Wie rechnet ein Computer?

Zahlendarstellung

Das Dezimalsystem

Das Dezimalsystem kennen wir alle. Das sind die Zahlen, die wir lesen und sofort verstehen können. Schauen wir noch einmal, wie diese Zahlen mathematisch funktionieren. Jede Ziffer bekommt eine unterschiedliche Bedeutung, je nachdem, wo sie in der Zahl steht. Ganz rechts sind die "Einer". Das heißt, eine 1 bedeutet auch 1. An der zweiten Stelle von rechts bedeutet die 1 schon 10 usw. Das lässt sich mathematisch so darstellen:

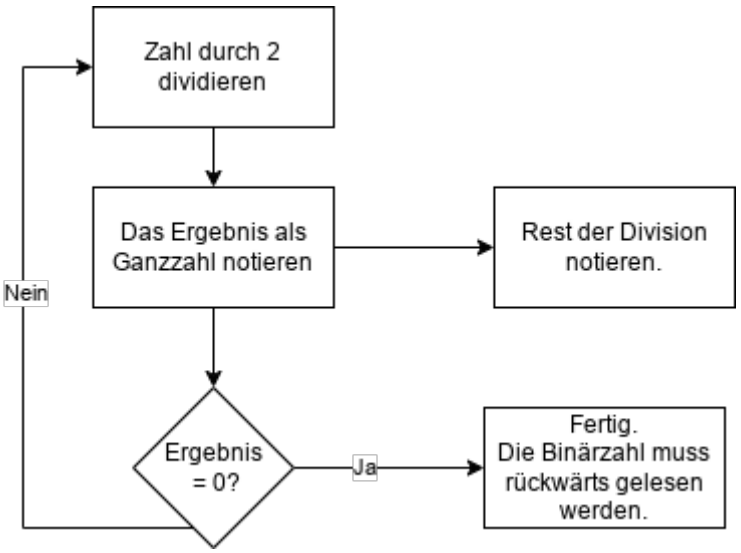
$$4711_{10} = 4 * 10^3 + 7 * 10^2 + 1 * 10^1 + 1 * 10^0$$

Dasselbe funktioniert auch, wenn wir weniger Ziffern zur Verfügung haben, z.B. nur zwei, nämlich 0 und 1:

$$1101_2 = 1 * 2 \ + 1 * 2 \ + 0 * 2 \ + 1 * 2 \ = 1 * 8 + 1 * 4 + 0 * 2 + 1 * 1 = 13_{10}$$

Auf diese Weise können Binärzahlen in Dezimalzahlen umgewandelt werden. Zur umgekehrten Rechnung kann man einen Algorithmus verwenden.

Algorithmus zum Berechnen einer Binärzahl



Beispiel:

```
190:2=95 Rest 0
95:2=47 Rest 1
```

```
47:2=□23□Rest 1
23:2=□11□Rest 1
11:2=□5□Rest 1
5:2=□2□Rest 1
2:2=□1□Rest 0
1:2=□0□Rest 1
```

Die Binärzahl lautet dann 10111110. Sie muss also von unten nach oben gelesen werden.

Programmierung dieses Algorithmus

So sieht die Grundstruktur des Programms aus:

```
import math
def convertDecToBin(dec):
    pass
    # Code

def convertToDec(number):
    pass
    # Code
number = input("Welche Zahl moechtest du umwandeln?\n")

zahl= convertDecToBin(int(number))
print(f"Die Zahl {number} als Binaerzahl lautet: {zahl}")
print(f"Die urspruengliche Zahl war: {convertToDec(zahl)}")
```

Addition von Binärzahlen

Addition von Dezimalzahlen

$$\begin{array}{rr} & 2752 \\ + & 4261 \\ \hline & 7013 \end{array}$$

Addition von Binärzahlen

$$\begin{array}{rr} & 10010 \\ + & 100111 \\ \hline & 111001 \end{array}$$

Die Addition funktioniert wie die Addition von Dezimalzahlen. Bei Dezimalzahlen stehen 10 Ziffern zur Verfügung. Ab der 10. Ziffer kommt es zu einem Überlauf, d. h., sie lässt sich nicht mehr darstellen. Daher muss die 10er-Stelle als Übertrag notiert werden und bei der Addition der nächsten Stelle berücksichtigt werden. Bei den Binärzahlen stehen nur zwei Ziffern zur Verfügung. Daher kommt es schon früher zu einem Überlauf. Dieser muss genauso berücksichtigt werden wie bei der Addition von Dezimalzahlen.

