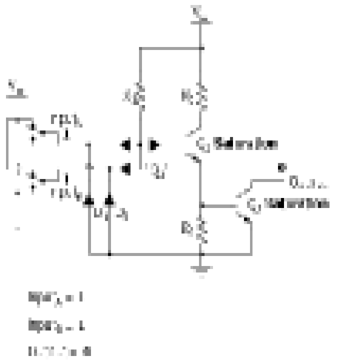
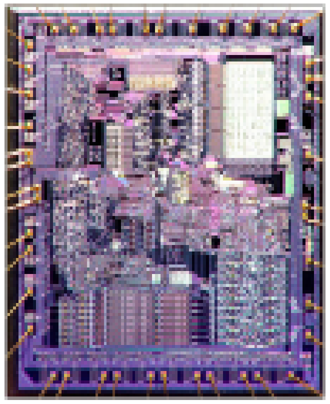
Computer Systemen en Computer Architectuur

Dinsdag 20 februari 2023 – 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

Systemen/Engineering

Software ontwikkeling

Theoretische informatica

Wiskunde

Eindwerk

Distributed Systems

Datastructuren en Graafalgoritmes

Numerieke lineaire algebra

Software Engineering

Artificiele Intelligentie

Programming Project databases

Netwerken

Compilers

Algoritmes en Complexiteit

Elementaire Statistiek

Numerieke Analyse

Operating Systems

Advanced Programming

Inleiding databases

Machines en Berekenbaarheid

Lineaire Algebra

Project Software Engineering

Computer Graphics

Talen en automaten

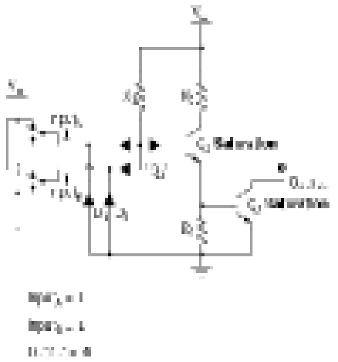
Calculus

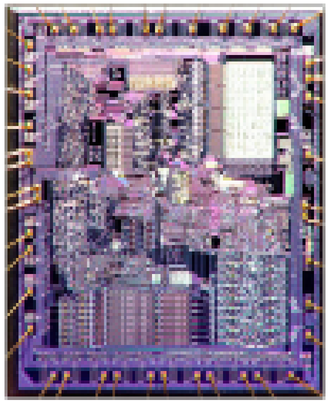
Gegevens-abstractie en data structuren

Computer Systemen en Architectuur

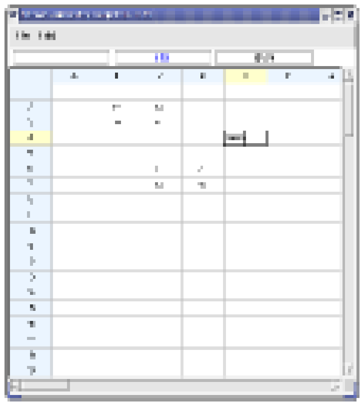
Inleiding programmeren

Discrete wiskunde





```
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
```

