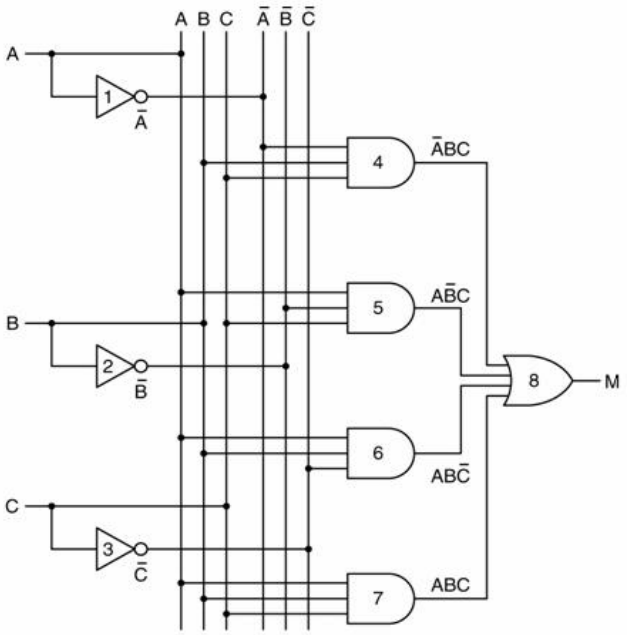
# Tietokoneen rakenne

# Digital logic



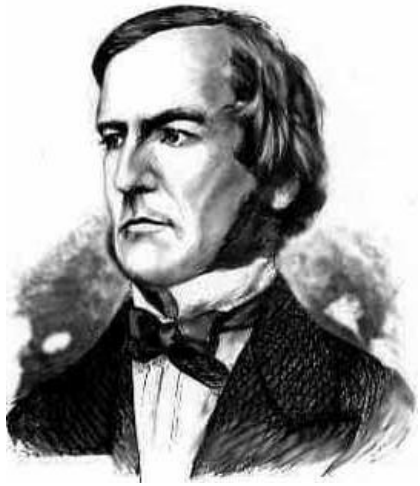

Stallings: Appendix B

- Boolean Algebra
- Combinational Circuits
- Simplification
- Sequential Circuits

# Boolean Algebra

# Boolean Algebra

- George Boole
  - ideas 1854
- Claude Shannon
  - apply to circuit design, 1938
  - “father of information theory”



**Topics:**

- Describe digital circuitry function
  - programming language?
- Optimise given circuitry
  - use algebra (Boolean algebra) to manipulate (Boolean) expressions into simpler expressions

(piirisuunnittelu)

# Boolean Algebra

- Variables: A, B, C
- Values: TRUE (1), FALSE (0)
- Basic logical operations:
  - binary: AND ( · ) $A \bullet B = AB$ ja, tulo,
    OR ( + ) $B + C$ tai, yhteenlasku,
  - unary: NOT ( ¯ ) $\overline{A}$ ei negaatio
- Composite operations, equations
  - precedence: NOT, AND, OR
  - parenthesis

$$D = A + \overline{B} \bullet C = A + ((\overline{B})C)$$

# Boolean Algebra

- Other operations
  - NAND
  - NOR

$$A \text{ NAND } B = \text{NOT } (A \text{ AND } B) = \overline{AB}$$

$$A \text{ NOR } B = \text{NOT } (A \text{ OR } B) = \overline{A + B}$$

- Truth tables
  - What is the result of the operation?

**Boolean Operators**

| P | Q | NOT P | P AND Q | P OR Q | P XOR Q | P NAND Q | P NOR Q |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

(Sta06 Table B.1)

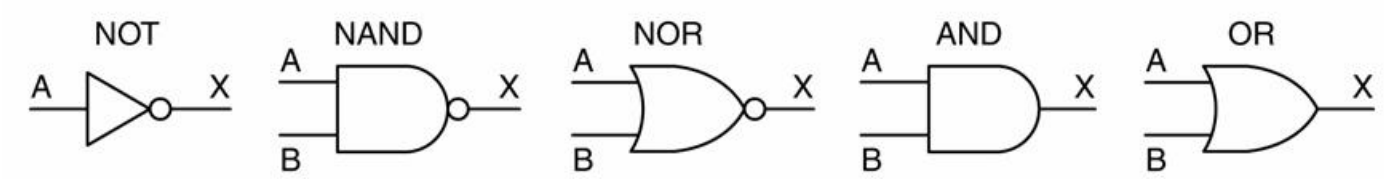
# Postulates and Identities

- How can I manipulate expressions?
  - Simple set of rules?

| | | |
|---|---|---|
| **Basic Postulates** | | |
| $A \cdot B = B \cdot A$ | $A + B = B + A$ | Commutative Laws |
| $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ | Distributive Laws |
| $1 \cdot A = A$ | $0 + A = A$ | Identity Elements |
| $A \cdot \overline{A} = 0$ | $A + \overline{A} = 1$ | Inverse Elements |
| **Other Identities** | | |
| $0 \cdot A = 0$ | $1 + A = 1$ | |
| $A \cdot A = A$ | $A + A = A$ | |
| $A \cdot (B \cdot C) = (A \cdot B) \cdot C$ | $A + (B + C) = (A + B) + C$ | Associative Laws |
| $\overline{A \ B} = \overline{A} + \overline{B}$ | $\overline{A + B} = \overline{A} \ \overline{B}$ | DeMorgan's Theorem |

(Sta06 Table B.2)

# Gates (veräjät / portit)




- Implement basic Boolean algebra operations
- Fundamental building blocks
  - 1 or 3 inputs, 1 output
- Combine to build more complex circuits
  - memory, adder, multiplier, ...

yhteenlaskupiiri, kertolaskupiiri

- Gate delay
  - change inputs, after gate delay new output available
  - 1 ns? 10 ns? 0.1 ns?

