Technische Hochschule **Rosenheim**

Modul - IT Systems (IT)

Bachelor Programme AAI

01 - Lecture: Logic Circuits

Prof. Dr. Marcel Tilly

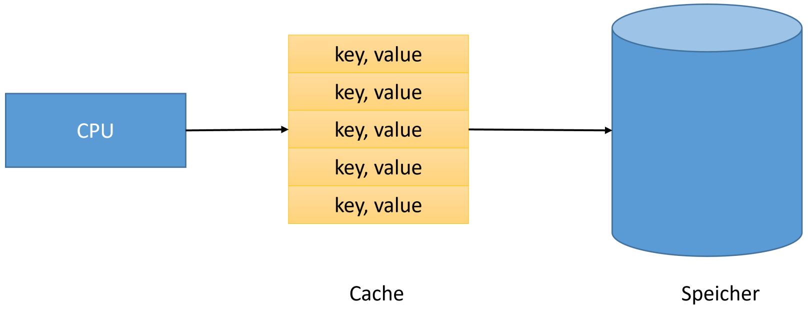
Faculty of Computer Science, Cloud Computing

Vers. 18.10.2021

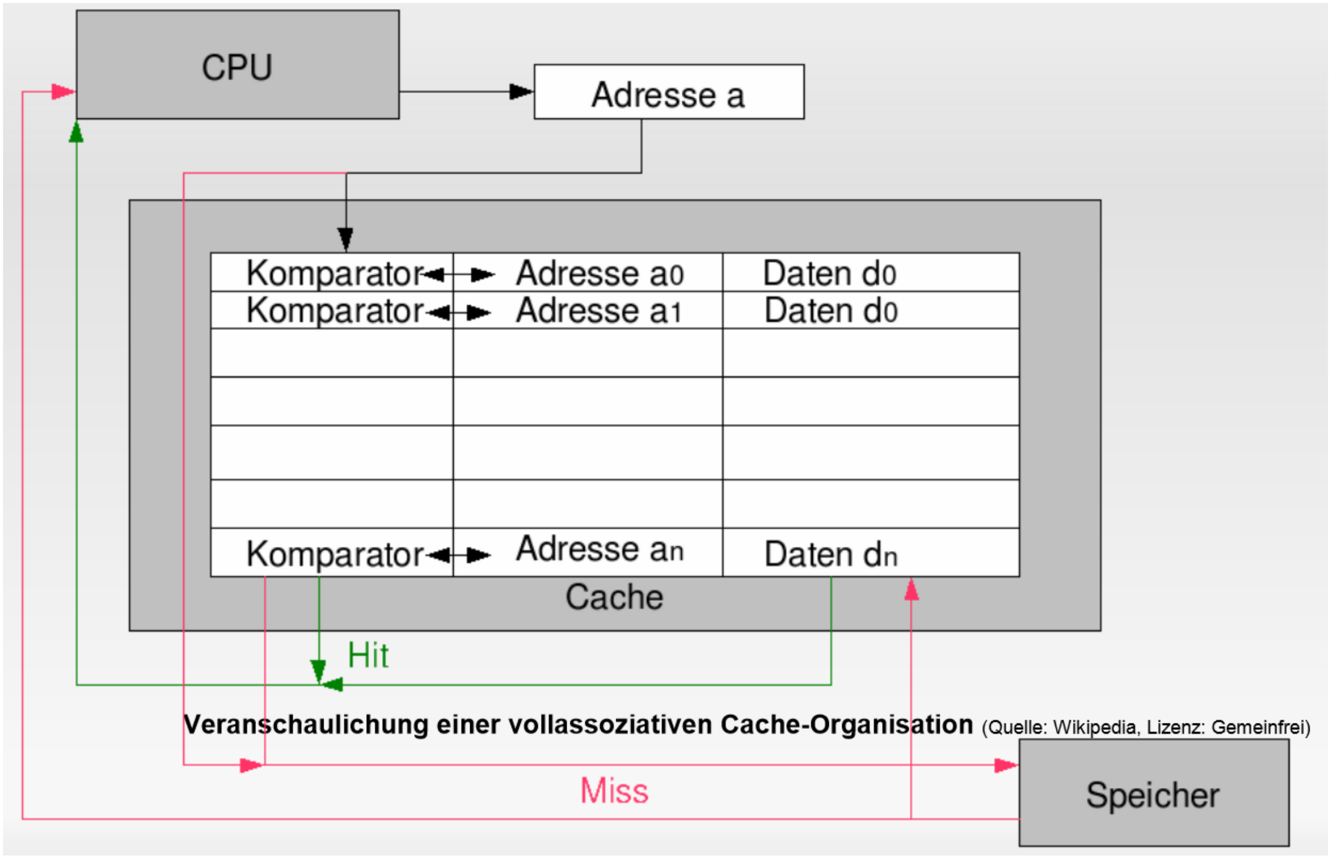
Technische Hochschule Rosenheim

Before we start ...

