAR |
| 007E | | tilde | TILDE |
| 00A0 | | nbsp | NON-BREAKING SPACE |
| 00A1 | | iexcl | INVERTED EXCLAMATION MARK |
| 00A2 | | cent | CENT SIGN |
| 00A3 | | pound | POUND SIGN |

Tabelle 1.20: The UNICODE to SGML (ISO DIS 6862.2) Mappings

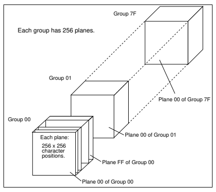


| UNIC | ROM | SYM | GRK | GK2 | HEB | ARB | NAME |
|---|---|---|---|---|---|---|---|
| 0020 | 20 | 20 | 20 | 20 | 20/A0 | 21/A1 | SPACE |
| 0021 | 21 | 21 | 21 | 21 | 21/A1 | 21/A1 | ECLAMATION MARK |
| 0022 | 22 | | 22 | 22 | 22/A2 | 22/A2 | QUOTATION MARK |
| 0023 | 23 | 23 | 23 | 23 | 23/A3 | 23/A3 | NUMBERSIGN |
| 0024 | 24 | | 24 | 24 | 24/A4 | 24/A4 | DOLLAR SIGN |
| 0025 | 25 | 25 | 25 | 25 | 25/A5 | 25 | PERCENT SIGN |
| 0026 | 26 | 26 | 26 | 26 | 26 | 26/A6 | AMPERSAND |
| 0027 | 27 | | 27 | 27 | 27/A7 | 27/A7 | APOSTROPHE-QUOTE |
| 0028 | 28 | 28 | 28 | 28 | 28/A8 | 28/A8 | OPENING PARENTHESIS |
| 0029 | 29 | 29 | 29 | 29 | 29/A9 | 29/A9 | CLOSING PARENTHESIS |
| 002A | 2A | | 2A | 2A | 2A/AA | 2A/AA | ASTERISK |
| 002B | 2B | 2B | 2B | 2B | 2B/AB | 2B/AB | PLUS SIGN |
| 002C | 2C | 2C | 2C | 2C | 2C/AC | 2C | COMMA |
| 002D | 2D | | 2D | 2D | 2D/AD | 2D/AD | HYPHEN-MINUS |
| 002E | 2E | 2E | 2E | 2E | 2E/AE | 2E/AE | PERIOD |
| 002F | 2F | 2F | 2F | 2F | 2F/AF | 2F/AF | SLASH |
| 0030 | 30 | 30 | 30 | 30 | 30/B0 | 30 | DIGIT ZERO |
| 0031 | 31 | 31 | 31 | 31 | 31/B1 | 31 | DIGIT ONE |
| 0032 | 32 | 32 | 32 | 32 | 32/B2 | 32 | DIGIT TWO |
| 0033 | 33 | 33 | 33 | 33 | 33/B3 | 33 | DIGIT THREE |
| 0034 | 34 | 34 | 34 | 34 | 34/B4 | 34 | DIGIT FOUR |
| 0035 | 35 | 35 | 35 | 35 | 35/B5 | 35 | DIGIT FIVE |
| 0036 | 36 | 36 | 36 | 36 | 36/B6 | 36 | DIGIT SIX |
| 0037 | 37 | 37 | 37 | 37 | 37/B7 | 37 | DIGIT SEVEN |
| 0038 | 38 | 38 | 38 | 38 | 38/B8 | 38 | DIGIT EIGHT |
| 0039 | 39 | 39 | 39 | 39 | 39/B9 | 39 | DIGIT NINE |
| 003A | 3A | 3A | 3A | 3A | 3A/BA | 3A/BA | COLON |
| 003B | 3B | 3B | 3B | 3B | 3B/BB | 3B | SEMICOLON |
| 003C | 3C | 3C | 3C | 3C | 3C/BC | 3C/BC | LESS-THAN SIGN |
| 003D | 3D | 3D | 3D | 3D | 3D/BD | 3D/BD | EQUALS-SIGN |
| 003E | 3E | 3E | 3E | 3E | 3E/BE | 3E/BE | GREATER-THAN SIGN |
| 003F | 3F | 3F | 3F | 3F | 3F/BF | 3F | QUESTION MARK |
| 0040 | 40 | | 40 | 40 | 40 | 40 | COMMERCIAL AT |
| 0041 | 41 | | 41 | 41 | 41 | 41 | LATIN CAPITAL LETTER A |
| 0042 | 42 | | 42 | 42 | 42 | 42 | LATIN CAPITAL LETTER B |
| 0043 | 43 | | 43 | 43 | 43 | 43 | LATIN CAPITAL LETTER C |
| 0044 | 44 | | 44 | 44 | 44 | 44 | LATIN CAPITAL LETTER D |
| 0045 | 45 | | 45 | 45 | 45 | 45 | LATIN CAPITAL LETTER E |
| 0046 | 46 | | 46 | 46 | 46 | 46 | LATIN CAPITAL LETTER F |
| 0047 | 47 | | 47 | 47 | 47 | 47 | LATIN CAPITAL LETTER G |
| ⋮ | | | | | | | |

