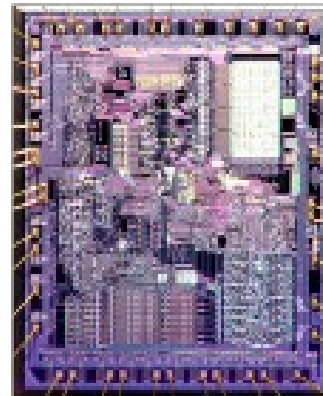
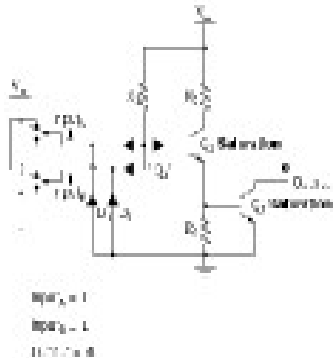


Computer Systemen en Computer Architectuur



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

Hans Vangheluwe



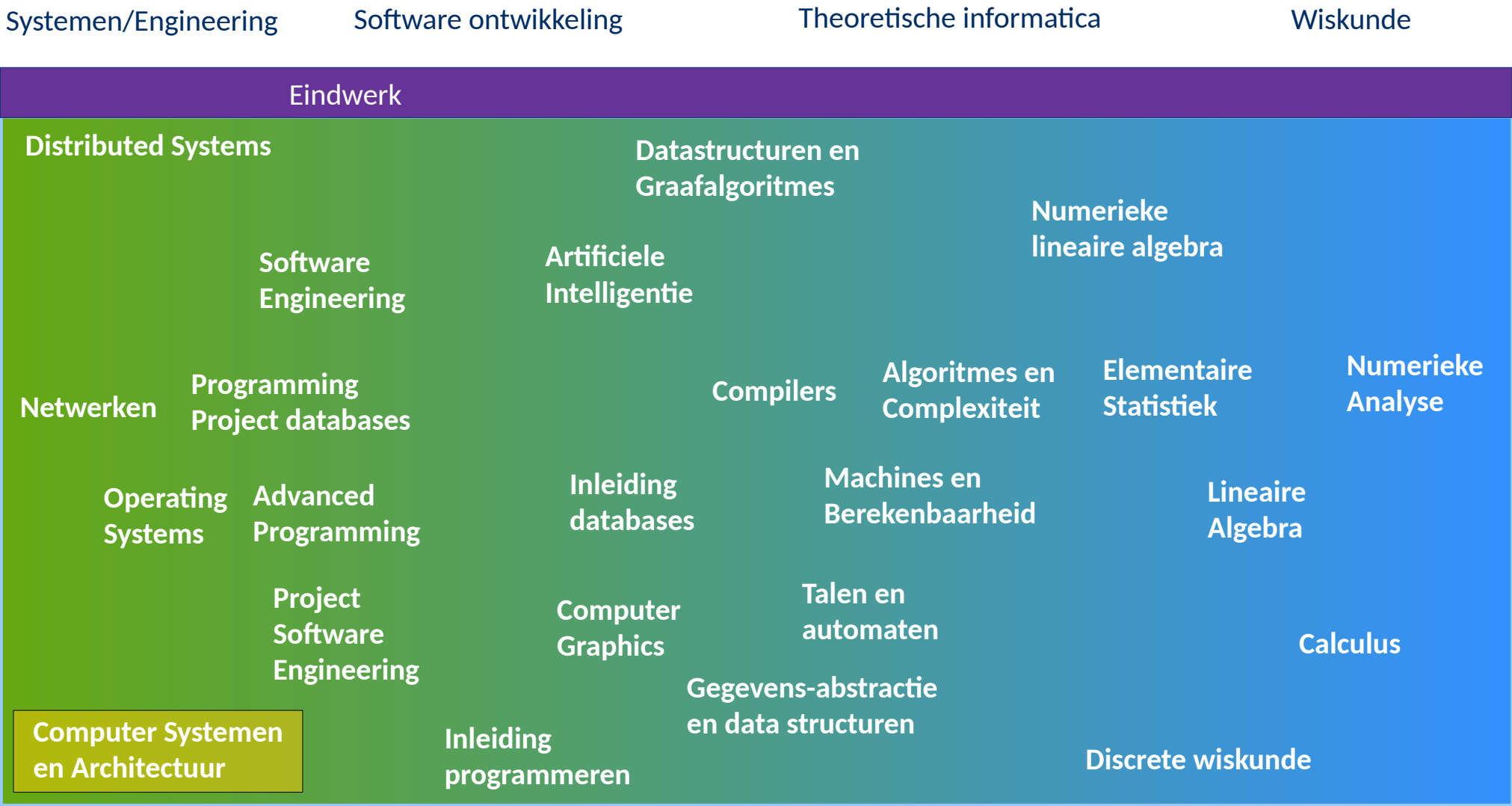
```
strcpy:
    addi $sp, $sp, -4
    sw   $a0, 0($sp)
    add  $s0, $zero, $zero
L1:    add $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2:    lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```

