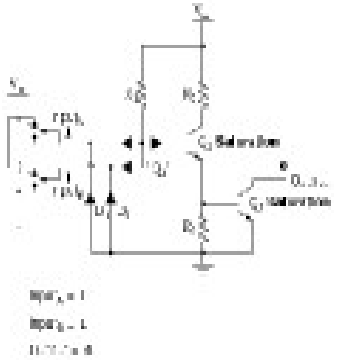
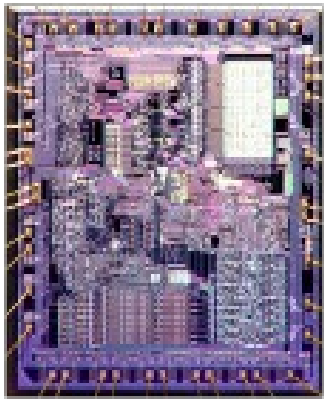
Computer Systemen en Computer Architectuur

Dinsdag 28 oktober 2025 – **getalvoorstellingen** ((meeloop)les)

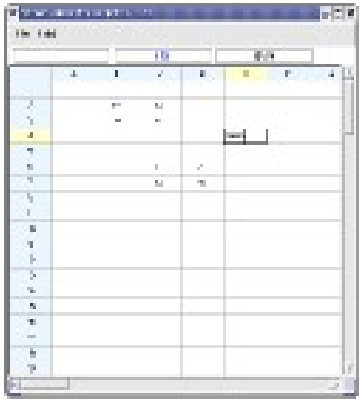
Hans Vangheluwe





```
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
```