Tabelle 1.21: UNICODE to Macintosh Mappings



Die Notwendigkeit, Control-Codes anderer Codierungen auch verfügbar zu haben, und die Weigerung von Japan und Korea, die vereinheitlichte CJK-Codierung (die mit 19000 statt insgesamt über 31000 Codepositionen ausgekommen wäre) zu akzeptieren, führt zum 32Bit Zeichencode *ISO 10646*, der in seinen ersten 65536 Codes die Zeichen des Unicodes benutzt:




Näheres zum ISO- bzw. Unicode lese man bei

`http://dns.hti.umich.edu/htistaff/pubs/1997/janete.01/`,
`http://www.indigo.ie/egt/standards/iso10646/bmp-today-table.html`,
`http://wwwcn.cern.ch/cnl/215/node47.html`

und

`http://www.unicode.org/`

nach. Die nun zur Verfügung stehenden 4294967296 Codes dürften voraussichtlich für eine Codierung auch der ausgefallensten (ausgestorbenen) Schriften ausreichen.

### 1.2.2 Zahlen

#### 1.2.2.1 Dezimalziffern

Um die zehn Ziffern des Dezimalsystems binär zu codieren, braucht man (mindestens) 4 Bit; im *BCD-Code* oder *8-4-2-1–Code* wird nun jede Dezimalziffer durch die Bitkombination dargestellt, die im Stellenwertsystem der Basis 2 den entsprechenden Wert hat:

$$\sum_{i=0}^{3} m_i \cdot 2^i, \quad m_i \in \{0, 1\}$$

In einem Byte kann man dann zwei Dezimalstellen, in 4 Byte (32 Bit) 8 Dezi-

| Wert | $m_3$ | $m_2$ | $m_1$ | $m_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |

Tabelle 1.22: Zifferndarstellung im BCD-Code

malstellen bzw. in 8 Byte (64 Bit) 16 Dezimalstellen unterbringen. Abgesehen davon, daß in 4 Byte im BCD-Code die Zahlen

$$00.000.000 \ldots 99.999.999,$$

im Dualsystem jedoch die (im Binärstellenwertsystem zu interpretierenden Zahlen)

$$0 \ldots \sum_{i=0}^{31} 1 \cdot 2^i,$$

also

$$0 \ldots 2^{32} - 1 = 0 \ldots 4.294.967.295$$

codiert werden können, ist auch das Rechnen mit BCD-codierten Zahlen aufwendiger: es benötigt mehr Schaltfunktionen / Gatter:



$$\begin{array}{rr r l}
 & 9_{10} & & 1001_2 \\
+ & 3_{10} & + & 0011_2 \\
\hline
 & 12_{10} & & 1100 \quad \leftarrow \text{kein BCD-Code, deshalb Korrektur um 6} \\
 & & + & 0110 \\
\hline
 & & 0001 & 0010
\end{array}$$

Ziffernaddition: 1 Addition, 1 Vergleich $> 9$ und ggf. 1 weitere Addition

Schließlich gibt es Binärmuster, die nicht als BCD-codierte Zahl auftreten können, die Abbildung ist also nicht bijektiv. Aus Gründen der besseren Ökonomie benutzen moderne Computer deshalb fast ausschließlich binär codierte Zahlen.