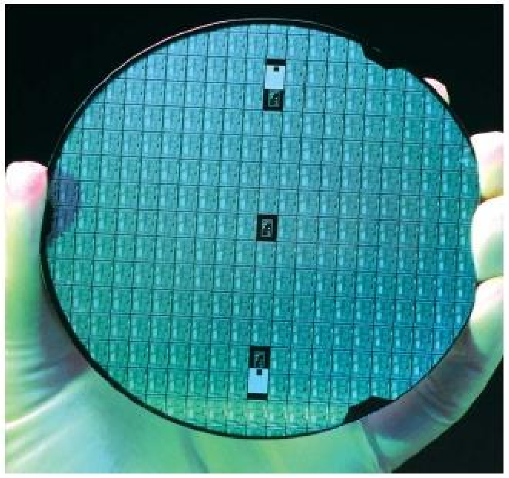


physics digital electronics

computer architecture / systems

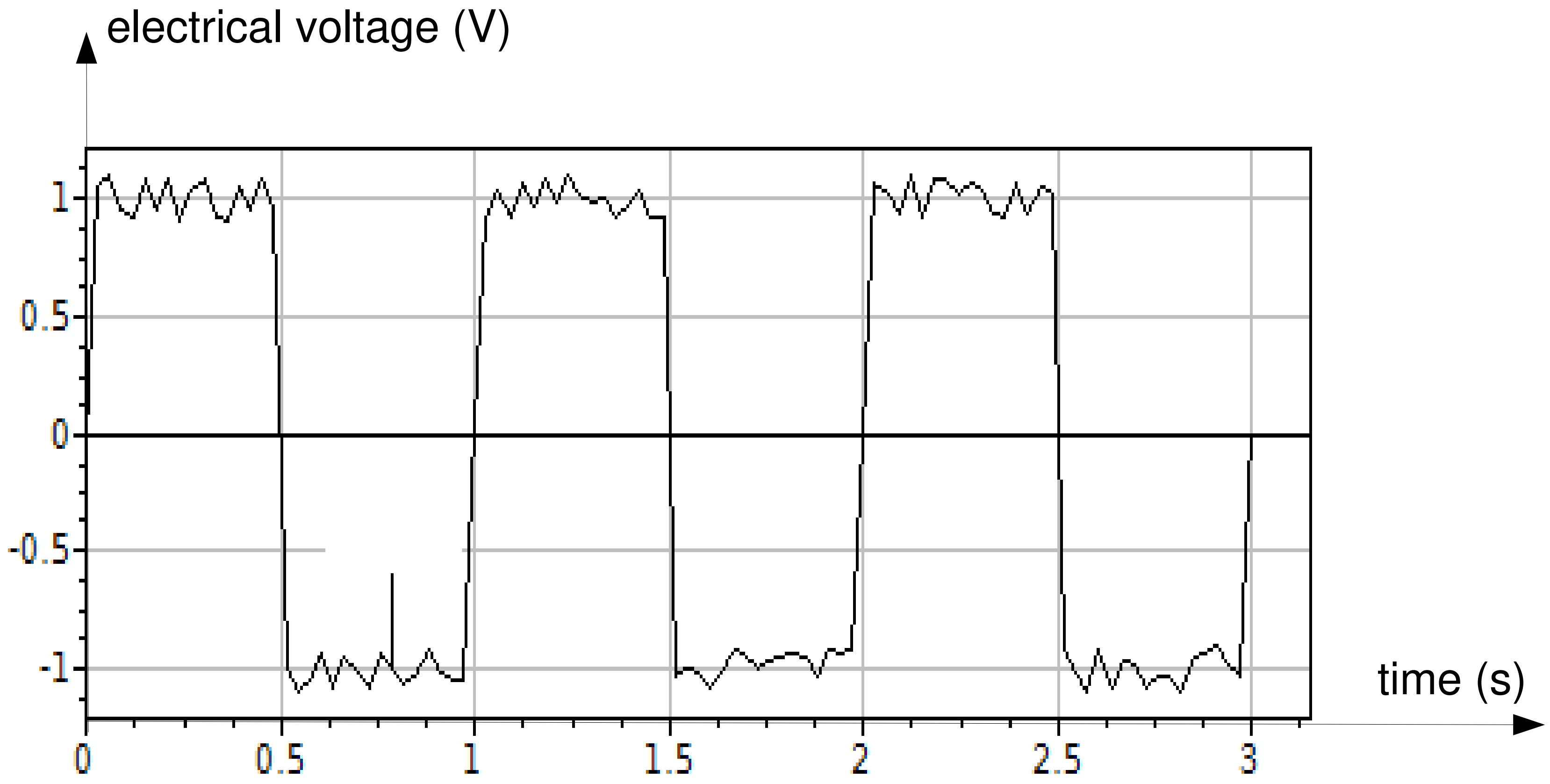
HL progr. languages operating systems

complex SW applications

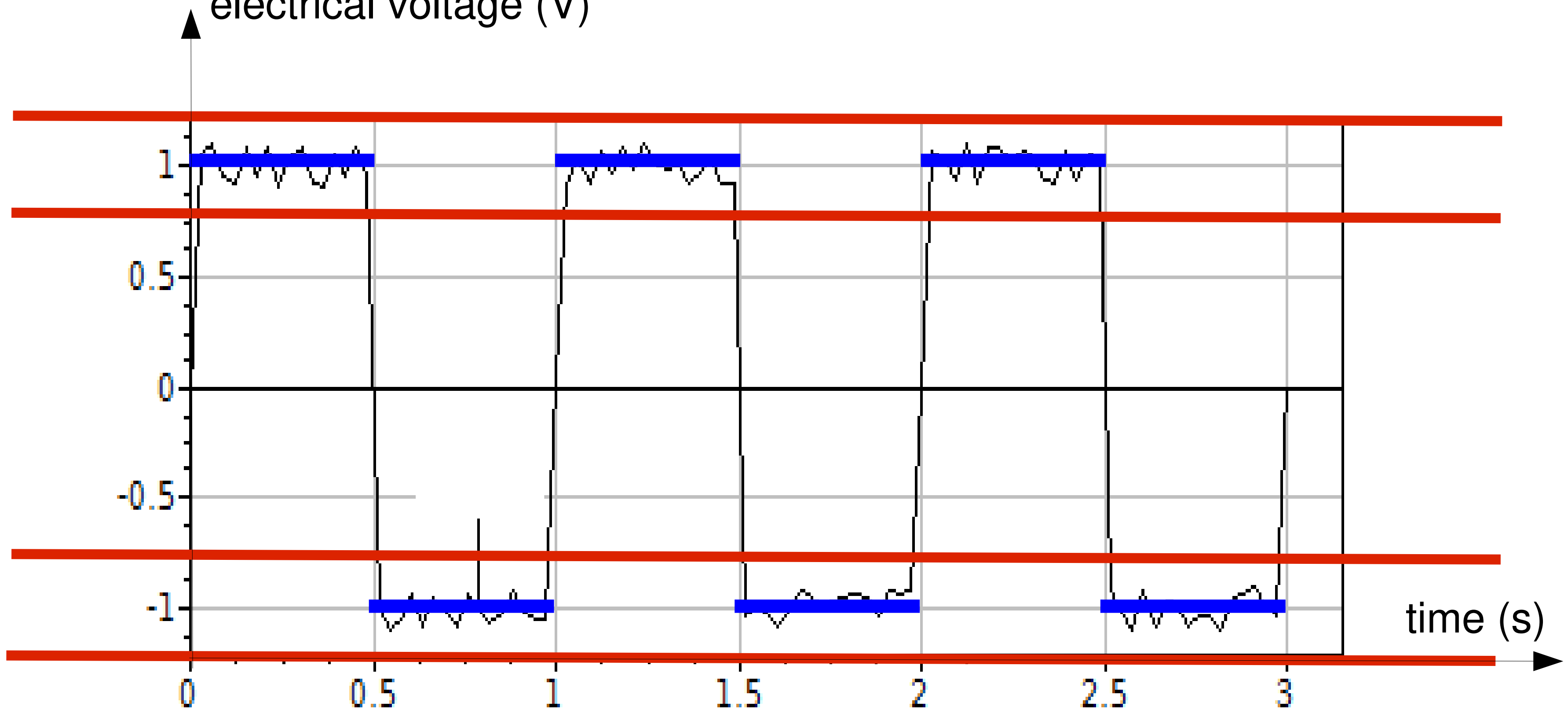


```
def visitFunction(self, function):
  if typeChecker.debug: typeChecker.typeCheck([function], [Function])
  numArg=0
  for argument in function.getArgs():
    if isinstance(argument, RangeRef):
      numArg += len(argument.getCellRefSet())
    else:
      numArg += 1
    argument.accept(self)

  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)
```

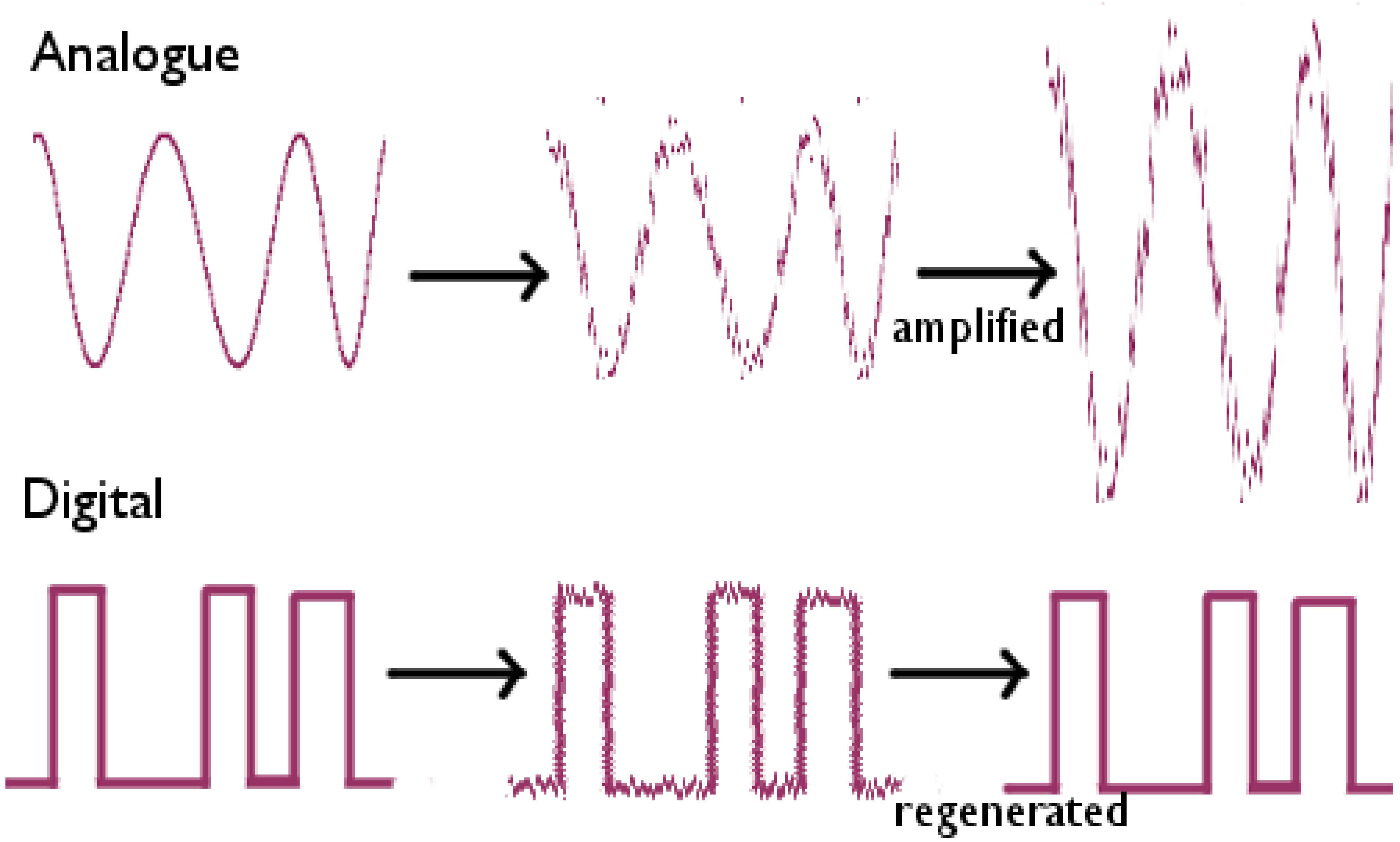
van Analoog ...



