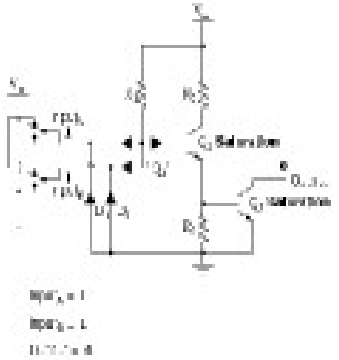
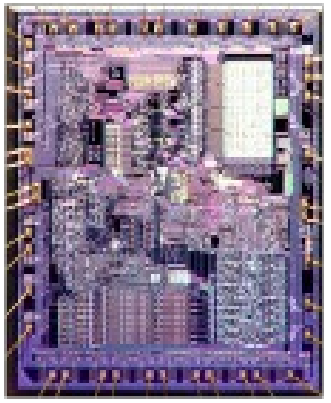
Computer Systemen en Computer Architectuur

Maandag 3 maart 2025 – **Binaire Logica: de Basis van Digitale Computers**

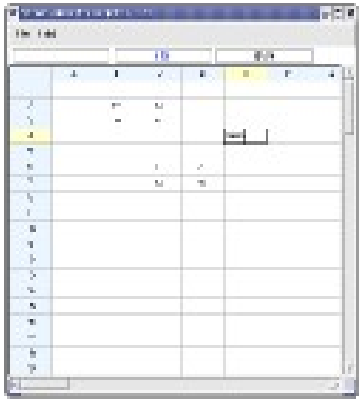
Hans Vangheluwe





```
strcpy:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
    add  $s0, $zero, $zero
L1: add  $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```

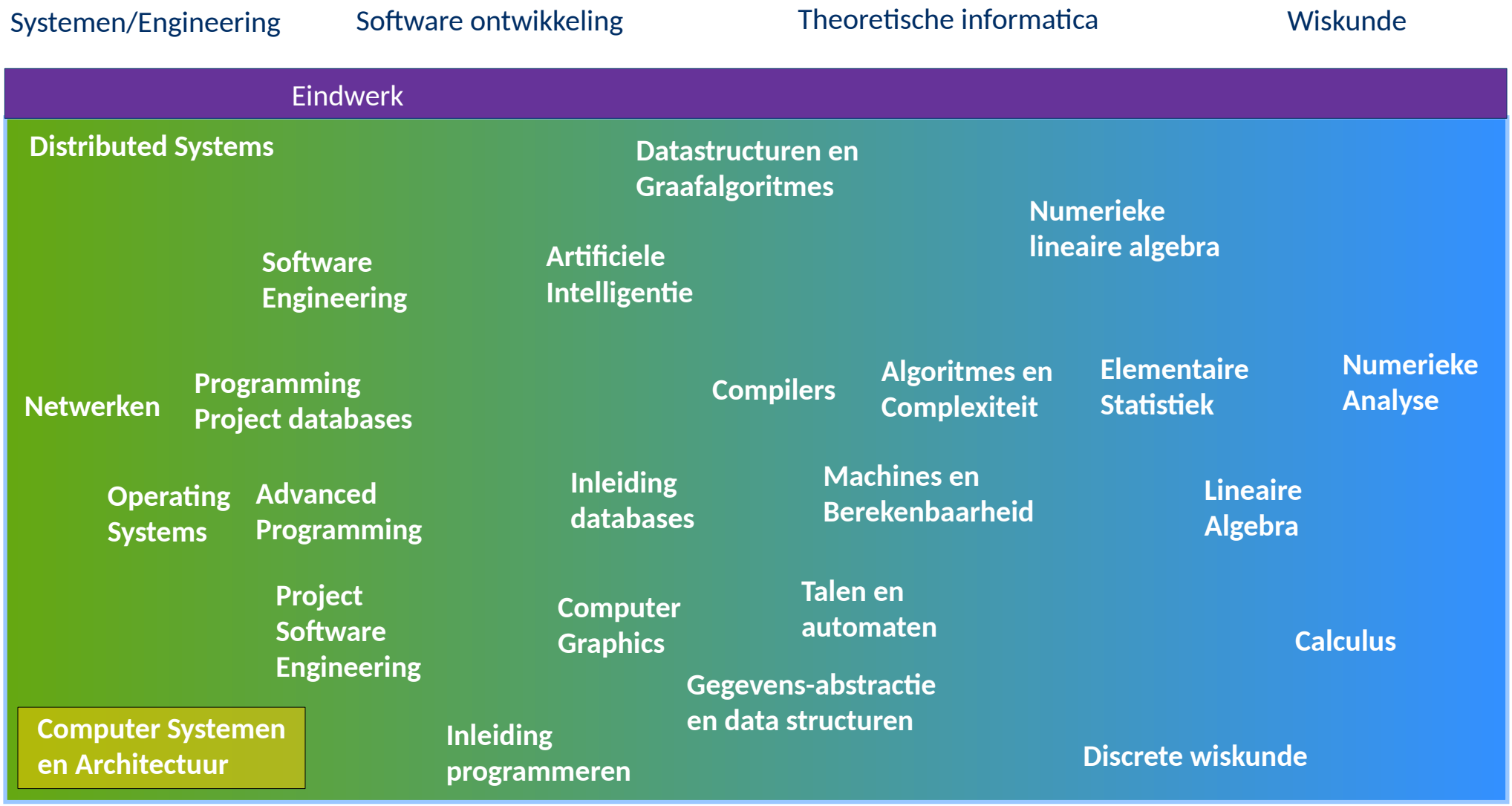
```
k = 1
xPowerK = x
Sign = 1; s = 0

while k <= N :
 term = sign*xPowerK/
        factorial(k)
 s = s + term
 k = k + 2
 xPowerK =
  xPowerK * xSquare
 sign = -sign
```

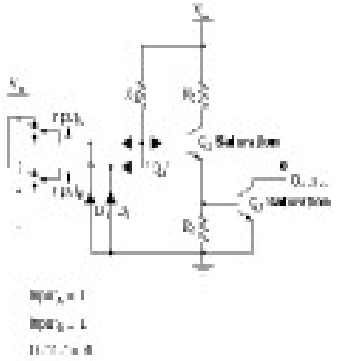


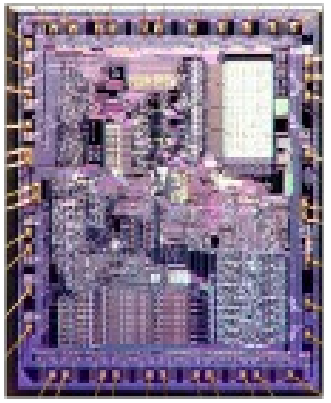
physics digital electronics — computer architecture / systems — HL progr. languages operating systems — complex SW applications

http://msdl.uantwerpen.be/people/hv/teaching/ComputerSystemsArchitecture/proefles.pdf

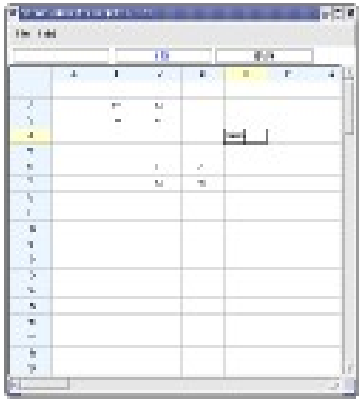
Bachelor Programma Informatica



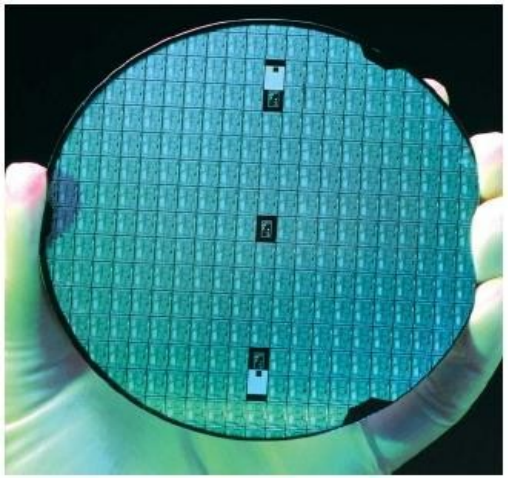


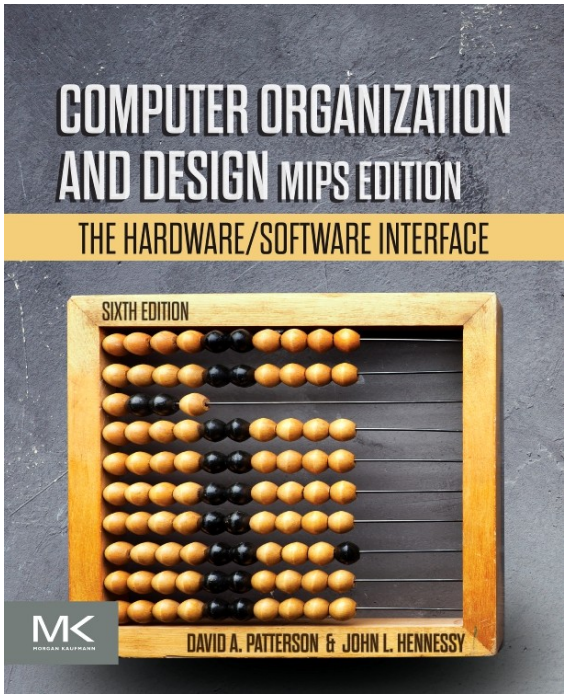


```
strcpy:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
    add  $s0, $zero, $zero
L1: add  $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```



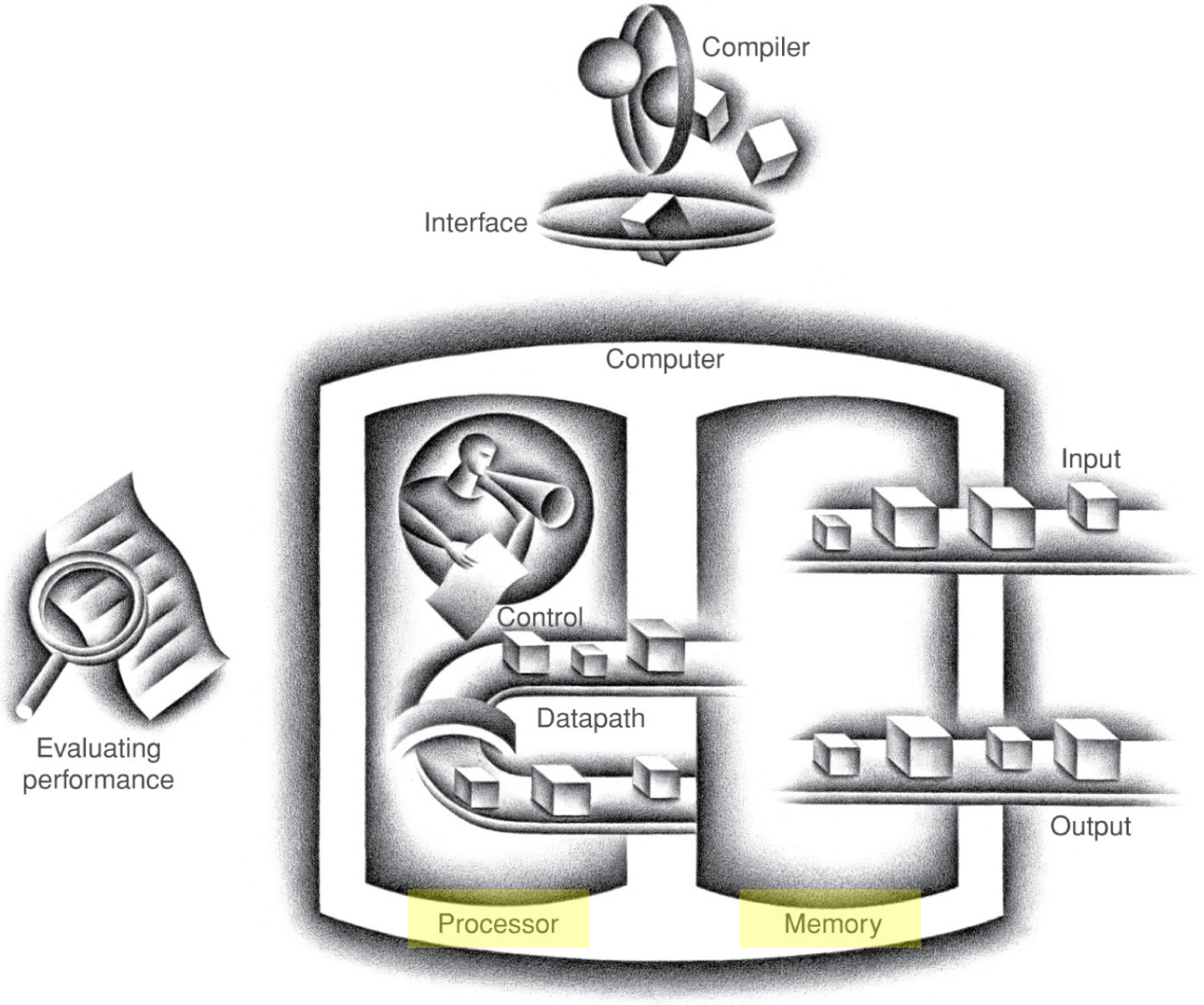
physics digital electronics — computer architecture / systems — HL progr. languages operating systems — complex SW applications