#### 1.2.2.2 Ein Ausflug in die technische Informatik

Zu Informationen der Halbleiter-Physik vergleiche etwa:

`http://jas.eng.buffalo.edu/applets/education/fab/NMOS/nmos.html`,
`http://www.gel.ulaval.ca/ chabot01/MOSFET/English.html`,
`http://jas.eng.buffalo.edu/applets/education/index2.html`.

Seien $x_i, y_i, z_i, c_i, c_{i+1} \in \{0, 1\}$. Für vorgegebene $c_i, x_i, y_i$ ist dann

$$z_i = x_i \oplus y_i \oplus c_i$$

das Ergebnisbit und

$$c_{i+1} = x_i \cdot y_i \vee x_i \cdot c_i \vee y_i \cdot c_i$$

der Übertrag.

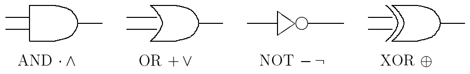



Abbildung 1.13: Gattersymbole



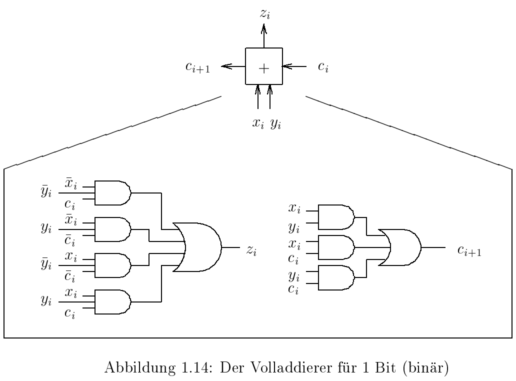



Abbildung 1.14: Der Volladdierer für 1 Bit (binär)

Allgemein gilt:

1. Jede Wertetabelle ist durch *vollständige* Beschaltung (ODER-Normalform) aller Wahrheitskombinationen realisierbar.
2. Es können durch Minimierung Gatter eingespart werden: KV-Diagramme (vgl. Übungen) oder Regeln der Schaltalgebra

Einen (experimentellen) Überblick zur Kombination von NOT-, AND- und OR-Gattern erhält man etwa mittels

`http://math.hws.edu/TMCM/java/xLogicCircuits/index.html`.

Bauen Sie etwa die obige Schaltung zur Übertragsberechung beim Einbit-Addierer mit Übertrageingang nach, und testen Sie.



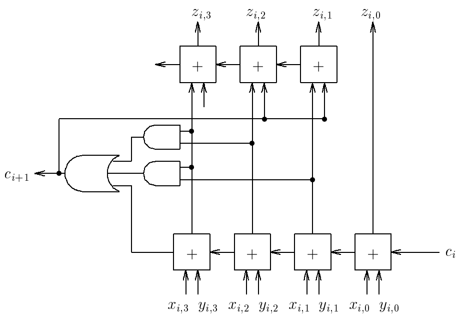
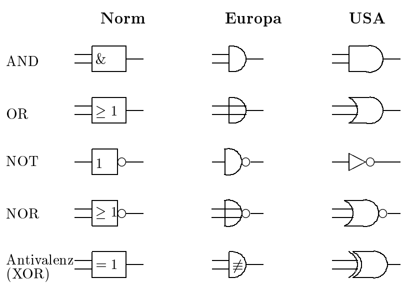



Abbildung 1.15: Der 4-Bit BCD-Addierer: es werden 7 statt 4 Volladdierern und 3 Gatter benötigt.