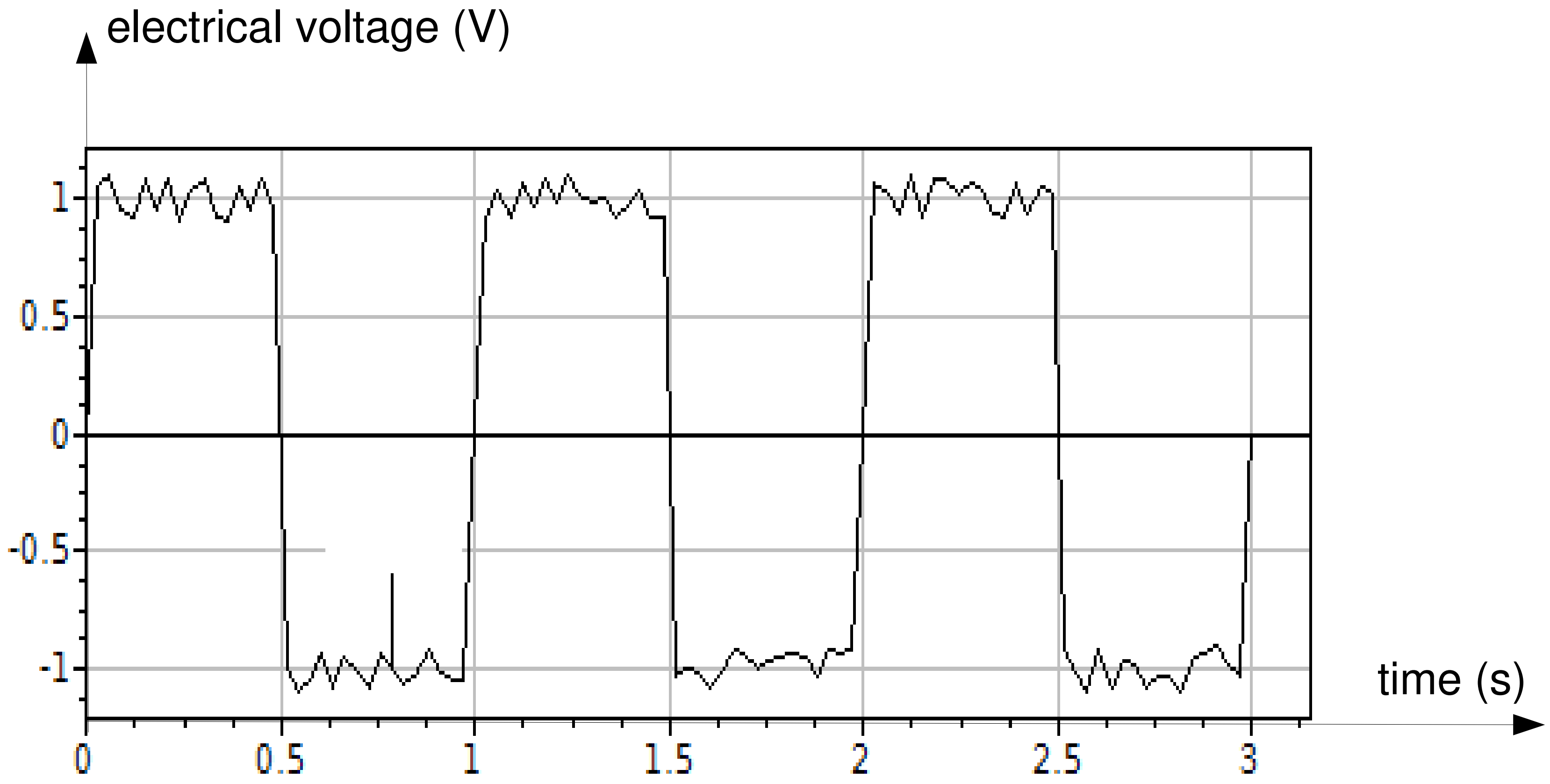
… naar Digitaal



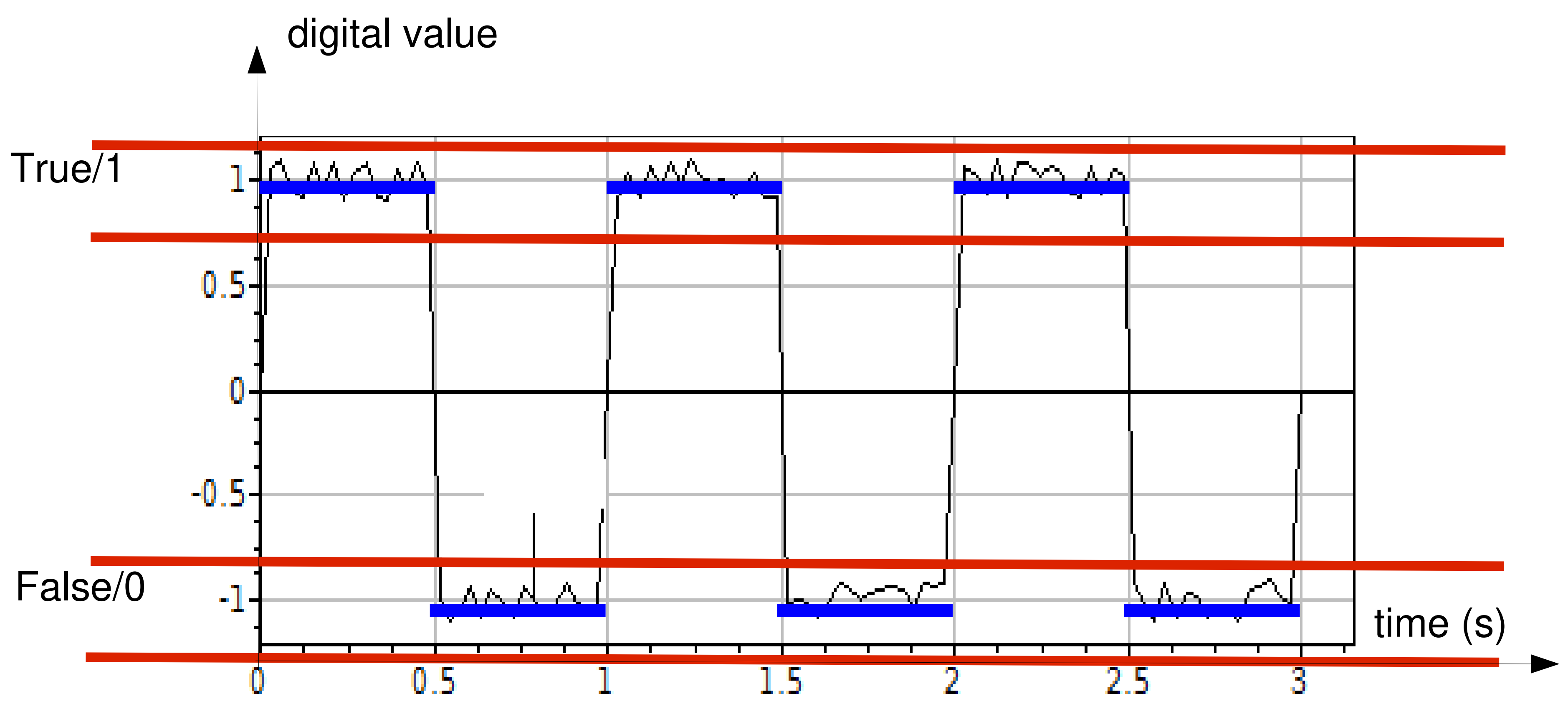
Analoog vs. Digitaal



van Analoog ...



… naar Digitaal



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

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

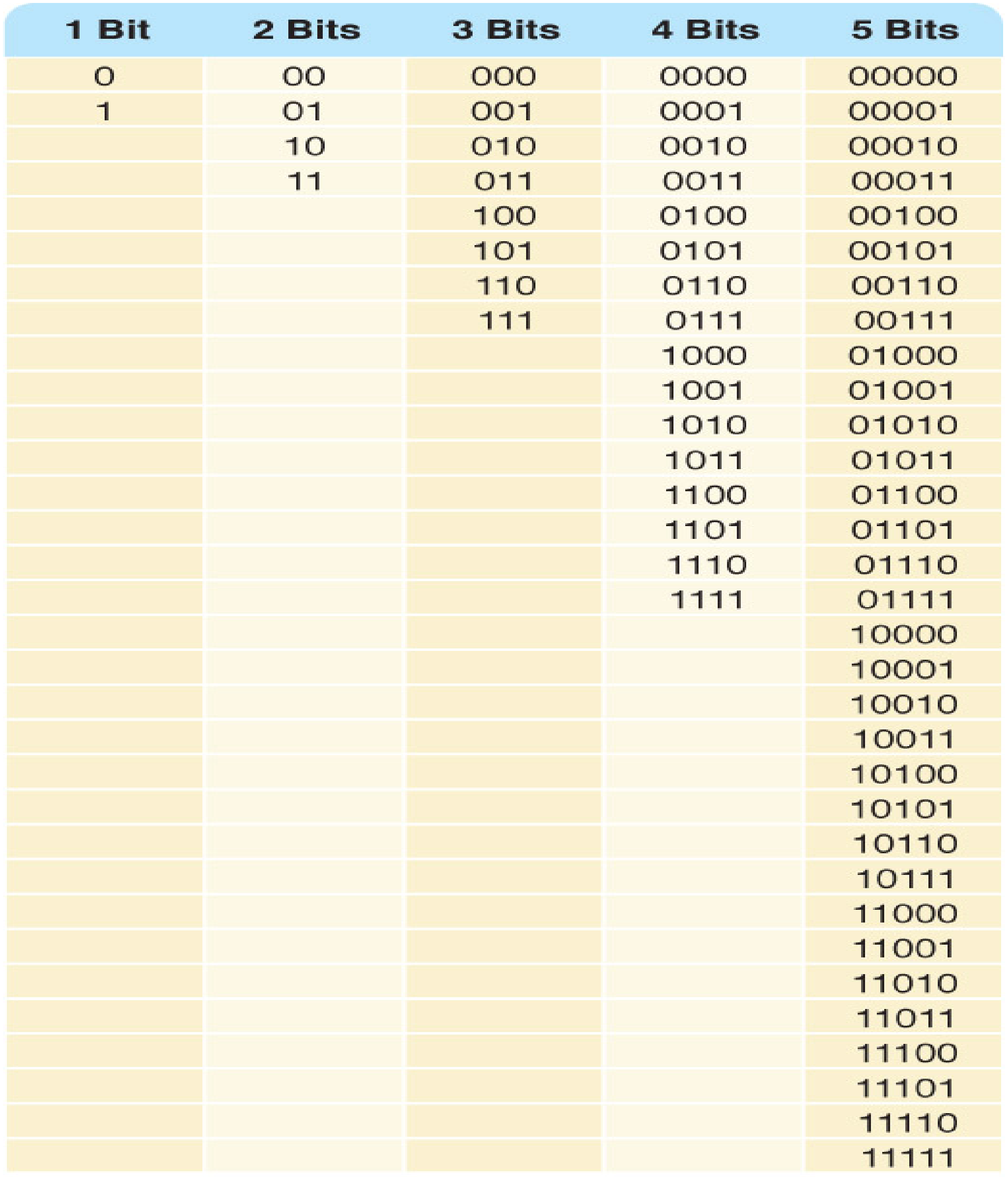
Met 1 bit, kan 2 **verschillende entiteiten voorstellen**

Met 2 bits, kan 4 verschillende entiteiten voorstellen

...