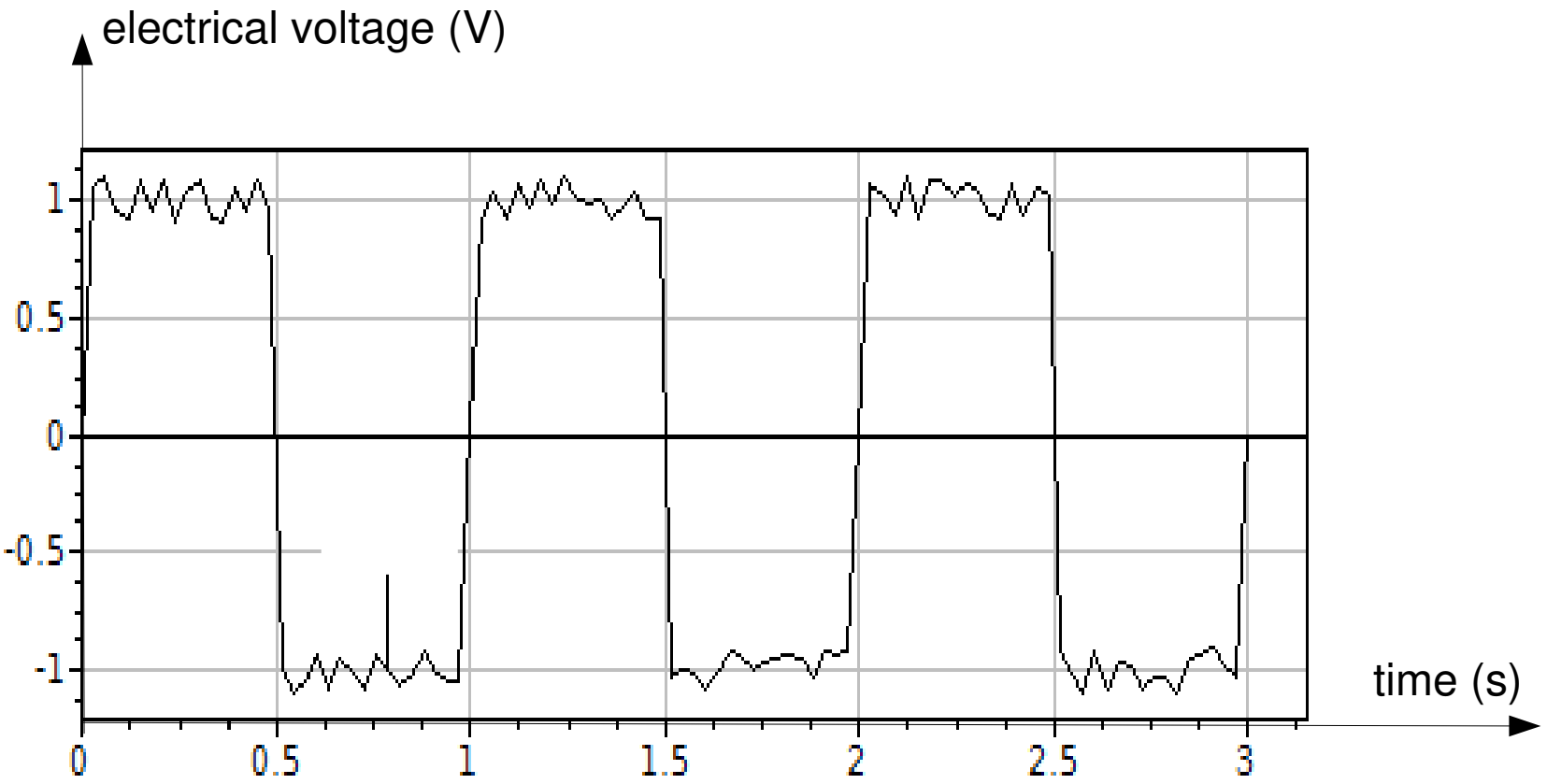


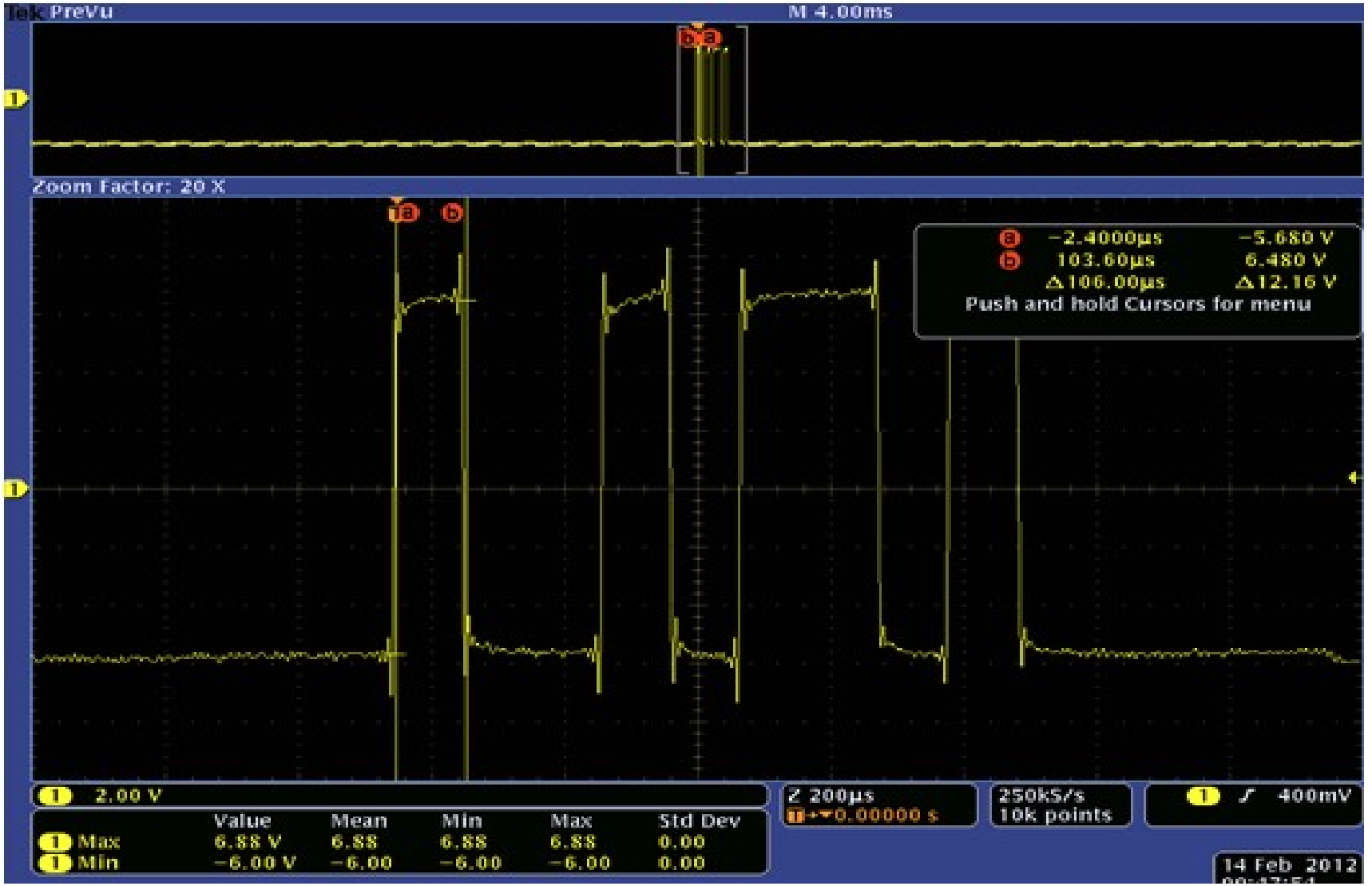

```
def visitFunction(self, function):
  if typeChecker.debug: typeChecker.typeCheck([function], [Function])
  numArg=0
  for argument in function.getArgs():
    if isinstance(argument, RangeRef):
      numArg += len(argument.getCellRefSet())
    else:
      numArg += 1
    argument.accept(self)

  args=self.__evalStack[-numArg:]
  self.__evalStack=self.__evalStack[0:-numArg]


  if len(args)>1 and self.__checkValueError(args):
    self.__evalStack.append(0)
    return

  execStr="answer = "+function.getName()+"("+str(args)+")"
  try:
    exec execStr
  except NameError, n:
    fName=split(n[0],"'")
    self.__nameError=True
    self.__nameErrorStr=fName[1]
    self.__evalStack.append(0)
    return
  self.__evalStack.append(answer)
```

(Functionele) Componenten van een Computer

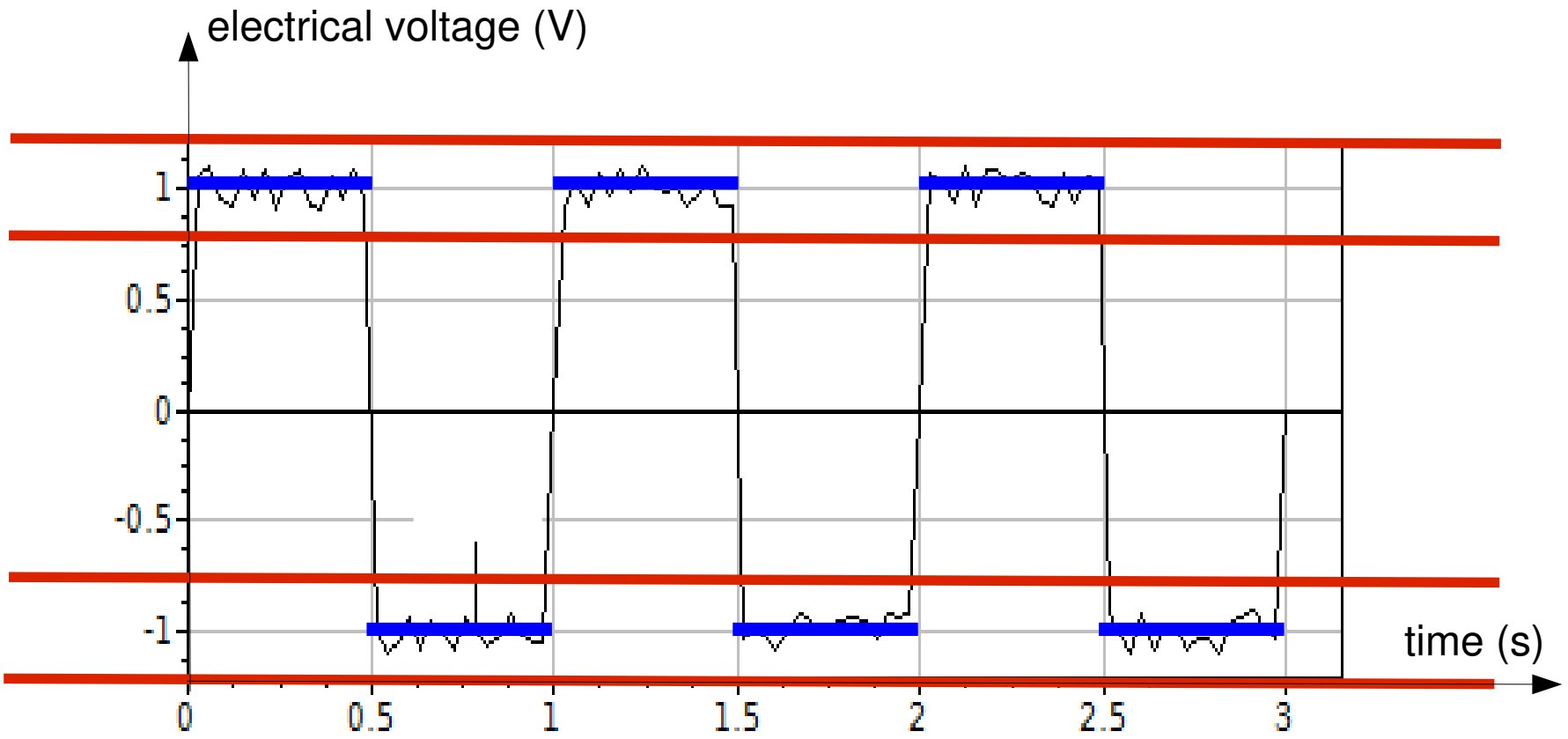


van Analoog ...