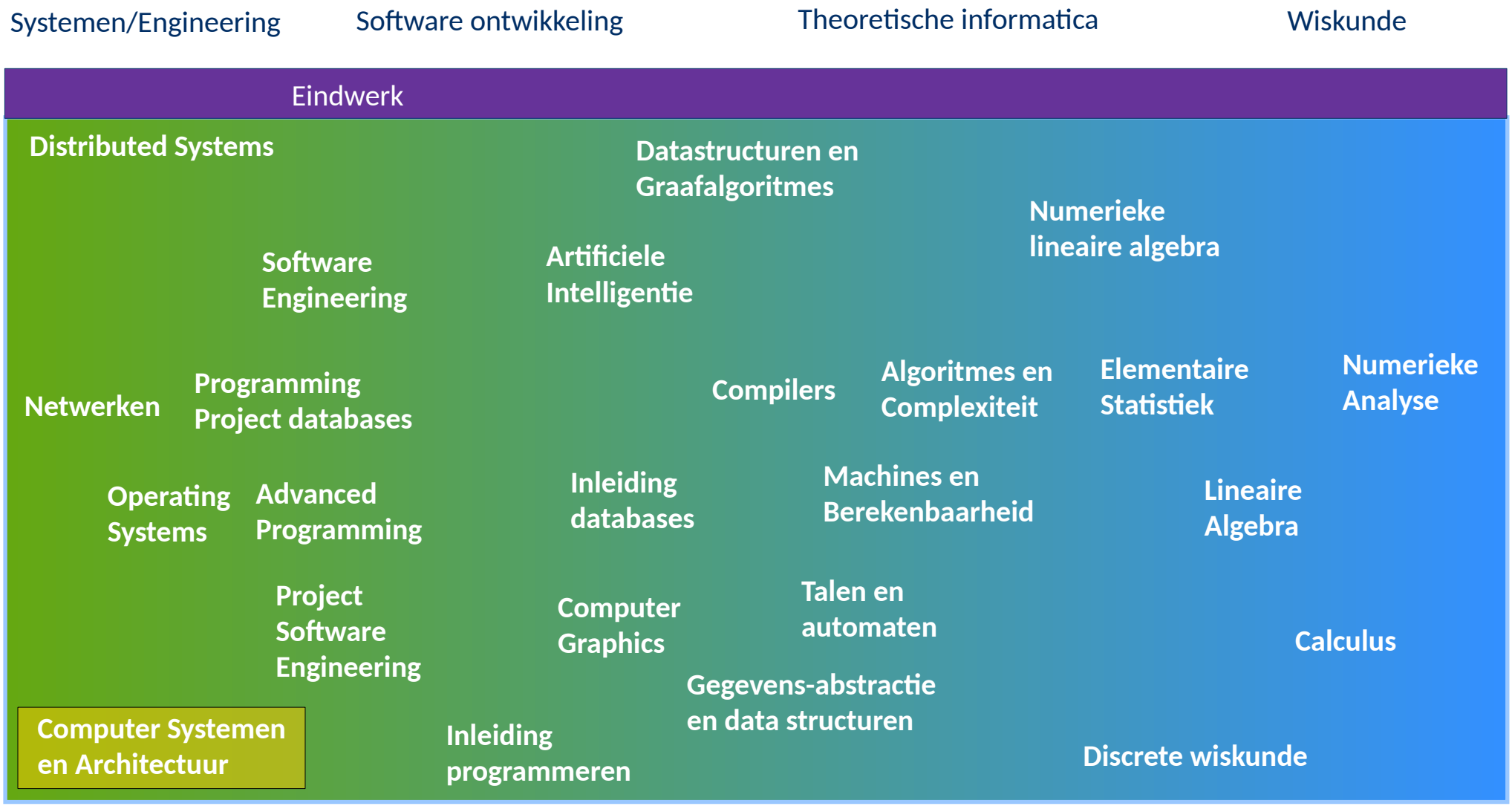
```
k = 1
xPowerK = x
Sign = 1; s = 0

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

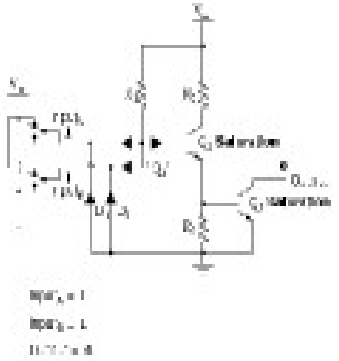


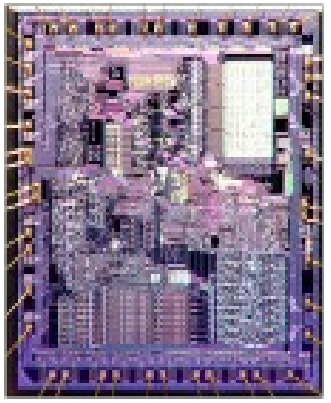
physics digital electronics — computer architecture / systems — HL progr. languages operating systems — complex SW applications

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

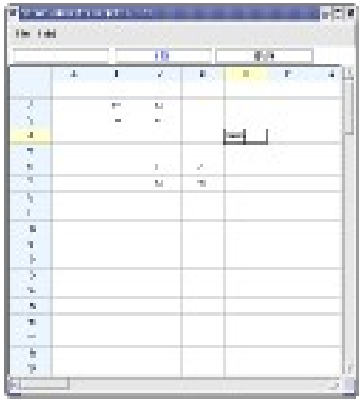
Bachelor Programma Informatica



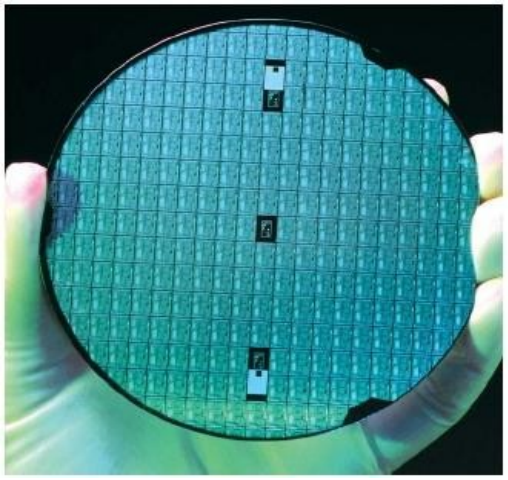


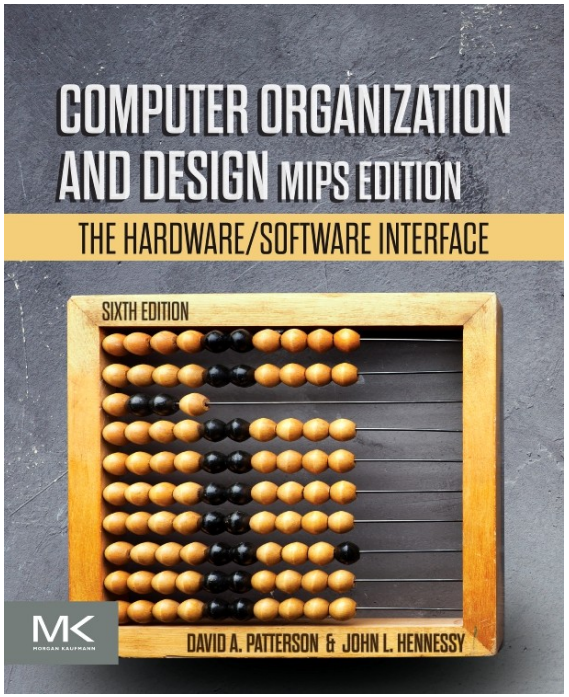


```
strcpy:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
    add  $s0, $zero, $zero
L1: add  $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```



