

# Temporal Locality

- if a memory location is accessed at a certain moment, it will very likely be accessed again in the near future
- examples
  - variables are used repeatedly in a program
  - program loops - instructions are repeated

# Spatial Locality

- if a memory location is accessed at a certain moment, the neighboring locations will very likely be accessed again in the near future
- examples
  - walking through arrays
  - the execution of instruction sequences - their codes are stored at consecutive memory addresses

# Physical Order and Logical Order

- instructions to be executed are stored in memory in the physical order
- they are read from memory and executed
  - the rule: in the same order they are memorized
  - exception: jumping over several instructions
- the result is the logical order of instructions
  - may differ from one run to another
  - an instruction can be executed 0, 1, 2, ... times

# II. Combinational Circuits and Boole Functions

# Analogical Signal vs. Digital Signal

- analogical signal - continuous
  - if it can have values  $a$  and  $b$ , then it can also have any value in  $[a,b]$
- digital signal - discrete
  - can only have a few distinct levels (values)
  - computer - 2-level digital signal (0 and 1)
  - there are also other computing systems apart from PCs

# Types of Circuits

- combinational circuits
  - the output values depend only on the input values
  - the same input values always yield the same output values
- sequential circuits
  - beside the inputs, the output values also depend on the state of the circuit
  - which evolves through time

# Truth tables

- how to describe how a combinational circuit works?
- apply each possible combination of input values
- and observe the output values for each such combination
- together, these relationships (input-output) make a truth table

# Circuits and Boole Functions

- each truth table has a corresponding Boole function
  - so each combinational circuit has a corresponding Boole function

inputs			outputs		
$I_1$	...	$I_n$	$O_1$	...	$O_m$
0	0...0	0	?	?...?	?
0	0...0	1	?	?...?	?
...	...	...	...	...	...
1	1...1	1	?	?...?	?



# II.1. Boole Functions

# Algebraic Structure

- a non-void set  $B$ , containing at least two elements:  $a, b, a \neq b$
- the set of binary operations  $\{ +, \cdot \}$
- one unary operation  $\{ - \}$
- closure:  
 $a + b \in B$   
 $a \cdot b \in B$   
 $\bar{a} \in B$

# Boole Functions

- $B = \{0,1\}$
- $f : B^n \rightarrow B^m$ 
  - function:  $n$  variables,  $m$  values
  - circuit:  $n$  inputs,  $m$  outputs
- there are  $(2^m)^{2^n}$  such distinct functions
  - $n = 1, m = 1$ : 4 unary functions of one value
  - $n = 2, m = 1$ : 16 2-variable Boole functions of one value

# Truth Tables

$a$	$f_0(a)$	$f_1(a)$	$f_2(a)$	$f_3(a)$
0	0	0	1	1
1	0	1	0	1
	$= 0$	$= a$	$= \bar{a}$	$= 1$

$a$	$b$	$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$	$F_{11}$	$F_{12}$	$F_{13}$	$F_{14}$	$F_{15}$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

# Axioms and Theorems in Boole Algebra (1)

identity	$X + 0 = X$	$X \cdot 1 = X$
constants	$X + 1 = 1$	$X \cdot 0 = 0$
idempotence	$X + X = X$	$X \cdot X = X$
involution	$\overline{\overline{X}} = X$	
complementarity	$X + \overline{X} = 1$	$X \cdot \overline{X} = 0$
commutativity	$X + Y = Y + X$	$X \cdot Y = Y \cdot X$
associativity	$(X + Y) + Z = X + (Y + Z)$	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
distributivity	$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$

# Axioms and Theorems in Boole Algebra (2)

unification	$X \cdot Y + X \cdot \bar{Y} = X$	$(X + Y) \cdot (X + \bar{Y}) = X$
absorption	$X + X \cdot Y = X$ $(X + \bar{Y}) \cdot Y = X \cdot Y$	$X \cdot (X + Y) = X$ $(X \cdot \bar{Y}) + Y = X + Y$
De Morgan laws	$\overline{X+Y+\dots} = \bar{X} \cdot \bar{Y} \cdot \dots$	$\overline{X \cdot Y \cdot \dots} = \bar{X} + \bar{Y} + \dots$
generalization (duality)	$\overline{f(X_1, \dots, X_n, 0, 1, +, \cdot)} = f(\bar{X}_1, \dots, \bar{X}_n, 1, 0, \cdot, +)$	

# The Computer - Elementary Operations

- in today's computers, elementary operations are the operations of Boole logic
  - which simulate (among others) the elementary arithmetic operations in base 2
- a combinational circuit actually implements a Boole function
  - how do we get the expression of the Boole function from the truth table?