Recap



Cache Organisation



Veranschaulichung einer vollassoziativen Cache-Organisation (Quelle: Wikipedia, Lizenz: Gemeinfrei)

Replacement Strategy

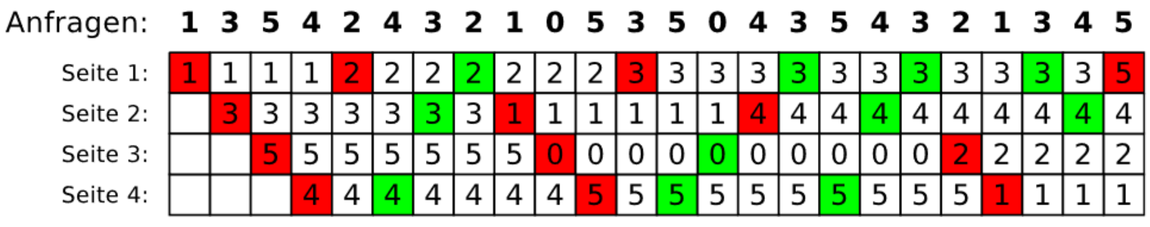
Overwrite/replace entries in the cache:

- *random* or *random procedure*: Occupancy of the *cache* as uniform as possible
  - A randomly selected entry is overwritten
- First in, first out (FIFO)
  - Oldest block is replaced (even if currently used)
- NRU (*not recently used*)
  - Division into 4 classes by means of *used* and *modified* bit
    1. not used, not modified entries
    2. not used, modified entries
    3. used, not modified entries
    4. used, modified entries
  - Overwrite a randomly selected entry of the lowest class that is not empty
- LRU (*least recently used*)
  - The longest unused block is replaced
- Measure is *Cache Hit Ratio* (CHR)

$$CHR = \frac{Number\ of\ Cache\ Hits}{Number\ of\ Cache\ Hits + Number\ of\ Cache\ Misses}$$

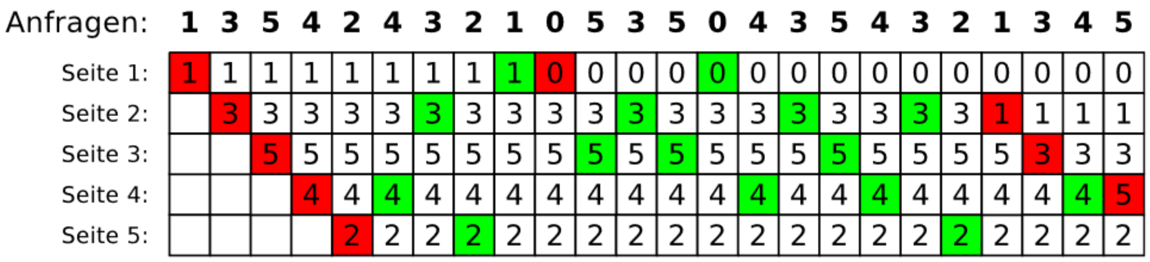
Example: FIFO

If a value has to be displaced in the FIFO exchange procedure, the entry that has been in the memory the longest is displaced. (= **First in, First out**)



Anfragen:	1	3	5	4	2	4	3	2	1	0	5	3	5	0	4	3	5	4	3	2	1	3	4	5
Seite 1:	🟥1	1	1	1	🟥2	2	2	🟩2	2	2	2	🟥3	3	3	3	🟩3	3	3	🟩3	3	3	🟩3	3	🟥5
Seite 2:		🟥3	3	3	3	3	🟩3	3	🟥1	1	1	1	1	1	🟥4	4	4	🟩4	4	4	4	4	🟩4	4
Seite 3:			🟥5	5	5	5	5	5	5	🟥0	0	0	0	🟩0	0	0	0	0	0	🟥2	2	2	2	2
Seite 4:				🟥4	4	🟩4	4	4	4	4	🟥5	5	🟩5	5	5	5	🟩5	5	5	5	🟥1	1	1	1

Hitrate: 11/24 = 0,4583333%
Missrate: 13/24 = 0,5416666%



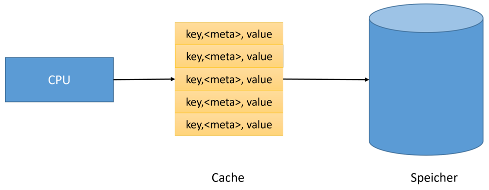
Anfragen:	1	3	5	4	2	4	3	2	1	0	5	3	5	0	4	3	5	4	3	2	1	3	4	5
Seite 1:	🟥1	1	1	1	1	1	1	1	🟩1	🟥0	0	0	0	🟩0	0	0	0	0	0	0	0	0	0	0
Seite 2:		🟥3	3	3	3	3	🟩3	3	3	3	3	🟩3	3	3	3	🟩3	3	3	🟩3	3	🟥1	1	1	1
Seite 3:			🟥5	5	5	5	5	5	5	5	🟩5	5	🟩5	5	5	5	🟩5	5	5	5	5	🟥3	3	3
Seite 4:				🟥4	4	🟩4	4	4	4	4	4	4	4	4	🟩4	4	4	🟩4	4	4	4	4	🟩4	🟥5
Seite 5:					🟥2	2	2	🟩2	2	2	2	2	2	2	2	2	2	2	2	🟩2	2	2	2	2

