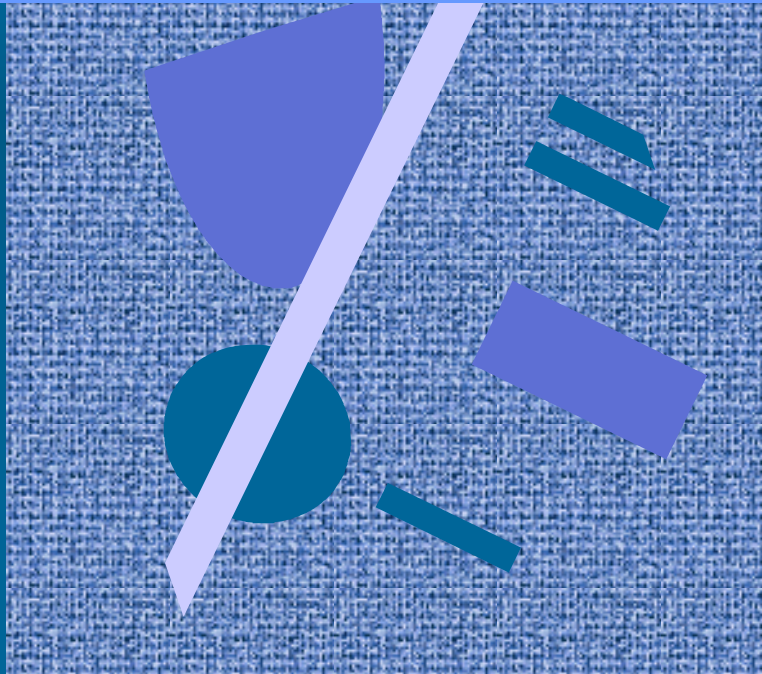


# Digital Logic

## Appendix A



Boolean Algebra  
Gates  
Combinatorial Circuits  
Sequential Circuits

# Boolean Algebra

- George Boole
  - ideas 1854
- Claude Shannon,
  - apply to circuit design, 1938 (piirisuunnittelu)
- Describe digital circuitry function
  - programming language?
- Optimise given circuitry
  - use algebra (Boolean algebra) to manipulate (Boolean) expressions into simpler expressions

# Boolean Algebra

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

# Boolean Algebra

- Other operations

- NAND

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

- NOR

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

- Truth tables

- What is the result of the operation?

Table A.1

		Q	
	AND	0	1
P	0	0	0
	1	0	1

# Postulates, Identities in Boolean Algebra

- How can I manipulate expressions?
  - Simple set of rules?
- Basic identities
  - commutative laws
  - distributive laws
  - identity elements
  - inverse elements
  - associative laws
  - DeMorgan's theorem

Table A.2

(vaihdantalait)

(osittelulait)

(identiteetit)

(vasta-alkiot)

(liitantalait)

(DeMorganin laki)

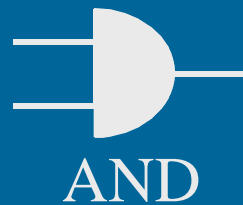
# Gates

(portit)

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

(yhteenlaskupiiri,  
kertolaskupiiri)

Fig. A.1



- Gate delay
  - change inputs, new output available after gate delay

(viive)

# Functionally Complete Set

- Can build all basic gates (and, or, not) from a smaller set of gates

- and, or, not (trivial!)

- and, not

- OR?

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

- or, not

- nand

- nor

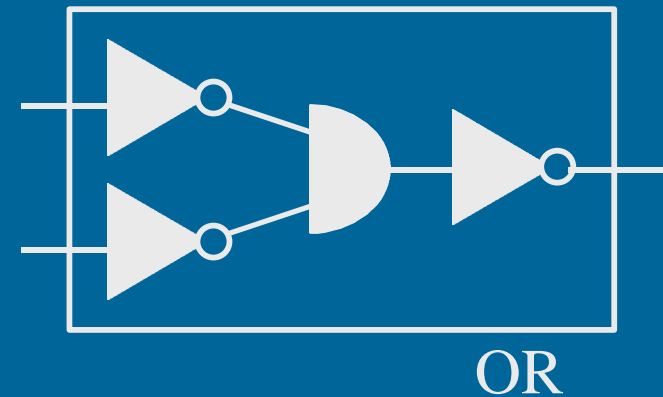


Fig A.2

Fig A.3

# Combinational Circuits <sup>(3)</sup>

(yhdistelmä-  
piirit)