http://tech-www.informatik.uni-hamburg.de/applets/cmos/cmosdemo.html (extra material)

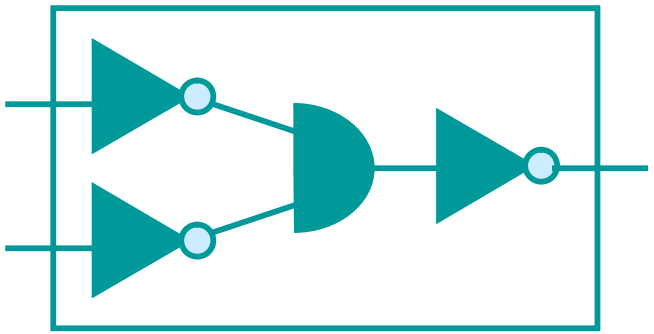
Sta06 Fig B.1

# Functionally Complete Set

funktionaalisesti täydellinen joukko

- **Can build all basic gates (AND, OR, NOT) from a smaller set of gates**
  - With AND, NOT
  - With OR, NOT
  - With NAND alone
  - With NOR alone

$$A + B = \overline{\overline{A} \bullet \overline{B}}$$



OR

Sta06 Fig B.2, B.3

# Combinational Circuits

yhdistelmäpiirit

- **Interconnected set of gates**
  - m inputs, n outputs
  - change inputs, wait for gate delays, new outputs
- **Each output**
  - depends on combination of input signals
  - can be expressed as Boolean function of inputs
- **Function can be described in three ways**
  - with Boolean equations (one equation for each output)
  - with truth table
  - with graphical symbols for gates and wires

# Describing the Circuit

- **Boolean equations**

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

- **Truth table**

<---------- inputs ----------> <- output ->

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

(Sta06 Table B.3)

- **Graphical symbols**

Sta06 Fig B.4

# Tietokoneen rakenne

## Simplification

Piirin yksinkertaistaminen

# Simplify Presentation (and Implementation)

- **Boolean equations**
  - Sum of products form (SOP)

Sta06 Table B.3

Sta06 Fig B.4

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

Boolean algebra

  - Product of sums form (POS)

$$F = (A + B + C) \bullet (A + B + \overline{C}) \bullet (\overline{A} + B + C) \bullet (\overline{A} + B + \overline{C}) \bullet (\overline{A} + \overline{B} + \overline{C})$$

Sta06 Fig B.5

- Which presentation is better?
  - Fewer gates? Smaller area on chip?
  - Smaller circuit delay? Faster?

# Algebraic Simplification

- Circuits become too large to handle?
- Use basic identities to simplify Boolean expressions

$$F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C}$$
$$= \overline{A}B + B\overline{C} = B(\overline{A} + \overline{C})$$

Sta06 Fig B.4

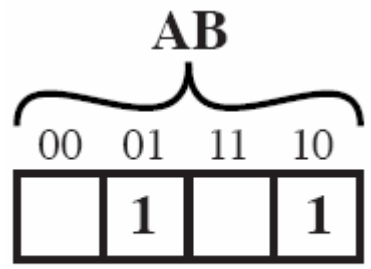
Sta06 Fig B.6

- May be difficult to do
- How to do it automatically?
- Build a program to do it “best”?

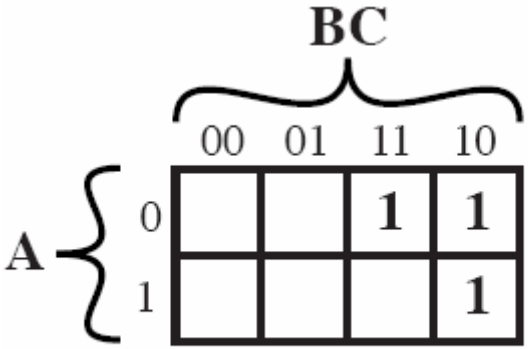
$$f = \overline{a}\overline{b}cd + \overline{a}bcd + ab\overline{c}\overline{d} + ab\overline{c}d$$
$$+ abcd + abc\overline{d} + a\overline{b}cd + a\overline{b}c\overline{d}$$

# Karnaugh Map

- Represent Boolean function (i.e., circuit) truth table in another way
  - Use canonical form: each term has each variable once
  - Use SOP presentation
- Karnaugh map squares
  - Each square is one product (input value combination)
  - Value is one (1) iff the product is present
    o/w value is “empty”




(a) $F = A\bar{B} + \bar{A}B$

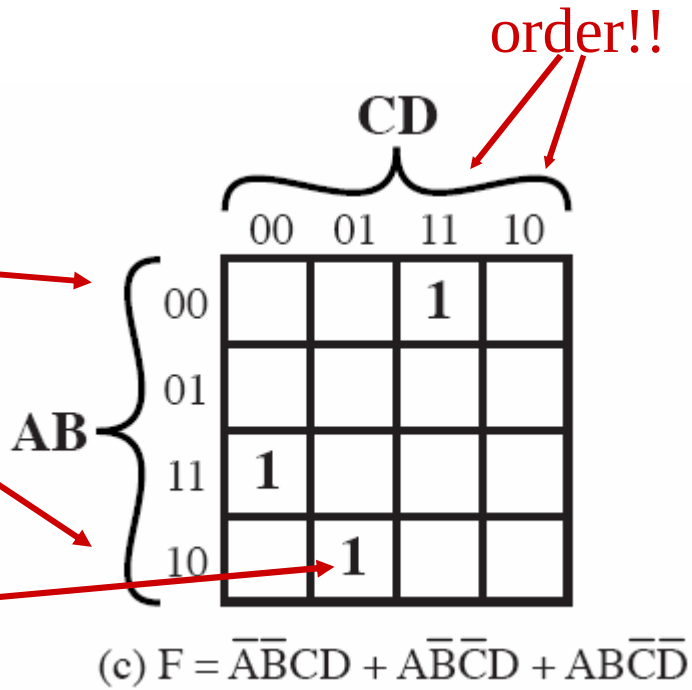



(b) $F = \bar{A}BC + \bar{A}BC + AB\bar{C}$

(Sta06 Fig B.7)