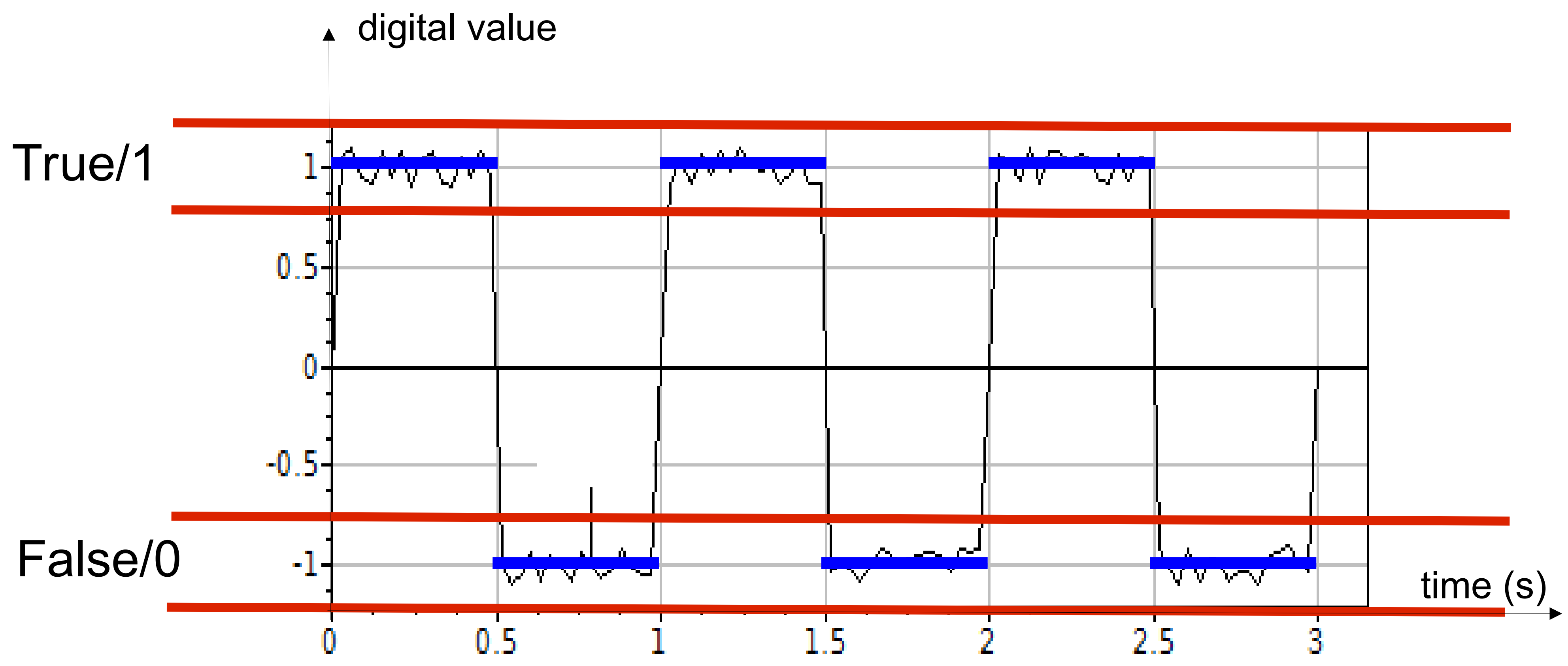
Met **N** bits, kan **2^N** verschillende entiteiten voorstellen

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



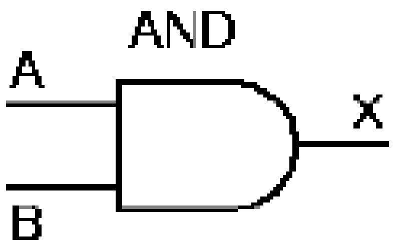
1 Bit	2 Bits	3 Bits	4 Bits	5 Bits
0	00	000	0000	00000
1	01	001	0001	00001
	10	010	0010	00010
	11	011	0011	00011
		100	0100	00100
		101	0101	00101
		110	0110	00110
		111	0111	00111
			1000	01000
			1001	01001
			1010	01010
			1011	01011
			1100	01100
			1101	01101
			1110	01110
			1111	01111
				10000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

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

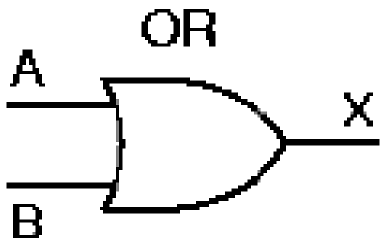


logical 0/1 – Boolean True/False – asserted/de-asserted – high/low

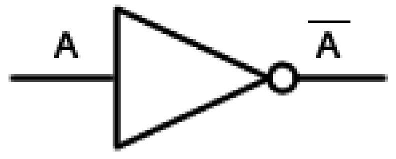
Universele Basis van bewerkingen: Logische Componenten



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



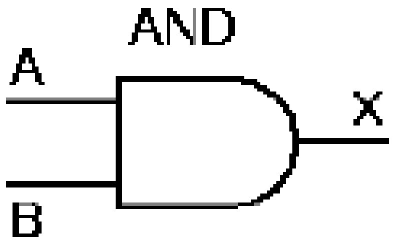
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



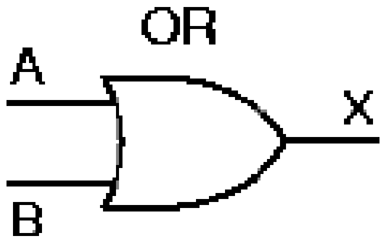
(a)

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

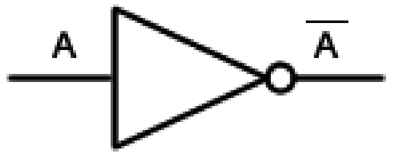
Universele Basis van bewerkingen: Logische Componenten



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

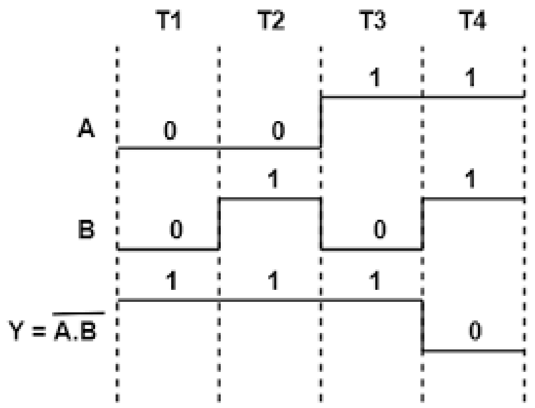


A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



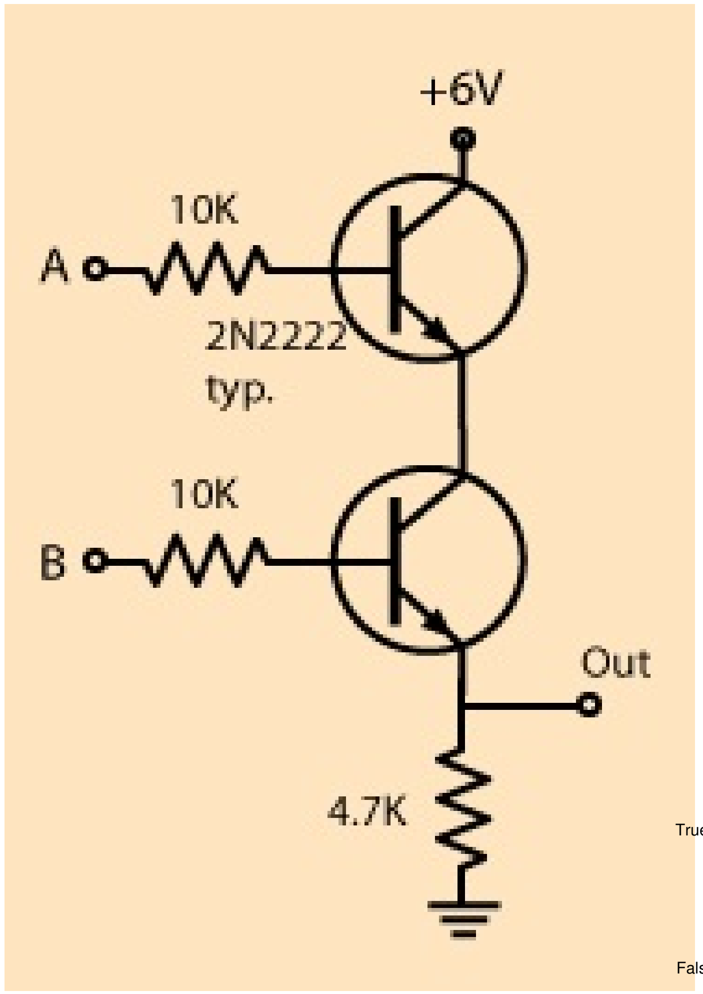
(a)

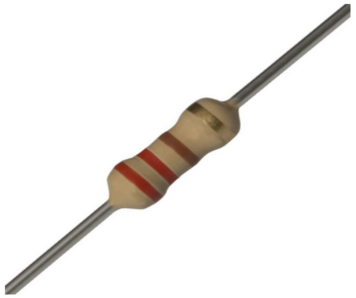
A	$\overline{A}$
0	1
1	0

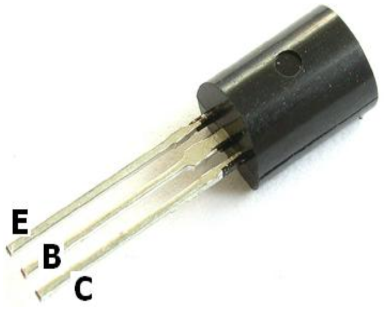


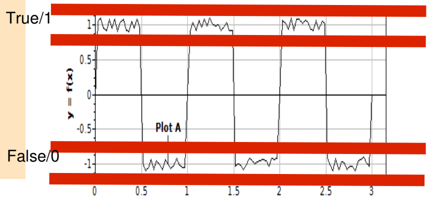
Universele Basis implementatie: Logische Componenten in electronica