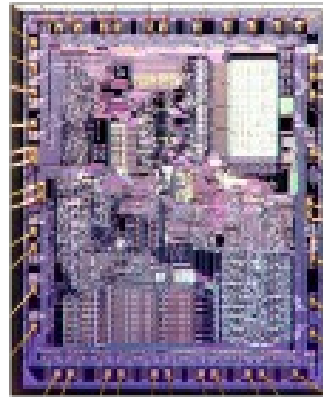
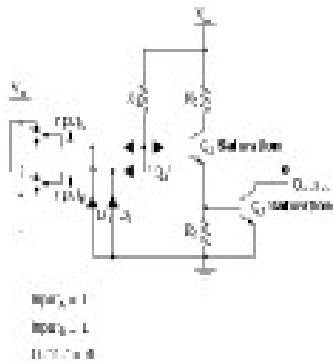
k = 1
xPowerK = x
Sign = 1; s = 0

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

Bachelor Programma Informatica

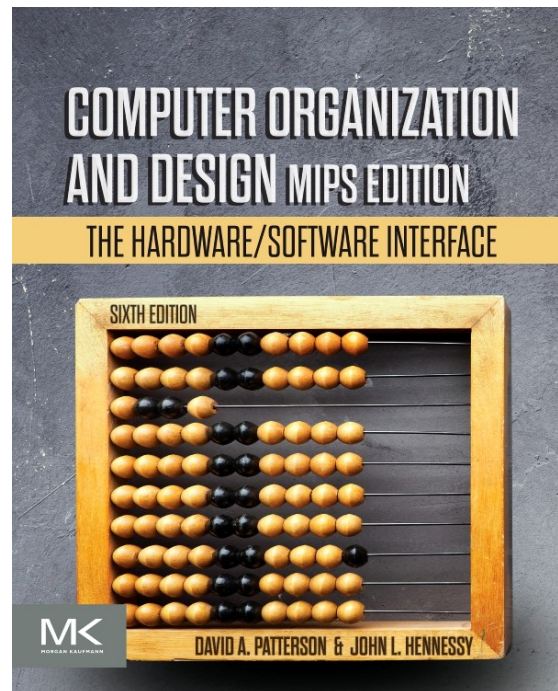
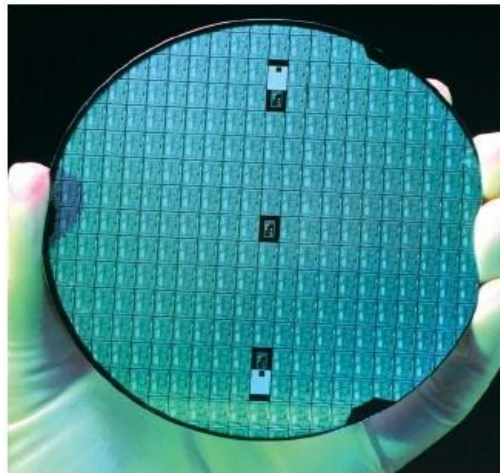




```

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 ...