Logische Schaltungen

Ein Computer kann nur Nullen und Einsen. Das weiß mehr oder weniger jeder. Doch wie kann ein Computer damit rechnen? Darum geht es in diesem Kapitel.

Simulation von logischen Schaltungen

Zur Übung simulieren wir die Schaltungen, aus denen ein Computerchip aufgebaut ist, mit Digital Logic Sim. Die ZIP-Datei muss nur entpackt werden und das Programm wird dann mit einem Doppelklick auf `Digital LOgic Sim.exe` gestartet.

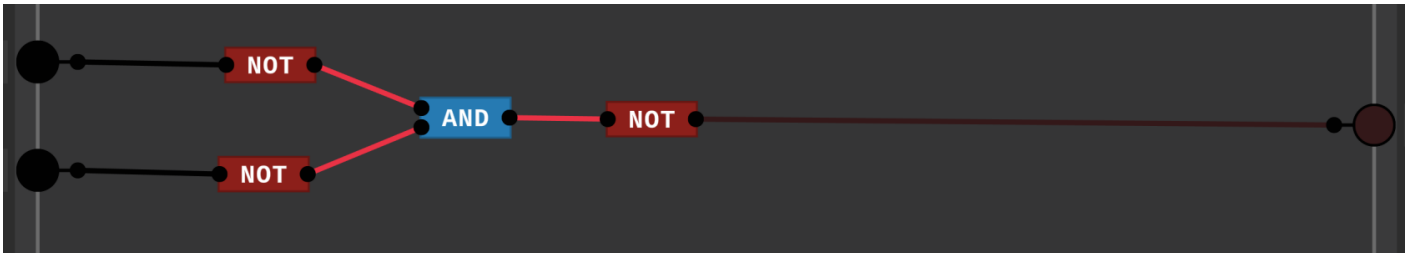
Mithilfe dieses Programms soll aus einer AND- und einer NOT-Schaltung ein Addierer aufgebaut werden, mit dem man zwei 1-Bit Binärzahlen addieren kann. Wer weitermachen möchte, kann damit natürlich auch einen 4-Bit Addierer bauen. Unter Windows werden die Daten unter diesem Verzeichnis gespeichert:

`C:\Benutzer\BENUTZERNAME\AppData\LocalLow\SebastianLague\Digital Logic Sim\V1\Projects`.

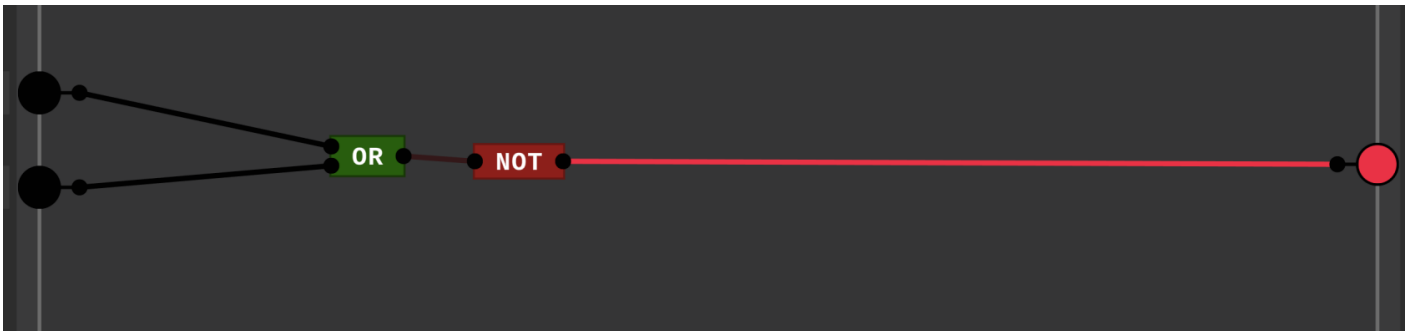
Um das Verzeichnis im *Windows Explorer* anzuzeigen muss unter *Ansicht* noch *Ausgeblendete Elemente* angewählt werden.