# Normal Forms

- disjunctive normal form (DNF)
  - for each row that yields value 1 on output – conjunction term ( $\cdot$ )
    - contains each variable: negated if the variable is 0 on that row, not negated if the variable is 1
  - these terms - connected through disjunction (+)
- conjunctive normal form (CNF): dual
- example:  $F_9(x,y) = \bar{x} \cdot \bar{y} + x \cdot y = (x + \bar{y}) \cdot (\bar{x} + y)$



## II.2. Logic Diagrams

# The Alphabet of Logic Diagrams

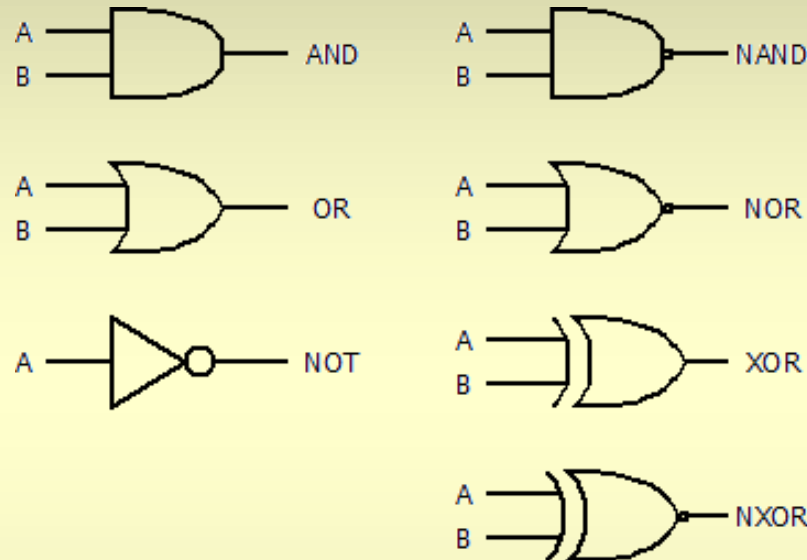
- logic gates are the implementations of some Boole functions
- so the behavior of each gate can be described by a truth table
  - corresponding to the Boole function associated with the gate
- elementary gates: AND, OR, NOT
- other gates: NAND, NOR, XOR, NXOR

# The Alphabet of Logic Diagrams

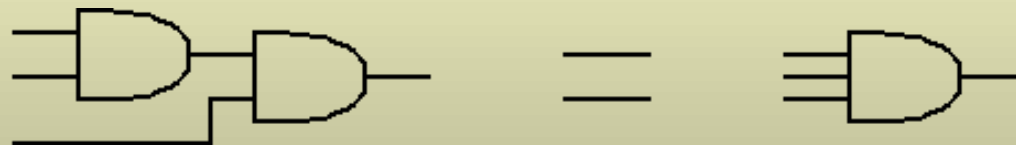
A	NOT
0	1
1	0

A	B	AND	OR	NAND	NOR	XOR	NXOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

# Gate Symbols



- associative binary operations can be extended operations with a finite number of operands



# Minimal Set of Generators

- set of generators - set of gate types which can implement any Boole function
  - minimal set of generators - set of generators with the minimal number of gate types
- it is possible with 3 (NOT, AND, OR)
  - normal forms (disjunctive, conjunctive)
  - also possible with (NOT + AND, NOT + OR)
  - minimal - 1 (NAND, NOR)

# Homework

- prove that the following sets of gate types are sets of generators
  - NOT + AND
  - NOT + OR
  - NAND
  - NOR

# **II.3. Circuit Implementation Through Boole Functions**

# Defining Boole Functions

- can be defined in several ways
  - truth table
  - expressions - variables and logical operations
  - graphic representation
  - sigma-notation ( $\Sigma$ )
- in the end, we need to have a Boole expression
  - which allows a gate implementation