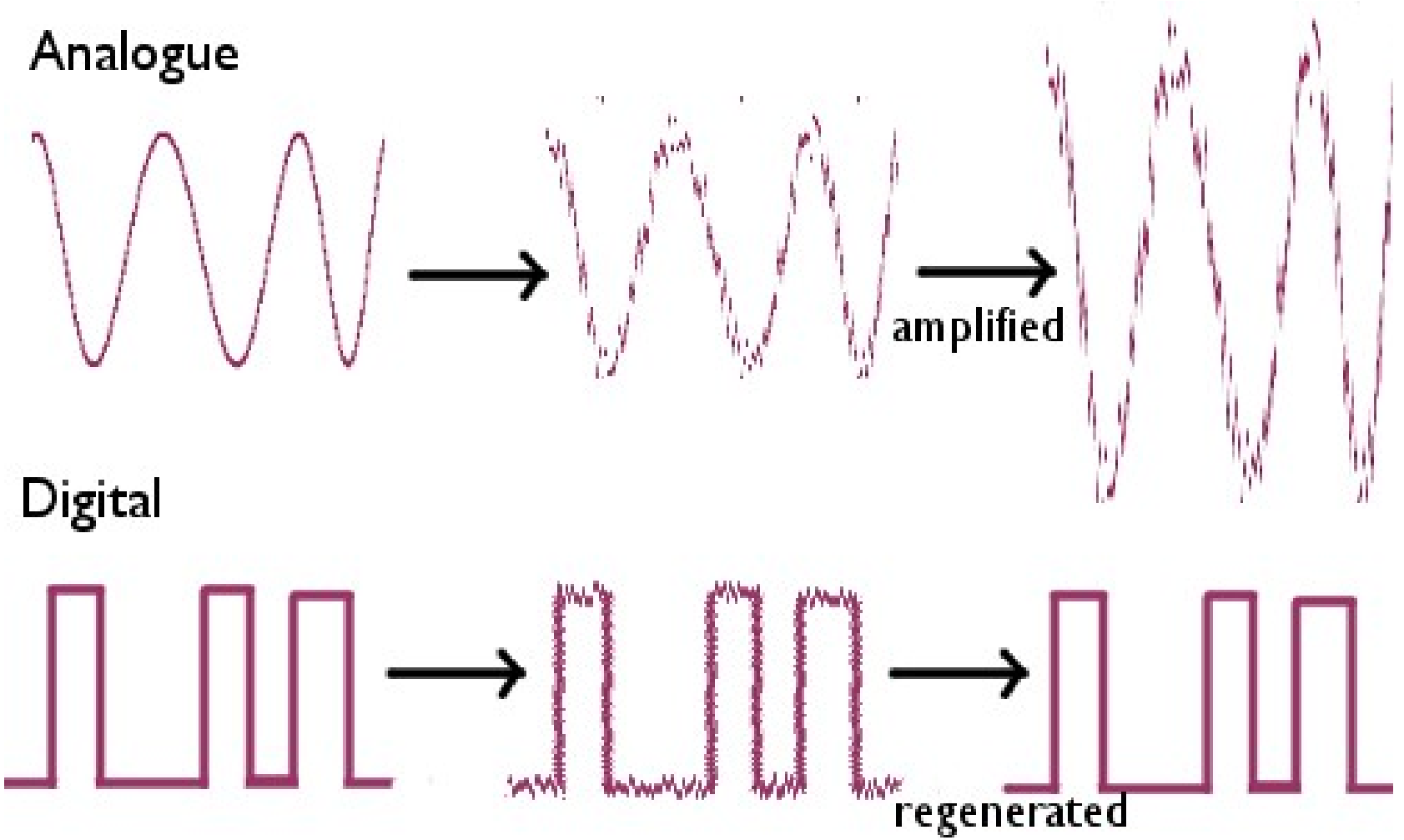



Tek PreVu
M 4.00ms
Zoom Factor: 20 X
a -2.4000µs -5.680 V
b 103.60µs 6.480 V
Δ106.00µs Δ12.16 V
Push and hold Cursors for menu
1 2.00 V
Value Mean Min Max Std Dev
1 Max 6.88 V 6.88 6.88 6.88 0.00
1 Min -6.00 V -6.00 -6.00 -6.00 0.00
Z 200µs
0.00000 s
250kS/s
10k points
1 400mV
14 Feb 2012

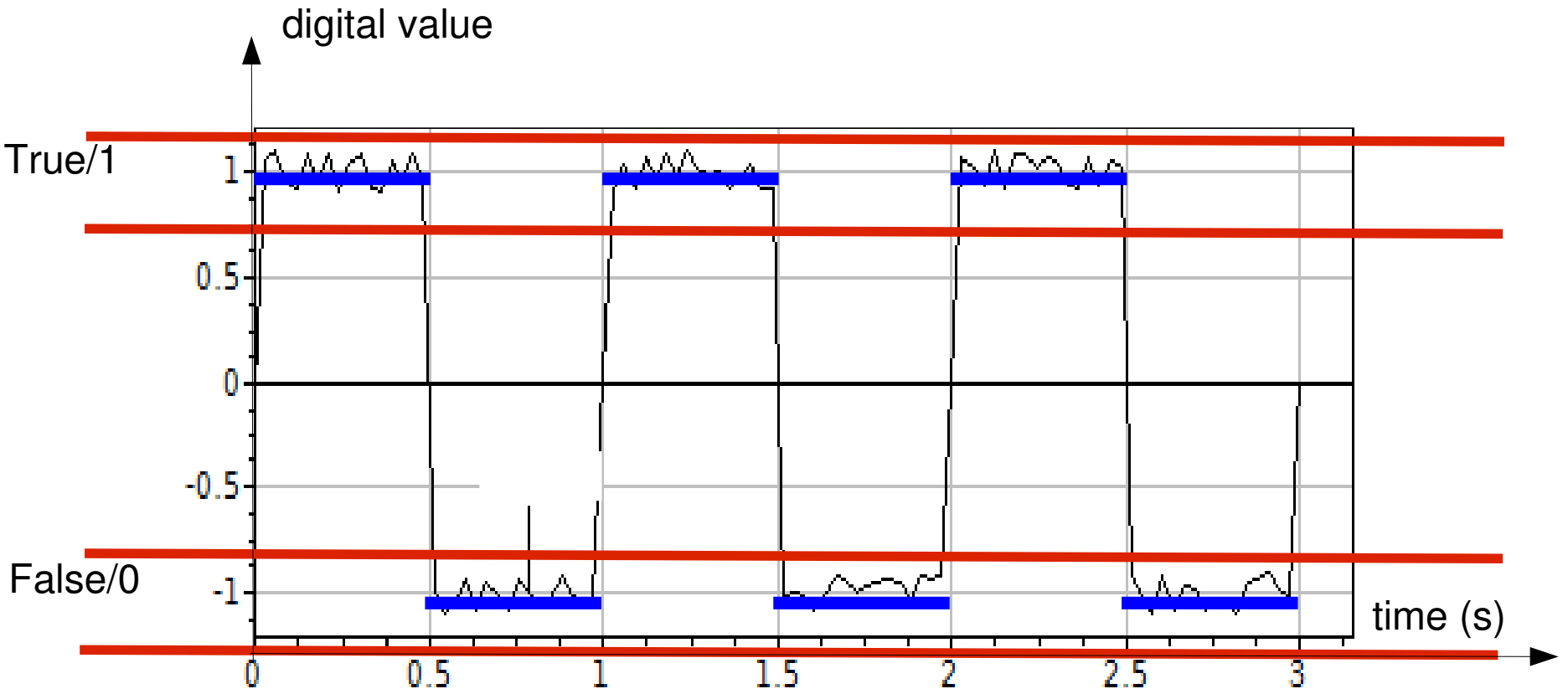
… naar Digitaal



Voordelen: Analoog vs. Digitaal



… naar Digitaal



logical 0/1 – Boolean True/False – asserted/de-asserted – high/low

BITs (**B**inary dig**IT**s) om informatie to **coderen**

Met 1 bit, kan 2 **verschillende entiteiten voorstellen**

Met 2 bits, kan 4 verschillende entiteiten voorstellen

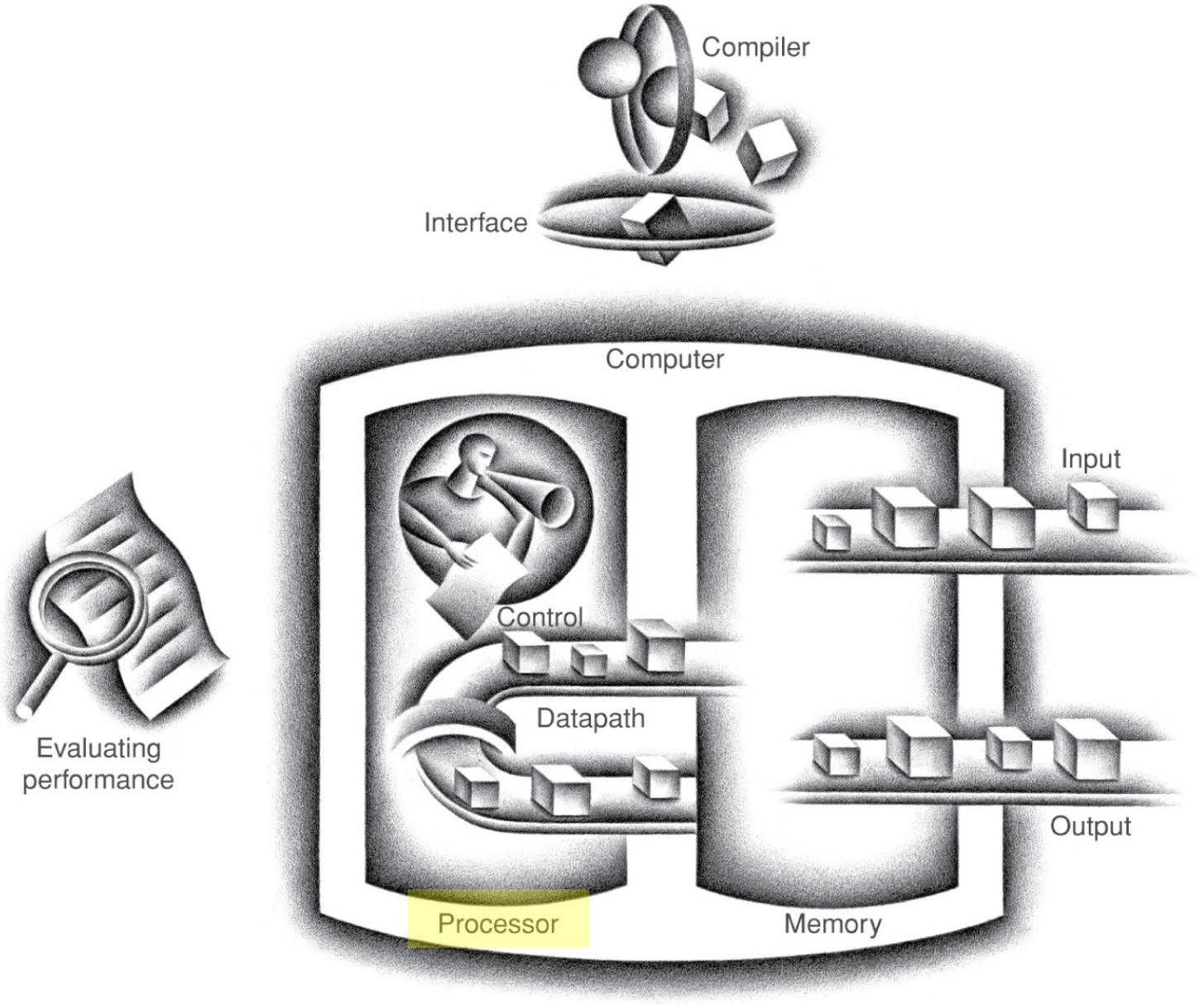
...