D:\2009S CSC 285\MIPS assembly code\Fibonacci.asm - MARS 3.7

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui \$t1,4097	6: la \$s0, fibs # load address ...
	0x00400004	0x34300000	ori \$t6,\$t1,0	
	0x00400008	0x3c011001	lui \$t1,4097	7: la \$s5, size # load address ...
	0x0040000c	0x3435004c	ori \$t2,\$t1,76	
	0x00400010	0x8eb50000	lw \$t2,0(\$t1)	8: lw \$s5, 0(\$s5) # load array size
	0x00400014	0x24120001	addiu \$t8,\$t0,1	9: li \$s2, 1 # 1 is the know...
	0x00400018	0xae120000	sw \$t8,0(\$t6)	10: sw \$s2, 0(\$s0) # F[0] = 1
	0x0040001c	0xae120004	sw \$t8,4(\$t6)	11: sw \$s2, 4(\$s0) # F[1] = F[0] = 1
	0x00400020	0x22b1ffff	addi \$t7,\$t5,-2	12: addi \$s1, \$s5, -2 # Counter for ...
	0x00400024	0x8e130000	lw \$t9,0(\$t6)	15: loop: lw \$s3, 0(\$s0) # Get value fr...
	0x00400028	0x8e140004	lw \$t9,4(\$t6)	16: lw \$s4, 4(\$s0) # Get value fr...
	0x0040002c	0x02749020	add \$t8,\$t9,\$t20	17: add \$s2, \$s3, \$s4 # F[n] = F[n-1...
	0x00400030	0xae120008	sw \$t8,8(\$t6)	18: sw \$s2, 8(\$s0) # Store newly ...
	0x00400034	0x22100004	addi \$t6,\$t6,4	19: addi \$s0, \$s0, 4 # increment ad...
	0x00400038	0x2231ffff	addi \$t7,\$t7,-1	20: addi \$s1, \$s1, -1 # decrement lo...
	0x0040003c	0x1e20ffff	bgtz \$t7,-7	21: bgtz \$s1, loop # repeat while...

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000013	0x68540020	0x69462065	0x616e6f62	0x20696363
0x10010060	0x626d756e	0x20737265	0x3a657261	0x0000000a	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages Run I/O

Assemble: assembling D:\2009S CSC 285\MIPS assembly code\Fibonacci.asm

Assemble: operation completed successfully.

Clear

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000

Labels

Label	Address
Fibonacci.asm	
loop	0x00400024
print	0x00400058
out	0x00400070
fibs	0x10010000
size	0x1001004c
space	0x10010050
head	0x10010052

Logisim circuit diagram showing a Fibonacci sequence generator. The circuit uses a 7-bit register (A) and a 4-bit register (D) to store the current Fibonacci number and its previous value. The circuit is controlled by a Clock and a Reset button. The output of the circuit is a 7-bit number (A) and a 4-bit number (D). The circuit is labeled "Is het een negatief getal?" (Is it a negative number?).

MARS

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

Logisim

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

Data Representation

(binary) Data Representation

- Numbers

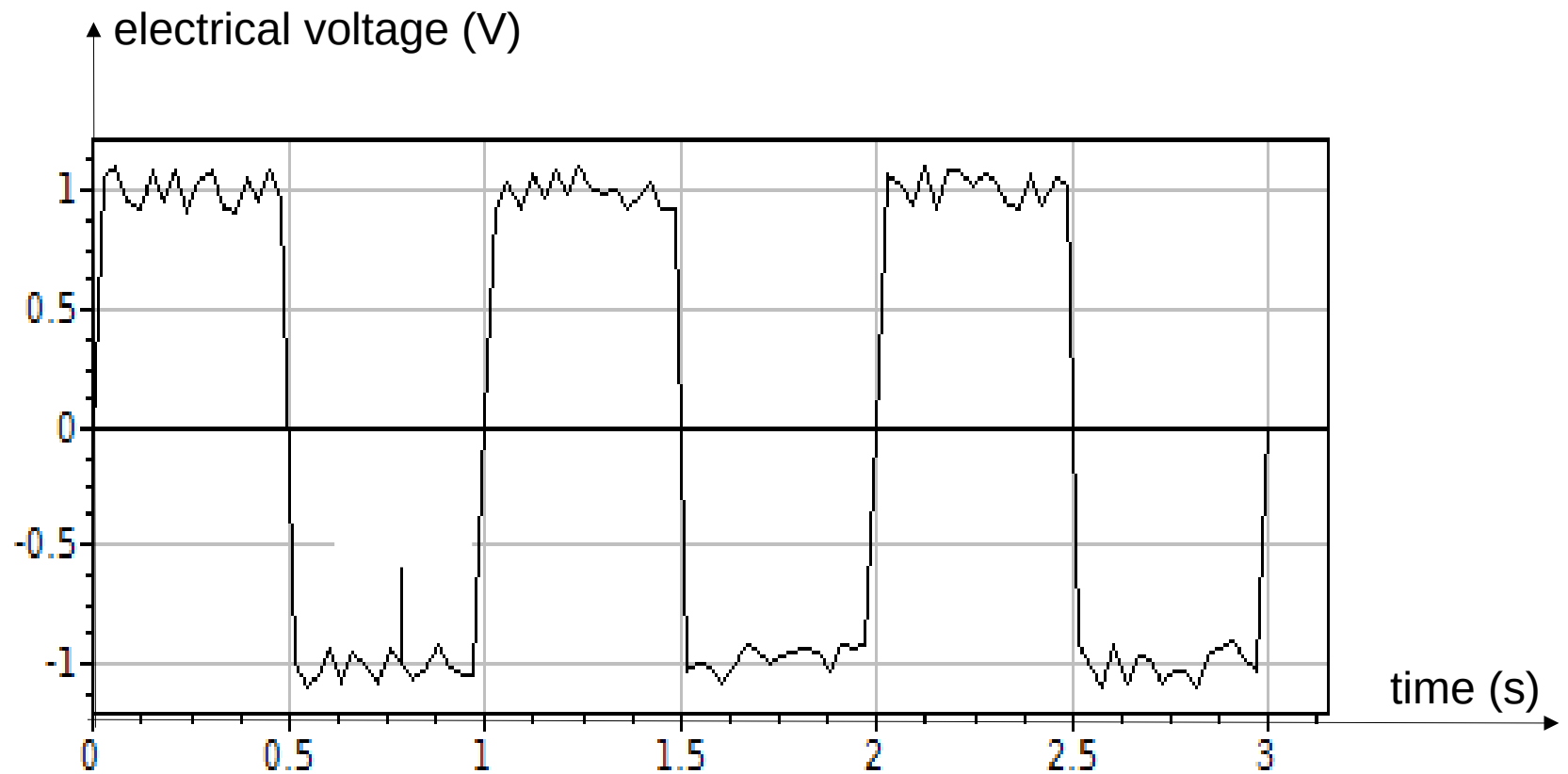
$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$