AND

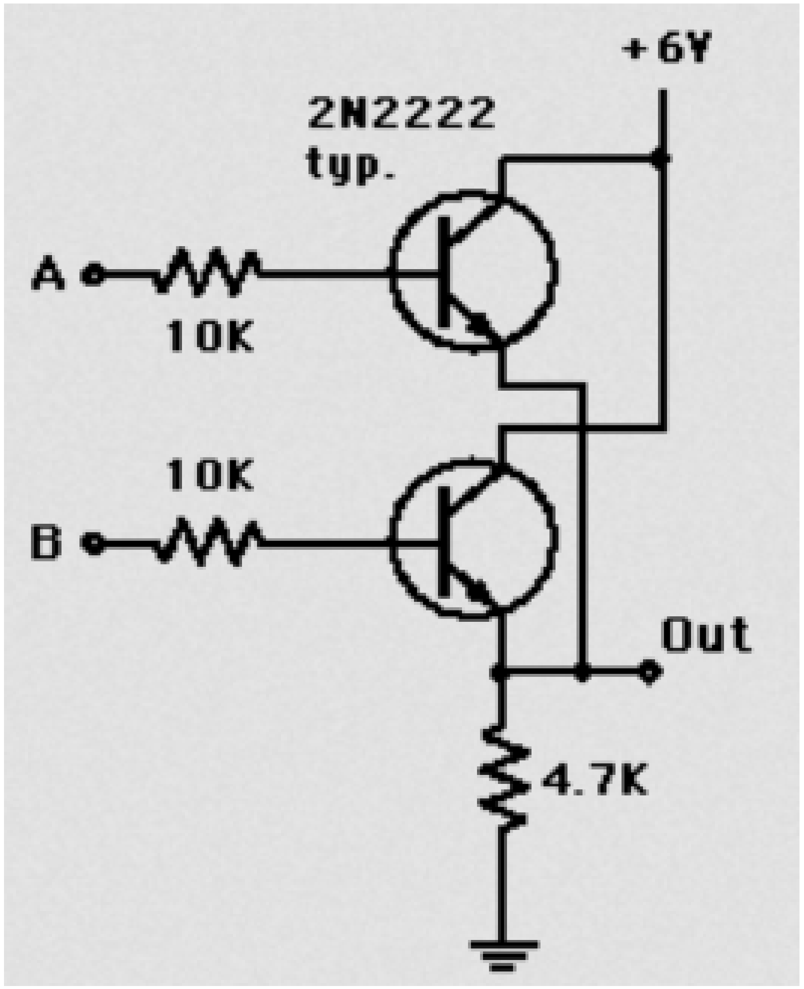






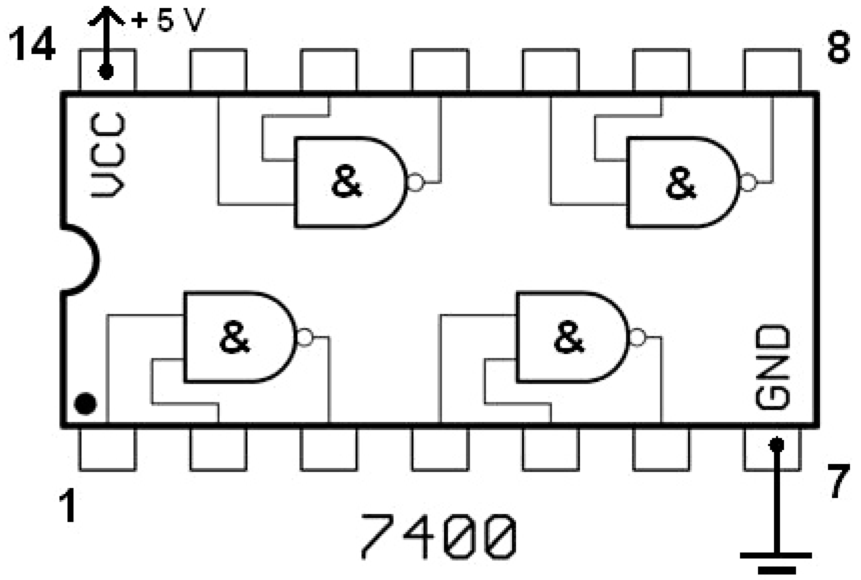


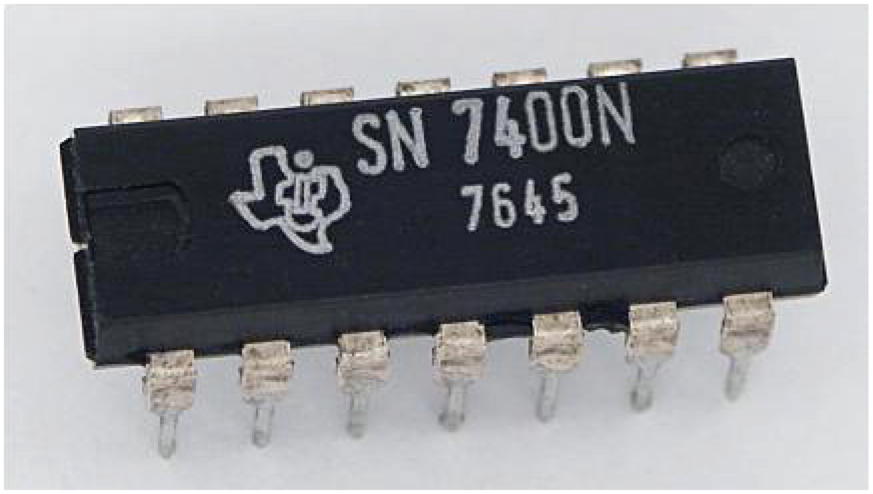
OR



Implementing Logic Components:

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

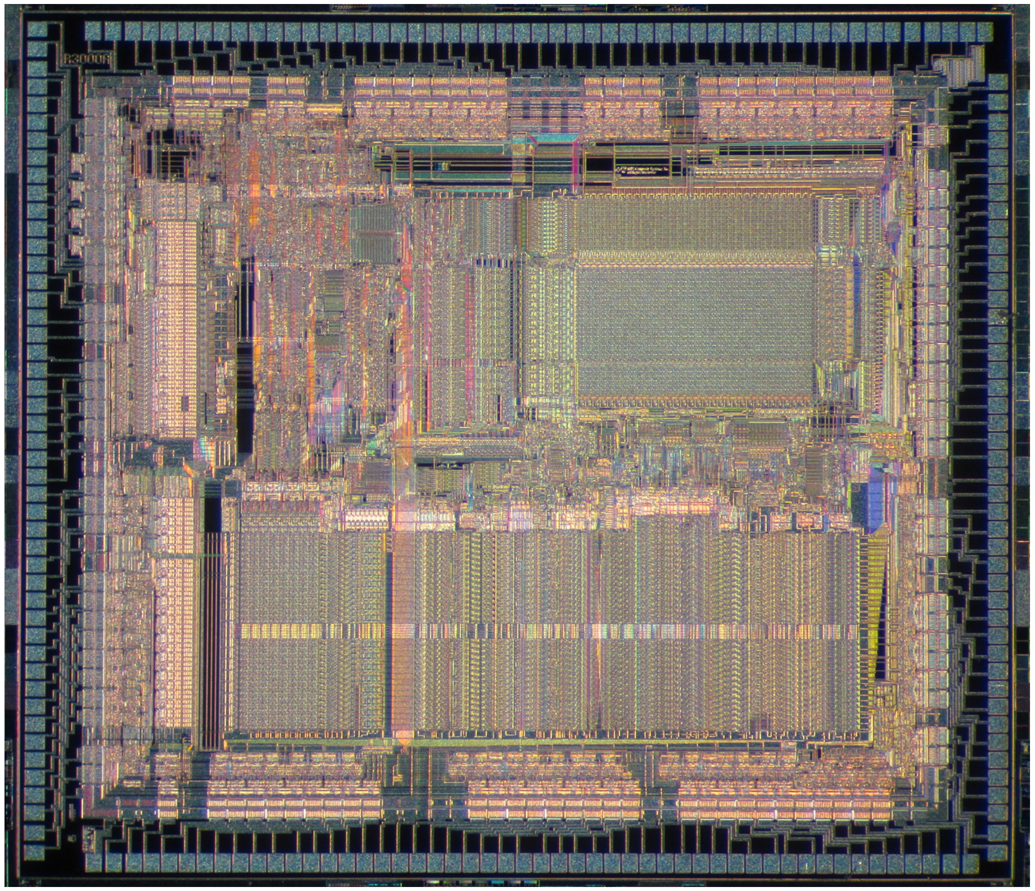




manufactured in the 45th week of 1976

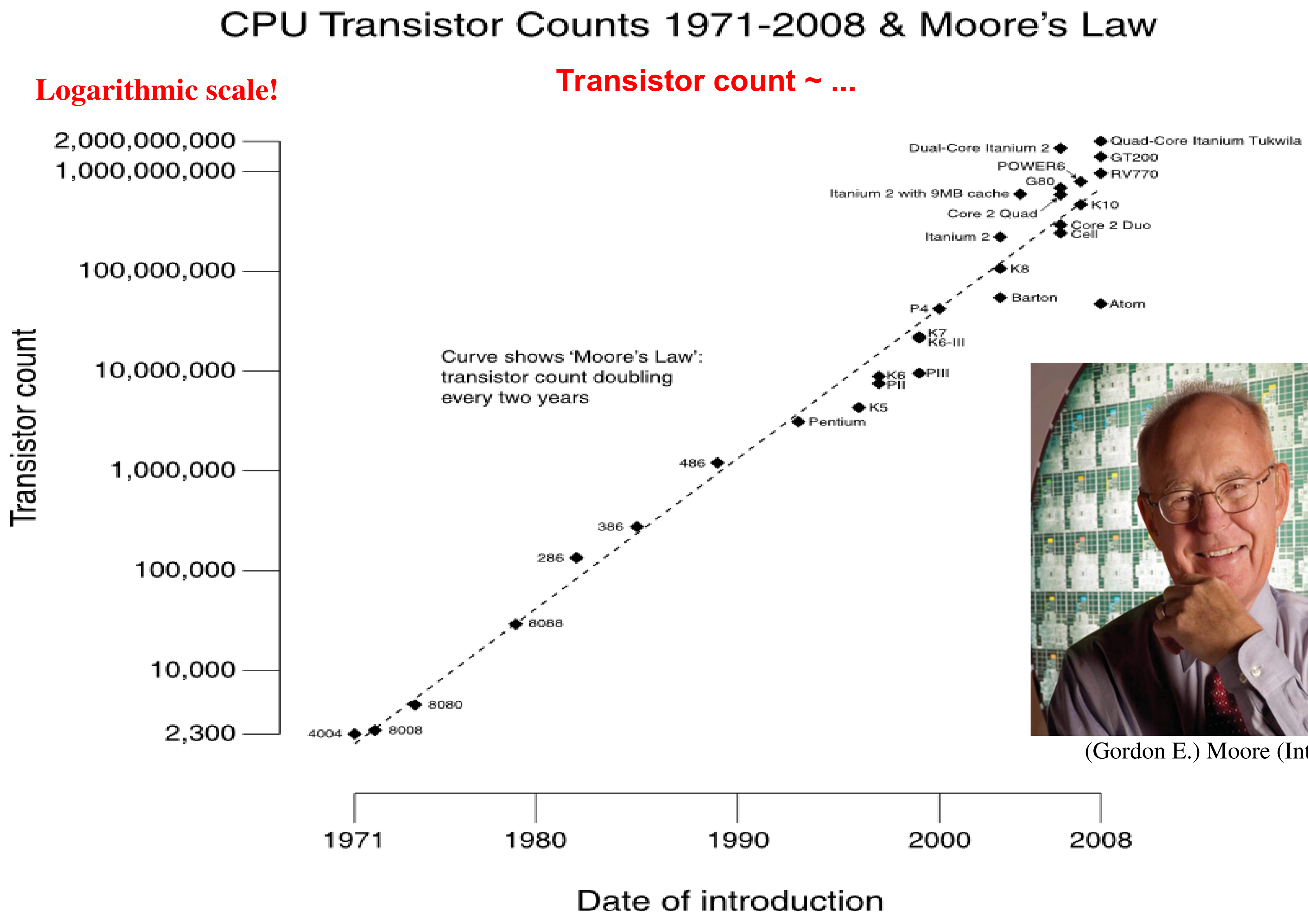
Implementing Logic Components:

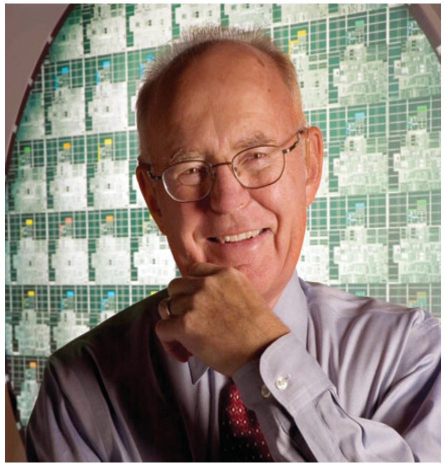
32 bit MIPS R3000 processor (115000 transistors)



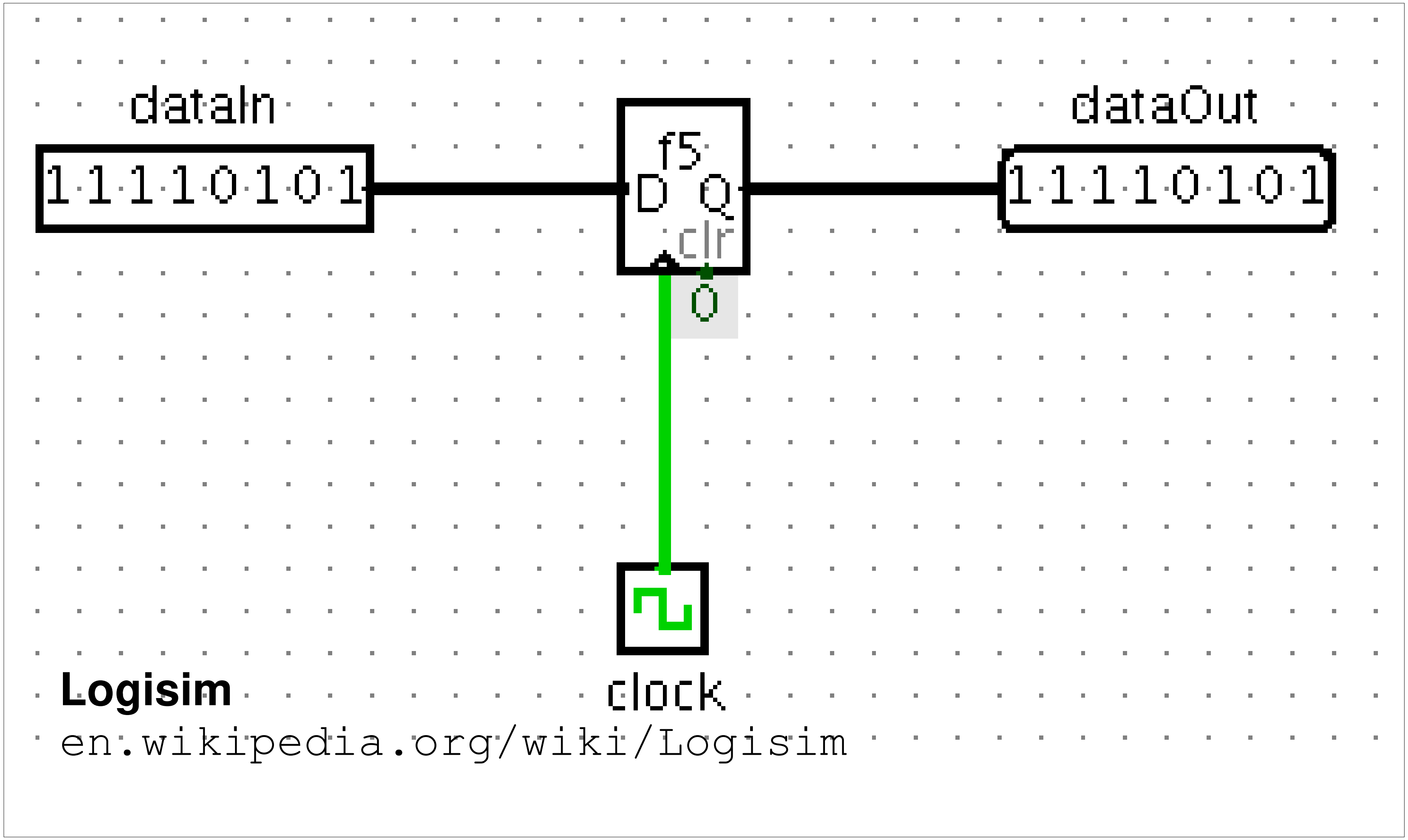
early 1990s

Exponentiele Groei!



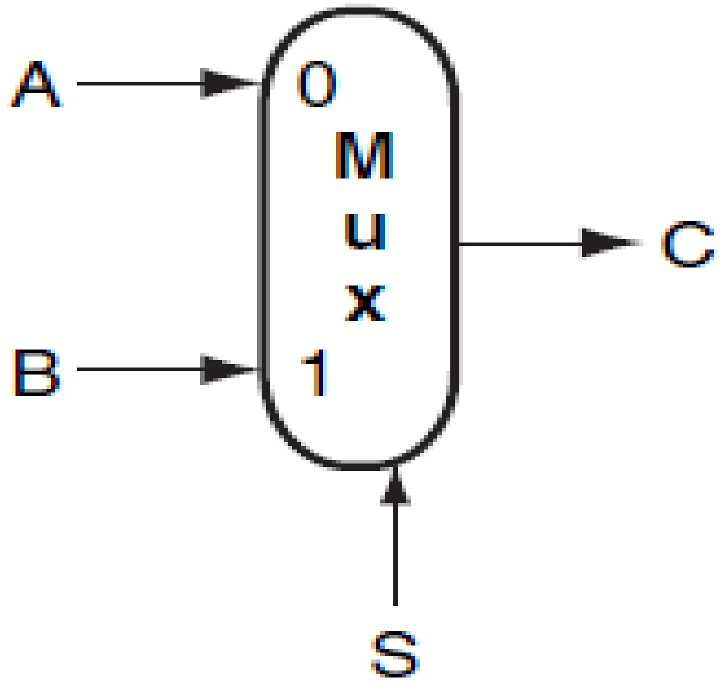

(Gordon E.) Moore (Intel)