# Karnaugh Map

- Adjacent squares differ only in one input value (wrap around)
- Square for input combination $A\bar{B}\bar{C}D = 1001$

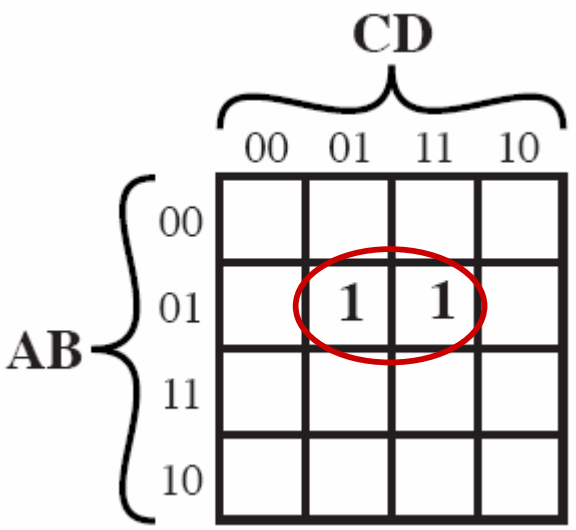



(Sta06 Fig B.7)

# Karnaugh Map Simplification

- If adjacent squares have value 1,
  input values differ
  only in one variable

- Value of that variable is irrelevant
  (when all other input variables
  are fixed for those squares)




- Can ignore that variable for those expressions
  - ... + $\bar{A}B\bar{C}D + \bar{A}BCD$ + ... ignore C ... + $\bar{A}BD$ + ...

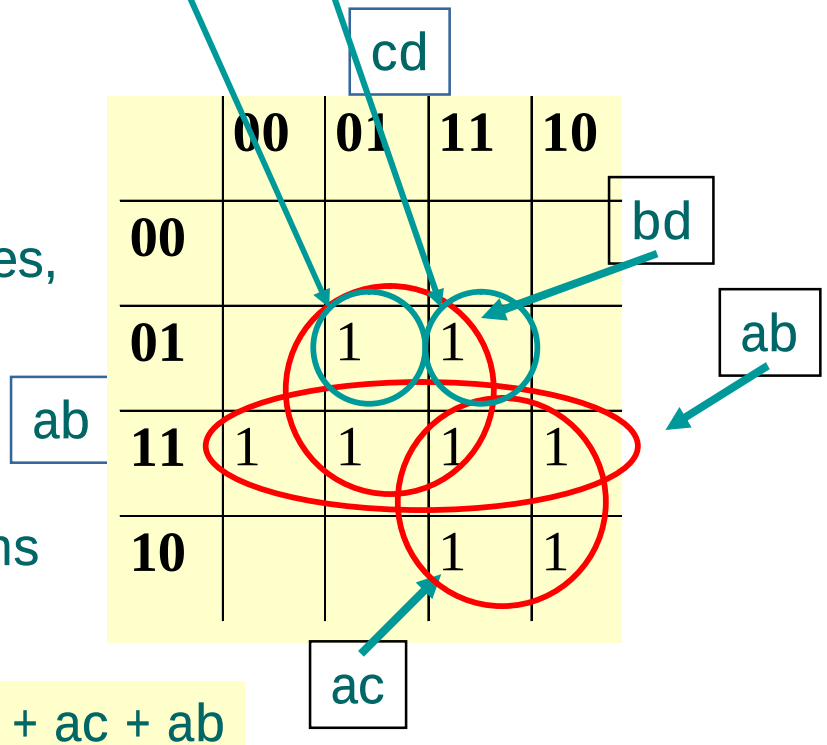
# Using Karnaugh Maps to Minimize Boolean Functions (8)

Original function

$$f = \bar{a}\bar{b}cd + \bar{a}bcd + ab\bar{c}\bar{d} + ab\bar{c}d + abcd + abc\bar{d} + a\bar{b}cd + a\bar{b}c\bar{d}$$

Canonical form (already OK)