Hitrate: 15/24 = 0,625%
Missrate: 9/24 = 0,375%

Cache Structure

A line of the cache contains various information:

- A data block containing one or more memory locations of the cache.
- If more than one, the least significant address bits correspond as "offset" to the position in the cache line
- A tag that specifies which data is currently stored there (usually most significant part of the address)
- Status bits as needed, e.g..
    - **Valid Bit**: Indicates whether the cache line has a valid allocation
    - **Dirty Bit**: For write-back caches, indicates whether the data has been modified



Write to a Cache

Write-policies

When writing, you have two options when using a cache that already contains the memory address:

- **Write-through method**: The data is written to memory and cache in parallel.
    - Advantages/Disadvantages?
- **Write-Back method**: The data is only updated in the cache and marked as changed ("dirty") there. If the cache wants to displace it, it has to write the changed date to memory first.
    - Advantages/Disadvantages?

Agenda

We will look at the following in this session:

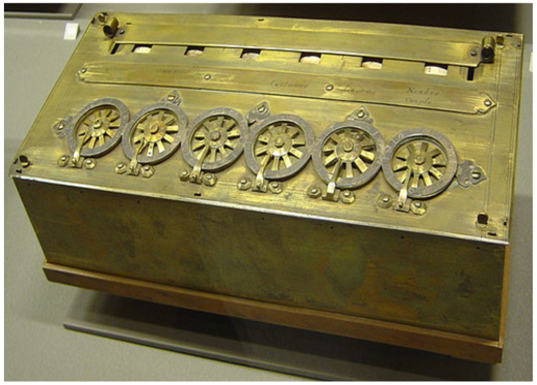
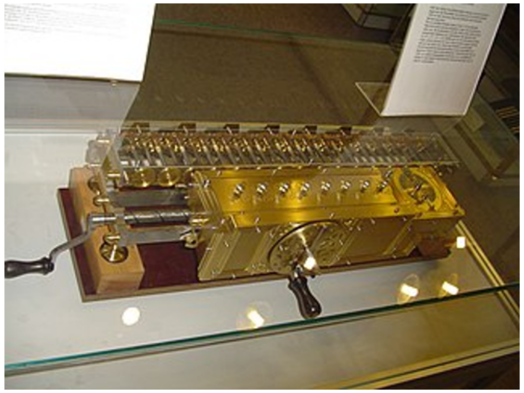
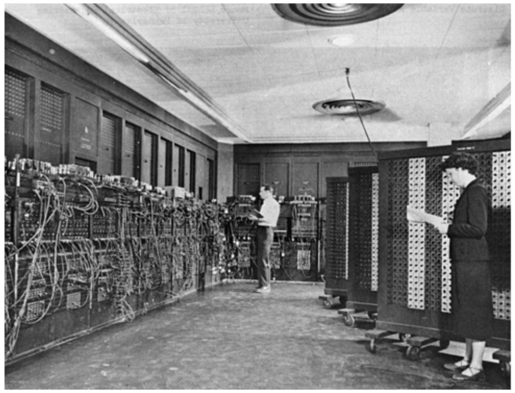
- Boolean algebra
- Boolean functions
- Truth tables
- Switching functions
- Logic gates
- Half and full adders

Objectives

Students ...

- ... know how to use Boolean algebra
- ... can calculate simple logical circuits
- ... can determine a Boolean function for a simple given logic circuit

What has happened so far?

Pacaline	Leibnitzrechenmaschine	ENIAC
		
Addition und Subtraktion	Addition und Subtraktion Division und Multiplikation	Addition und Subtraktion Division und Multiplikation Quadratwurzel

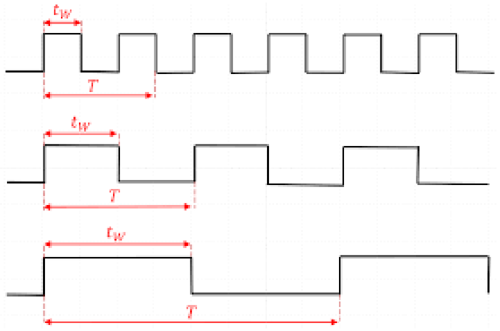
But how do they calculate at all or better asked: **How does a computer calculate?**

Initial situation

The electronics in a computer works *digital*.