## $\Sigma$ -notation(1)

- example - "majority of k inputs"
  - function value: 1 if most input variables have value 1, 0 otherwise
    - for 3 variables:  $f(x_1, x_2, x_3) = \Sigma(3, 5, 6, 7)$
- $\Sigma$ -notation corresponds to the disjunctive normal form
  - each number in the brackets is a conjunction term
  - $\Sigma$  denotes the disjunction of terms

## $\Sigma$ -notation(2)

- how many variables are necessary?
  - the lowest power of 2 that is equal to or greater than the highest number in the brackets
    - for our example:  $2^2 < 7 < 2^3 \rightarrow n = 3$
- the term corresponding to a number
  - all variables, connected through conjunction
  - each variable is negated if assigned to a bit with value 0; not negated for 1
    - example:  $3_{(10)} = 011_{(2)} \rightarrow \overline{x_1} \cdot x_2 \cdot x_3$

## Minimization (1)

- the disjunctive normal form of the function majority of 3

$$f(A,B,C)=\bar{A}\cdot B\cdot C+A\cdot\bar{B}\cdot C+A\cdot B\cdot\bar{C}+A\cdot B\cdot C$$

- large number of elementary gates
- a simpler equivalent expression (**the same Boole function**) would make the circuit
  - faster
  - cheaper
  - more reliable

## Minimization (2)

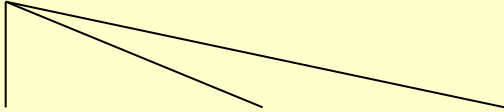
- how to get a simpler expression from the disjunctive normal form ?
  - equivalent rewriting
    - use the laws and axioms of Boole algebra
  - perfect induction
  - Veitch-Karnaugh method
  - Quine-McCluskey method
  - hybridization (combine the above methods)

# Minimization - Algebraic Rewriting

- same example

$$f = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

(idempotence)


$$= \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C$$

(unification)


$$= B \cdot C + A \cdot C + A \cdot B$$

- difficult for complex expressions

# Homework

- find the disjunctive normal form and study minimization through algebraic rewriting for the function "odd"
  - the function value is: 1 for an odd number of inputs with value 1; 0 otherwise

## **II.4. Minimization of Boole Functions Using Karnaugh Diagrams**

# The Veitch-Karnaugh Method

- provides a visual way of putting together the conjunction terms in DNF for which unification can be applied
- unification is possible if two terms differ on one variable only
  - which is negated for one term and not negated for the other one
- such terms become neighbors in a Karnaugh diagram



# Structure of a Karnaugh Diagram

2-dimensional table

- variable names
  - for rows and columns, respectively
- label area
  - label - bit string of length  $n$
  - each bit corresponds to a variable (input)
  - all possible input combinations are present
- function value (output) area

# Examples of Diagrams

2 variables

A \ B	0	1
	0	1
0	1	1
1		

3 variables

A \ BC	00	01	11	10
	0	1	1	1
0		1		1
1		1	1	

4 variables

AB \ CD	00	01	11	10
	00	01	11	10
00	1	1		
01				
11		1		1
10	1			1

# Grey Code

- labels are not written in increasing order, but in Grey order
- any two consecutive labels, including the first and the last, differ by one bit
  - 2 bits: 00, 01, 11, 10
  - 3 bits: 000, 001, 011, 010, 110, 111, 101, 100
  - 4 bits: 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000

# Diagram Adjacencies (1)

- two positions are adjacent if their corresponding labels differ by a single bit
  - Grey code: adjacency  $\rightarrow$  neighborhood
- for an  $n$ -variable function, a location has  $n$  adjacent locations
  - $n < 5$ : adjacent locations are found visually (up, down, left, right)
  - $n \geq 5$ : there are also other adjacencies, not directly visible