Der BCD-Addierer ist aufwendiger:

Gatter




und ihre Realisierungen:

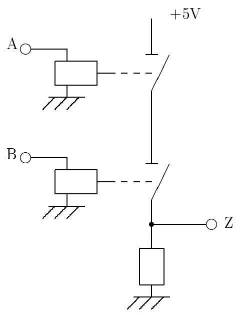



Abbildung 1.16: Relaisschaltung: AND

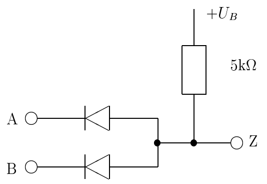



Abbildung 1.17: DL bzw. DTL (positive Logik): AND



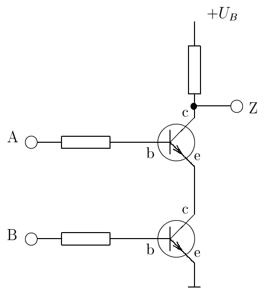



Abbildung 1.18: RTL insbes. DCTL: NAND

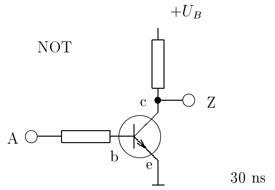



Abbildung 1.19: DTL, Dioden-Transistor-Logik: NOT

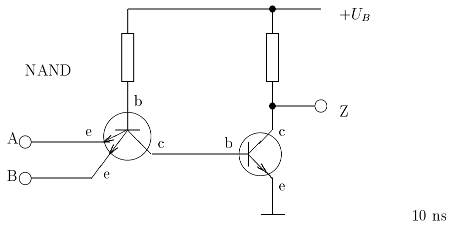



Abbildung 1.20: TTL, Transistor-Transistor-Logik: NAND



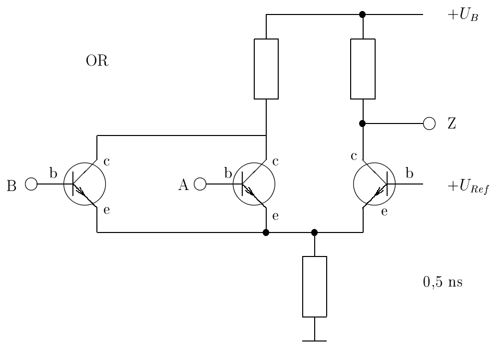



Abbildung 1.21: ECL: OR

Welche Gatter braucht man, um alle möglichen Logikverknüpfungen aufzubauen?

- AND, OR, NOT: Hier ist ein Gatter redundant, denn aus den Morganschen Gesetzen erhält man

$$\text{NOT(A AND B)} = \text{(NOT A) OR (NOT B)}.$$

Speicherbausteine können durch Kippstufen realisiert werden:

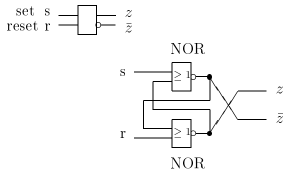



Abbildung 1.22: Speicher = Flipflops = bistabile Kippschaltungen



**RAM** random access memory

1. Informationsverlust bei Stromausfall bei Halbleiter-RAMs, nicht bei Kernspeicher
2. Arten:
   - statisch: SRAM, TTL, Flipflops
   - dynamisch: DRAM, CMOS, interne Kapazität wird zur Speicherung genutzt, dadurch ist etwa alle 2ms ein Auffrischen nötig

**ROM** read only memory, Nur-Lese-Speicher

1. Inhalt überlebt Stromausfall
2. nur einmalig im Werk (bei Fertigung) programmierbar
3. (deshalb:) nur in großen Stückzahlen rentabel

**EPROM** erasable programmable ROM

1. Inhalt überlebt Stromausfall
2. einige Male durch starke Ultraviolettstrahlen lösch- und danach mit +27V neu programmierbar
3. (deshalb:) in kleinen Stückzahlen rentabel

**EEPROM** electrical erasable programmable ROM

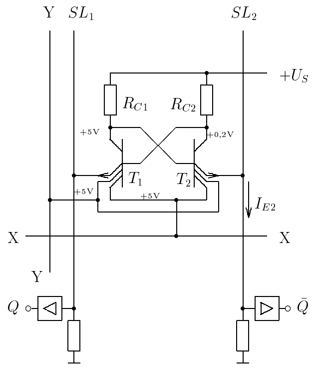
1. Bei einer entsprechenden Konstruktion ist jedes Bit einzeln löschbar
2. elektrisches Löschen
3. ca. 50 oder heute $10^4 - 10^5$ mal änderbar