- Interconnected set of gates
  - change input, wait for gate delays, new output
- Output is Boolean function of input signals
  - m (binary, Boolean) inputs
  - n (binary, Boolean) outputs
- Described in three ways
  - Boolean equations (one for each output)

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

- truth table

Table A.3

- graphical symbols describe implementation with gates and wires

Fig A.4



# Simplify Presentation (and Implementation) <sup>(3)</sup>

- Boolean equations
  - Sum of products form (SOP)

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

Fig A.4

- Product of sums form (POS)

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

Boolean algebra

Fig A.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

$$\begin{aligned} F &= \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} \\ &= \overline{A}B + B\overline{C} = B(\overline{A} + \overline{C}) \end{aligned}$$

Fig A.5

Fig A.6

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

# Karnaugh Map Squares

- Each square represents complete input value combination (canonical form)

order!!

- adjacent squares differ only in one input value (wrap around)

CD:	00	01	11	10
AB				
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Square for input value combination

$$\overline{A}\overline{B}CD = 1001$$

# Karnaugh Maps

- Represent Boolean function (I.e., circuit) truth table in another way Fig A.7

- each square is **one product** in sums-of-products (SOP) presentation
- value is one (1) iff corresponding input values give value 1
- value is function value for those input values

CD

	00	01	11	10
00			1	
01				
11	1			
10		1		

AB

$$F = \overline{A}BCD + ABC\overline{D} + AB\overline{C}D$$

# Karnaugh Map Simplification

- Starting point:
  - Adjacent squares differ only in one input variable value

1	1
$c$	$\bar{c}$

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  
to corresponding values for those squares)



Can ignore that variable for those expressions

# Using Karnaugh Maps to Minimize Boolean Functions <sup>(6)</sup>

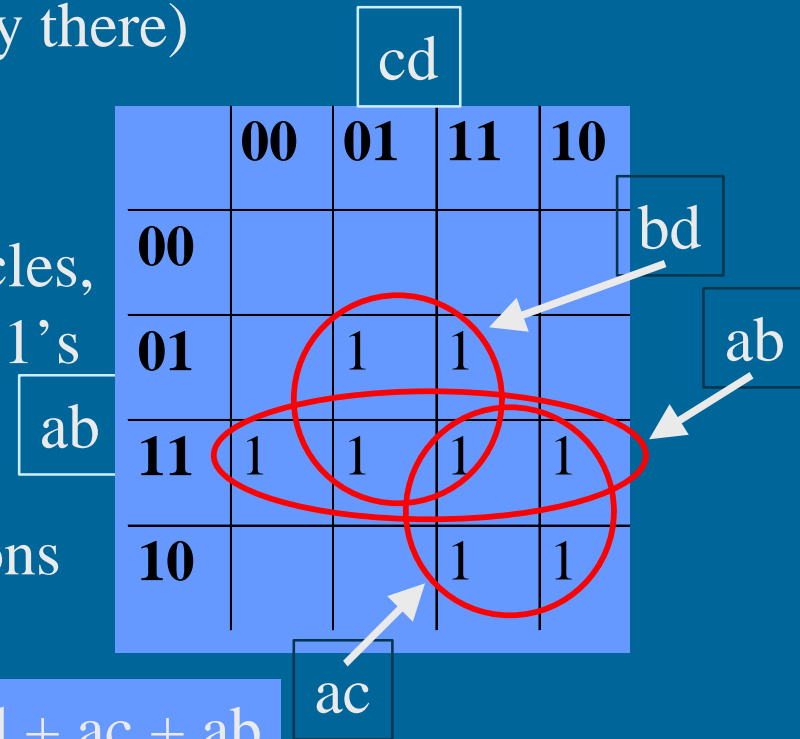
Original function

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

Canonical form (now already there)