What does *digital* mean?

- in *digital electronic* it works with only two voltage levels



- that's why computers use '0' and '1' to transmit information
- a binary system exactly fulfills the requirements of the *digital electronic* (abstract)
- '0' and '1' are the complementary or inverse values of each other

Boolean Algebra

Basis of the processing of an information on bit level is the Boolean algebra. (Origin: George Boole (1815-1864))

Purpose.

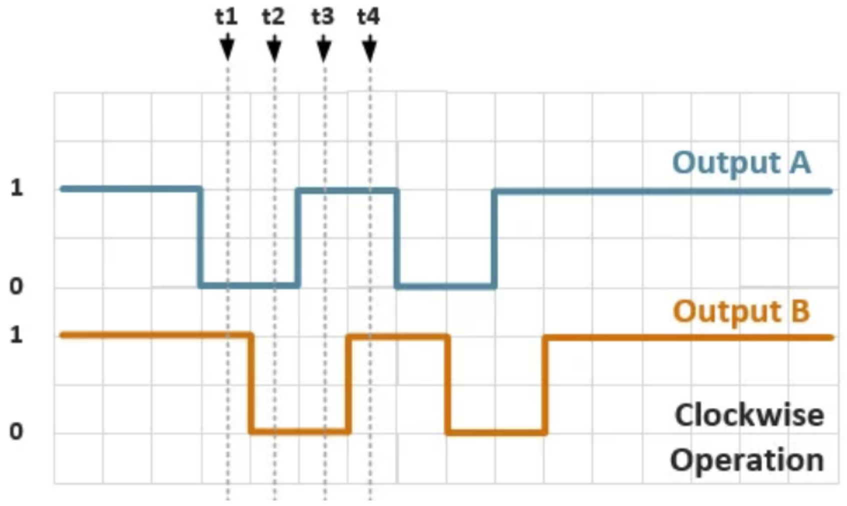
- To determine *true* and *false* statements.
- Boolean algebra operates on the two truth values: true (*true*) and false (*false*).

> **Note.**
>
> - Boolean algebra is closely related to logic.
> - Two states false and true → two-valued (binary). These can also be described by 0 and 1.

Electronics

In application (digital electronics), Boolean algebra is also called switching algebra.

- T (True), H (High), 1 (Bit set)
- F (False), L (Low), 0 (Bit not set)



Clockwise
Sequence

	A	B
t1	0	1
t2	0	0
t3	1	0
t4	1	1

Operations

In Boolean algebra, one deals with links of truth values by logical operators whose results are in turn truth values.

Basic logical operations:

connection	name	notation
not a	negation	$\neg a$
a and b	conjunction	$a \land b$
a or b	disjunction	$a \lor b$
if a then b	implication	$a \Rightarrow b$
a exactly if b	equivalence	$a \Leftrightarrow b$

Boolean algebra - formal definition

- An *algebra* is determined by a set of values, by its operations on this set of values, and by the associated laws of computation.
- Boolean algebra is based on the binary set {0, 1} and the operations ∨, ∧ and ¬.
- An *operation* is composed of the operator that specifies which activity is performed and the operands to which the operation refers.
- The operations of Boolean algebra are defined as follows:
  - One-place operator ¬, with only one operand (here a):

bit a	**¬a (negation)**
0	1
1	0

Boolean algebra - formal definition

- Two-digit operators ∧ and ∨, with two operands (here a and b):

bit a	bit b	a ∧ b (AND)	a ∨ b (OR)
1	0	0	1
1	1	1	1
0	0	0	0
0	1	0	1

> **Note**: The AND operation (∧) has priority over the OR operation (∨).
>
> So: **a ∨ b ∧ c = a ∨ (b ∧ c)**

Boolean algebra - formal definition

In Boolean algebra, the following laws of arithmetic apply:

	$\vee$ (OR)	$\wedge$ (AND)
commutative law	$a \vee b = b \vee a$	$a \wedge b = b \wedge a$
associative law	$(a \vee b) \vee c = a \vee (b \vee c)$	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$
distributive law	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
absorption Law	$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$
neutral Element	$0 \vee a = a$	$1 \wedge a = a$
inverse element	$a \vee \neg a = 1$	$a \wedge \neg a = 0$

Proof

How can these laws be proved?

- Commutative law: $a \lor b = b \lor a$
- Associative law: $(a \lor b) \lor c = a \lor (b \lor c)$
- Neutral element: $0 \lor a = a$

Let's try it out!

Truth functions

definition: A function $f : B^n -> B$ **is called an n-digit Boolean function**. There exist 2^{2^n} n digit functions.