**NVRAM** non volatile RAM

1. etwa das im PC vorhandene batteriegebufferte Memory-Modul im Uhr-Chip zur Aufnahme der aktuellen Hardware-Konfiguration
2. oft ein EEPROM

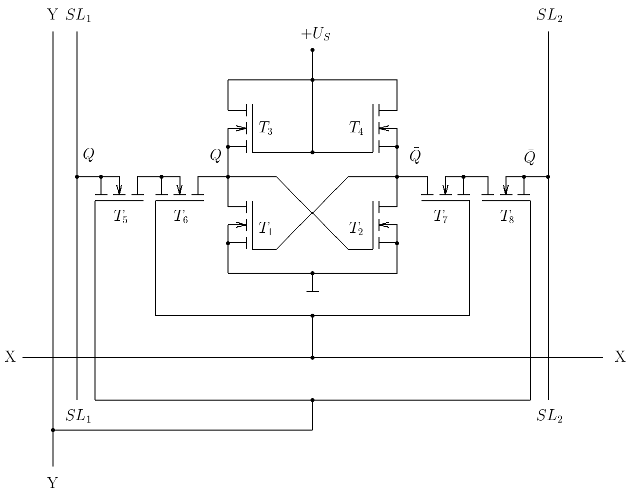
**Realisierung von Speicherzellen:**

SRAM: TTL



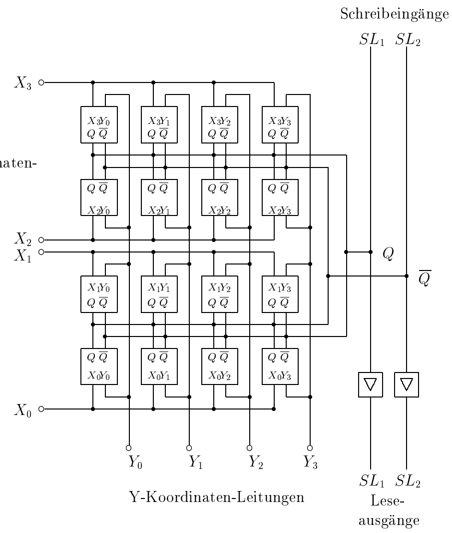



SRAM: N-MOS



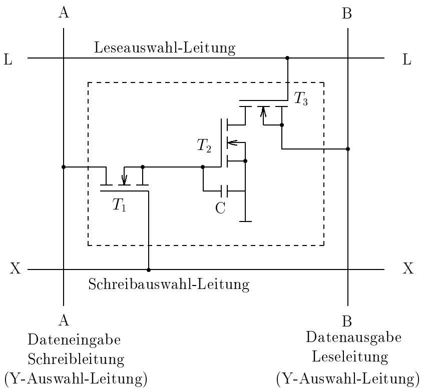



RAM-Speichermatrix:



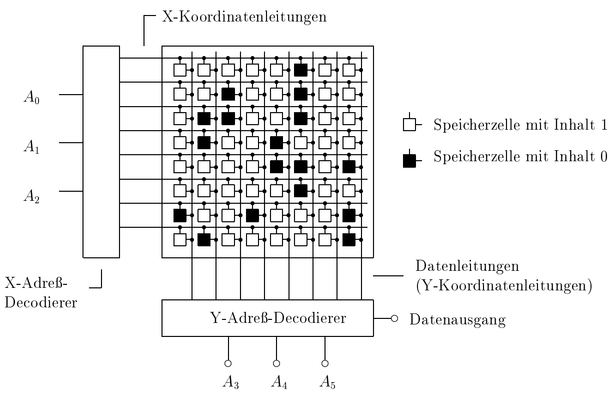


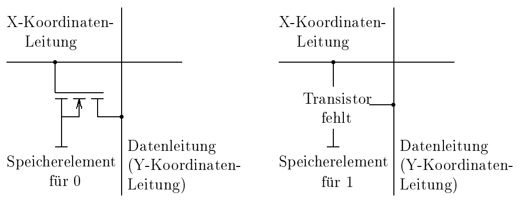

DRAM:




$1 \hat{=} +5V$
$0 \hat{=} 0V$, Masse

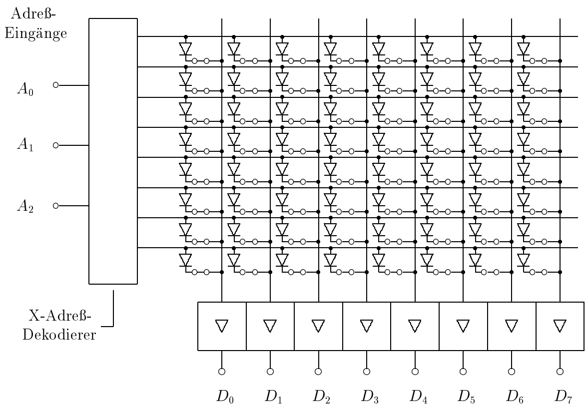
ROM:



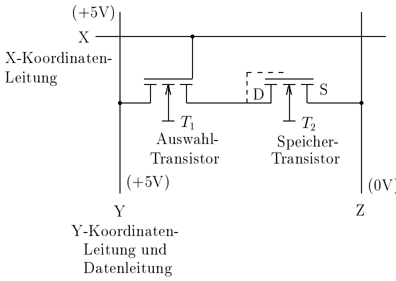






PROM:




EPROM:






#### 1.2.2.3 Vorzeichenlose ganze Zahlen

| Wortlänge | | Zahlenwerte |
|---|---|---|
| Bit | | 0...1 |
| Byte | 8 Bit | $0...2^8 - 1 = 255$ |
| Halbwort | 16 Bit | $0...2^{16} - 1 = 65535$ |
| Wort | 32 Bit | $0...2^{32} - 1 = 4\,294\,967\,295$ |
| Doppelwort | 64 Bit | $0...2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$ |
| Quadwort | 128 Bit | $0...2^{128} - 1$ |
| | | $= 340\,282\,366\,920\,938\,463\,463\,374\,607\,431\,768\,211\,455$ |

Kurzschreibweise für lange Binärfolgen:

| | |
|---|---|
| $1011\,1010_2$ | dual |
| \o272 | oktal |
| \xBA | hexadezimal |

Es besteht die Möglichkeit, vorzeichenlose ganze Zahlen im Stellenwertsystem zur Basis $b$ darzustellen. Dazu sei $x \in \mathbb{N}$. Dann gilt für beliebiges $b \in \mathbb{N} \setminus \{0, 1\}$:

```
i := 0;
Wiederhole
   z_i := x rem b;
   x := x div b;
   i := succ (i);
bis ( x = 0 )
```

liefert die Stellen von $x$ zur Basis $b$:

$$x = \sum_{i=0}^{k} z_i b^i$$

**Aufgabe 1.3**

1. *Beweise die Korrektheit des angegebenen Verfahrens.*
2. *Wie berechnet man den Wert von $(z_N \cdots z_2 z_1 z_0)_b$ ? (Hinweis: Man benutze das Horner-Schema*

$$(\ldots((z_k \cdot b + z_{k-1}) \cdot b + z_{k-2}) \cdot b + \ldots) \cdot b + z_0$$

*und gebe den entsprechenden Pseudocode an.)*



#### 1.2.2.4 Addition statt Subtraktion

Wie berechnet man $1236 + 995$? Nach einer Kopfrechenregel rechnet man

$$1236 + \underbrace{1000 - 5}_{=995} = 2236 - 5 = 2231,$$

d.h. man rechnet

$$\underbrace{1236 - 5}_{\text{Subtraktion}} = \underbrace{1236 + 995}_{\text{Addition}} - \underbrace{1000}_{\text{Korrektur}} .$$

Eine noch bessere Methode – insbesondere für den Computer – ist die Benutzung der Zehnerkomplements, d.h. man rechnet

$$1236 - 5 = \underbrace{1236}_{4stellig} + \underbrace{9995}_{4stellig} - 10000$$

und kann wegen des Korrekturterms den Übertrag in der 5. Stelle einfach vergessen.