Karnaugh Map




Find smallest number of circles, each with largest number ($2^i$) of 1's

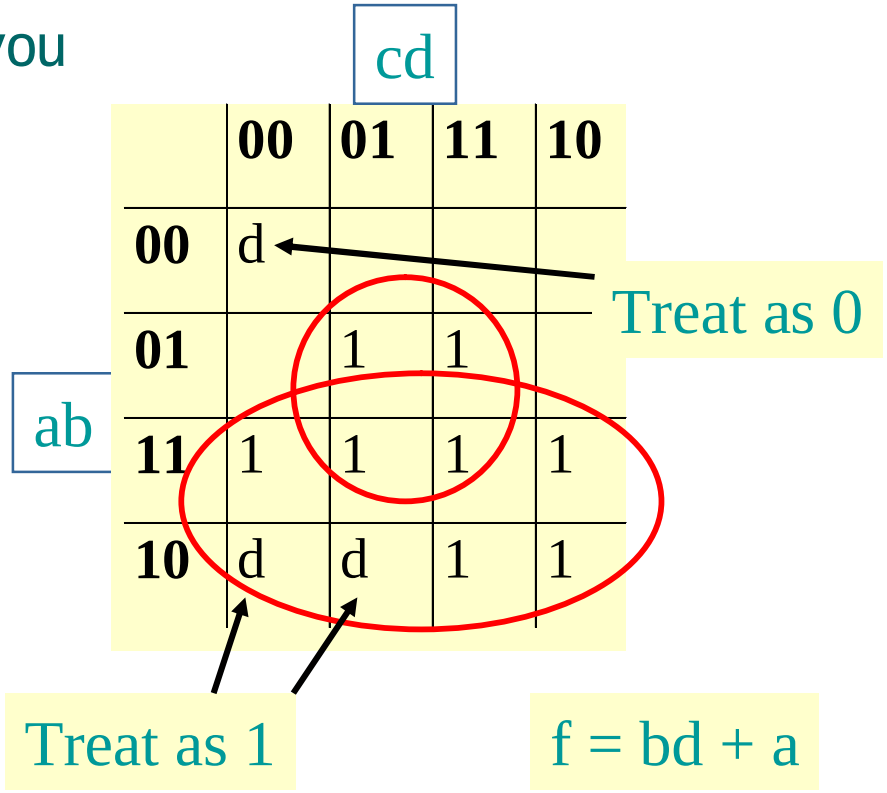
- can wrap-around

Select parameter combinations corresponding to the circles

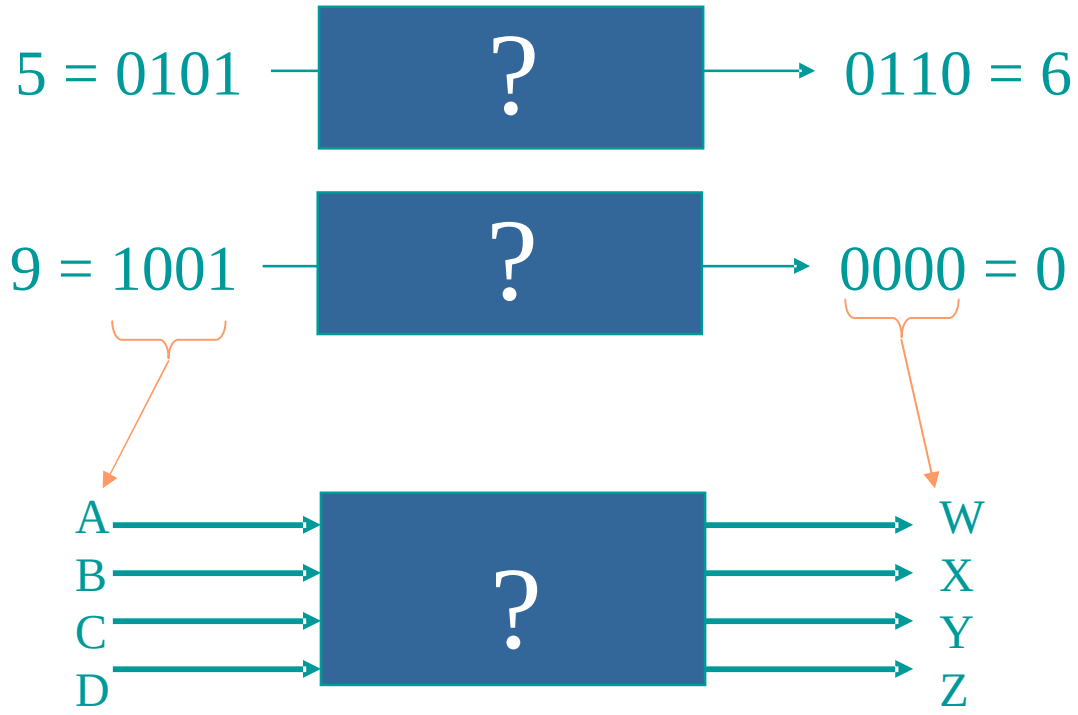
Get reduced function f = bd + ac + ab

# Impossible Input Variable Combinations

- **What if some input combinations can never occur?**
  - Mark them ”don’t care”, “d”
  - treat them as 0 or 1, whichever is best for you
  - more room to optimize



# Example: Circuit to add 1 (mod 10) to 4-bit BCD decimal number (3)




- Truth table?
- Karnaugh maps for W, X, Y and Z?