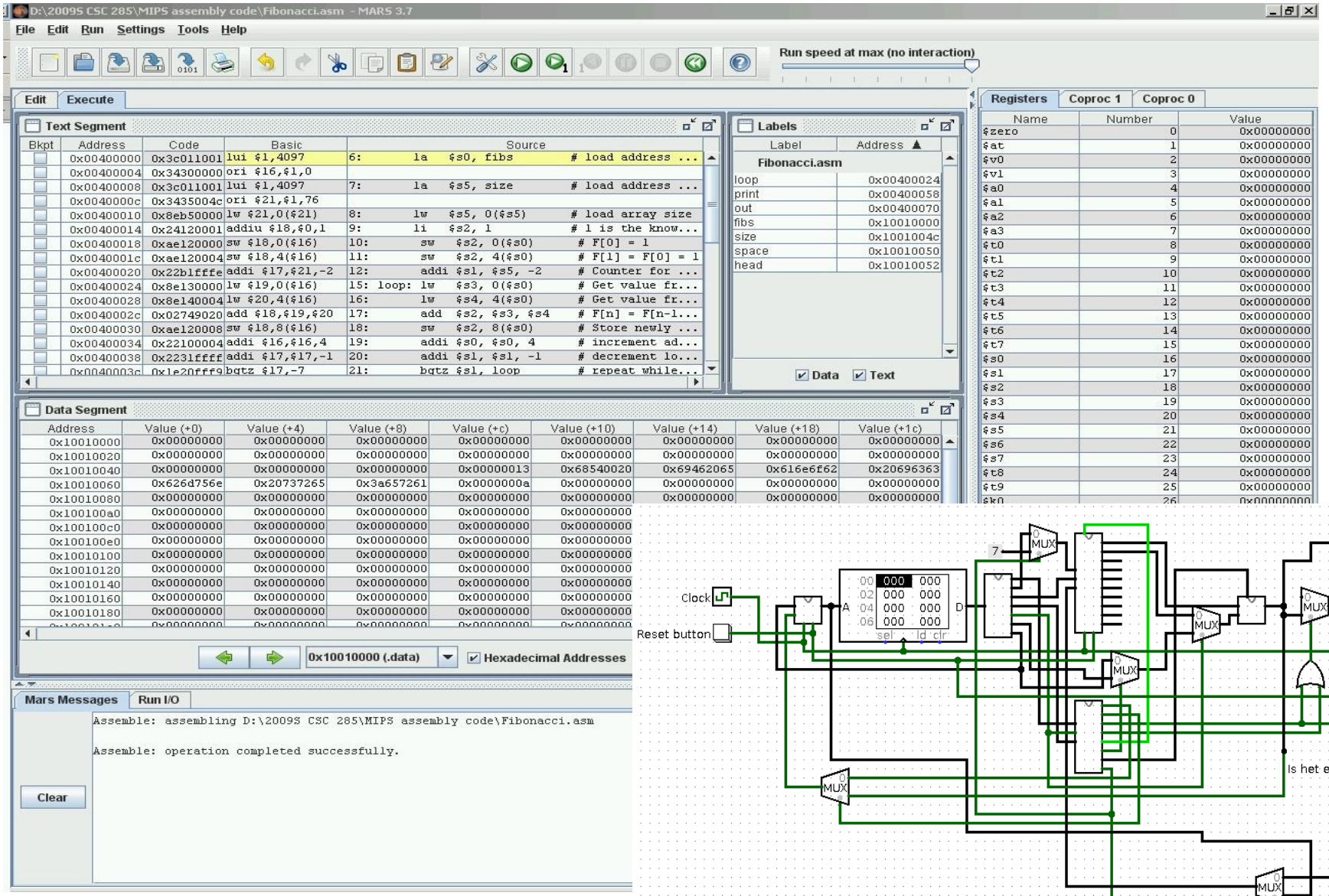
physics digital electronics — computer architecture / systems — HL progr. languages operating systems — complex SW applications






```
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)