## Diagram Adjacencies (2)

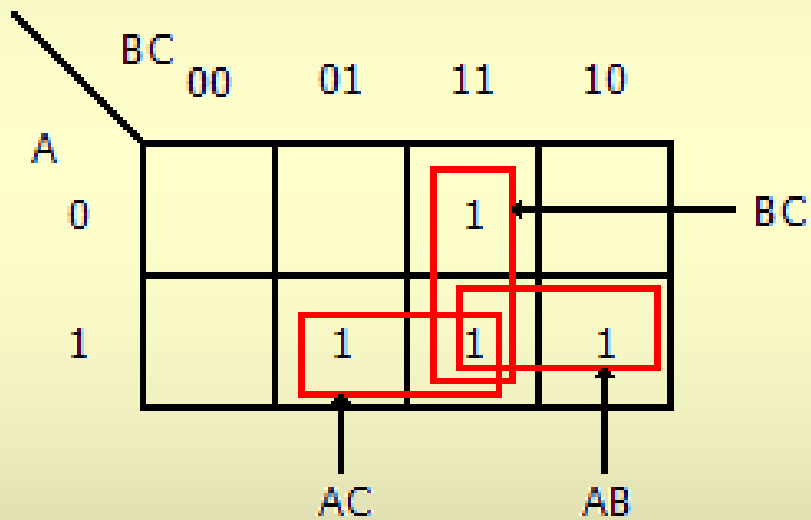
- there may be more than 2 adjacent locations
  - extend the unification to more than 2 variables
- in Karnaugh diagrams, they correspond to blocks with  $2^k$  locations
  - power of 2 both for rows and for columns
    - including power 0
    - rectangle-shaped
  - for each location, the block must contain precisely  $k$  adjacent locations

# Karnaugh Minimization

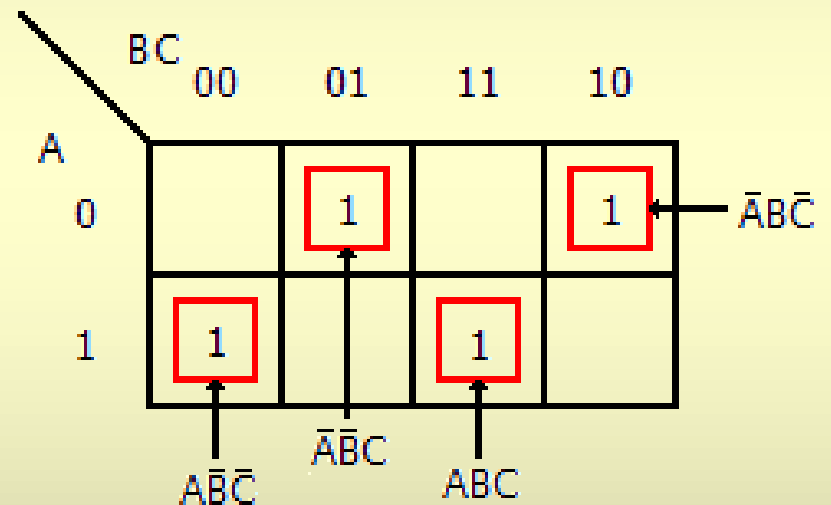
- look for blocks containing only 1s
  - corresponding to an adjacency (see before)
  - the blocks - as large and as few as possible
- for each block with  $2^k$  locations (all 1s)
  - we have a conjunction term of  $n-k$  variables
  - contains the variables whose values are constant for all locations in the block
    - 0: variable is negated; 1: variable is not negated
  - all these terms are connected by disjunction