Met **N** bits, kan $\mathbf{2^N}$ verschillende entiteiten voorstellen

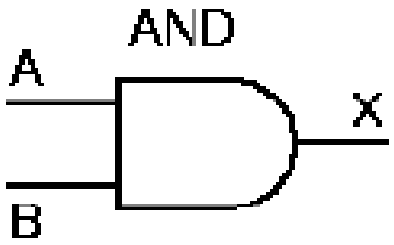
Voorbeeld:
{Rood, Groen, Blauw}
gecodeerd als {00, 01, 10}

1 Bit	2 Bits	3 Bits	4 Bits	5 Bits
0	00	000	0000	00000
1	01	001	0001	00001
	10	010	0010	00010
	11	011	0011	00011
		100	0100	00100
		101	0101	00101
		110	0110	00110
		111	0111	00111
			1000	01000
			1001	01001
			1010	01010
			1011	01011
			1100	01100
			1101	01101
			1110	01110
			1111	01111
				10000
				10001
				10010
				10011
				10100
				10101
				10110
				10111
				11000
				11001
				11010
				11011
				11100
				11101
				11110
				11111

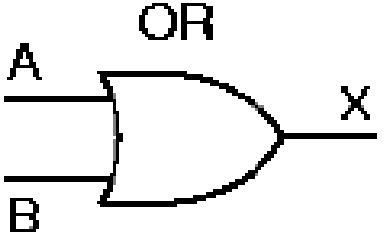
(Functionele) Componenten van een Computer



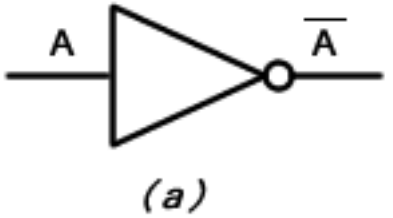
Universele Basis van bewerkingen: Logische Componenten



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1



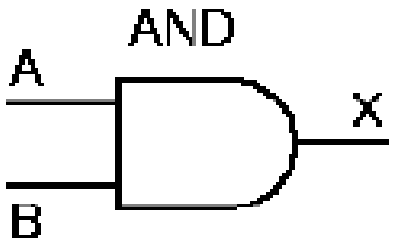
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



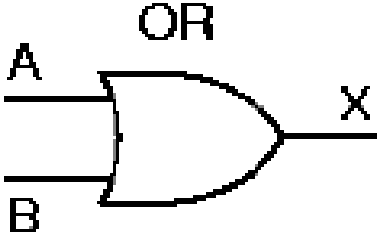
(a)

A	$\overline{A}$
0	1
1	0

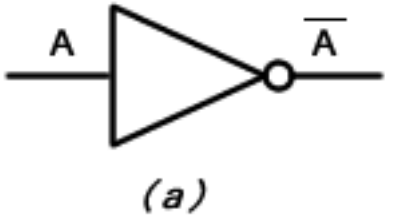
Universele Basis van bewerkingen: Logische Componenten



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

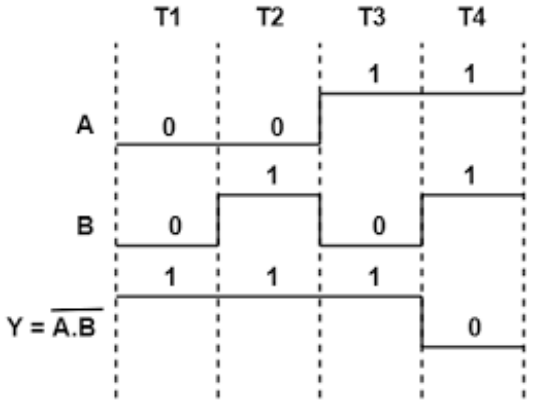


A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



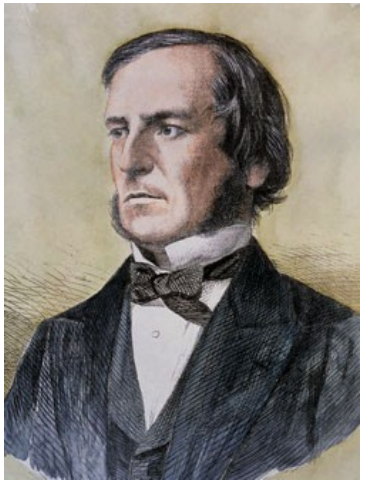
(a)

A	$\overline{A}$
0	1
1	0



Booleaanse algebra: AND (.), OR (+), NOT ($^-$)

- Identity law: $A + 0 = A$ and $A \cdot 1 = A$.
- Zero and One laws: $A + 1 = 1$ and $A \cdot 0 = 0$.
- Inverse laws: $A + \overline{A} = 1$ and $A \cdot \overline{A} = 0$
- Commutative laws: $A + B = B + A$ and $A \cdot B = B \cdot A$.
- Associative laws: $A + (B + C) = (A + B) + C$ and $A \cdot (B \cdot C) = (A \cdot B) \cdot C$.
- Distributive laws: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ and $A + (B \cdot C) = (A + B) \cdot (A + C)$.



George Boole
(1815 - 1864)

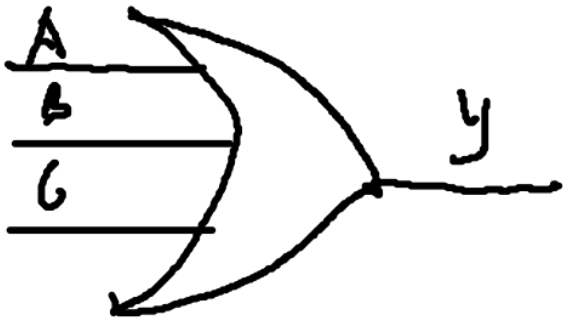
Wetten van De Morgan:

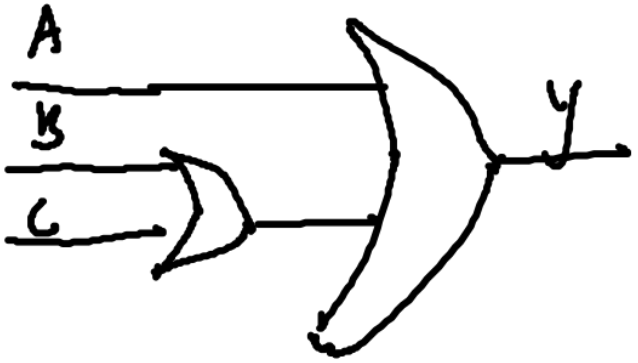
- $\overline{A + B} = \overline{A} \,.\, \overline{B}$
- $\overline{A \,.\, B} = \overline{A} + \overline{B}$

Augustus De Morgan
(1806-1871)

Associativiteit

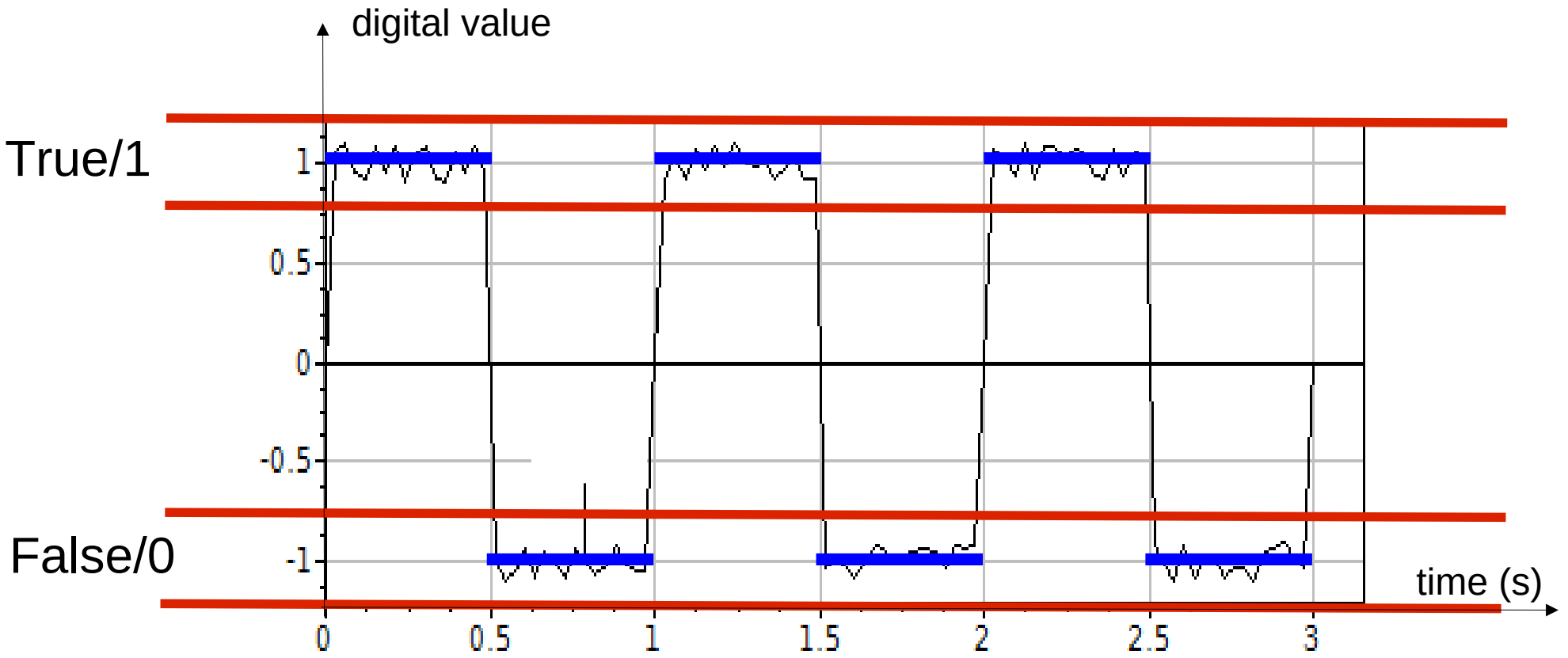
$$y = A + B + C = A + (B + C) = (A + B) + C$$





Digitale **signalen** zijn gebaseerd op Analoge Signalen

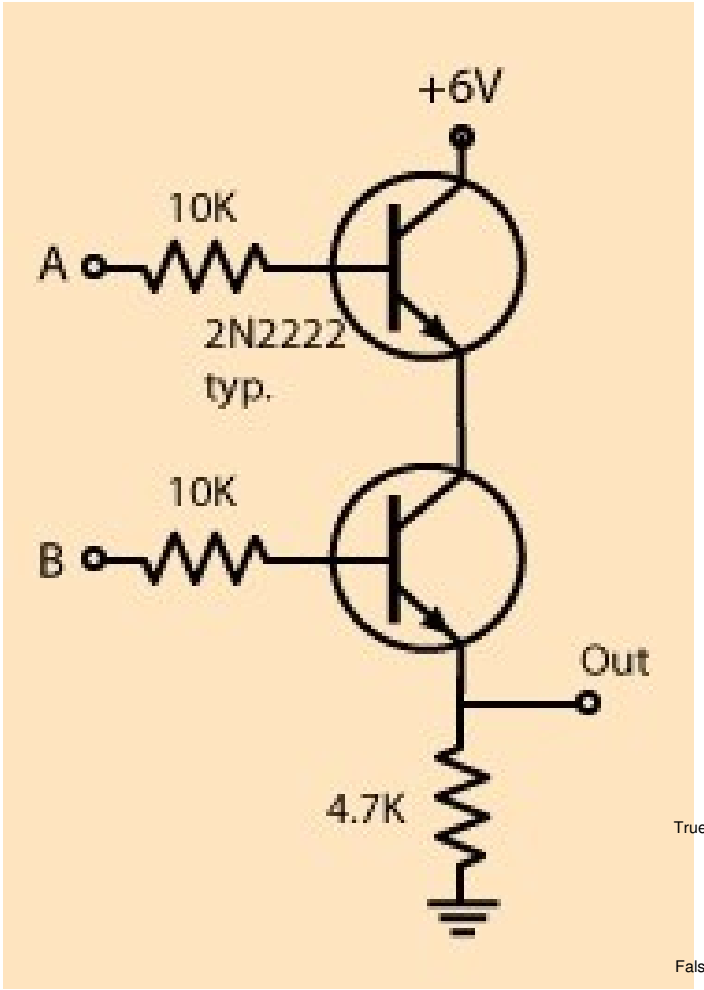
→ Digitale (logische) **bewerkingen** op Analoge Signalen



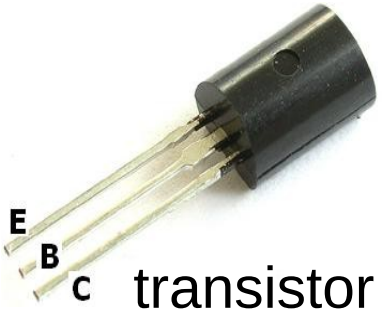
logical 0/1 – Boolean True/False – asserted/de-asserted – high/low