```

Oefeningen ... al doende ...



MARS

http://courses.missouristate.edu/KenVollmar/MARS/

Logisim

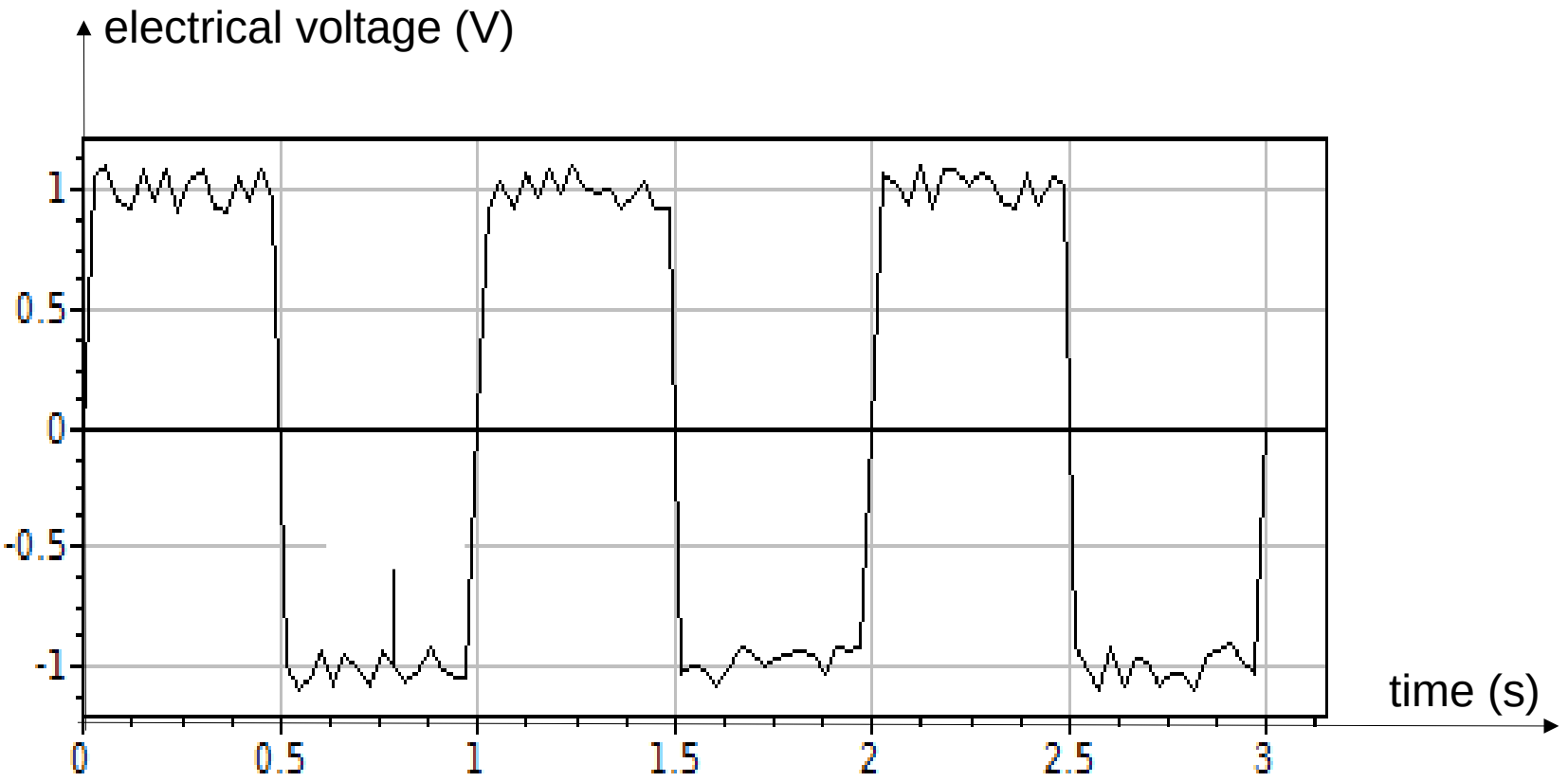
https://sourceforge.net/projects/circuit/