Karnaugh Map

Find smallest number of circles, each with largest number of 1's

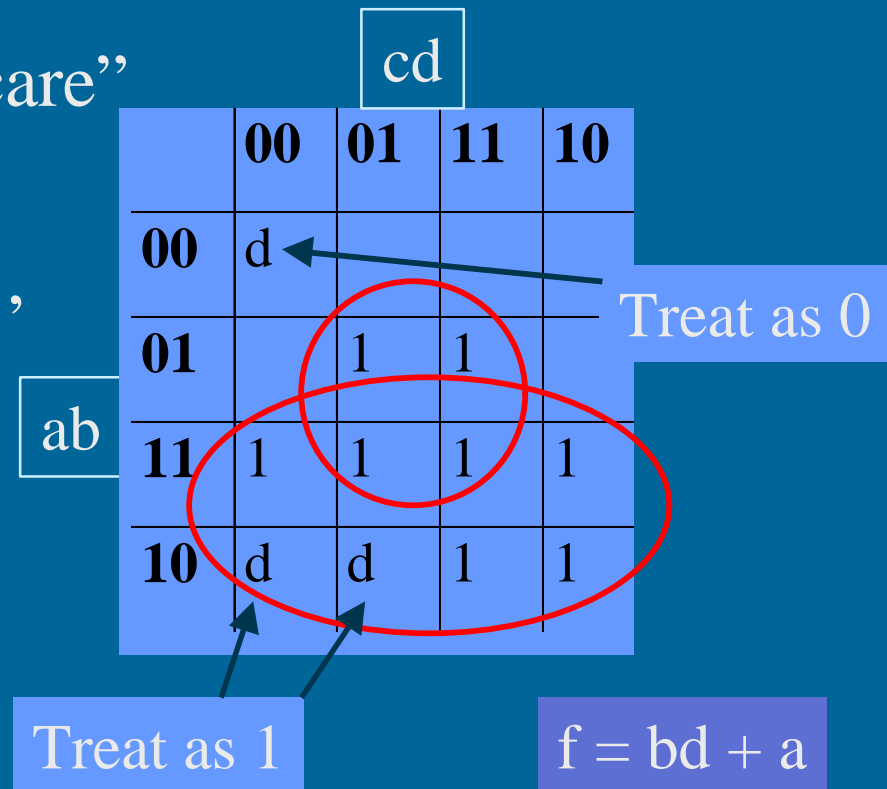


Select parameter combinations corresponding to the circles

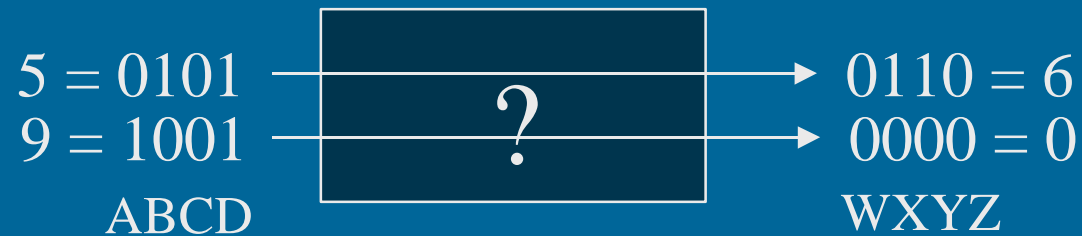
Get reduced function  $f = bd + ac + ab$

# Impossible Input Variable Combinations

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



# Example: Circuit to add 1 (mod 10) to 4-bit BCD decimal number <sup>(4)</sup>



Truth table?

Table A.4

Karnaugh maps for W, X, Y and Z?

Fig. A.10



# Quine-McKluskey Method

- Another method to minimise Boolean expressions
- Why?
  - Karnaugh maps become complex with 6 input variables
- Quine-McKluskey method
  - tabular method
  - automatically suitable for programming
  - details skipped

# Basic Combinational Circuits

- Building blocks for more complex circuits
  - Multiplexer
  - Encoders/decoder
  - Read-Only-Memory
  - Adder

# Multiplexers (2)

- Select one of many possible inputs to output
  - black box Fig A.12
  - simple truth table Tbl A.7
  - implementation Fig A.13
- Each input “line” can be many parallel lines
  - select one of three 16 bit values Fig A.14
    - $C_{0..15}$ ,  $IR_{0..15}$ ,  $ALU_{0..15}$
  - simple extension to one line selection

# Encoders/Decoders

- Only one of many Encoder input or Decoder output lines can be 1
- Encode the line number as output
  - hopefully less pins (wires) needed this way
  - Example:
    - encode 8 input wires with 3 output pins
    - route 3 wires around the board
    - decode 3 wires back to 8 wires at target