Universele Basis implementatie: Logische Componenten in electronica

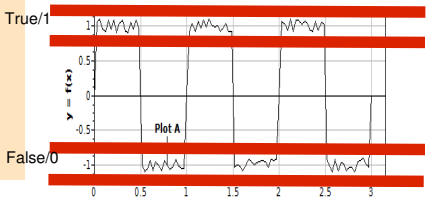
AND



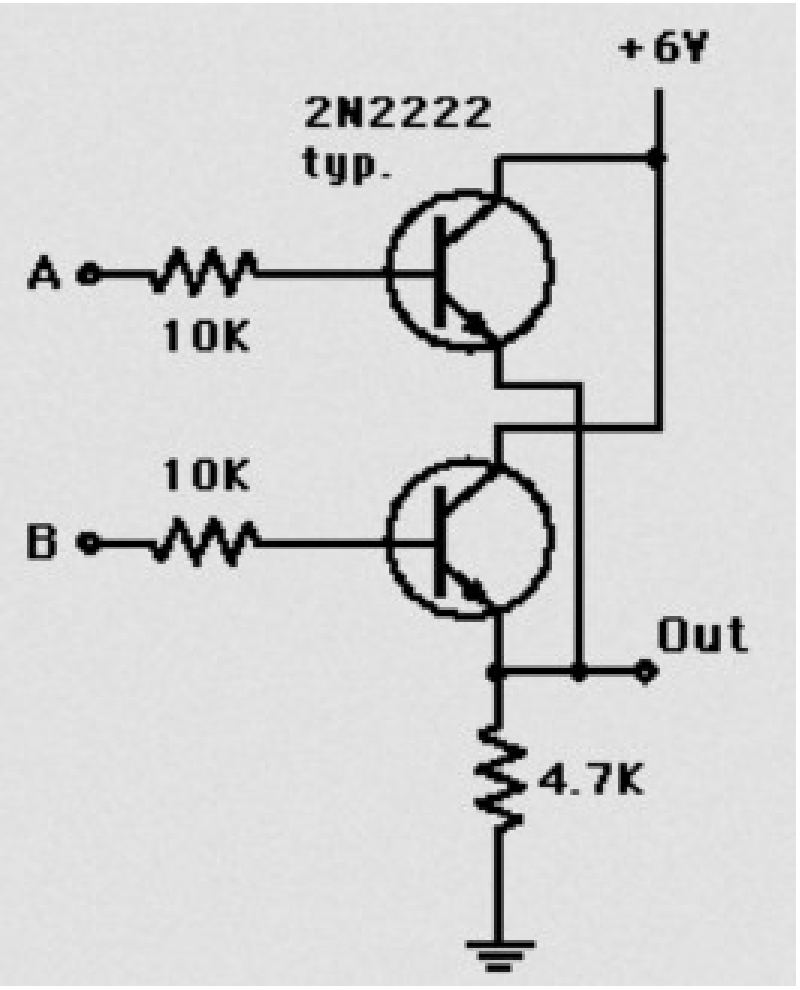
weerstand



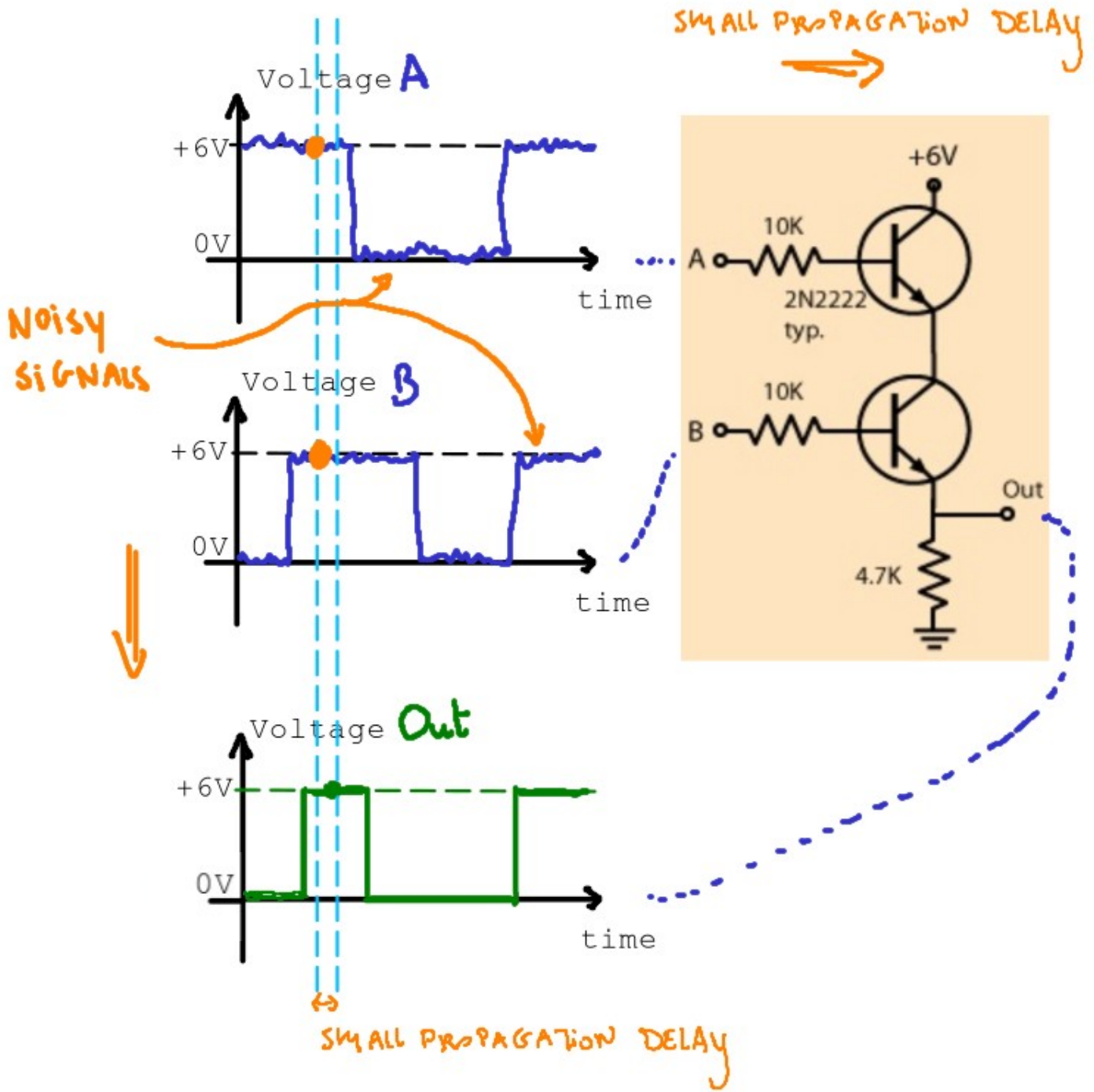
transistor



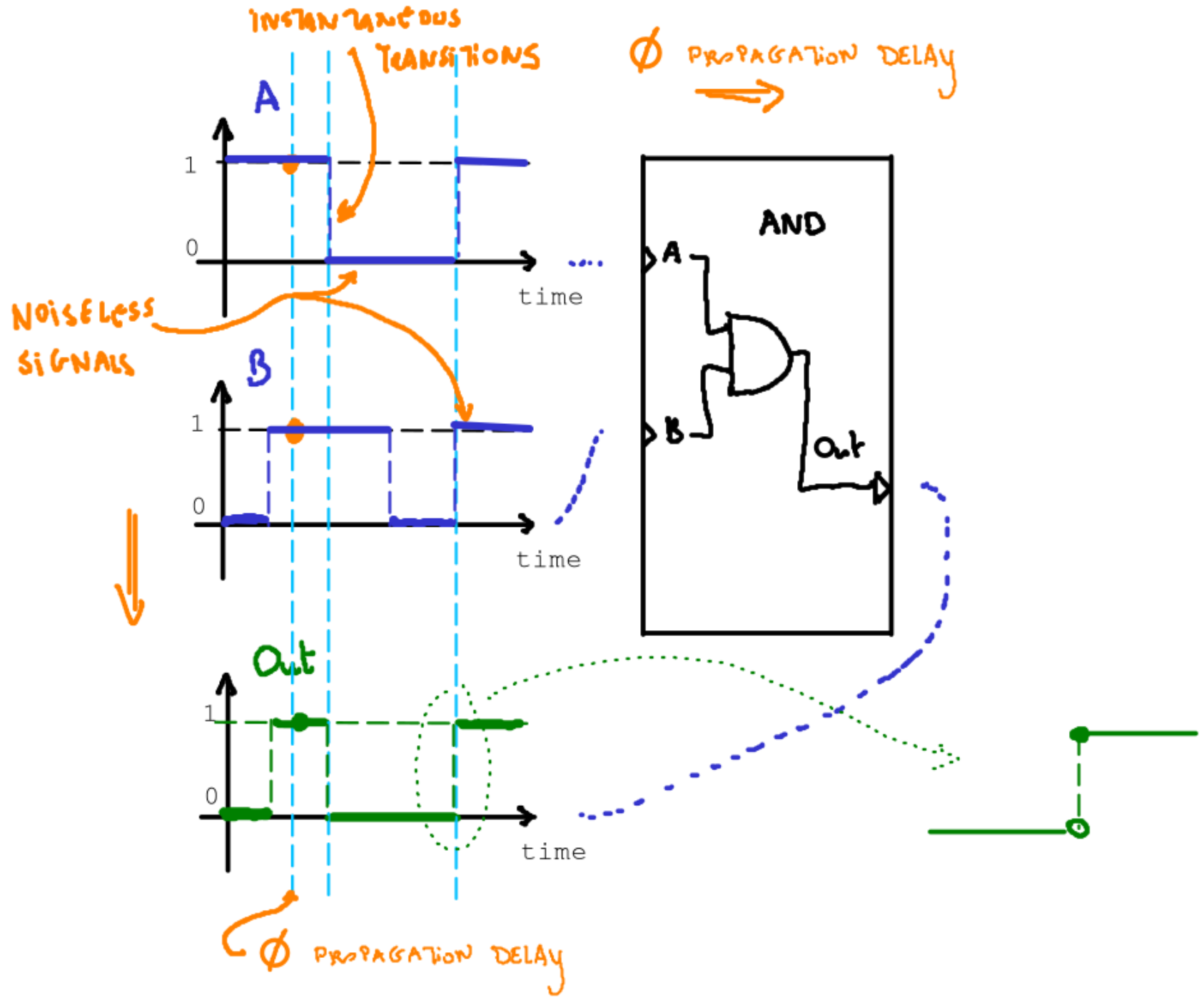
OR



physical (analog) view

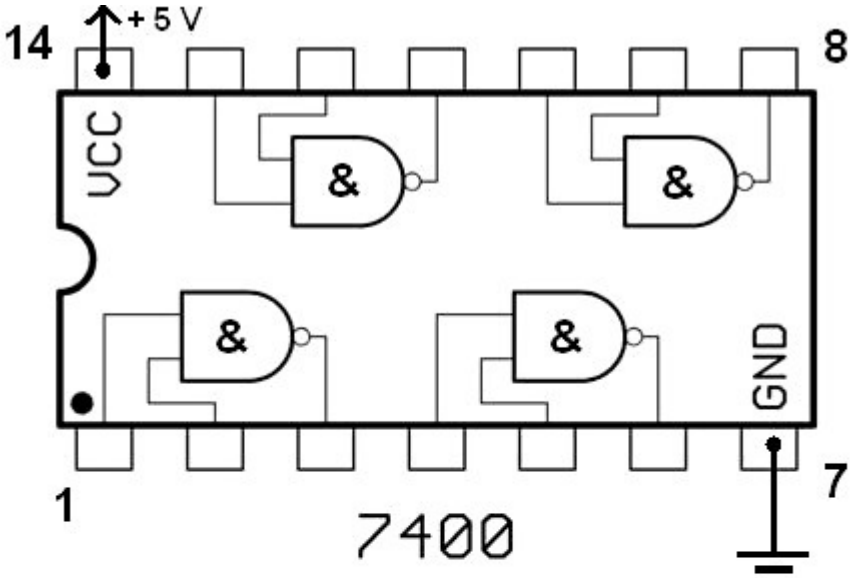


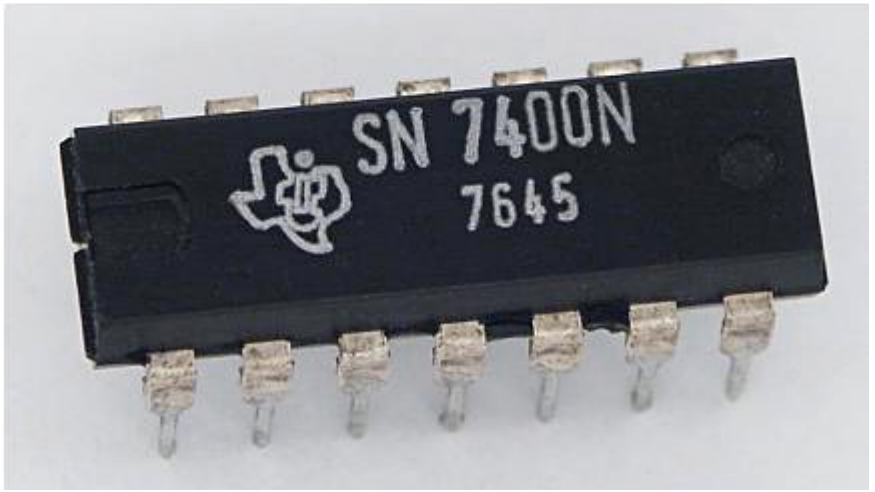
logical (digital) view (idealized)



Logische Componenten Implementeren:

SN 7400N met 4 NAND gates (~ 8 transistors)