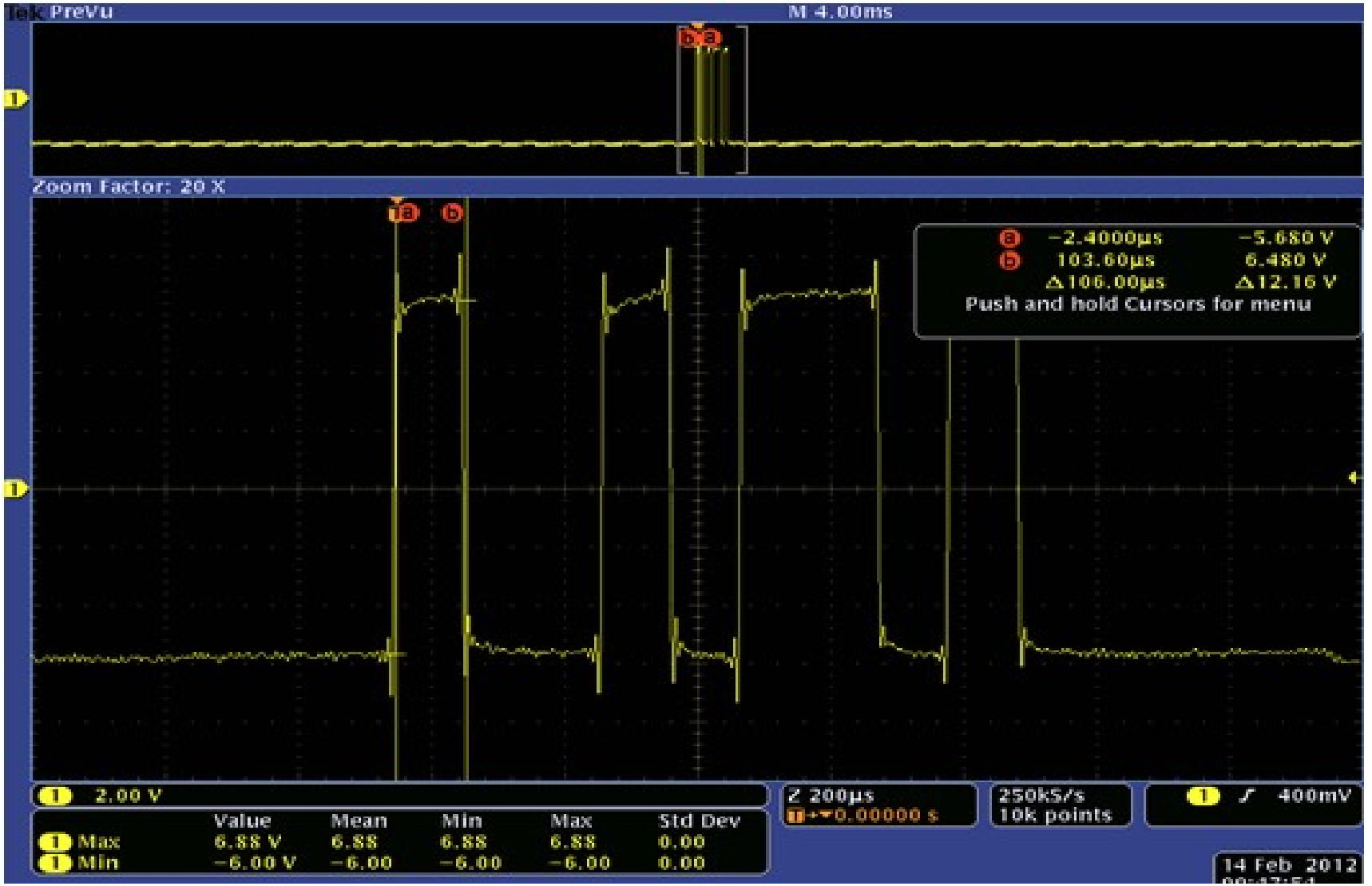
Data Representation

(binary) Data Representation

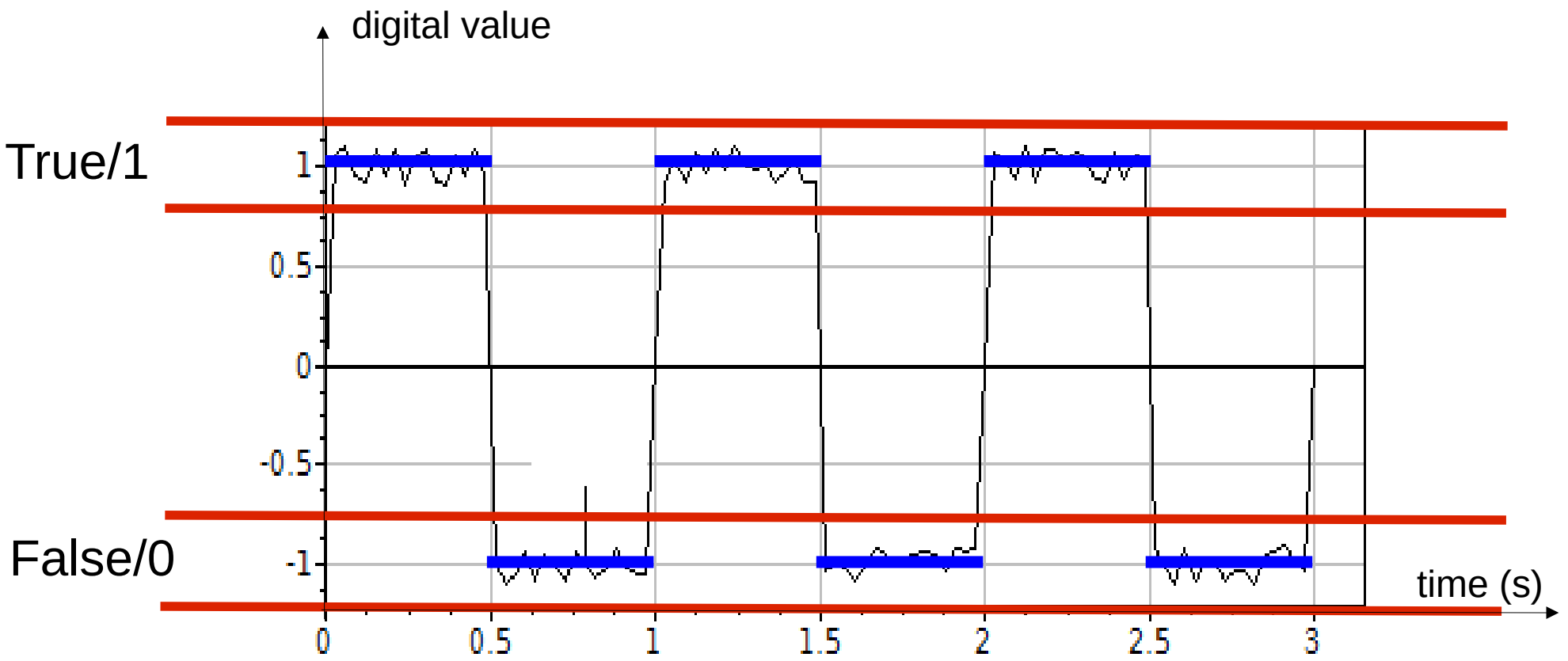
- Numbers
  $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
- Letters (and “strings”)
- Instructions
- ....

from Analog ...