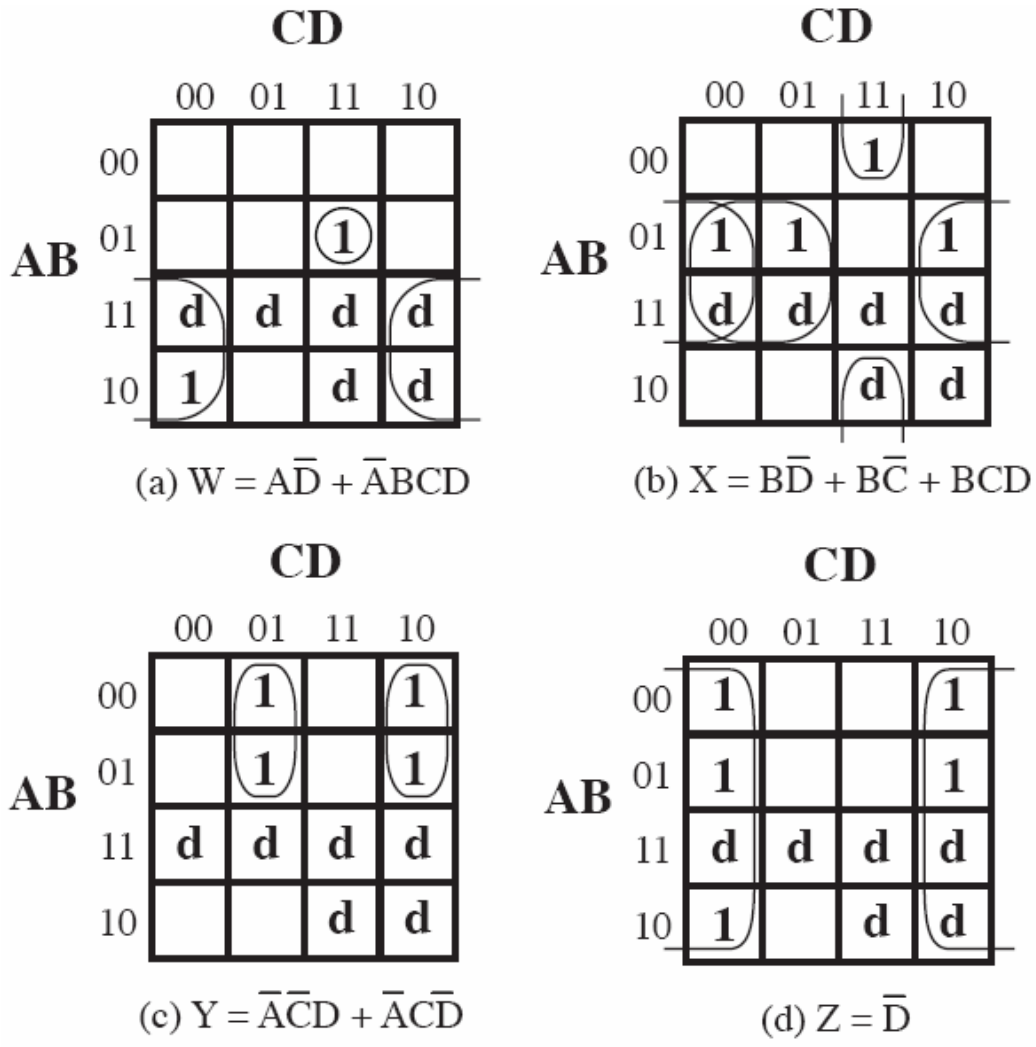
# Example cont.: Truth Table

**Truth Table for the One-Digit Packed Decimal Incrementer**

| | Input | | | | | Output | | | |
|---|---|---|---|---|---|---|---|---|---|
| Number | A | B | C | D | Number | W | X | Y | Z |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 4 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 5 | 0 | 1 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 6 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 7 | 0 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 8 | 1 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 9 | 1 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Don't care condition | 1 | 0 | 1 | 0 | | d | d | d | d |
| | 1 | 0 | 1 | 1 | | d | d | d | d |
| | 1 | 1 | 0 | 0 | | d | d | d | d |
| | 1 | 1 | 0 | 1 | | d | d | d | d |
| | 1 | 1 | 1 | 0 | | d | d | d | d |
| | 1 | 1 | 1 | 1 | | d | d | d | d |

(Sta06 Table B.4)

# Example cont: Karnaugh Map




(Sta06 Fig B.10)

# Other Methods to simplify Boolean expressions

- **Why?**
  - Karnaugh maps become complex with 6 input variables
- **Quine-McKluskey method**
  - Tabular method
  - Automatically suitable for programming
- **Luque Method**
  - Based on dividing circle in different ways
  - Can be fractally expanded to infinitely many variables
- **Interesting, but not part of this course**
- **Details skipped**

# Basic

# Combinatorial Circuits

Building blocks for more complex circuits

- Multiplexer
- Encoders/decoder
- Read-Only-Memory
- Adder

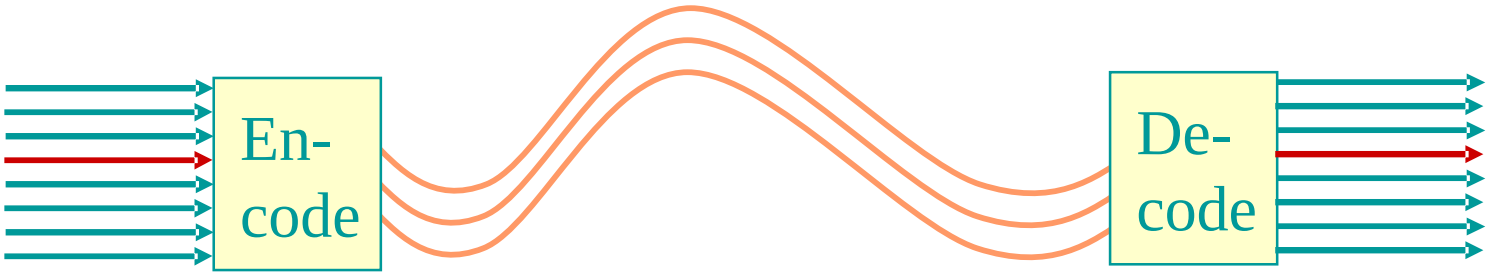
# Multiplexers

- Select one of many possible inputs to output
  - black box Sta06 Fig B.12
  - truth table Sta06 Table B.7
  - implementation Sta06 Fig B.13
- Each input/output "line" can be many parallel lines
  - select one of three 16 bit values
    - $C_{0..15}$ , $IR_{0..15}$ , $ALU_{0..15}$
  - simple extension to one line selection Sta06 Fig B.14
    - lots of wires, plenty of gates …

# Encoders/Decoders

- Only one of many Encoder input or Decoder output lines can be 1
- Encode that line number as output
  - hopefully less pins (wires) needed this way
  - optimise for space, not for time
  - Example:
    - encode 8 input wires with 3 output pins
    - route 3 wires around the board
    - decode 3 wires back to 8 wires at target

Sta06 Fig B.15



# Read-Only-Memory (ROM) (5)

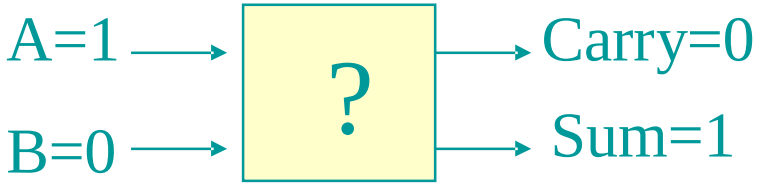
- **Given input values, get output value**
  - Like multiplexer, but with fixed data
- **Consider input as address, output as contents of memory location**
- **Example**
  - Truth tables for a ROM Sta06 Table B.8
    - 64 bit ROM
    - 16 words, each 4 bits wide
  - Implementation with decoder & or gates Sta06 Fig B.20