Fig A.15



# Read-Only-Memory (ROM) <sup>(4)</sup>

- Given input values, get output value
- Consider input as address, output as contents of memory location
- Truth tables for a ROM Table A.8
  - 64 bit ROM
  - 16 words, each 4 bits wide

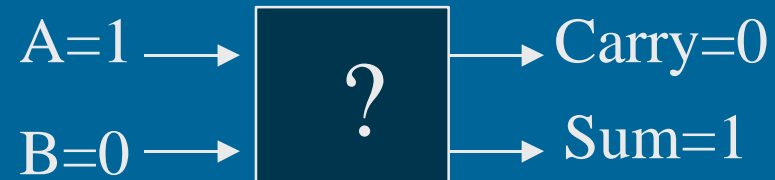
$$\text{Mem (7)} = 4$$

$$\text{Mem (11)} = 14$$

- Implementation with decoder & or gates

# Adders (4)

1-bit adder



1-bit adder with carry



Implementation

Table A.9, Fig A.22

Build 4-bit adder from 4 1-bit adders

Fig A.21

# Sequential Circuit

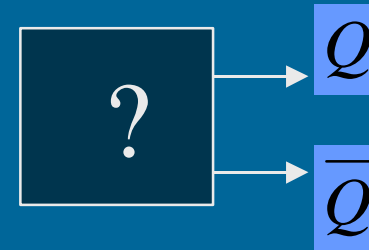
(sarjalliset piirit)

- Circuit has (modifiable) internal state
- Output of circuit depends (also) on internal state
  - not only from current inputs
  - $\text{output} = f(\text{input}, \text{state})$
  - $\text{new state} = f(\text{input}, \text{state})$
- Processor control, registers

# Flip-Flop

(kiikku)

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

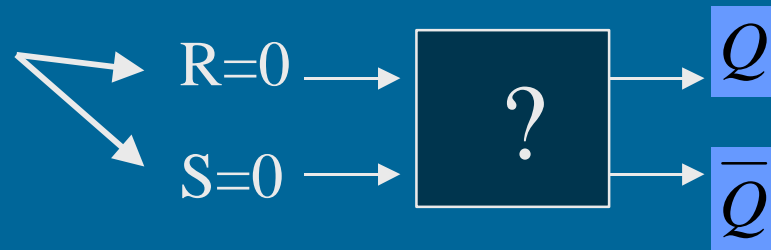




# S-R Flip-Flop or S-R Latch (4)

(laituri)

Usually  
both 0

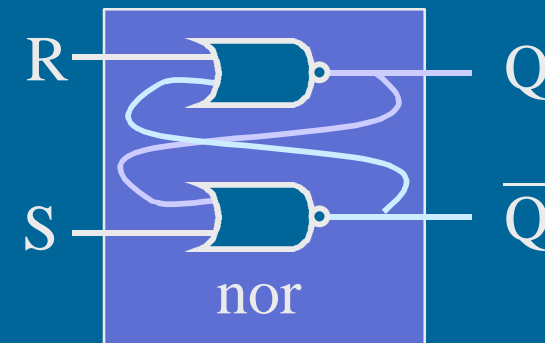


“Write 1” = “set S=1 for a short time” = “set”



“Write 0” = “set R=1 for a short time” = “reset”

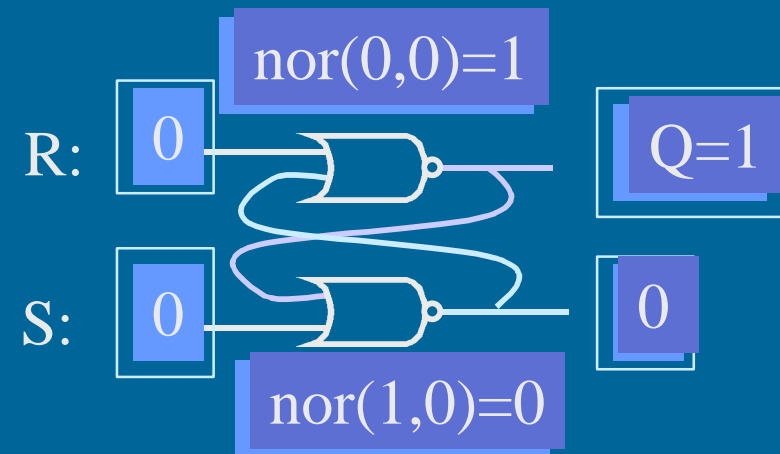
$\text{nor}(0, 0) = 1$   
 $\text{nor}(0, 1) = 0$   
 $\text{nor}(1, 0) = 0$   
 $\text{nor}(1, 1) = 0$