- Example : **n=1 (one-digit function)** , $y = f_0(x), \ldots, f_3(x)$

Combining all possible assignments of arguments and results in the following

$$2^2 = 4$$

one-digit truth functions

a	constant 0	identity a	negation ¬a	constant 1
0	0	0	1	1
1	0	1	0	1
	f0	f1	f2	f3

Truth functions

and for n=2 the

$$2^4 = 16$$

two-digit truth functions:

a,b	0,0	0,1	1,0	1,1	notation	designation
f1	0	0	0	0	Constant 0	
f2	0	0	1	a∧b	conjunction (AND)	
f3	0	0	1	0	a ∧ ¬b	negation of implication
f4	0	0	1	1	a	identity a
f5	0	1	0	0	¬a ∧ b	negation of implication
f6	0	1	0	1	b	identity b

Truth functions

a,b	0,0	0,1	1,0	1,1	Spelling	Designation
f7	0	1	0	????	antivalence (XOR)	
f8	0	1	1	a ∨ b	disjunction (OR)	
f9	1	0	0	¬(a∨b)	non-or (NOR)	
f10	1	0	1	????	equivalence	
f11	1	0	1	0	¬b	negation of b
f12	1	0	1	1	¬(¬a∧b)	implication
f13	1	1	0	0	¬a	negation of a
f14	1	1	0	1	¬(a∧¬b)	implication
f15	1	1	0	¬(a∧b)	Not-And (NAND)	
f16	1	1	1	1	constant 1	

Truth functions

a,b	0,0	0,1	1,0	1,1	Spelling	Designation
f7	0	1	1	0	$(a \land \neg b) \lor (\neg a \land b)$	antivalence (XOR)
f8	0	1	1	$a \lor b$	disjunction (OR)	
f9	1	0	0	$\neg(a \lor b)$	non-or (NOR)	
f10	1	0	0	1	$(a \land b) \lor (\neg a \land \neg b)$	equivalence (XNOR)
f11	1	0	1	0	$\neg b$	negation of b
f12	1	0	1	1	$\neg(\neg a \land b)$	implication
f13	1	1	0	0	$\neg a$	negation of a
f14	1	1	0	1	$\neg(a \land \neg b)$	implication
f15	1	1	0	$\neg(a \land b)$	Not-And (NAND)	
f16	1	1	1	1	constant 1	

Task 1

Two-digit logical operations

By combining the operations conjunction, disjunction and negation, all possible one-digit and two-digit logical operations can be expressed.

Example:

- f1: $a \land \neg a = 0$
- f6: $b \land 1 = b$
- f14: $a \Rightarrow b = \neg a \lor b$
- f16: $a \lor \neg a = 1$

Can you prove this?

Laws of Boolean Algebra

Technische Hochschule **Rosenheim**

All rules for calculating with logical operations result from the definition of Boolean algebra:

- Involutive law

$\neg(\neg a) = a$

- Merge rules

$\neg a \lor a \land b = \neg a \lor b$

$a \lor \neg a \land b = a \lor b$

$a \land (\neg a \lor b) = a \land b$

$\neg a \land (a \lor b) = \neg a \land b$

- de Morgan's laws

$\neg(a \land b) = \neg a \lor \neg b$

$\neg(a \lor b) = \neg a \land \neg b$

- Idempotence law

$a \lor a = a$

$a \land a = a$

Boolean function f

Previous: 1 input (a) or 2 inputs (a and b) and one output.

Generalization: n inputs 1 output.

> Note: A *boolean function* assigns a binary output variable y to n binary input variables x1,..,xn.

$y = f(x_1, x_2, ..x_n)$ with $y \in 0, 1$ and $x_i \in 0, 1$

Truth tables

Boolean functions can be expressed by truth tables.

bit a	bit b	bit c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Boolean Normal Form

Purpose: Using the theorem of Boolean normal form, one can construct a Boolean expression from a truth table of the Boolean function. In this case, the switching function is expressed only by the 3 base operators (¬,∨,∧):

bit a	bit b	bit c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

find solution for

```
f(a,b,c) = ... ∨... ∨ ...
```

Normal forms

- The truth table is a unique definition of a boolean function
- However, there are infinitely many different realizations using logic gates or descriptions in the form of a boolean expression
- Normal forms (also canonical forms):
    - Standard representation for a Boolean expression in a unique algebraic form.