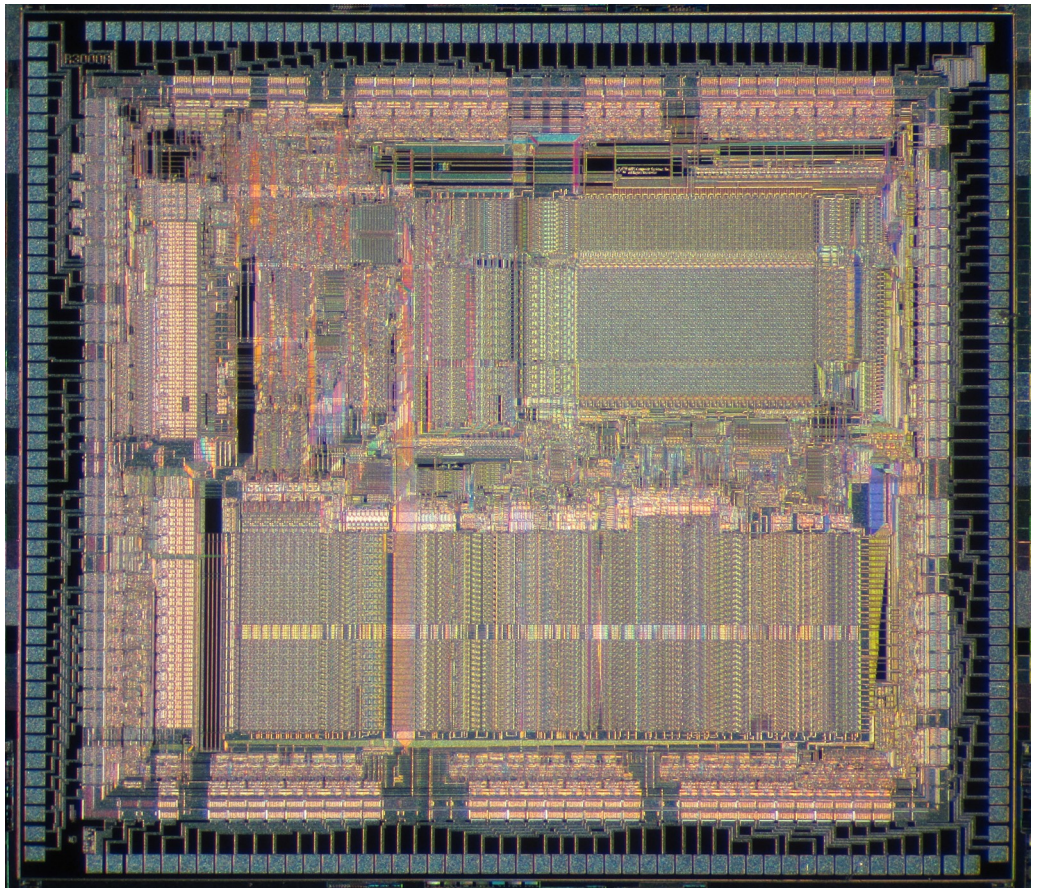




manufactured in the 45th week of 1976

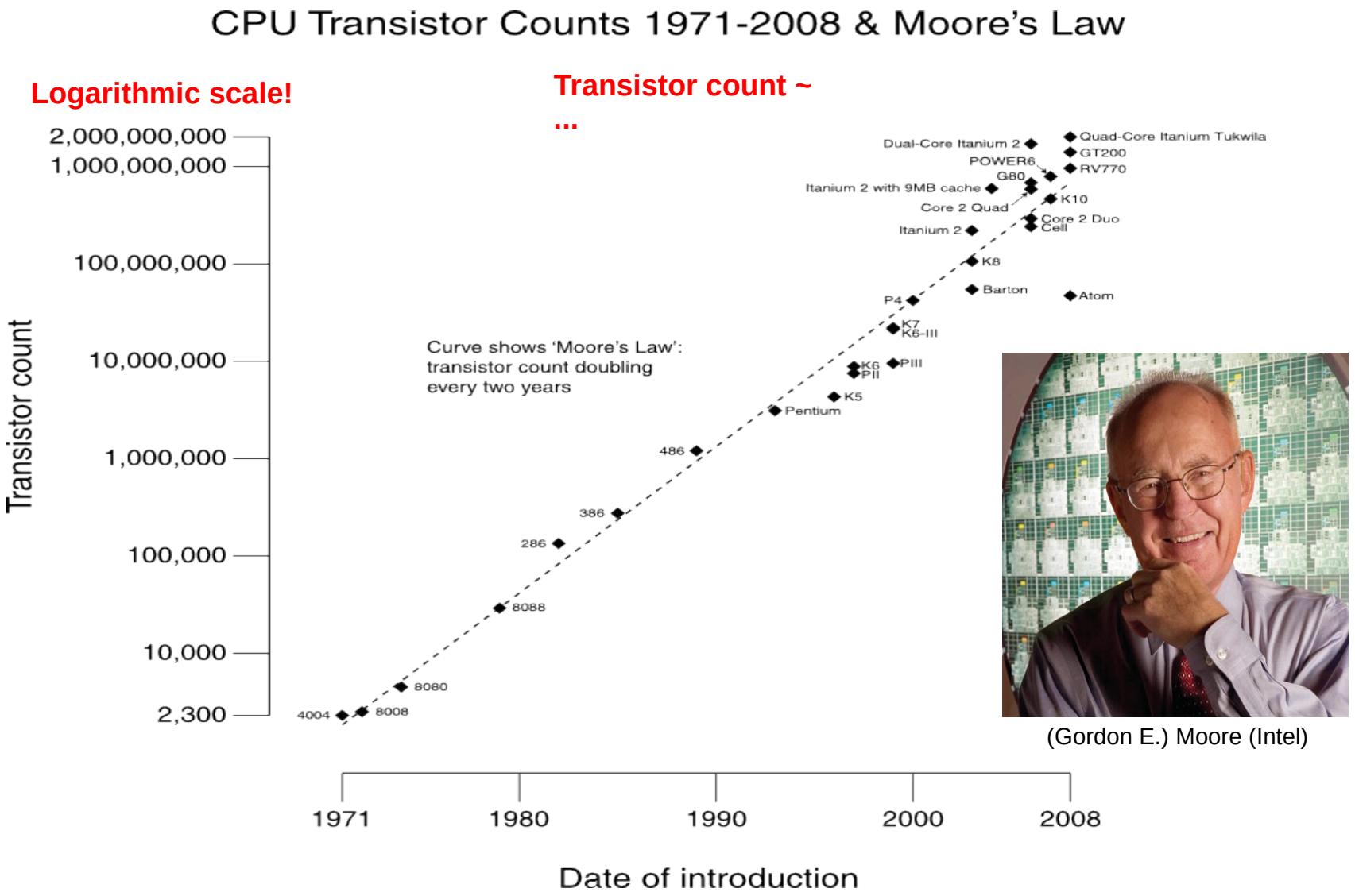
Logische Componenten Implementeren:

32 bit MIPS R3000 processor (115000 transistors)



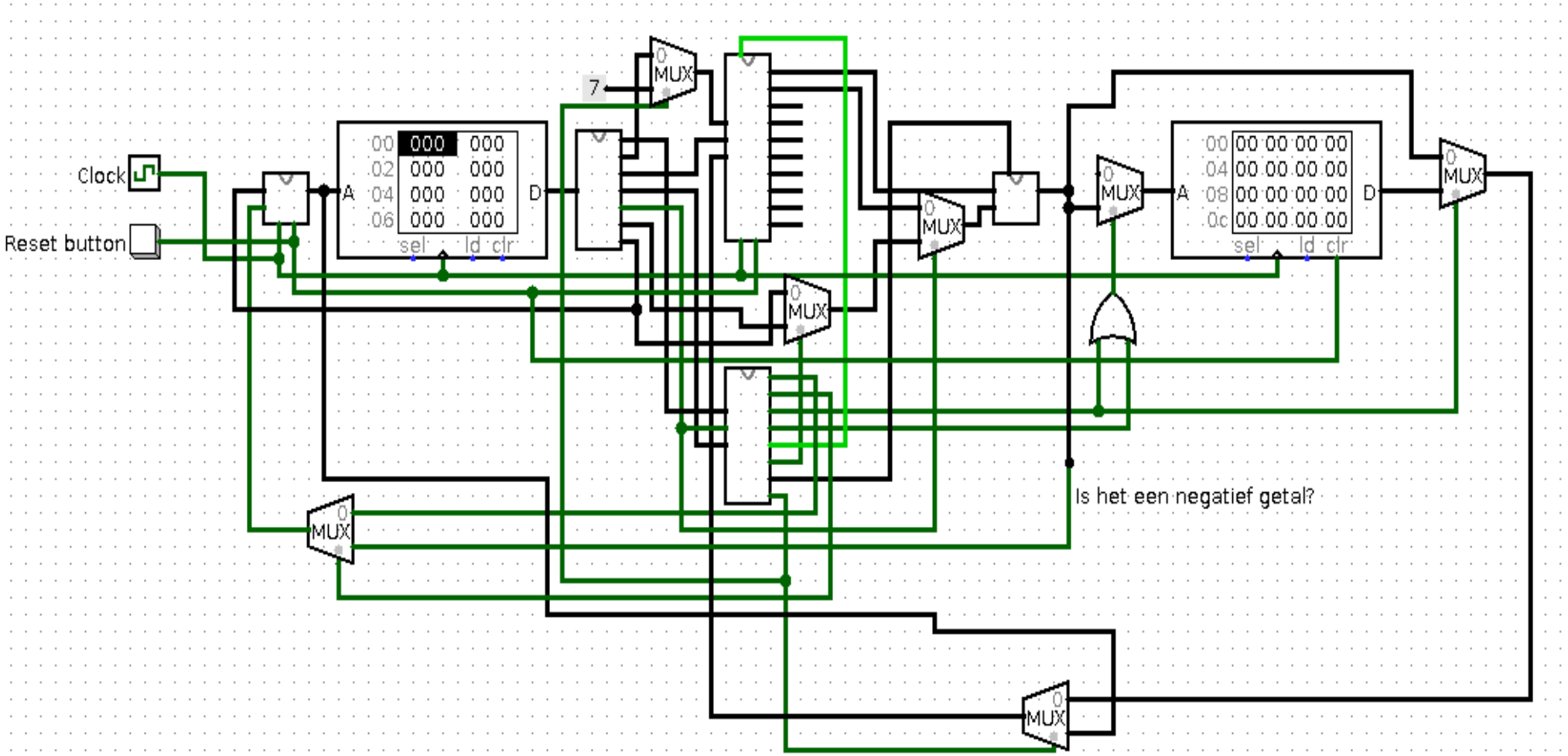
early 1990s

Exponentiële Groei!



(Gordon E.) Moore (Intel)

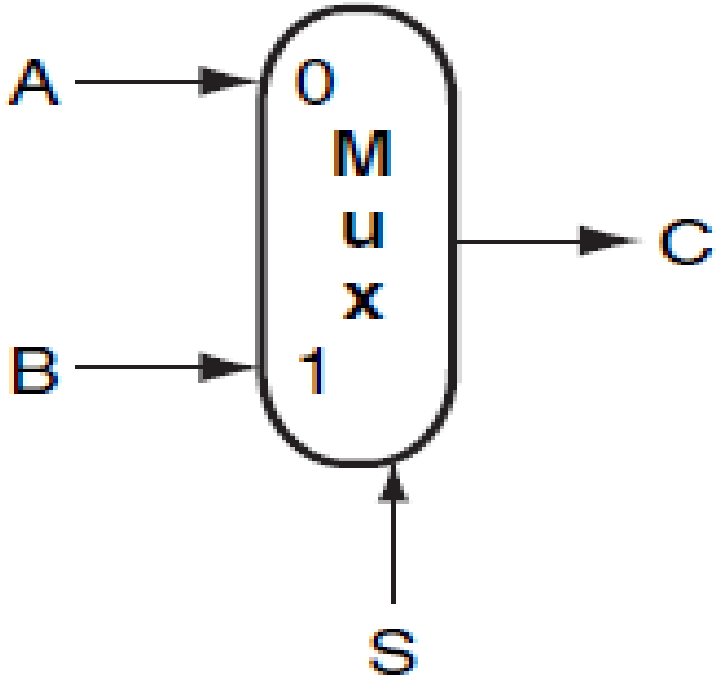
Logische Circuits hiërarchisch ontwerpen
Modelleren en Simuleren



Logisim

`https://sourceforge.net/projects/circuit/`

Multiplexor (1 bit)



specificatie: waarheidstabel

A	B	S	C
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

- Selectie
- Van parallel naar serieel

how many logic gates?
how long does it take?