# S-R Latch Stable States (3)

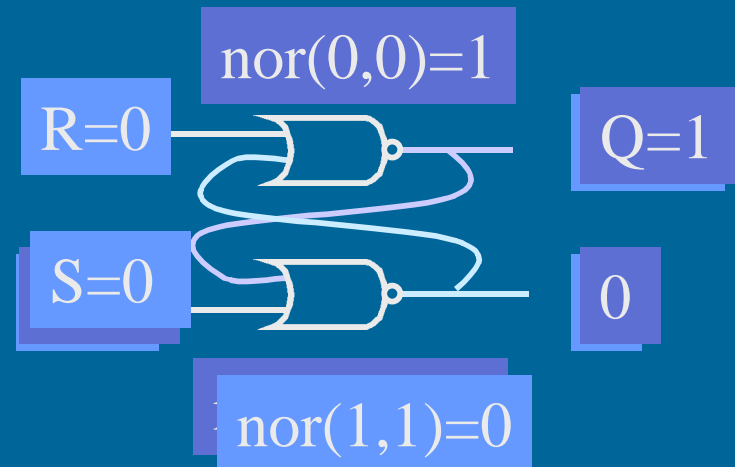
- 1 bit memory (value = value of Q)
- bi-stable, when R=S=0
  - Q=0?
  - Q=1?

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



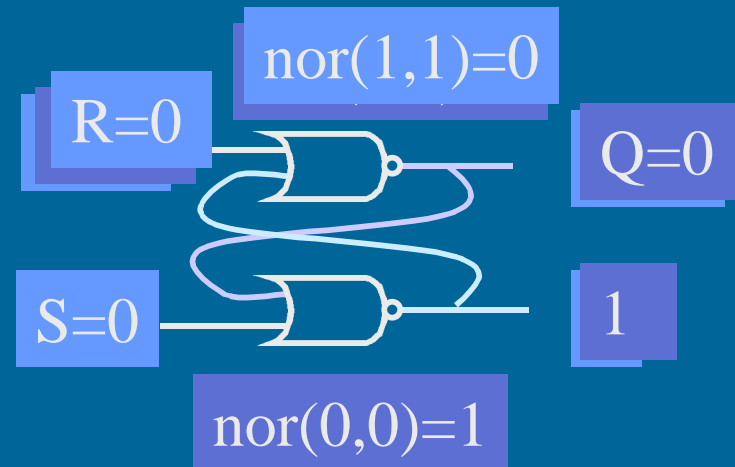
# S-R Latch Set (1) and Reset (0) <sup>(n)</sup>

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



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

$\text{nor}(0,0) = 1$   
 $\text{nor}(0,1) = 0$   
 $\text{nor}(1,0) = 0$   
 $\text{nor}(1,1) = 0$



# Clocked Flip-Flops

- State change can happen only when clock is 1
  - more control on state changes
- Clocked S-R Flip-Flop Fig. A.26
- D Flip-Flop Fig. A.27
  - only one input D
    - $D = 1$  and CLOCK  $\rightarrow$  write 1
    - $D = 0$  and CLOCK  $\rightarrow$  write 0
- J-K Flip-Flop Fig. A.28
  - Toggle Q when  $J=K=1$  Fig. A.29

# Registers (2)

- Parallel registers
  - read/write
  - CPU user registers
  - additional internal registers
- Shift Registers
  - shifts data 1 bit to the right
  - serial to parallel?
  - ALU operation?
  - rotate?