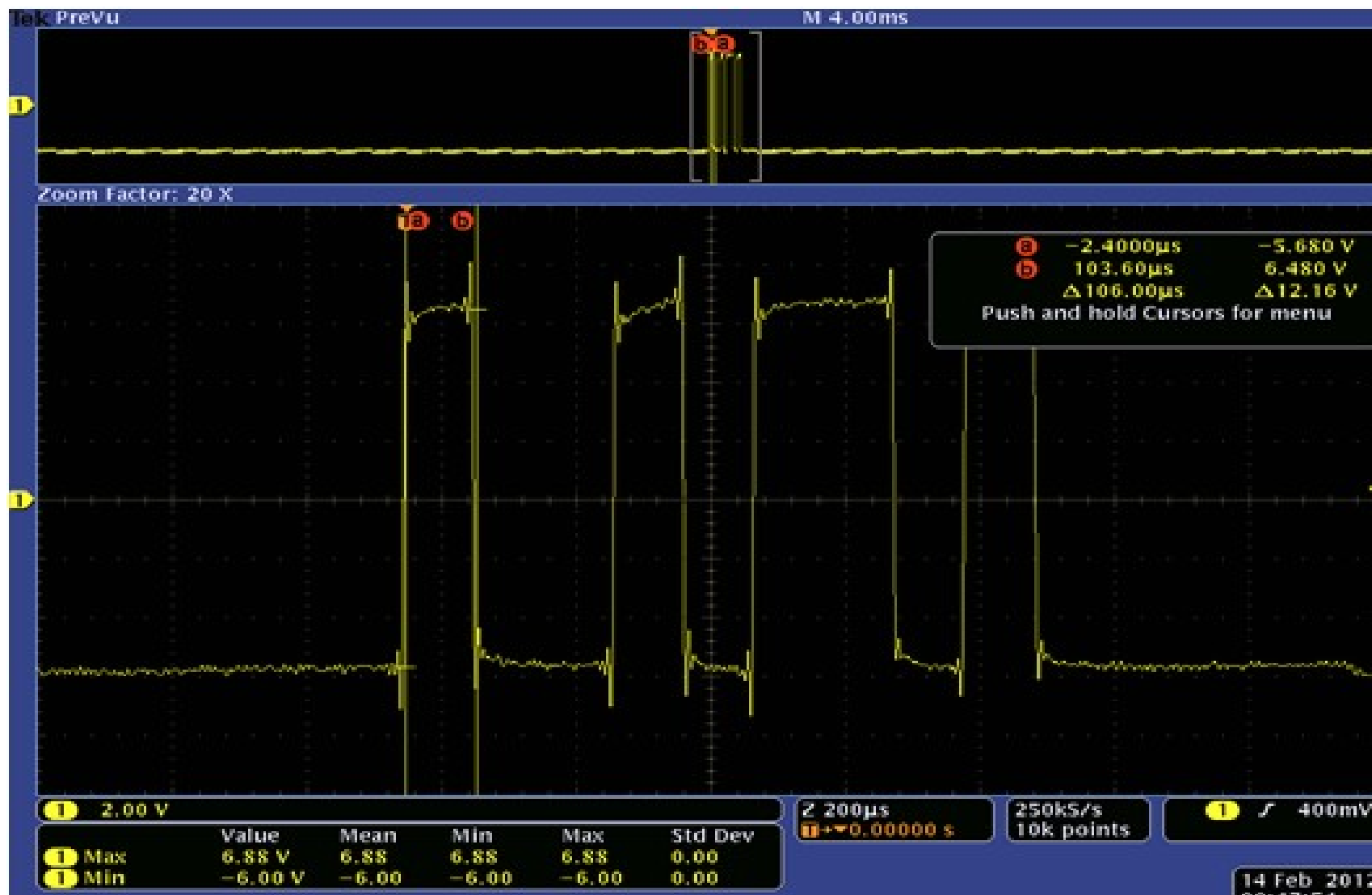
- Letters (and “strings”)