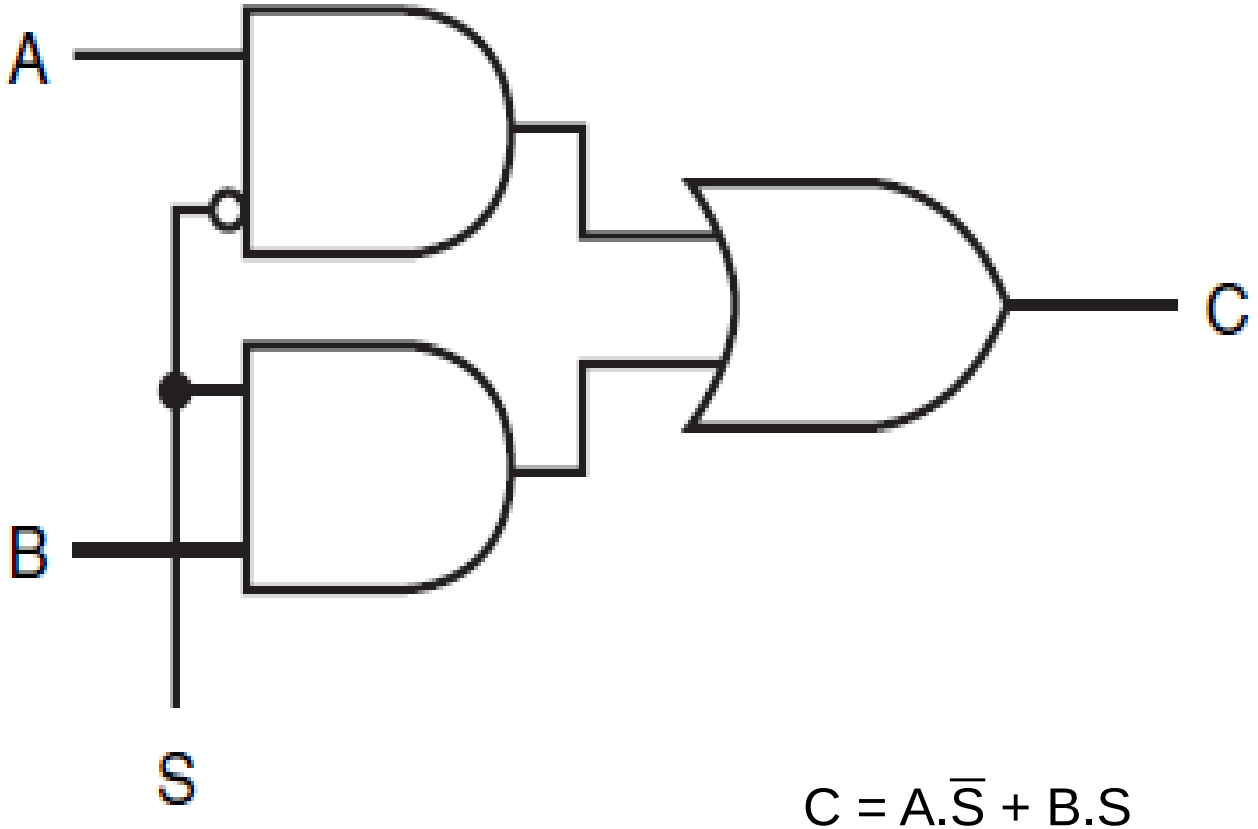
Sum of Products implementatie

A	B	S	C
0	0	0	0
0	1	0	0
1	0	0	**1**
1	1	0	**1**
0	0	1	0
0	1	1	**1**
1	0	1	0
1	1	1	**1**

$$C = A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S$$

how many logic gates?
how long does it take?

“optimale” implementatie



$C = A.\overline{S} + B.S$

how many logic gates?
how long does it take?

Bewijs van equivalentie?

$C = A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S \quad \Leftrightarrow \quad C = A.\overline{S} + B.S$

1. Identieke waarheidstabellen

2. Logica-gebaseerd “symbolisch” bewijs

Bewijs van equivalentie (1.)

$C = A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S \quad \Leftrightarrow \quad C = A.\overline{S} + B.S$

$C1 = A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S$
$C2 = A.\overline{S} + B.S$

C1 == C2 ?

A	B	S	C1	C2
0	0	0	0	0
0	1	0	0	0
1	0	0	**1**	**1**
1	1	0	**1**	**1**
0	0	1	0	0
0	1	1	**1**	**1**
1	0	1	0	0
1	1	1	**1**	**1**

Bewijs van equivalentie (2.)

$C = A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S \quad \Leftrightarrow \quad C = A.\overline{S} + B.S$

Bewijs van equivalentie (2.)?

$C = A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S \quad \Leftrightarrow \quad C = A.\overline{S} + B.S$

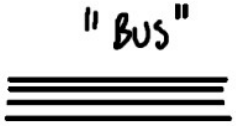
C	$= A.\overline{B}.\overline{S} + A.B.\overline{S} + \overline{A}.B.S + A.B.S$	$\Rightarrow$ (distributiviteit . +)
	$= (A.\overline{B} + A.B).\overline{S} + (\overline{A}.B + A.B).S$	$\Rightarrow$ (distributiviteit . +)
	$= A.(\overline{B} + B).\overline{S} + (\overline{A} + A).B.S$	$\Rightarrow$ (+ inverse “uitgesloten derde”)
	$= A.1.\overline{S} + 1.B.S$	$\Rightarrow$ (associativiteit .)
	$= (A.1).\overline{S} + (1.B).S$	$\Rightarrow$ (eenheidselement voor .)
	$= A.\overline{S} + B.S$	**q.e.d.** □

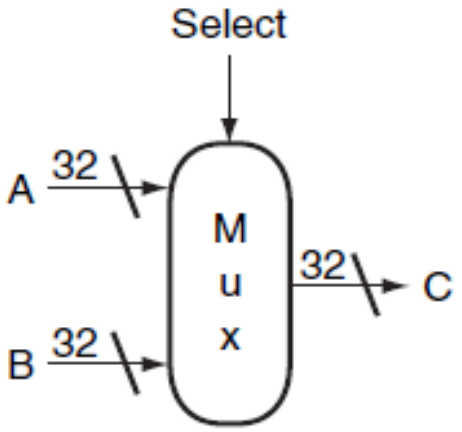
8. Sinus rectus peripheriæ, & complementi ſui æquepoſſunt radio.

Complementum peripheriæ dicimus reliquam peripheriam datæ ad circuli quadrantem. Sit igitur in præmiſſa figura, rectæ E F, *ſinus rectus peripheriæ* G E *vel* B C E: *& complementi ſui* C E *ſinus rectus* E D, *vel æqualis illi* A F *per trigeſimam quartam primi elementorum. Dico* A F *&* E F, *æquepoſſe radio* A E. *Nam per penultimam primi Euclidis, in Triangulis rectangulis, quadrata laterum rectum angulum continentium, æqualia ſunt lateri rectum angulum ſubtendenti. Sed* A F E *eſt Triangulum rectangulum ad* F *per ſeptimam hujus, crura verò rectum ambientia ſunt* A F *&* E F: *æquepoſſunt ergo radio* A E *rectum angulum ſubtendenti; quod erat demonſtrandum.*

Philippe van Lansberge in 1604

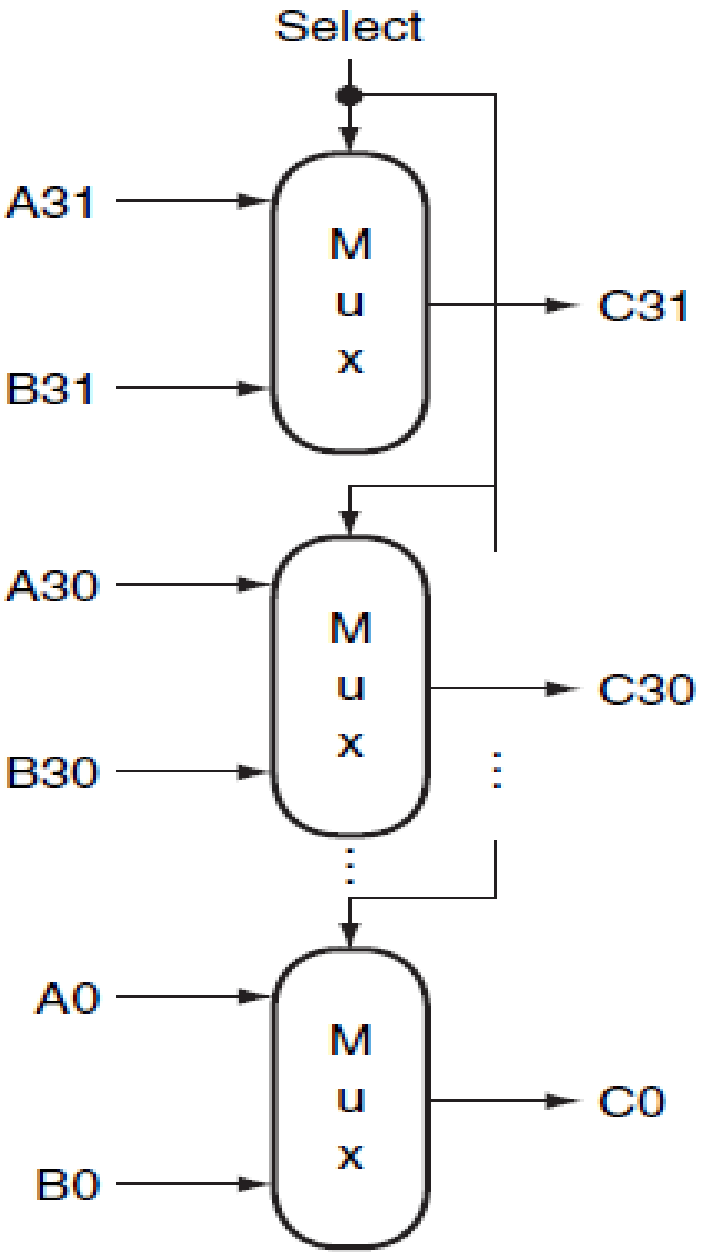
Multiplexor (32 bit), “bus”





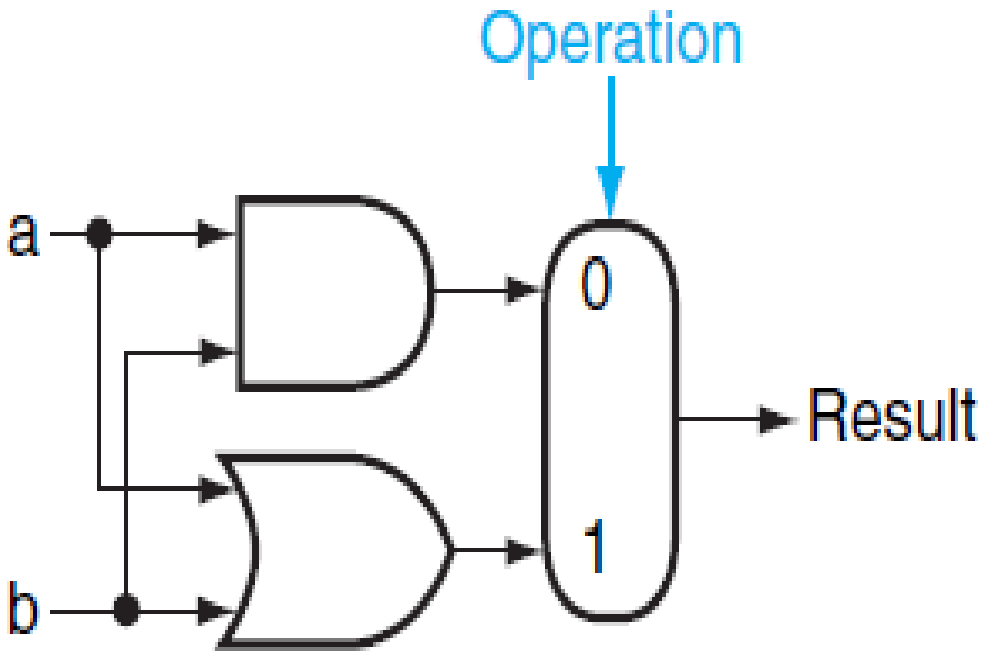
how many logic gates?
how long does it take?

Multiplexor (32 bit) implementatie

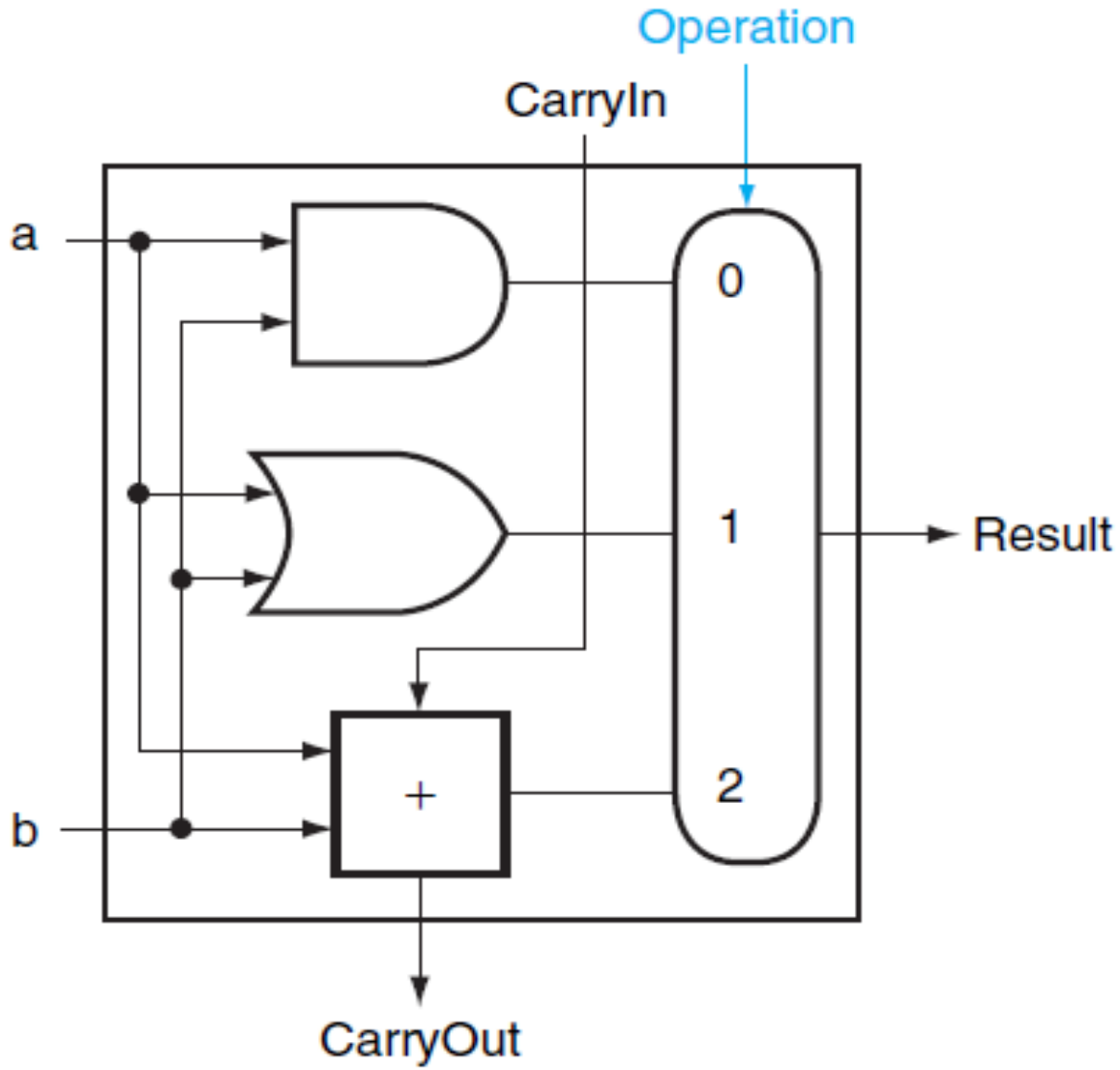


how many logic gates?
how long does it take?