Hier sind die Schaltbilder der einzelnen Elemente. Diese müssen in dieser Reihenfolge angelegt werden.

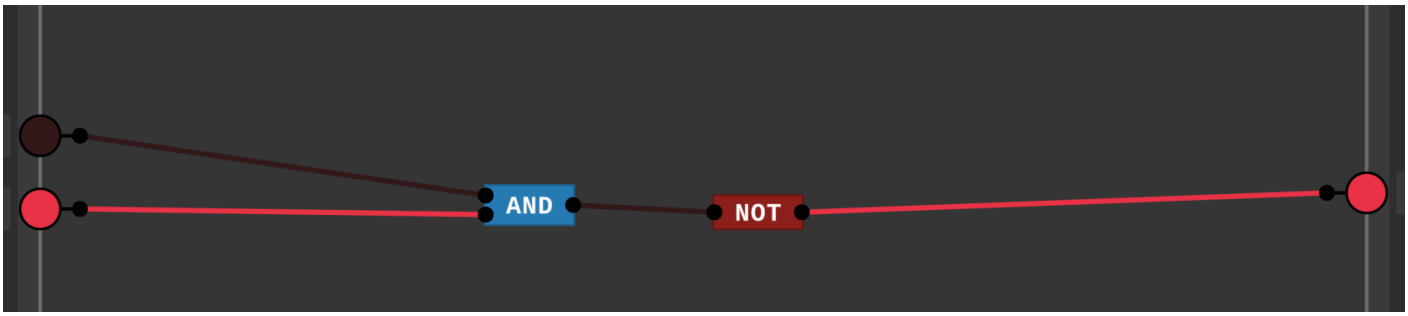
OR



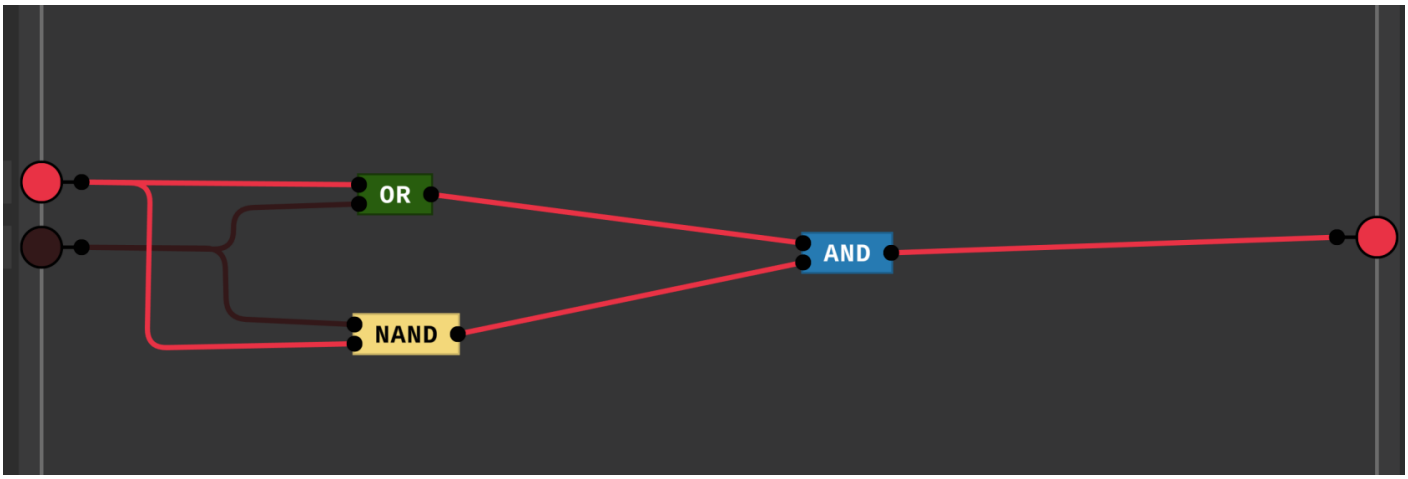
NOR