Mem (7) = 4

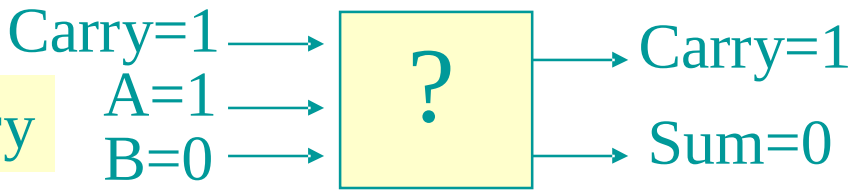
Mem (11) = 14

# Adders

1-bit adder




1-bit adder with carry




Implementation

Sta06 Table B.9, Fig B.22

Build a 4-bit adder from four 1-bit adders

Sta06 Fig B.21

# Sequential Circuits

sarjalliset piirit

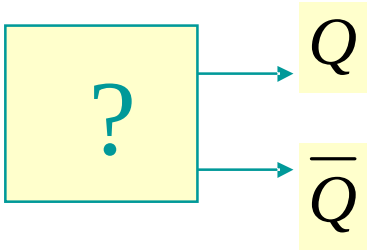
- Flip-Flop
- S-R Latch
- Registers
- Counters

# Sequential Circuit (sarjallinen piiri)

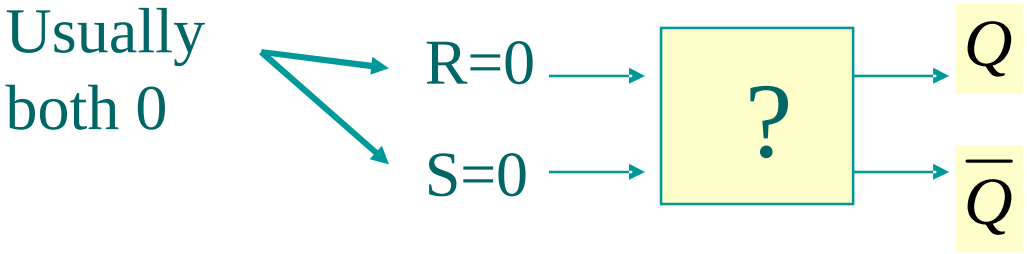
- Circuit has (modifiable) internal state
  - remembers its previous state
- Output of circuit depends (also) on internal state
  - not only from current inputs
  - output = $f_o$(input, state)
  - new state = $f_s$(input, state)
- Circuits needed for
  - processor control
  - registers
  - memory

# Flip-Flop (kiikku)

- 2 states for Q (0 or 1, true or false)
- 2 outputs
  - complement values
  - both always available on different pins
- Need to be able to change the state (Q)

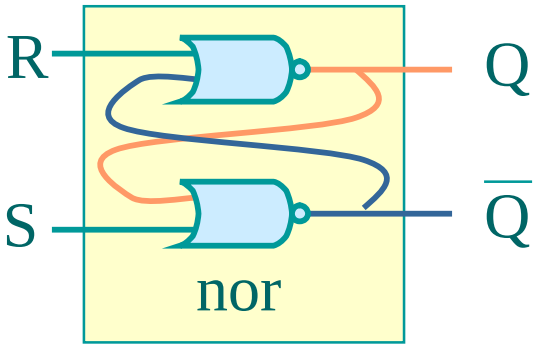


# S-R Flip-Flop or S-R Latch




S = “SET” = “Write 1” = “set S=1 for a short time”
R = “RESET” = “Write 0” = “set R=1 for a short time”

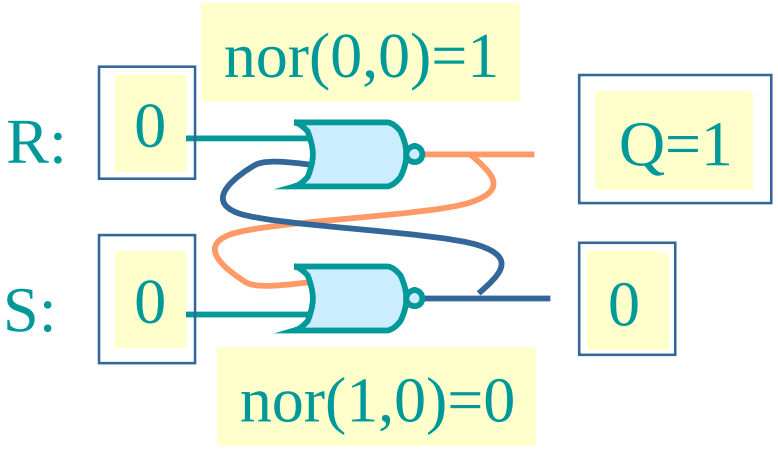
nor (0, 0) = 1
nor (0, 1) = 0
nor (1, 0) = 0
nor (1, 1) = 0



# S-R Latch Stable States (4)

- 1 bit memory (value = value of Q)
- bi-stable, when R=S=0
  - Q=0?
  - Q=1?