Logische Circuits Simuleren



Logisim
en.wikipedia.org/wiki/Logisim

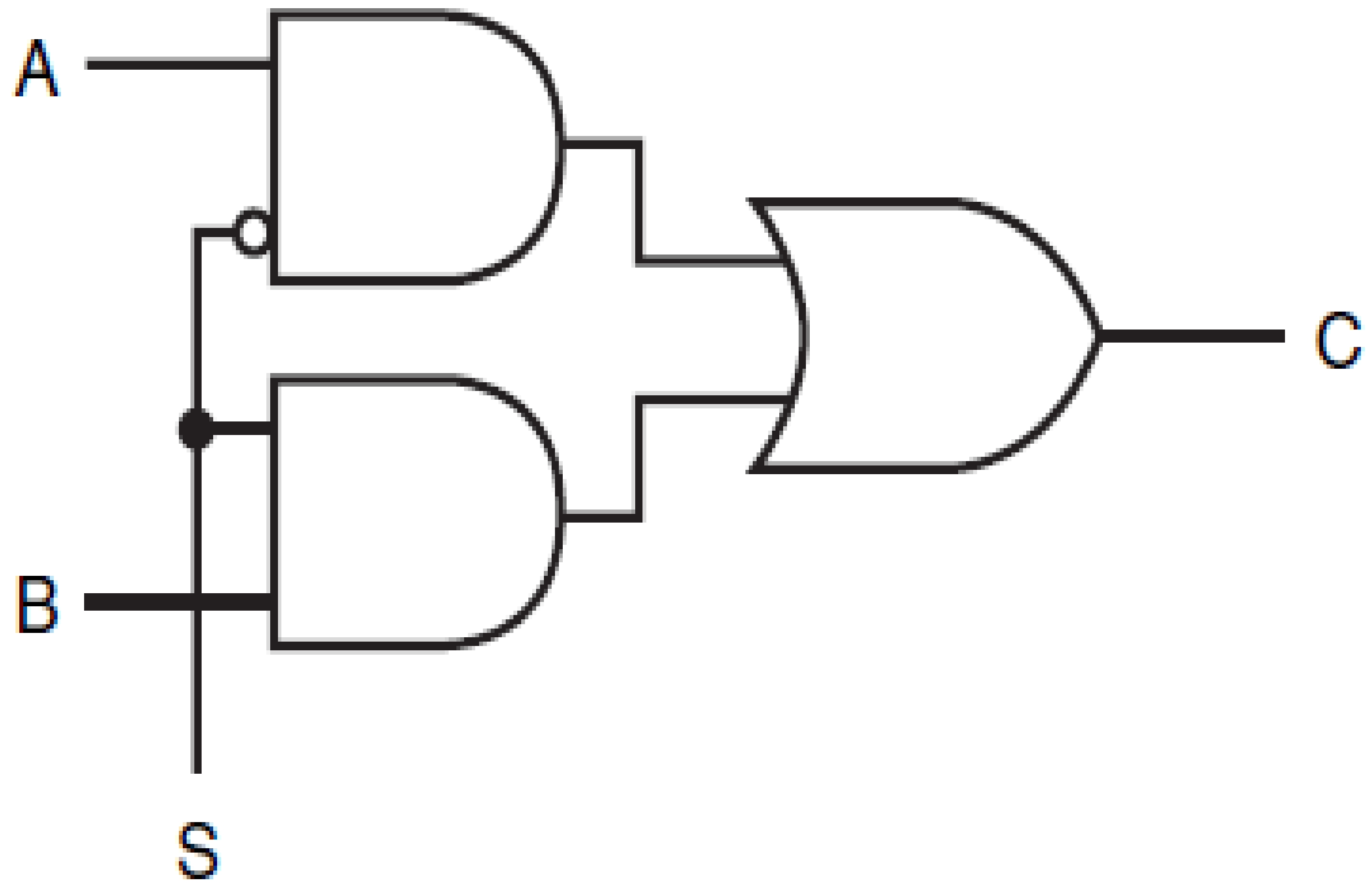
Multiplexor (1 bit)



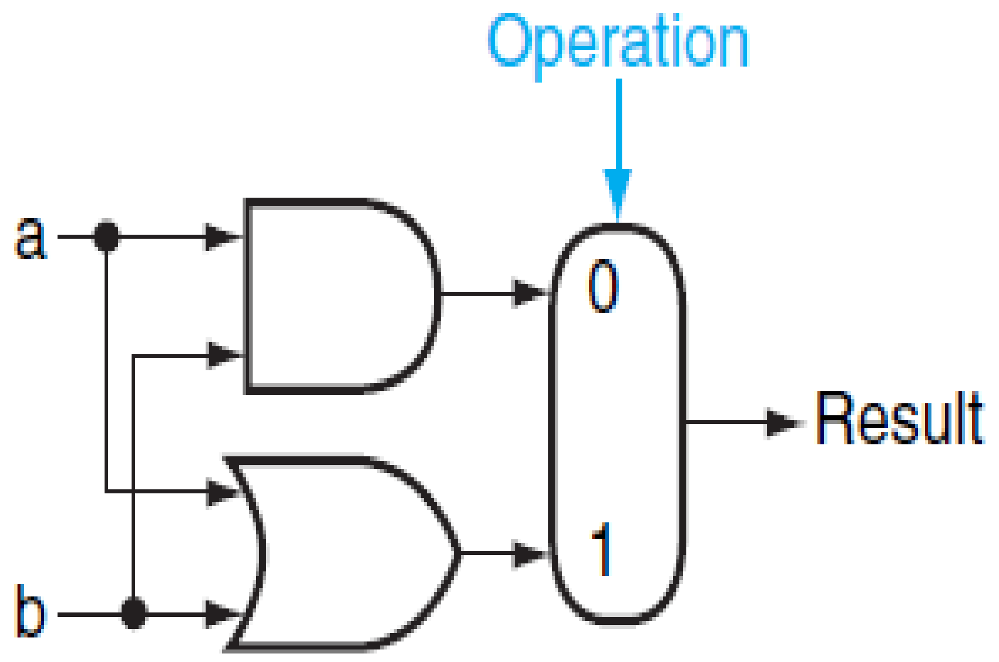
A	B	S	C
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

- Selection
- From parallel to serial

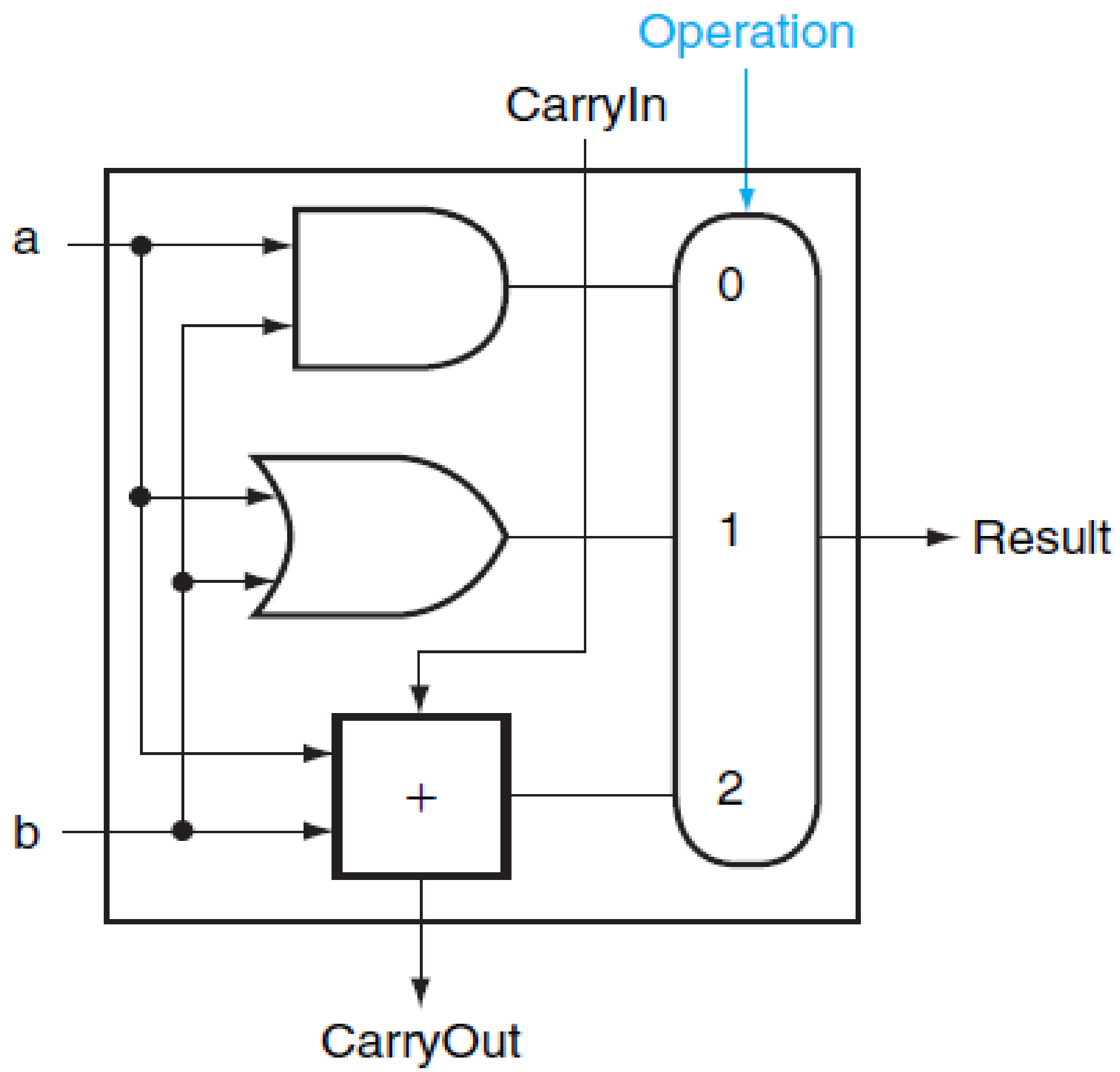
Multiplexor (1 bit) implementation

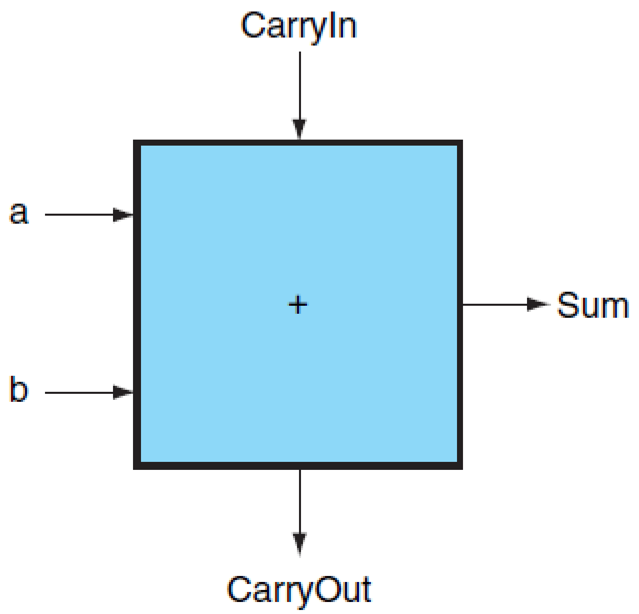


1-bit AND, OR



1-bit ALU (AND, OR, +)