- Instructions

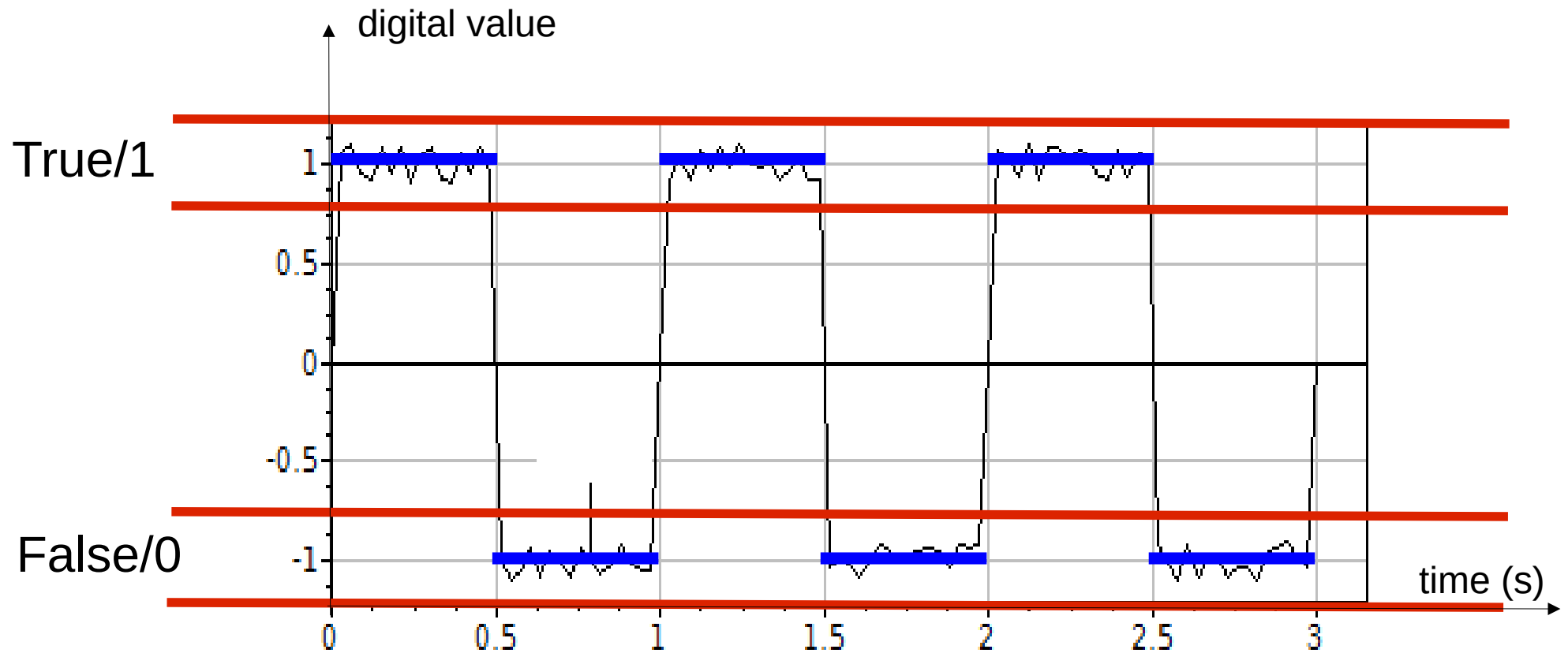
-

from Analog ...