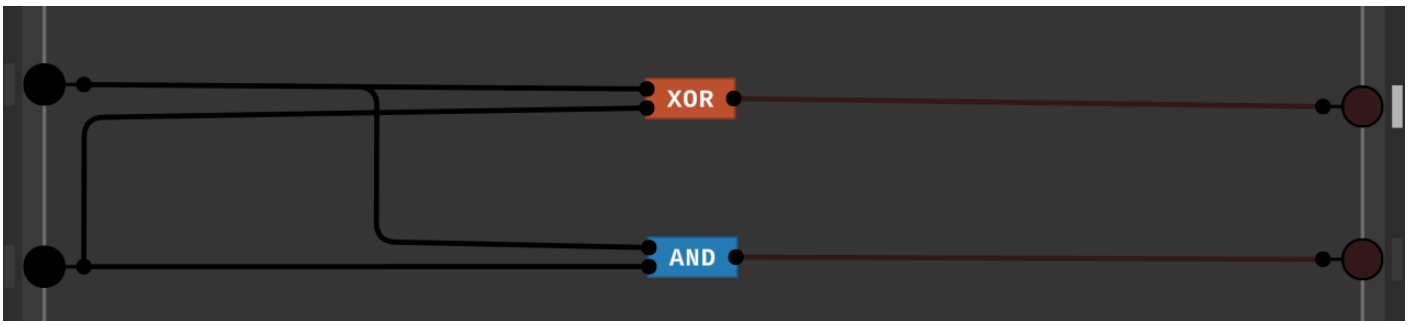
NAND



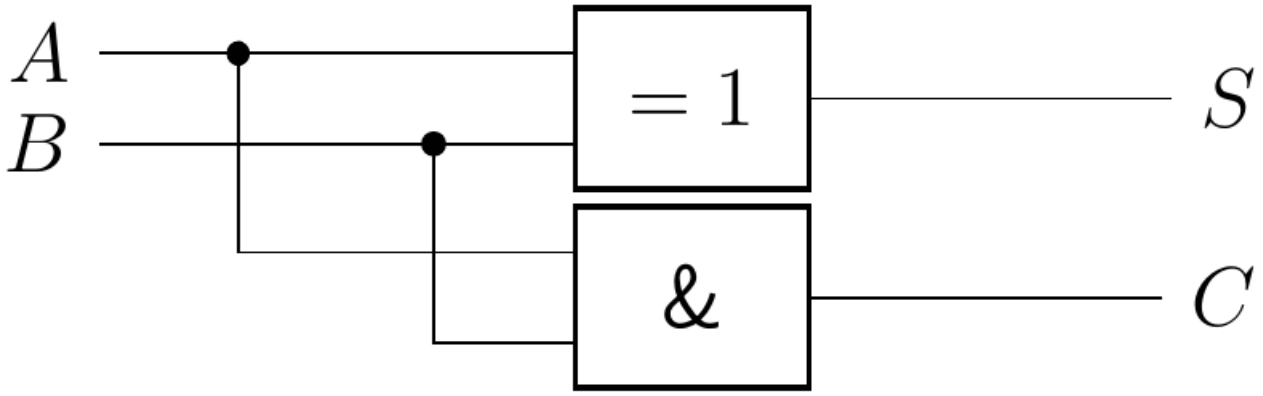
XOR



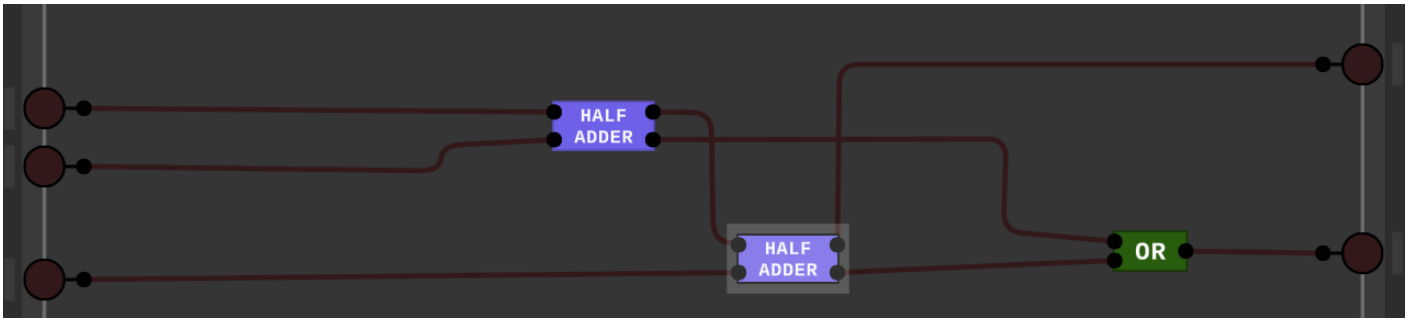
HALF ADDER



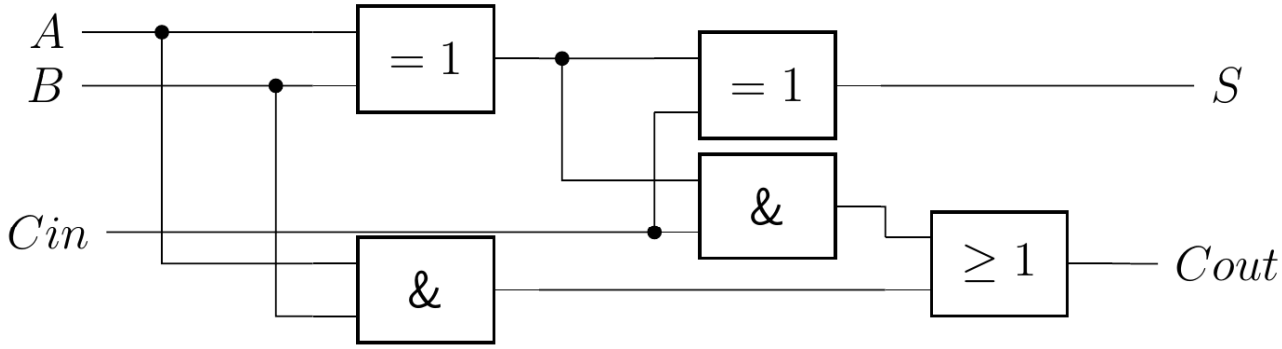
oder die Version mit Schaltsymbolen:



ADDER

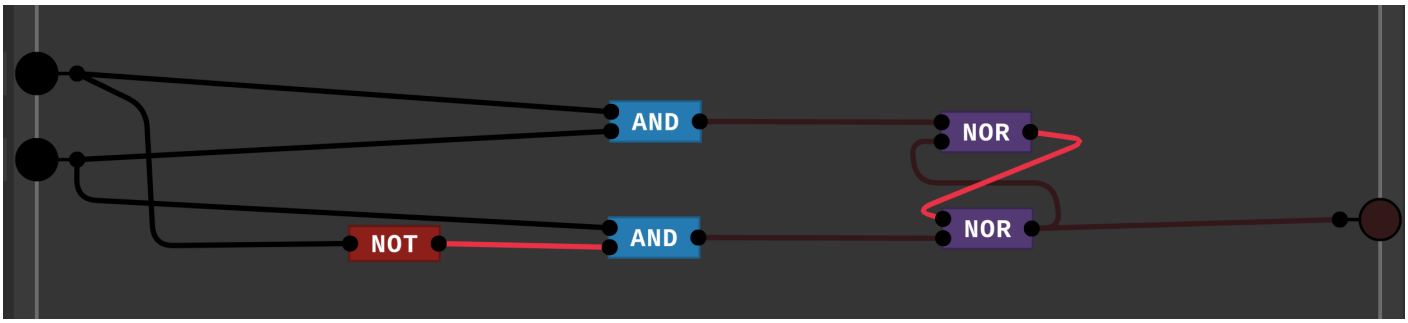


oder eine Version ohne den Halbaddierer:

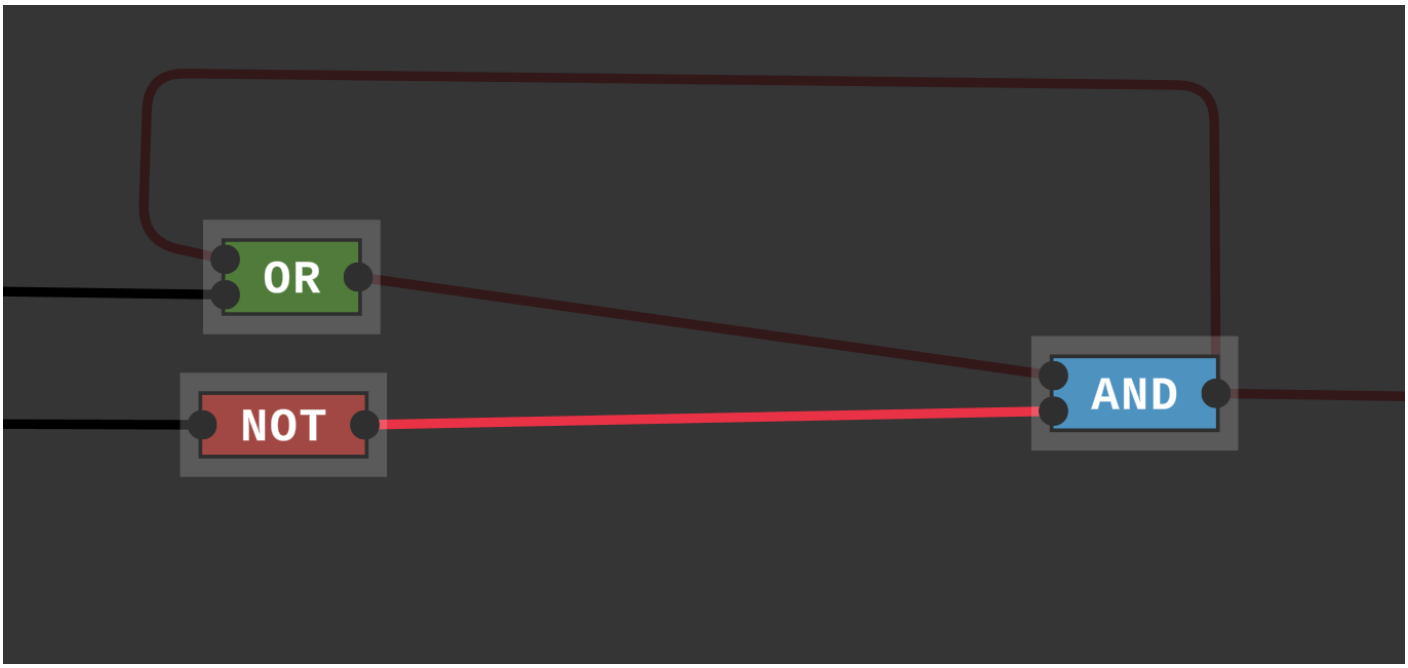


Simulation von Computerspeicher

D-Latch

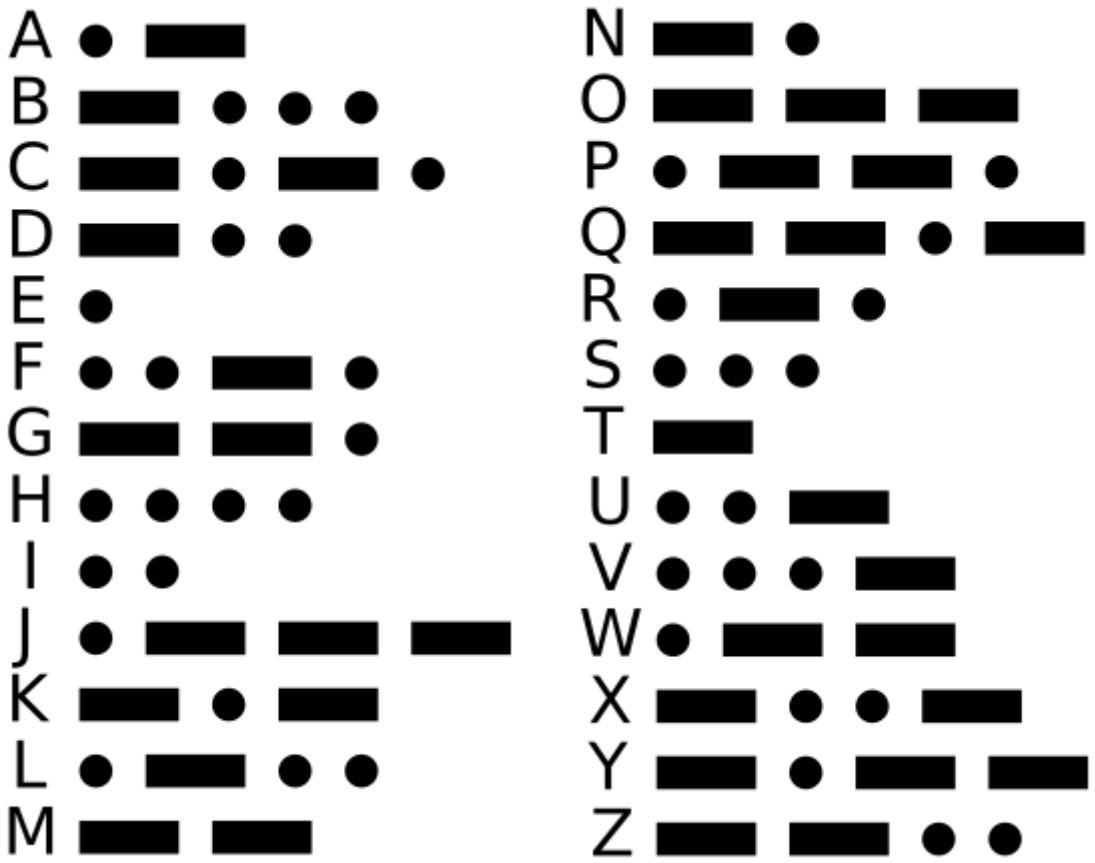


SR-Latch



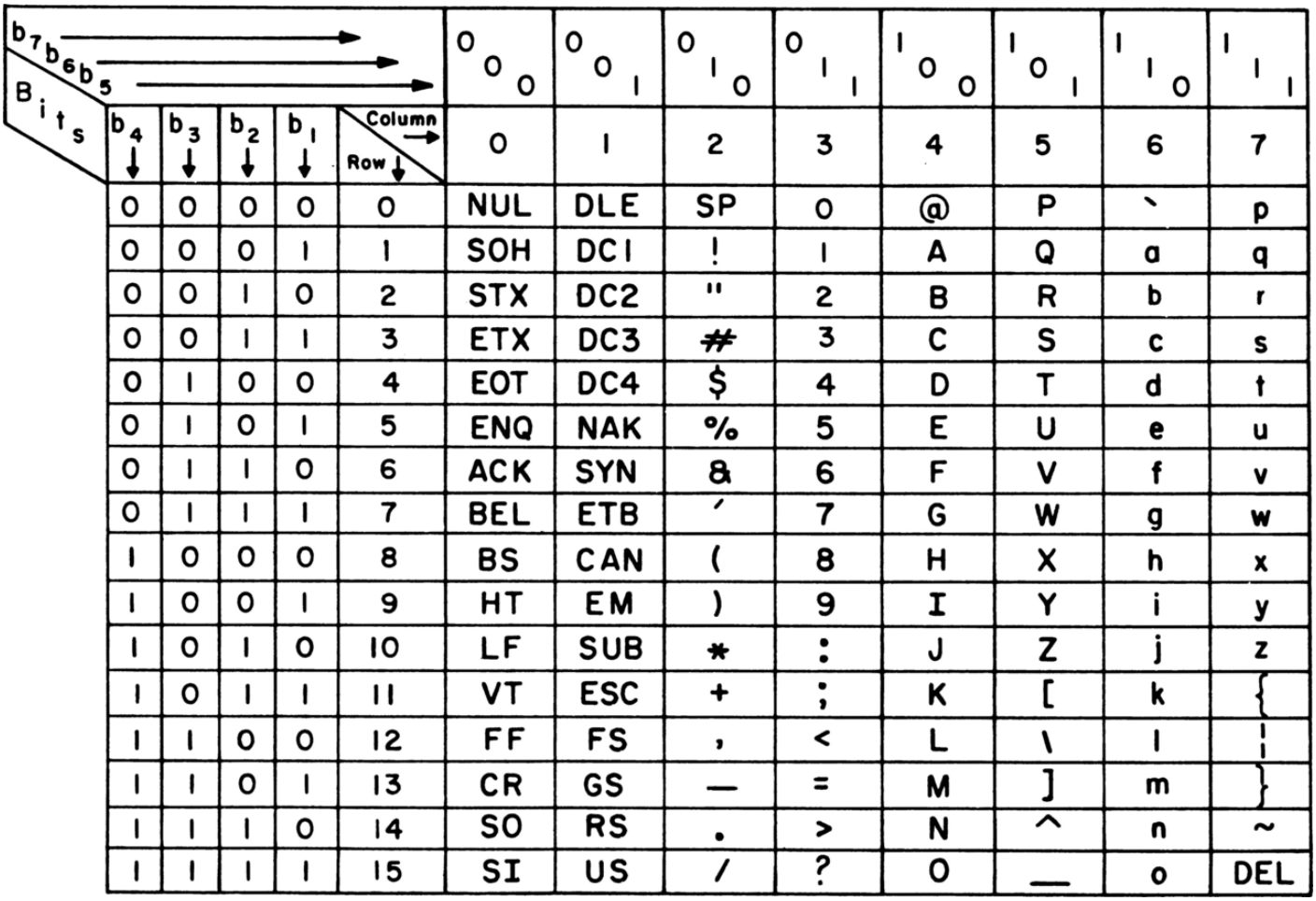
Kodierung von Buchstaben

Ein Vorläufer der heutigen Kodierung von Buchstaben ist das Morsealphabet.



So wie ein Computerchip, konnte man mit Morsegeräten nur zwischen einem kurzen und einem langen Signal unterscheiden. Jeder Buchstabe bekam also eine Kombination aus kurzen und langen Impulsen zugewiesen.

Für die Kodierung von Buchstaben und anderen Zeichen auf dem Computer wurde auch eine Kodierungstabelle entwickelt. Die erste Tabelle hier ASCII (American Standard Code for Information Interchange)



$b_7 b_6 b_5$ →	000	001	010	011	100	101	110	111
Bits b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Weitere Informationen