**Definition 1.1**
*Das $((N+1)$-stellige) 10er-Komplement von $\sum_{i=0}^{N} y_i \cdot 10^i, y_i \in \{0, \ldots, 9\}$ ist definiert als*

$$10^{N+1} - \sum_{i=0}^{N} y_i \cdot 10^i$$

*Es kann entweder durch eine $(N+2)$-stellige Subtraktionen oder durch Bestimmung des 9er Komplements mit anschließender Korrekturaddition mit 1 bestimmt werden.*

**Definition 1.2**
*Das $((N+1)$-stellige) 9er-Komplement von $\sum_{i=0}^{N} y_i \cdot 10^i, y_i \in \{0, \ldots, 9\}$ ist definiert als*

$$\sum_{i=0}^{N} 9 \cdot 10^i - \sum_{i=0}^{N} y_i \cdot 10^i = \sum_{i=0}^{N} (9 - y_i) \cdot 10^i$$

*Es kann durch $(N+1)$ 1-ziffrige Subtraktionen bestimmt werden.*

**Bemerkung 1.1**
*9er-Komplement und 10er-Komplement sind selbstinvers.*

**Bemerkung 1.2**

$$\textit{9er-Komplement}\left(\sum_{i=0}^{N} y_i\cdot 10^i\right) = \textit{10er-Komplement}\left(\sum_{i=0}^{N} y_i\cdot 10^i\right) - 1$$

Mit Hilfe des 10er Komplements kann also im Beispiel der Subtraktion wie folgt vorgegangen werden:

$$\sum_{i=0}^{N} x_i\cdot 10^i - \sum_{i=0}^{N} y_i\cdot 10^i = \sum_{i=0}^{N} x_i\cdot 10^i + \left(\sum_{i=0}^{N}(9-y_i)\cdot 10^i + 1 \underbrace{-\sum_{i=0}^{N} 9\cdot 10^i - 1}_{=-10^{N+1}}\right),$$

d.h. man addiert das 9er Komplement von $y$, korrigiert um $+1$ und *vergißt* einen eventuell auftretenden Überlauf in der $(N+2)$-ten Stelle. Codiert man $-y$ durch das Zehnerkomplement von $y$, so benötigt die Vorzeichenwechseloperation $(N+1)$ einstellige Subtraktionen und eine $(N+1)$-stellige Korrekturaddition, die Subtraktion $x-y = x+(-y)$ braucht jedoch nur eine $(N+1)$-stellige Addition.

$$\sum_{i=0}^{1} x_i 10^i \in \{0,...,99\}$$

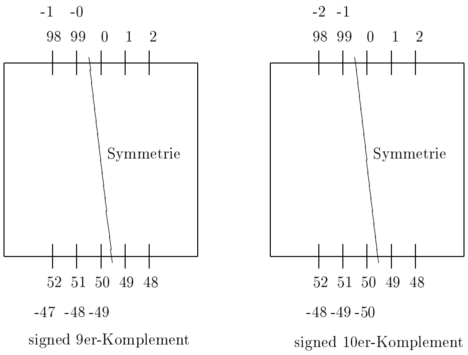



Abbildung 1.23: Darstellung von Zahlen im 9er bzw. 10er Komplement



Es stellt sich die Frage, welche $N$-stelligen Zahlen nun negativ und welche positiv interpretiert werden müssen.

Nach Abbildung 1.23 ergibt sich für die zweistelligen Zahlen im 9er Komplement ein dargestellter Zahlenbereich von

$$\{-49,-48,\ldots,-0,0,1,\ldots,49\} = \{-(10^N\ div\ 2+1),\ldots,-0+0,\ldots,10^N\ div\ 2-1\}$$

und im 10er Komplement

$$\{-50,-49,\ldots,-1,0,1,\ldots,49\} = \{-(10^N\ div\ 2),\ldots,-0+0,\ldots,10^N\ div\ 2-1\}$$

Im 10er Komplement ergibt sich also

$$Wert(x_N\ldots x_1x_0) = \begin{cases} \sum_{i=0}^{N} x_i 10^i & \text{, falls } x_N \in \{0,1,2,3,4\} \\ -10^{N+1} + \sum_{i=0}^{N} x_i 10^i & \text{, falls } x_N \in \{5,6,7,8,9\} \end{cases}$$

Wie ist nun für zwei vorzeichenbehaftete (*signed*) 10er Komplement-Zahlen $a, b$ der Wert $W := Wert(a_N\ldots a_0) + Wert(b_N\ldots b_0)$ zu berechnen?