1-bit AND, OR



1-bit ALU (AND, OR, +)



Binaire voorstelling/codering van gehele getallen

n-bit string "$x_{n-1}x_{n-2}\ldots x_1x_0$" has/encodes **value** x

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

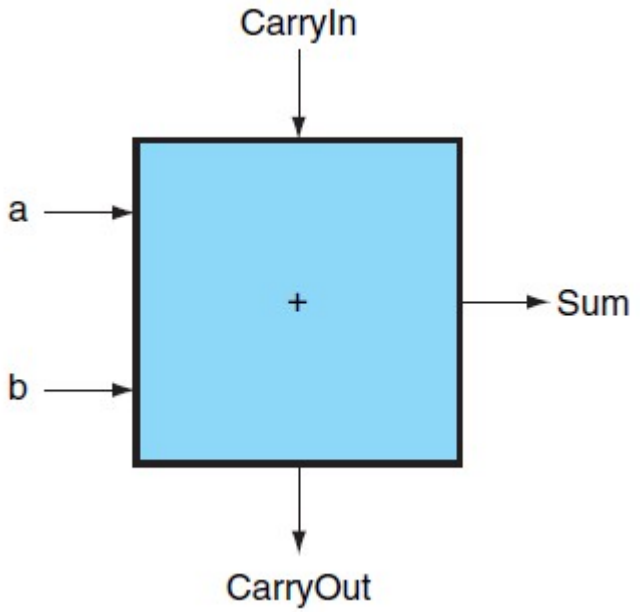
x_0 Least Significant bit (**LSb**)

x_{n-1} Most Significant bit (**MSb**)

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$


10100010010100010101
11101100101100101010
0101 THERE ARE 101
001011 ONLY 10 10010
10010 TYPES OF 0010
101110 PEOPLE: 01011
0010 THOSE WHO 000
100 UNDERSTAND 011
110101 BINARY 10101
0001 AND THOSE 000
1010 WHO DON'T 000
10101110110011010101
11001010101010101000
10100100101001010000
10101110110011010101

1-bit adder

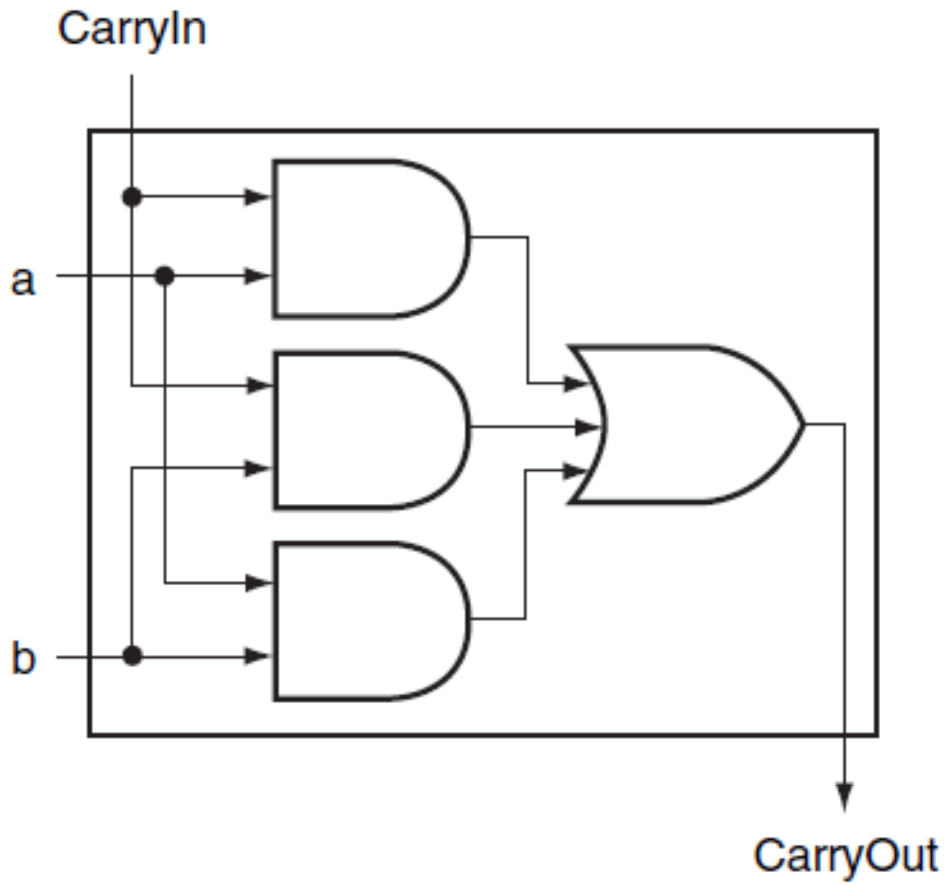


Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

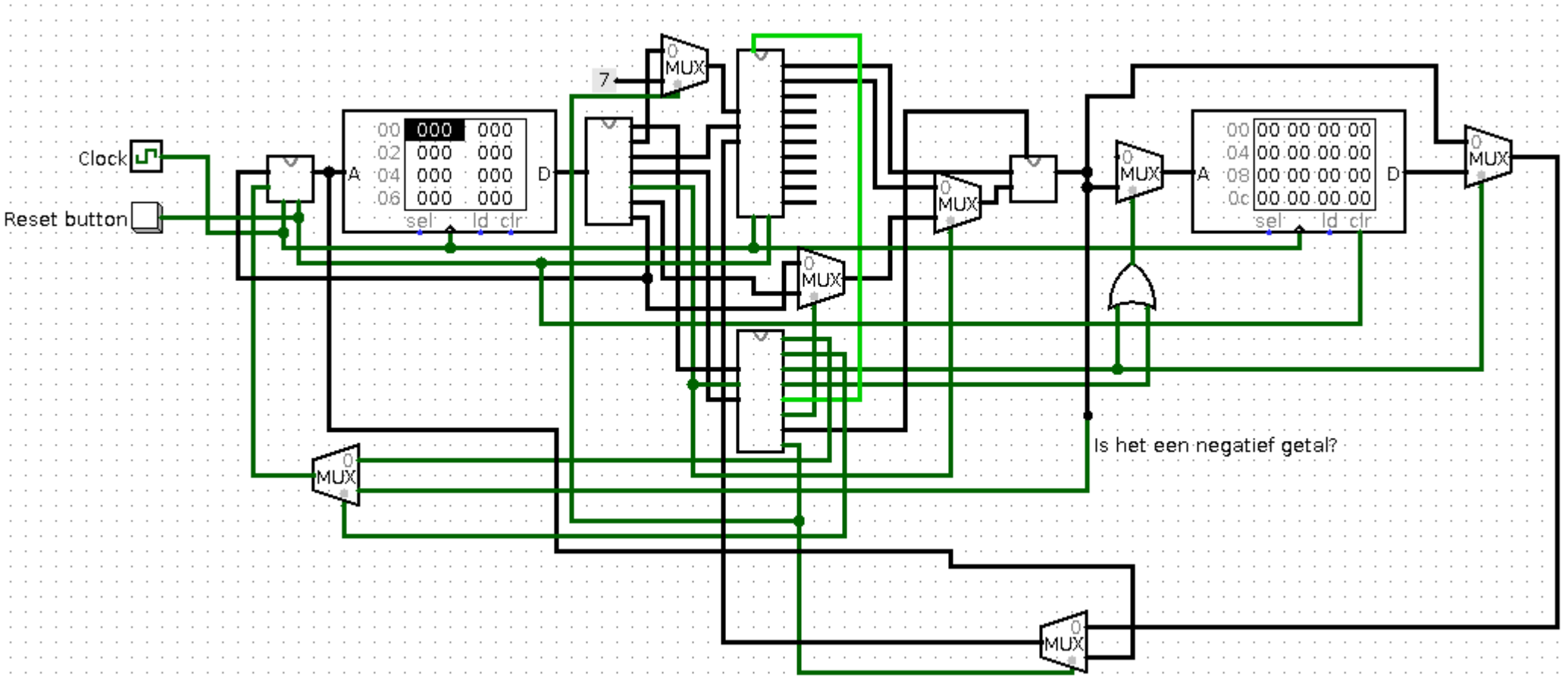
CarryOut

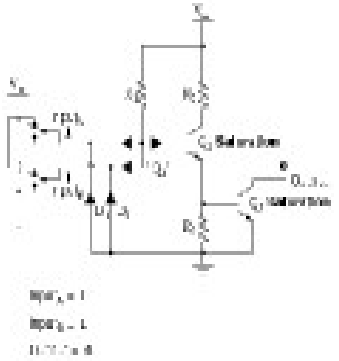
Inputs		
a	b	CarryIn
0	1	1
1	0	1
1	1	0
1	1	1

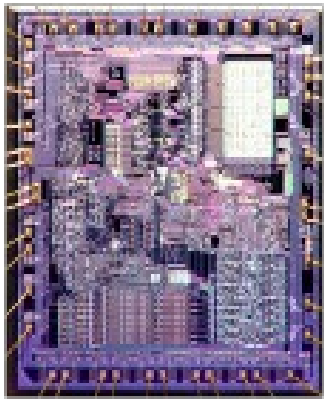
CarryOut = (b.CarryIn) + (a.CarryIn) + (a.b) + **(a.b.CarryIn)**
= (b.CarryIn) + (a.CarryIn) + (a.b)



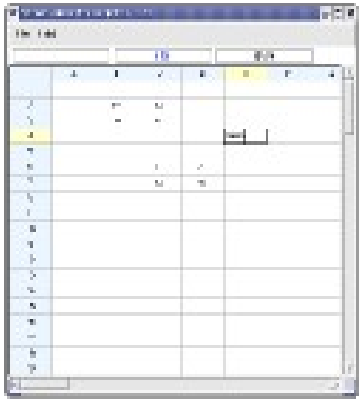
Vanaf nu: “only connect”







```
strcpy:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
    add  $s0, $zero, $zero
L1: add  $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```

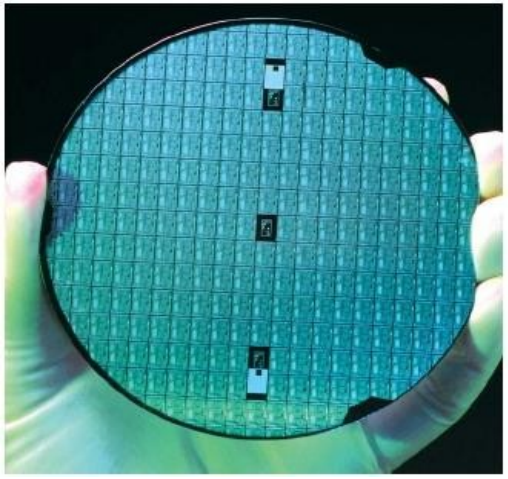


physics digital electronics

computer architecture / systems

HL progr. languages operating systems

complex SW applications



```
def visitFunction(self, function):
  if typeChecker.debug: typeChecker.typeCheck([function], [Function])
  numArg=0
  for argument in function.getArgs():
    if isinstance(argument, RangeRef):
      numArg += len(argument.getCellRefSet())
    else:
      numArg += 1
    argument.accept(self)

  args=self.__evalStack[-numArg:]
  self.__evalStack=self.__evalStack[0:-numArg]


  if len(args)>1 and self.__checkValueError(args):
    self.__evalStack.append(0)
    return

  execStr="answer = "+function.getName()+"("+str(args)+")"
  try:
    exec execStr
  except NameError, n:
    fName=split(n[0],"'")
    self.__nameError=True
    self.__nameErrorStr=fName[1]
    self.__evalStack.append(0)
    return
  self.__evalStack.append(answer)
```