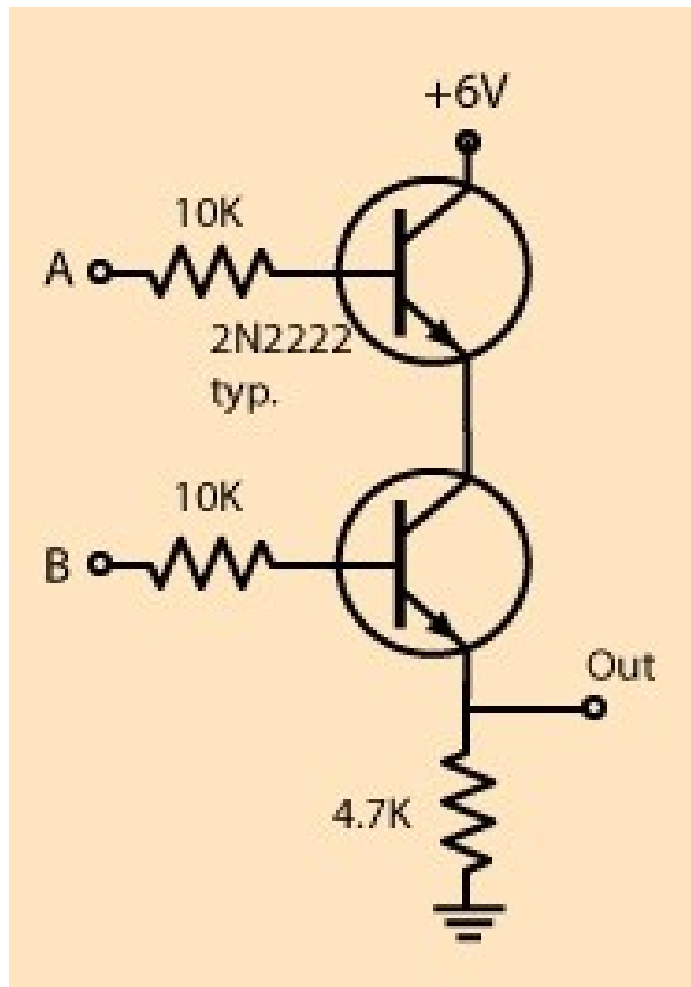
... to Digital



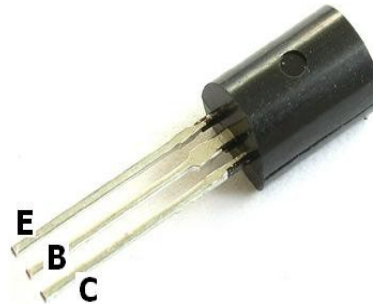
logical 0/1 – Boolean True/False – asserted/de-asserted – high/low