1. $a_N, b_N \in \{0,1,2,3,4\}$:

$$\begin{aligned} W &= \sum_{i=0}^{N} a_i 10^i + \sum_{i=0}^{N} b_i 10^i \\ &= \sum_{i=0}^{N} (a_i+b_i)10^i \\ &= \sum_{i=0}^{N} \underbrace{\left(a_i + b_i + \left(\sum_{j=0}^{i-1}(a_j+b_j)10^j\right) \text{ div } 10^i\right)}_{=:z_i} \cdot 10^i \end{aligned}$$

Falls $z_N \geq 5$ ergibt sich ein Overflow, d.h. $W$ ist nicht im signed 10er Komplement darstellbar! Die Rechnung leistet besser der folgende iterative Algorithmus:

```
uebertrag:= 0;
i:= 0;
Wiederhole
   z_i := x_i + y_i + uebertrag;
   uebertrag:= z_i div 10;
   z_i := z_i mod 10;
   i:=i+1
bis (i≥ N + 1);
falls (uebertrag≠ 0) ...
```



2. $a_N, b_N \in \{5,6,7,8,9\}$: der Algorithmus funktioniert ähnlich, ist $z_N \in \{0,1,2,3,4\}$ so ergibt sich ein Underflow!

3. $a_N \in \{0,1,2,3,4\}, b_N \in \{5,6,7,8,9\}$ oder $a_N \in \{5,6,7,8,9\}, b_N \in \{0,1,2,3,4\}$: wieder ähnlich, aber keine Probleme mit eventuellem Over- oder Underflow.

#### 1.2.2.5 Negative Zahlen im Dualsystem

Auf Großrechnern werden negative Zahlen zuweilen im 1er-Komplement dargestellt, meist jedoch ist eine Darstellung im 2er-Komplement üblich:

| | |
|---|---|
| Bit 0..1 | $0,-1$ |
| Byte 00..$FF$ | $-128,..,+127$ |
| | Dabei: oberstes Bit$=1 \Leftrightarrow$ Wert $<0$ |
| | Eigentlich: $0,..,127,-128,-127,..,-1$ |
| Halbwort (16 Bit) | $-2^{15}=-32768,..,+32767=2^{15}-1$ |
| Wort (32 Bit) | $-2^{31},..,2^{32}-1$ |
| Doppelwort (64 Bit) | $-2^{63},..,2^{63}-1$ |
| Quadwort (128 Bit) | $-2^{127},..,2^{127}-1$ |

Ein Vorzeichenwechsel wird dabei durch

$$x \mapsto \overline{x}+1$$

realisiert, wobei $\overline{x}$ das bitweise Komplement von $x$ darstellt, also $0 \leftrightarrow 1$. Dies funktioniert allerdings bei der kleinsten negativen Zahl nicht. Bei der Addition tritt ein Überlauf genau dann auf, wenn im obersten Bit ein Übertrag stattfindet oder (xor) aus dem obersten Bit heraus ein Übertrag erfolgt. Leider stellen viele Programmiersprachen (z.B. Pascal) einen solchen Überlauf nicht fest!
Die Multiplikation zweier Zahlen erfolgt analog zum schriftlichen Rechenverfahren, in CISC-CPU's häufig mittels Microprogrammen, sonst mittels Parallel- oder durch Seriell-Addierer. Gleiches gilt für die Division. Spezialfunktionen wie *log* oder *arctan* werden durch abbrechende Taylorreihen, Newtoniteration, Kettenbrüche, Funktionaltheoreme o.ä. realisiert. Dies geschieht meist in Software.