Fig. A.30

Fig. A.31

# Counters (4)

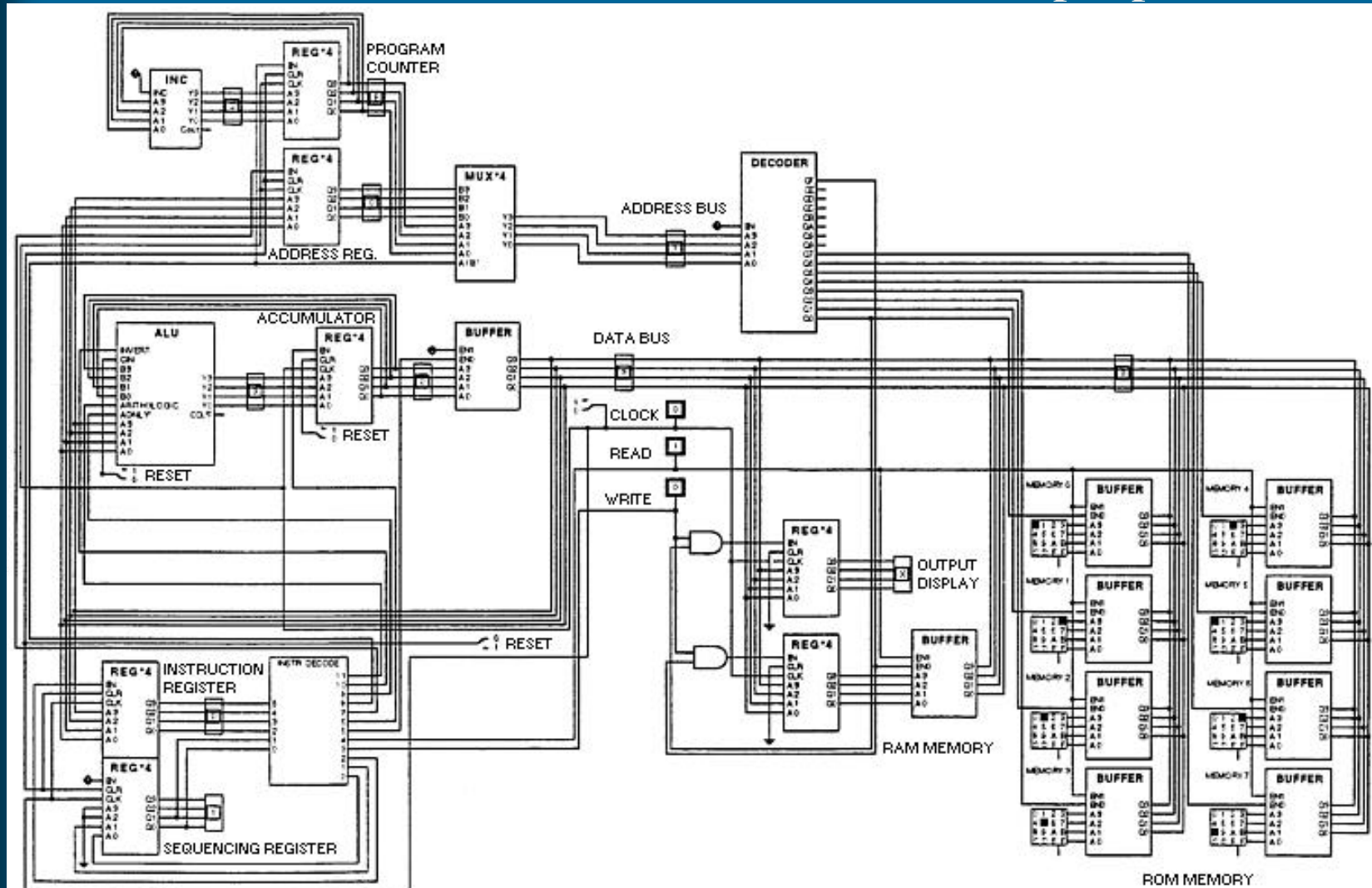
- Add 1 to stored counter value
- Counter
  - parallel register plus increment circuits
- Ripple counter
  - asynchronous
  - increment least significant bit, and handle “carry” bit as far as needed
- Synchronous counter
  - modify all counter flip-flops simultaneously
  - faster, more complex, more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

# Summary

- Boolean Algebra  $\rightarrow$  Gates  $\rightarrow$  Circuits
  - can implement all with NANDs or NORs
  - simplify circuits: Karnaugh, Quine-McKluskey
- Components for CPU design
  - ROM, adder
  - multiplexer, encoder/decoder
  - flip-flop, register, shift register, counter

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

## Simple processor



[http://www.gamezero.com/team-0/articles/math\\_magic/micro/stage4.html](http://www.gamezero.com/team-0/articles/math_magic/micro/stage4.html)

19.9.2000

Copyright Teemu Kerola 2000

32