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

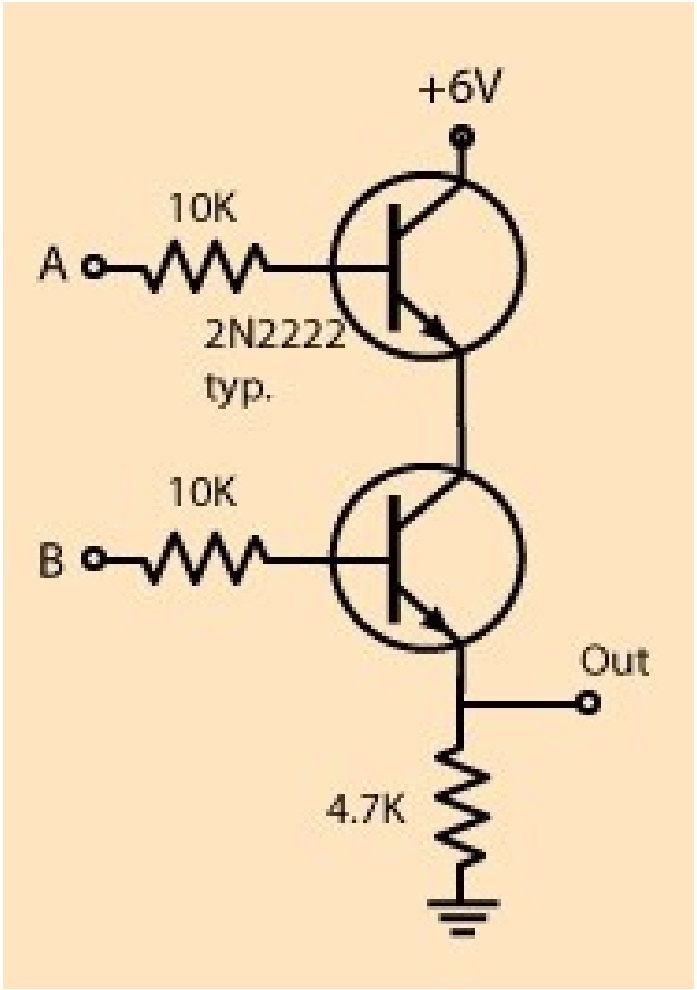
… to Digital



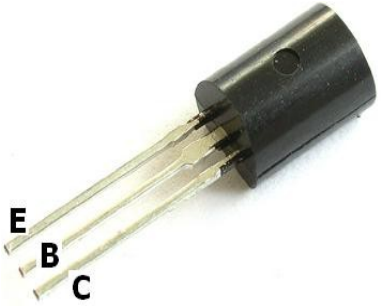
logical 0/1 – Boolean True/False – asserted/de-asserted – high/low

Implementing **Digital** (Logic) Components using **Analog** Electrical Components

AND



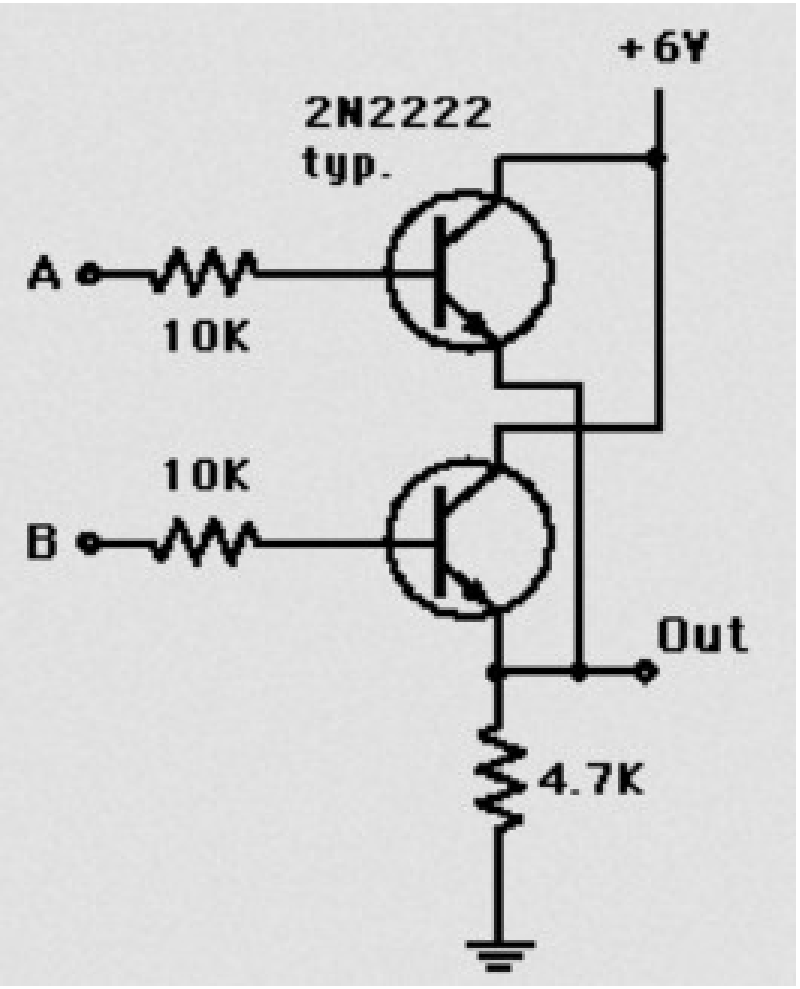
resistor



transistor

Used to amplify electrical signals.
When used “in saturation” acts as a switch.

OR



Binary representation

- in computing and telecommunications
- a **bit** is a basic *unit of information storage*
- “**bi**nary digi**t**”
- the maximum amount of **information** that can be stored in only two distinct states

`0` or `1`

Bit sequences
(bit) “string” (vs. char string)

```
                0   1 bit    bit
             0111   4 bits   nibble
         01100001   8 bits   byte
0101010110110111  16 bits   half-word
              ...  32 bits   word
              ...  64 bits   word
```

French: “octet”

A **word** is a natural unit of data used by a particular processor design (historically: 8, 16, 24, 32, or 64 bits)

Bit is abbreviated as “b”. e.g., k**b**ps (kilo bits per second)
Byte is abbreviated as “B”. e.g., k**B** (kilo Bytes)

Binary representation ++

With 1 bit, can **represent** 2 **distinct entities**

With 2 bits, can represent 4 distinct entities

...