# Examples

majority of 3

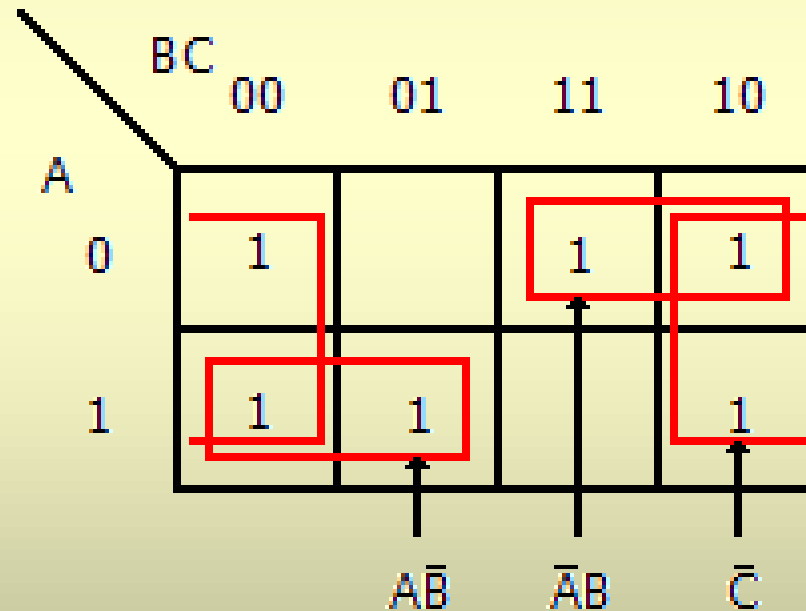


odd



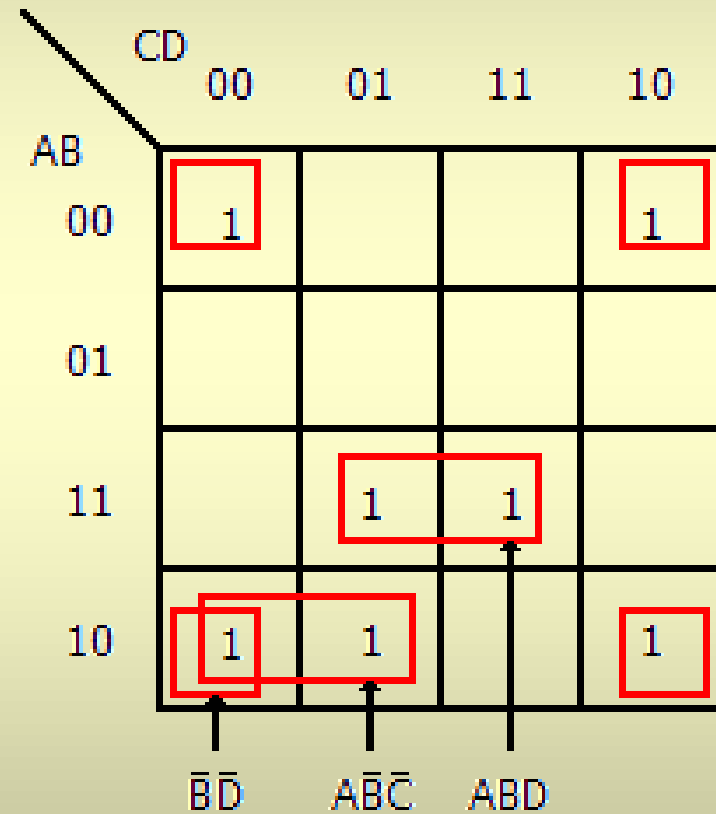
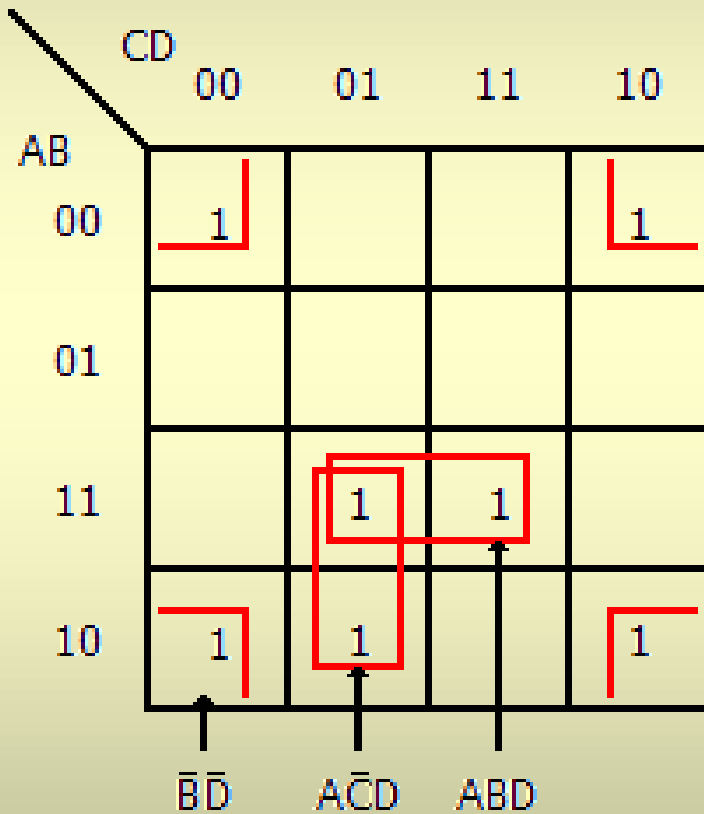
# Adjacency of Extreme Rows/Columns