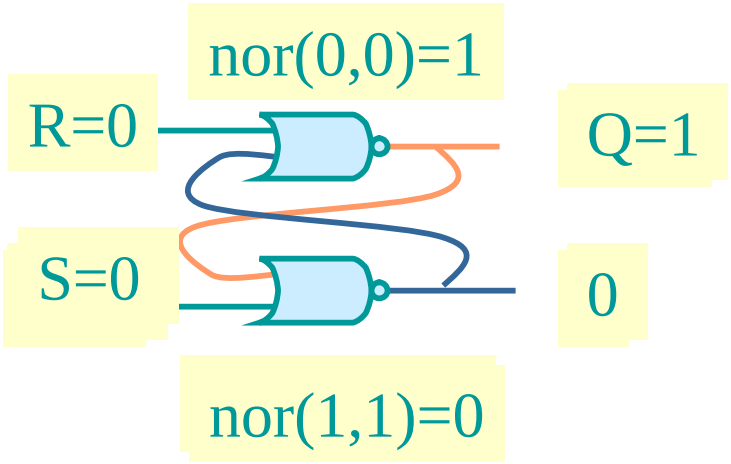
nor (0, 0) = 1
nor (0, 1) = 0
nor (1, 0) = 0
nor (1, 1) = 0




- output = $f_o$(input, state),
- state = $f_s$(input, state)

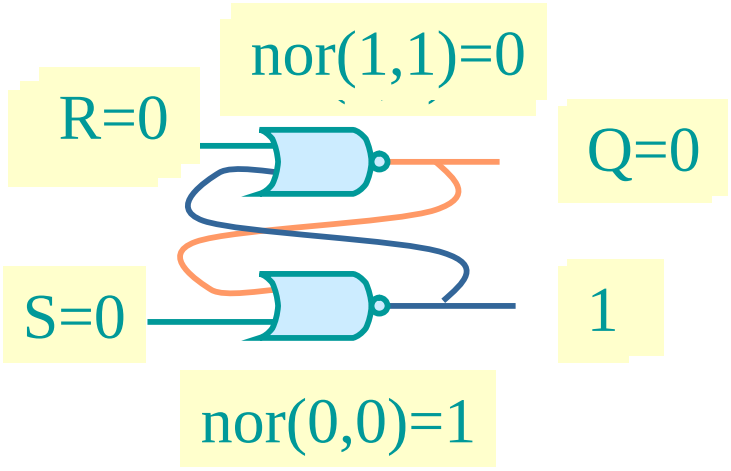
# S-R Latch Set (=1) and Reset (=0) (17)

Write 1: S= $0 \rightarrow 1 \rightarrow 0$




Write 0: R= $0 \rightarrow 1 \rightarrow 0$

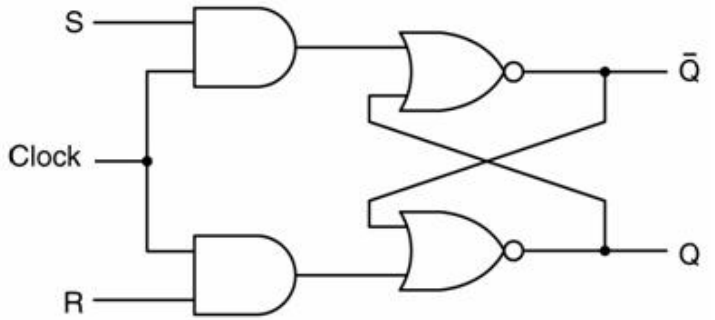
nor (0, 0) = 1
nor (0, 1) = 0
nor (1, 0) = 0
nor (1, 1) = 0



# Clocked Flip-Flops

- **State change can happen only when clock is 1**
  - more control on state changes

- **Clocked S-R Flip-Flop** Sta06 Fig B.26 Sta06 Fig B.29
- **D Flip-Flop** Sta06 Fig B.27
  - only one input D
    - D = 1 and CLOCK ž write 1
    - D = 0 and CLOCK ž write 0
- **J-K Flip-Flop** Sta06 Fig B.28
  - Toggle Q when J=K=1

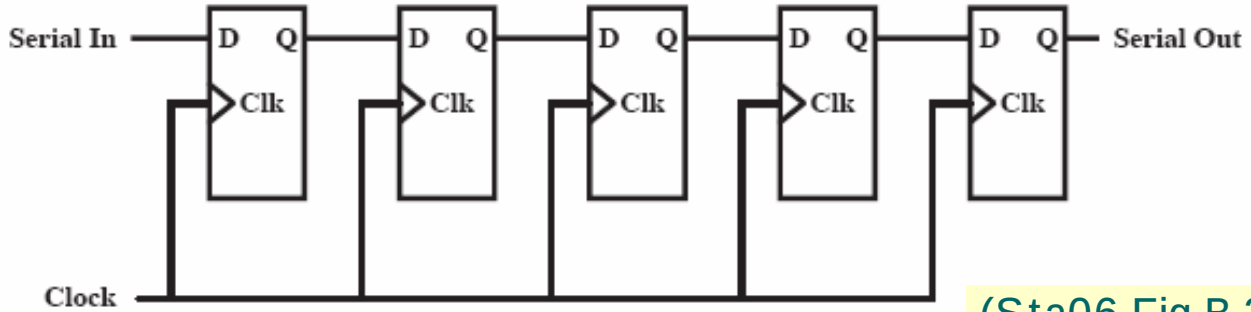


# Registers

- **Parallel registers**
  - read/write
  - CPU user registers
  - additional internal registers

Sta06 Fig B.30

- **Shift Registers**
  - shifts data 1 bit to the right
  - serial to parallel?
  - ALU ops?
  - rotate?