Minterm

Given a Boolean function $y = f(x_n, \ldots, x_1, x_0)$ and the literals x_i.

minterm: $(x_n \wedge \cdots \wedge x_2 \wedge x_1 \wedge x_0)$

- The minterm evaluates to 1 for exactly one particular configuration of the variable x_i and to 0 otherwise
- More precisely, the minterm evaluates to 1 if for all negated variables $x_i = 0$ and all non-negated variables $x_i = 1$.
- Example of a minterm: $y = \neg x_2 \wedge \neg x_1 \wedge x_0$

Maxterm

Given a Boolean function $y = f(x_n, \ldots, x_1, x_0)$ and the literals x_i.

maxterm: $(x_n \vee \cdots \vee x_2 \vee x_1 \vee x_0)$

- The maxterm evaluates to 1 for exactly one particular configuration of the variable x_i and to 0 otherwise.
- More precisely: The maxterm evaluates to 0 if for all negated variables $x_i = 1$ and all not negated $x_i = 0$.
- Example of a minterm: $y = \neg x_2 \vee \neg x_1 \vee x_0$

Disjunctive Normal Form

- A **disjunctive normal form (DNF)** is an OR-operation ("or", ∨) of minterms
  - all configurations of minterms in which $y = f(x_n, \ldots, x_1, x_0) = 1$ must occur, and
  - all literals are connected by conjunctions ∧ ("and").

Example:

f(a,b,c)= (a ∧ b ∧ c) ∨ (¬a ∧ b ∧ c) ∨ (a ∧ ¬b ∧ c) ∨ … (¬a ∧ b ∧ ¬c)

Start from value table (disjunction)

- each line of the value table corresponds to a full conjunction
- terms with function value "0" do not contribute to the function result

a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

- form full conjunction for rows with function value "1
- none for f(a,b) = 0
- line 2: ¬a ∧ b
- line 3: a ∧ ¬b
- Result: f(a,b)= (¬a ∧ b) v (a ∧ ¬b)

Conjunctive Normal Form

- A conjunctive normal form (KNF) is an AND-connection of maxterms.
- All configurations of maxterms in which $y = f(x_n, \ldots, x_1, x_0) = 0$ must occur

a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

- form full conjunction for rows with function value "0
- none for f(a,b) = 1
- line 1: a v b
- line 4: ¬a v ¬b
- Result: f(a,b)= (a v b) ∧ (¬a v ¬b)

Example

So what does this mean for our example? Disjunction or conjunction?

bit a	bit b	bit c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$f(a,b,c) =$

Example

So what does this mean for our example? Disjunction or conjunction?

bit a	bit b	bit c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$f(a,b,c) = (a \land b \land c) \lor (a \land \neg b \land \neg c) \lor (\neg a \land \neg b \land c)$

Shannon's theorem

Definition

The negation of a logical formula composed only of the three basic operations ($\neg, \vee, \wedge$), variable names (a,b,..), and constants (0, 1) is obtained by replacing all variables and constants with their complement ($1 \rightarrow 0$, $0 \rightarrow 1$, $x \rightarrow \neg x$) and interchanging the operations $\wedge$ and $\vee$. It is important to note that one must completely parenthesize before substitution and interchange, and the parentheses are preserved.

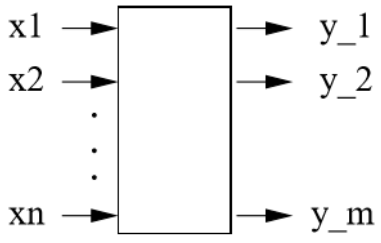
It is valid:

$$\boxed{\neg(f(a,b,..,\wedge,\vee,0,1)) = f(\neg a,\neg b,..,\vee,\wedge,1,0)}$$

Switching functions

- Previously: Boolean functions, i.e. only 1 output.
- Switching functions uniquely assign a binary sequence at the output to a binary sequence at the input:

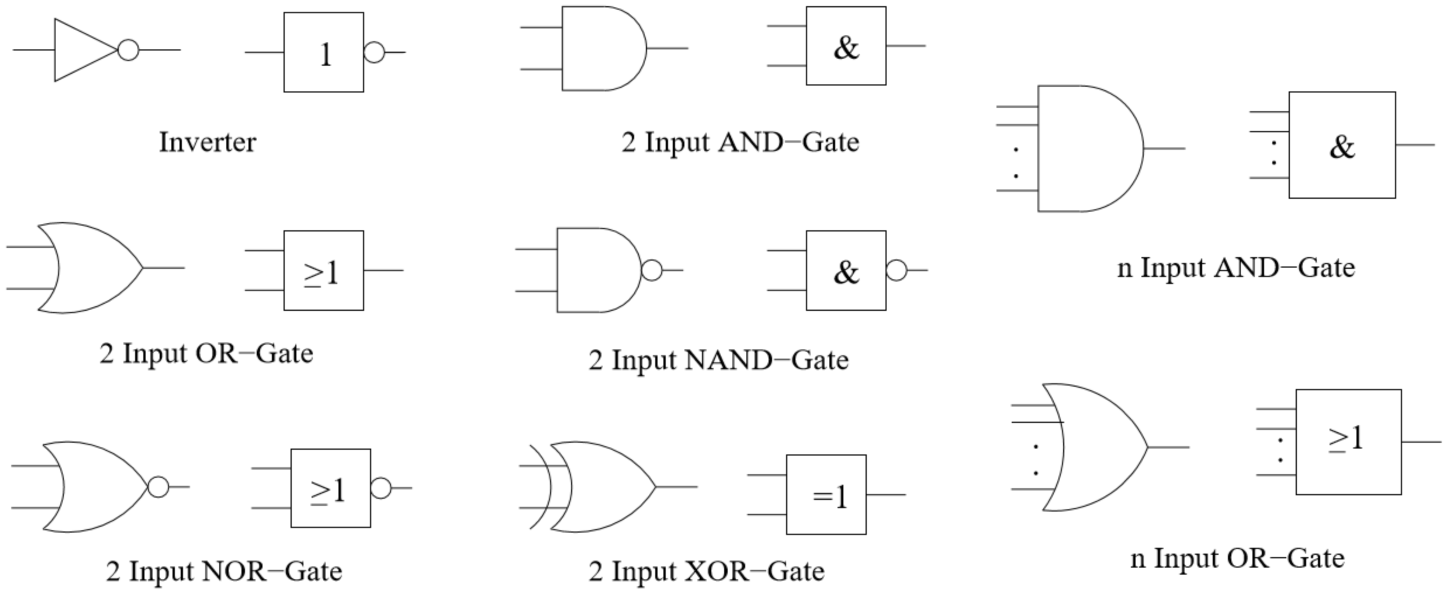
$$f_s : 0, 1^n \rightarrow 0, 1^m$$



Schaltfunktion
mit n Inputs
und m Outputs

Logical gates

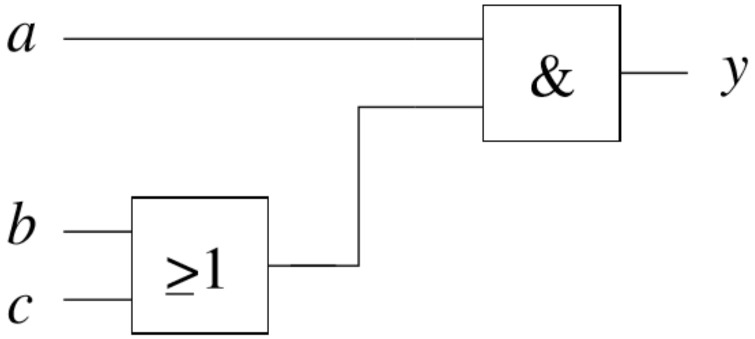
Switching networks and switching functions are built up from a few basic components or logical gates. Common logical gates are (American symbols and DIN 40700 [since 1976]):



Logical gates - example

Any switching functions and switching networks can be composed from the logical gates.

Example: y = a ∧ (b ∨ c)



What does this circuit do?

Logical gates - example

Truth table

bit a	bit b	bit c	b ∨ c	a ∧ (b ∨ c)
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Truth table → boolean expression

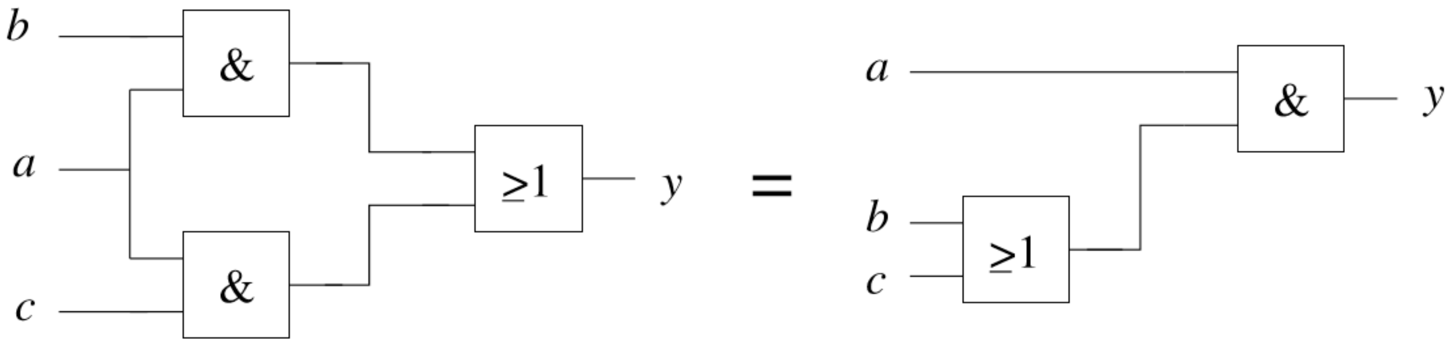
In practice, switching functions(networks) are obtained by:

- Representing desired behavior as a truth table
- Constructing the switching functions from the truth table (cf. Boolean normal form).
- Using the laws of Boolean algebra to transform and simplify the switching functions.