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

AND



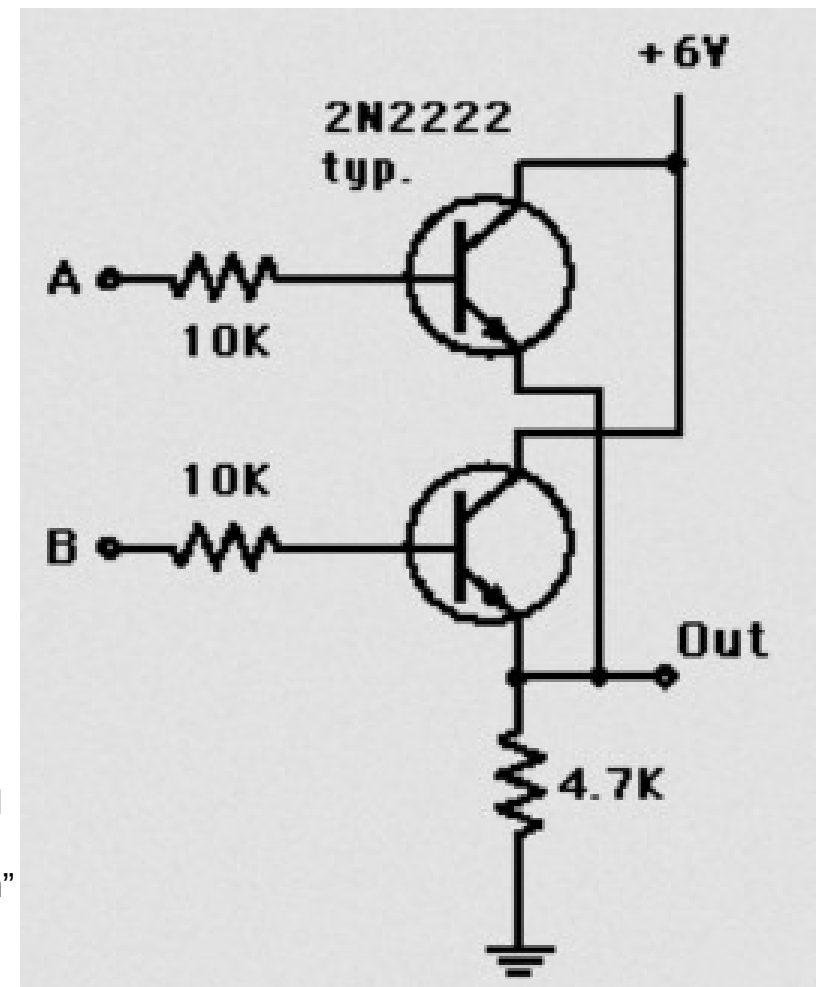
resistor



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*
- “**binary digit**”
- 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	French: “octet”
0101010110110111	16 bits	half-word	
...	32 bits	word	
...	64 bits	word	

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 **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 ($\mathbb{N}$)

" $x_{n-1}x_{n-2} \dots x_1x_0$ " is an n-bit, **base 2** (binary) encoding of value x

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

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 1011₂

= 0 + ... + 1×2^3 + 0×2^2 + 1×2^1 + 1×2^0

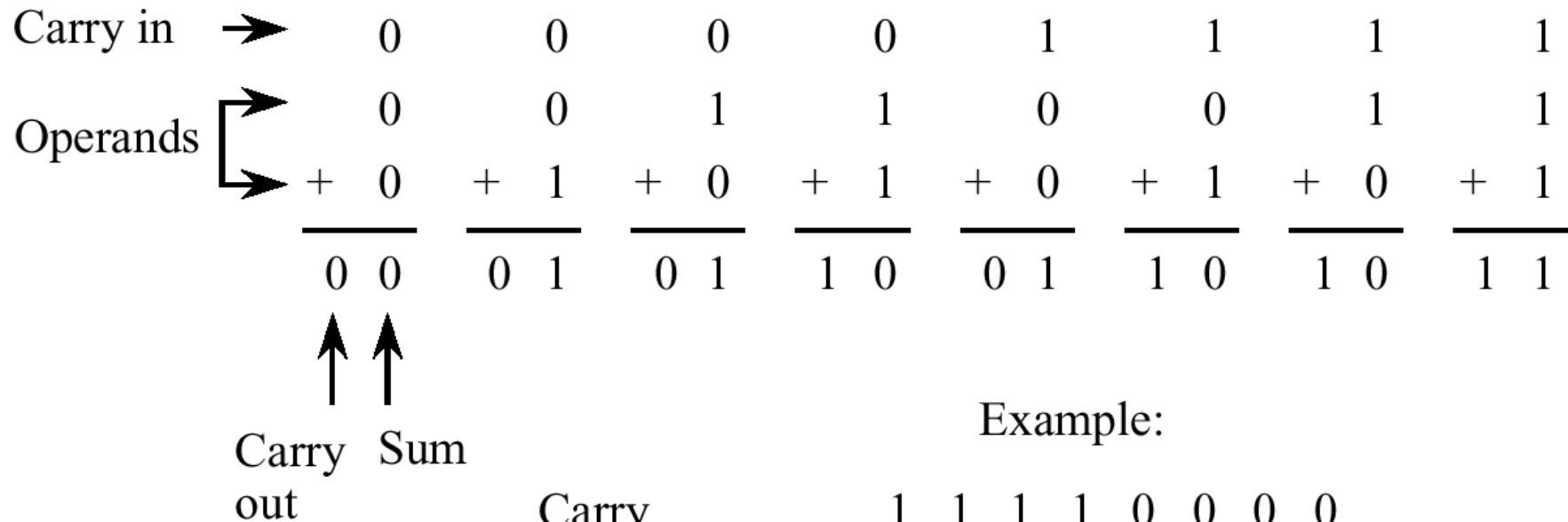
= 0 + ... + 8 + 0 + 2 + 1 = 11₁₀

Powers of 2

[illegible]