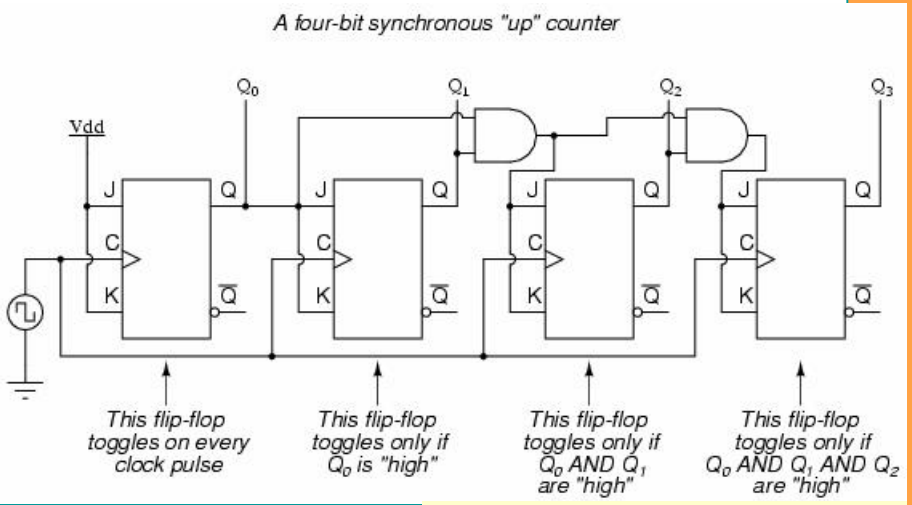



(Sta06 Fig B.31)

# Counters

- **Add 1 to stored counter value**
- **Counter**
  - parallel register plus increment circuits
- **Ripple counter** (aalto, viive)
  - asynchronous
  - increment least significant bit, and handle “carry” bit as far as needed
- **Synchronous counter**
  - modify all counter flip-flops simultaneously
  - faster, more complex, more expensive

Sta06 Fig B.32



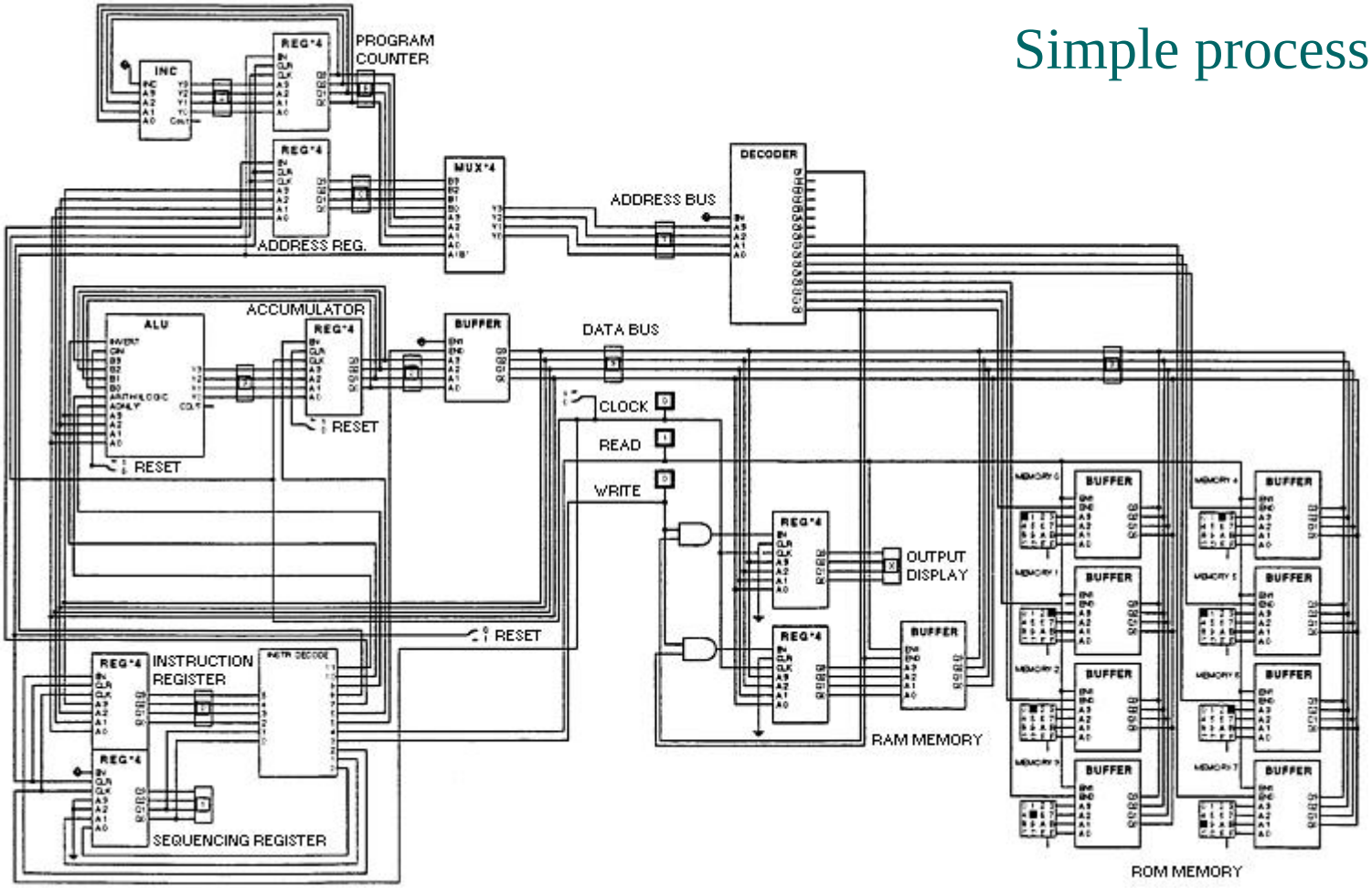

(http://www.allaboutcircuits.com)

# Summary

- **Boolean Algebra ž Gates ž Circuits**
  - can implement all with NANDs or NORs
  - simplify circuits:
    - Karnaugh, (Quine-McKluskey, Luque, ...)
- **Components for CPU design**
  - ROM, adder
  - multiplexer, encoder/decoder
  - flip-flop, register, shift register, counter

# -- End of Appendix B: Digital Logic --

## Simple processor



# Kertauskysymyksiä

- DeMorganin laki?
- Miten boolen funktio minimoidaan Karnaugh kartan avulla?
- Mitä eroa sarjallisessa piirissä on verrattuna ”normaaliin” kombinatoriseen piiriin?
- Miten S-R kiikku toimii?