Simplification of switching networks

Example:

Distributive law: (a ∧ b) ∨ (a ∧ c) = a ∧ (b ∨ c)



Decoder

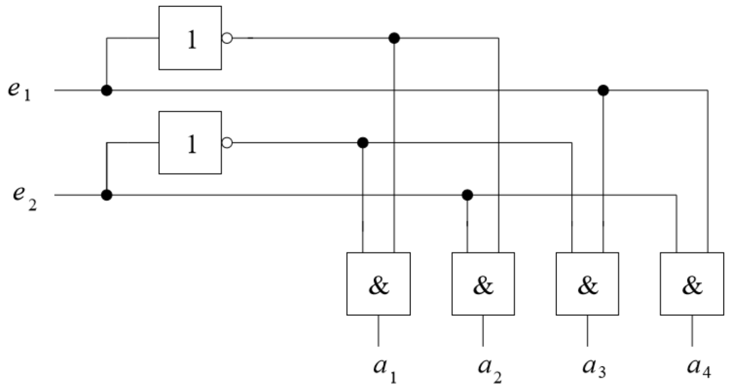
The *decoder* has n-inputs and

$$2^n = m$$

outputs, where for each input combination there is exactly one output which is **true**.

2-to-4 decoder:

e1	**e2**	**¬e1**	**¬e2**	**a1**	**a2**	**a3**	**a4**
0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0
1	0	0	1	0	0	1	0
1	1	0	0	0	0	0	1



Example: From BCD (*Binaray Coded Decimal*) to Decimal (00 -> 0, 01 -> 1, 10 -> 2, 11 -> 3)

Encoder

The inverse function to the decoder is the *encoder*. From 2^n inputs, exactly one of which is true, it produces an n-bit output.

4-to-2 Encoder:

i1	**i2**	**i3**	**i4**	**a1**	**a2**
1	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	0	1
0	0	0	1	1	1

What could such an encoder look like? - Exercise!

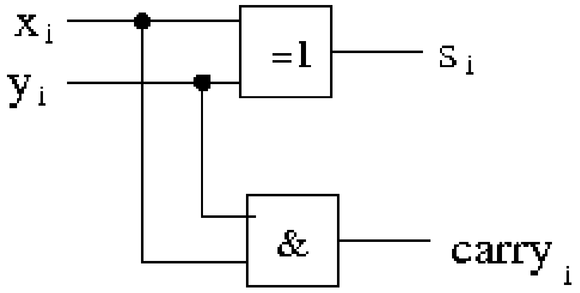
Half adder

A *half adder (HA)* is a switching network that adds two single-digit dual numbers without considering previous carries (carry) and returns the sum (S) and possibly a carry (C).

The resulting carry is generated separately.

Truth table of a half adder:

x_i	y_i	s_i	c_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

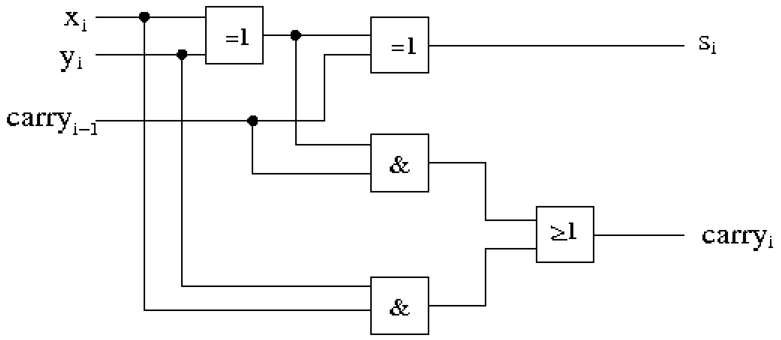


Full adder

The *full adder* differs from the half adder in that the previous carry is taken into account.

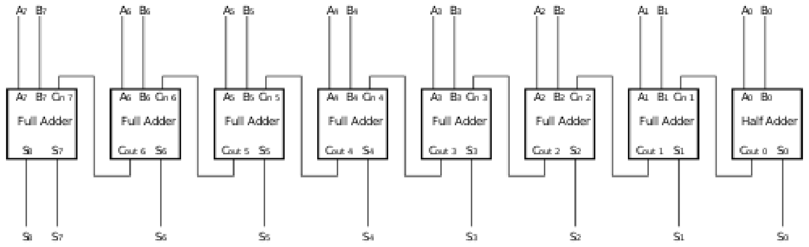
Truth table of a full adder:

x-i	y_i	c_i	s_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Full adder

- With the help of the *full adder* two n-digit dual numbers can be added in the computer.
  - For this one uses a chain of n full adders. - At the first addition (i=0) the incoming carry (c_-1) is set to 0.
- Other types of calculations can also be realized with the full adder.
  - The subtraction can be traced back to an addition with the two's complement.
  - With the help of additional circuits (shift registers) it is also possible to multiply and divide with adder circuits.



Summary

We looked at the following in detail:

- Boolean algebra
- Boolean functions
- Switching functions
- Logic gates
- Half and full adders