1010001001010001010
111011001011001010101
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
101011101100110101010
11001010101010101000
1010010010100101000
101011101100110101010

(intermezzo) Binary addition of Unsigned Integers ($\mathbb{N}$)



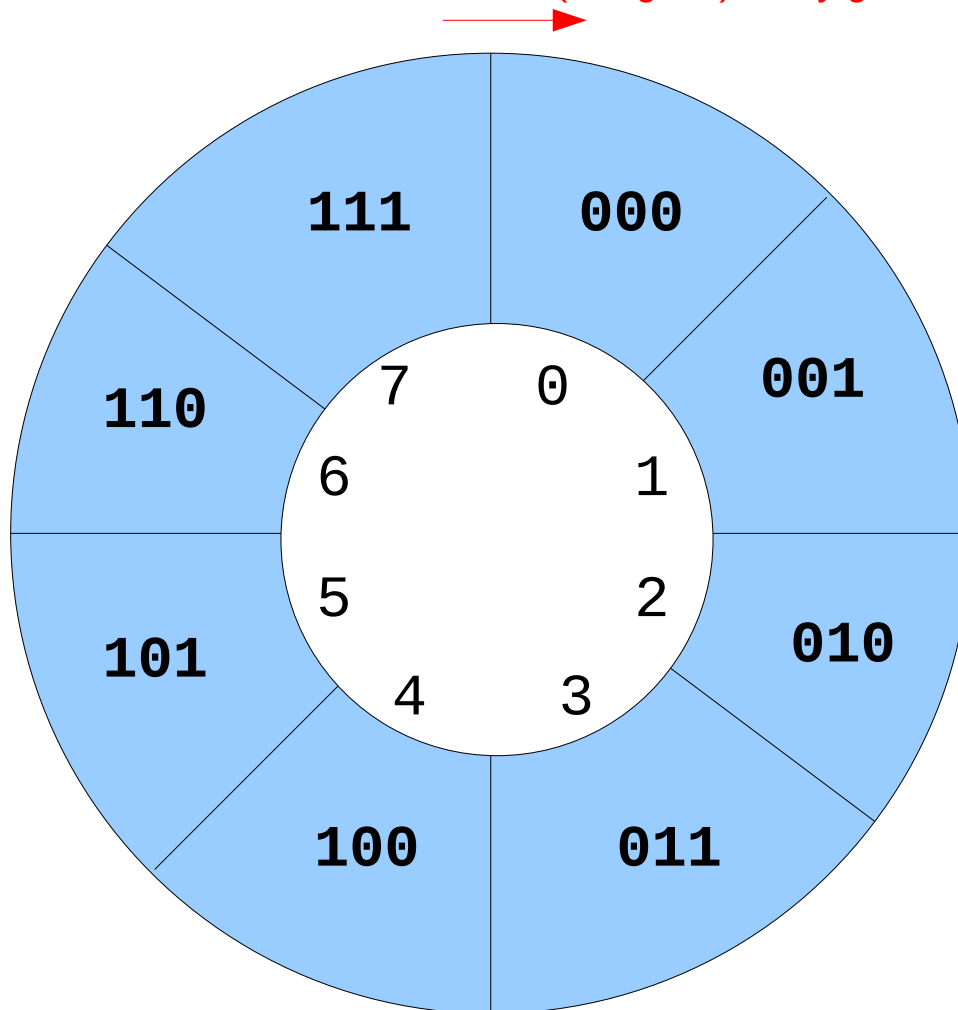
Example:

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	$(90)_{10}$
Sum		1	1	0	1	0	1	1	$(214)_{10}$

unsigned “modulo” arithmetic

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

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



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

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

used in for example address calculation (PC)

Binary Coded Decimal (BCD)

representation of unsigned Int ($\mathbb{N}$) / Real ($\mathbb{R}$)

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
(0.001100110011...)
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?

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?](#)