$$f = \Sigma(0,2,3,4,5,6)$$





# Expression Depends on Groups



# Avoiding Redundancies

non-minimal simplification

		CD			
		00	01	11	10
AB	00			1	
	01	1	1	1	
	11		1	1	1
	10		1		

minimal simplification

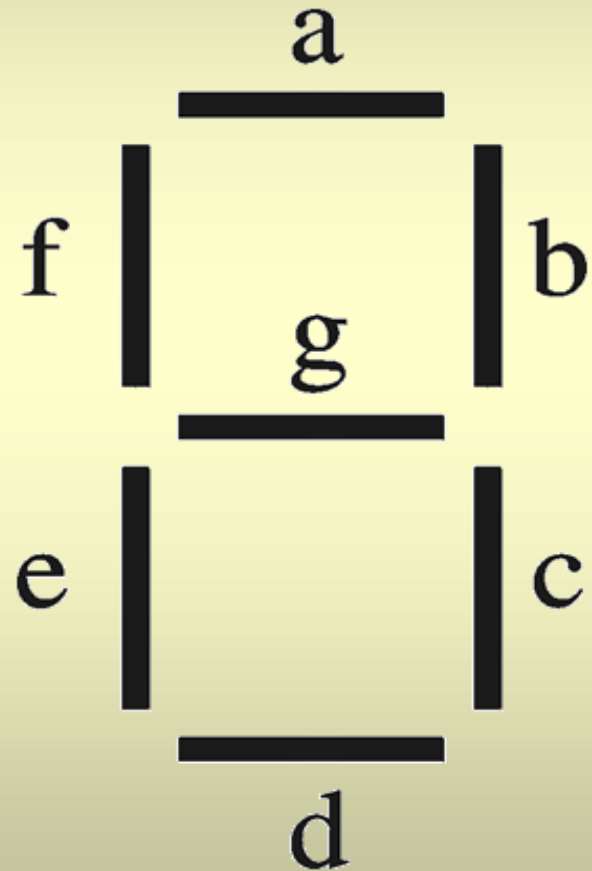
		CD			
		00	01	11	10
AB	00			1	
	01	1	1	1	
	11		1	1	1
	10		1		

# Impossible Value Combinations

- certain value combinations will never show up on input
  - according to the behavior we seek
  - but the diagram must be filled for all value combinations of the variables
- in the locations corresponding to these combinations we can write either 0 or 1
  - in order to get a simpler expression

# Example - Displaying Decimal Digits

- 7 segment display
- selecting the segments for each digit
  - 0 - switched off
  - 1 - switched on
- input (command) - 4 variables
  - a decimal digit can be written on 4 bits



## Segment $d$ - Truth Table

No	A	B	C	D	$d$
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0

No	A	B	C	D	$d$
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	*
11	1	0	1	1	*
12	1	1	0	0	*
13	1	1	0	1	*
14	1	1	1	0	*
15	1	1	1	1	*

# Simpler Expressions

"safe" simplification

		CD			
		00	01	11	10
AB	00	1		1	1
	01		1		1
	11				
	10	1	1		

exploiting impossible combinations

		CD			
		00	01	11	10
AB	00	1		1	1
	01		1		1
	11	*	*	*	*
	10	1	1	*	*

# Homework: 2-bit Comparison

- 4 variables: A, B, C, D
- make 2 numbers
  - $N_1 = AB$
  - $N_2 = CD$
- 3 outputs - correspond to the truth values
  - $LT = (N_1 < N_2)$
  - $EQ = (N_1 = N_2)$
  - $GT = (N_1 > N_2)$

A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

# Homework: 2-bit Multiplication

- 4 variables: A, B, C, D
- make 2 numbers
  - $N_1 = AB$
  - $N_2 = CD$
- 4 outputs - the product  
 $N_1 \cdot N_2$

A	B	C	D	P8	P4	P2	P1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1



# Homework: BCD Increment by 1

- 4 variables
  - make a BCD numbers
  - between 0 and 9
- 4 outputs - the input number, incremented
  - the result is also a BCD number

I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	*	*	*	*
1	0	1	1	*	*	*	*
1	1	0	0	*	*	*	*
1	1	0	1	*	*	*	*
1	1	1	0	*	*	*	*
1	1	1	1	*	*	*	*