With **N** bits, can represent $\mathbf{2^N}$ distinct entities

Example: {Red, Green, Blue} encoded as {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

Binary representation/approximation of numbers

Binary representation/encoding of **Unsigned Integers** ($\mathbb{N}$)

Binary Coded Decimal (BCD) representation/encoding of **Unsigned Integer** ($\mathbb{N}$) **/ Real** numbers ($\mathbb{R}$)

Binary representation/encoding of **Signed Integer Numbers** ($\mathbb{Z}$)

- Signed magnitude
- One's complement
- Two's complement
- Biased (Excess B)

Fixed-Point binary **approximation**/representation ($\mathbb{Q}$) of **Real** numbers ($\mathbb{R}$)

Floating Point binary **approximation**/representation of **Real** numbers ($\mathbb{R}$)

Binary representation/encoding of Unsigned Integers ($\mathbb{N}$)

"$x_{n-1}x_{n-2}...x_1x_0$" is an n-bit, **base** 2 (binary) encoding of 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: $[0, +2^n - 1] \subseteq \mathbb{N}_0$
- Example

  $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$

  $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$

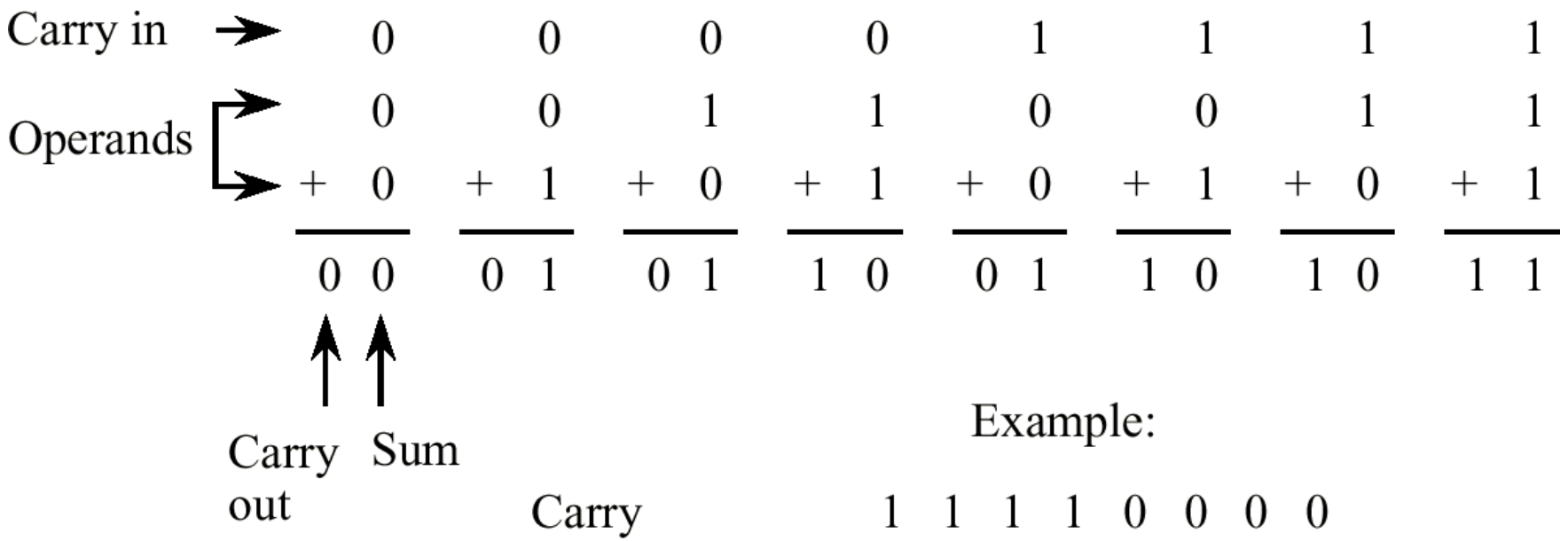
  $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$

Powers of 2

N	2^N	binary encoding as unsigned integer
0	1	0000 0000 0000 0001
1	2	0000 0000 0000 0010
2	4	0000 0000 0000 0100
3	8	0000 0000 0000 1000
4	16	0000 0000 0001 0000
5	32	0000 0000 0010 0000
6	**6**4	0000 0000 0100 0000
7	128	0000 0000 1000 0000
8	256	0000 0001 0000 0000
9	512	0000 0010 0000 0000
10	**1**,024	0000 0100 0000 0000
11	2,048	0000 1000 0000 0000
12	4,096	0001 0000 0000 0000
13	8,192	0010 0000 0000 0000
14	16,384	0100 0000 0000 0000
15	32,768	1000 0000 0000 0000
16	65,536	1 0000 0000 0000 0000
…	…	…
32	4,294,967,296	1 0000 0000 0000 0000 0000 0000 0000 0000


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

(intermezzo) Binary addition of Unsigned Integers (ℕ)