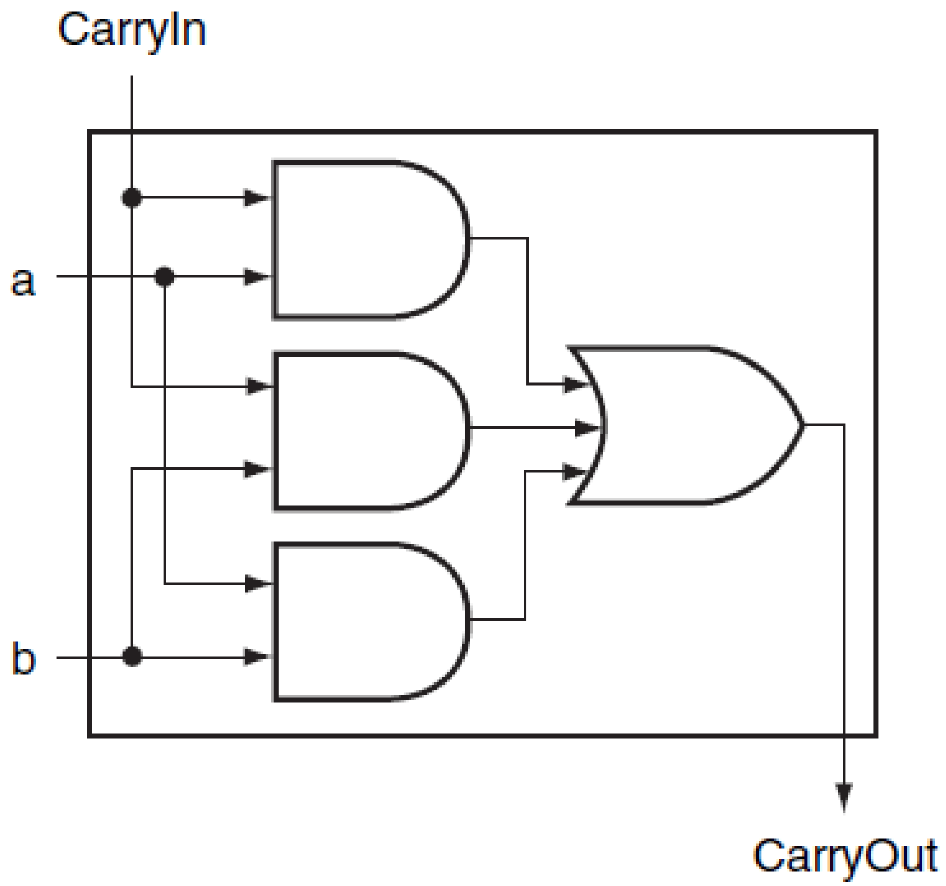
1-bit adder

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

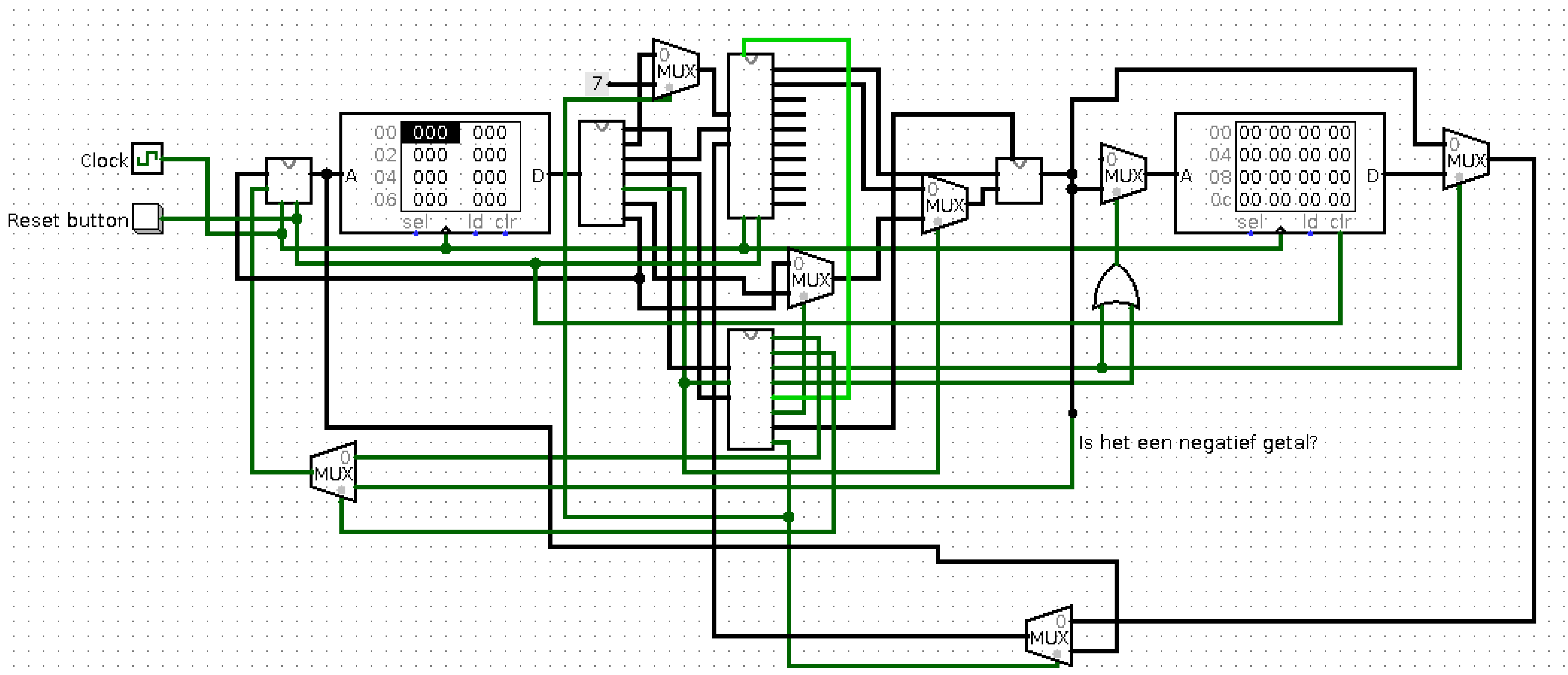
Inputs		
a	b	CarryIn
0	1	1
1	0	1
1	1	0
1	1	1

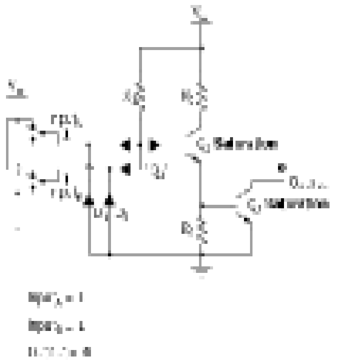
CarryOut

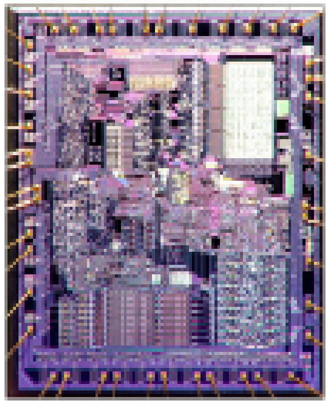
CarryOut = (b.CarryIn) + (a.CarryIn) + (a.b) + **(a.b.CarryIn)**
= (b.CarryIn) + (a.CarryIn) + (a.b)



Vanaf nu: “only connect”

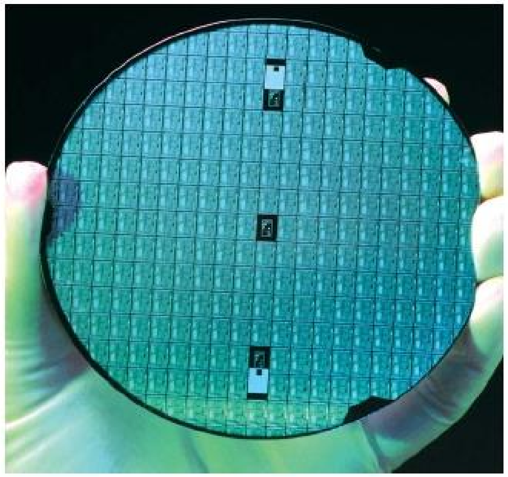






```
strcpy:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
    add  $s0, $zero, $zero
L1: add  $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```

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)
```