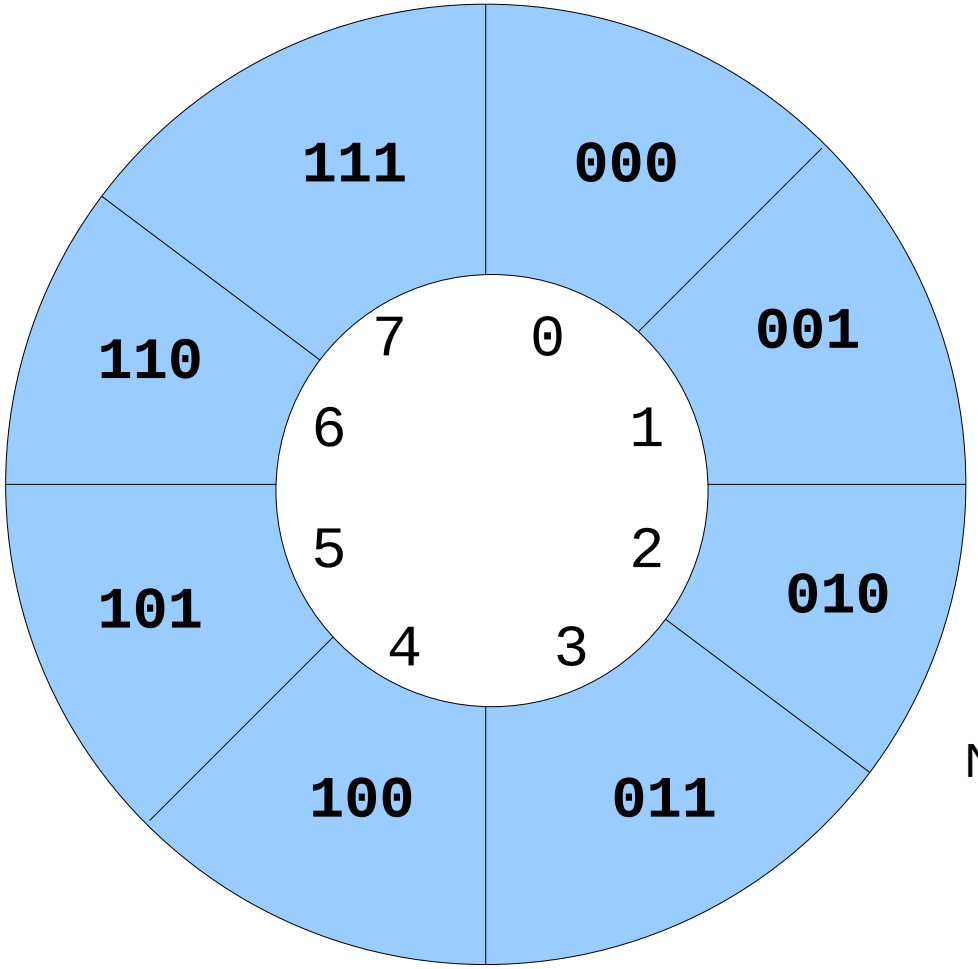
Example:

Carry		1	1	1	1	0	0	0	0	
Addend: A		0	1	1	1	1	1	0	0	$(124)_{10}$
Augend: B	+	0	1	0	1	1	0	1	0	$(90)_{10}$
Sum		1	1	0	1	0	1	1	0	$(214)_{10}$

unsigned “modulo” arithmetic

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

+1 (unsigned): carry gets discarded, no “overflow”



000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

N bits: $[[2^N]] = (2^N) \text{ MOD } 2^N = 0;$
$(2^N) \text{ DIV } 2^N = 1$
$[[2^N+k]] = (2^N+k) \text{ MOD } 2^N = k$

used in for example address calculation (PC)

Binary Coded Decimal (BCD) representation of unsigned Int (ℕ) / Real (ℝ)

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Every Decimal digit gets represented by 4 bits (Binary digits)

Decimal:	127	:	1	2	7
BCD:	000100100111	:	0001	0010	0111

Used mostly in

- Mainframes/servers (financial applications)
- Embedded microcontrollers/small processors (computation <<<)
- Only digital logic (no processor), display (7-segment)

Word size often 10 or 12 decimal digits (*e.g.,* in calculators)

Binary Coded Decimal (BCD)

Advantages:

- Easy to print, display (e.g., 7-segment), ... thanks to **per-digit** conversion
- Numbers such as decimal `0.2` have
  - an **infinite** representation in binary
    (`0.00110011001...`)
  - a **finite** representation in binary-coded decimal
    (`0000.0010`)
- Scaling by factors of 10 by **shifting** (clever compiler ...)
- 10-based **rounding** is easy

Disadvantages:

- More **complex** to implement +,-,*,/ (+: 15-20% more circuitry)
- Slower
- More storage space required

  e.g., 15_{10}: 1111_{binary} vs. 00010101_{BCD}

Praktische Vragen?

Universiteit Antwerpen

Login NL | EN

Studeren Onderzoek Studentenleven Over UAntwerpen Bibliotheek Vacatures Kalender Contact

INFORMATICA

Bachelor Master Iets voor jou? Waarom UAntwerpen? Jobmogelijkheden Contact

Studeren › Opleidingen › Alle opleidingen › Informatica

Vragen over de opleiding informatica?

Twijfel je nog over je studiekeuze en heb je nog vragen over de opleiding of jouw voorkennis? Stuur dan een e-mail naar word.wetenschapper@uantwerpen.be.

Heb je nog vragen over je inschrijving voor de opleiding, het studieprogramma of jouw studietraject? Surf dan naar de Helpdesk Wetenschappen, raadpleeg er de FAQ of kies voor 'Maak een nieuw ticket aan'.

We beantwoorden je vraag zo snel mogelijk.

Of misschien stel je je wel één van deze vragen:

- Wat is het verschil tussen informatica aan de hogeschool en aan de universiteit?
- Moet ik al kunnen programmeren voor ik kan starten?
- Wat kan je gaan doen met een diploma informatica?

https://www.uantwerpen.be/nl/studeren/aanbod/alle-opleidingen/informatica-studeren/contact/