Digital logic

- opposite of analog circuitry : V & I only allowed two values : on/off, true/false, yes/no, high/low, 1/Ø
- inherently more reliable, less prone to noise
- easily realized in simple electronic circuits (e.g. TTL : transistor-transistor logic)
- naturally leads to (and can be described by) performing arithmetics with binary numbers

- Our "usual" number system,

  decimal : base = 10, digits = Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9

cf. binary : base = 2, digits = Ø, 1 ← = bits

cf. octal : base = 8, digits = Ø, 1, 2, 3, 4, 5, 6, 7

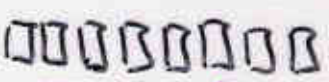
cf. hexadecimal: base = 16, digits = Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- positional : the same digit has a different worth depending on position:

$3483_{10} = 3\times10^3 + 4\times10^2 + 8\times10^1 + 3\times10^0$

(10^3, 10^0; more significant ← digit → less)

convention:



MS → LS

m = most, l = least

$1\,\emptyset\,1\,1\,\emptyset\,1_2 = 1\times2^5 + \emptyset\times2^4 + 1\times2^3 + 1\times2^2 + \emptyset\times2^1 + 1\times2^\emptyset$

(2^4, 2^2, 2^0) $= 32 + 8 + 4 + 1 = 45_{10}$

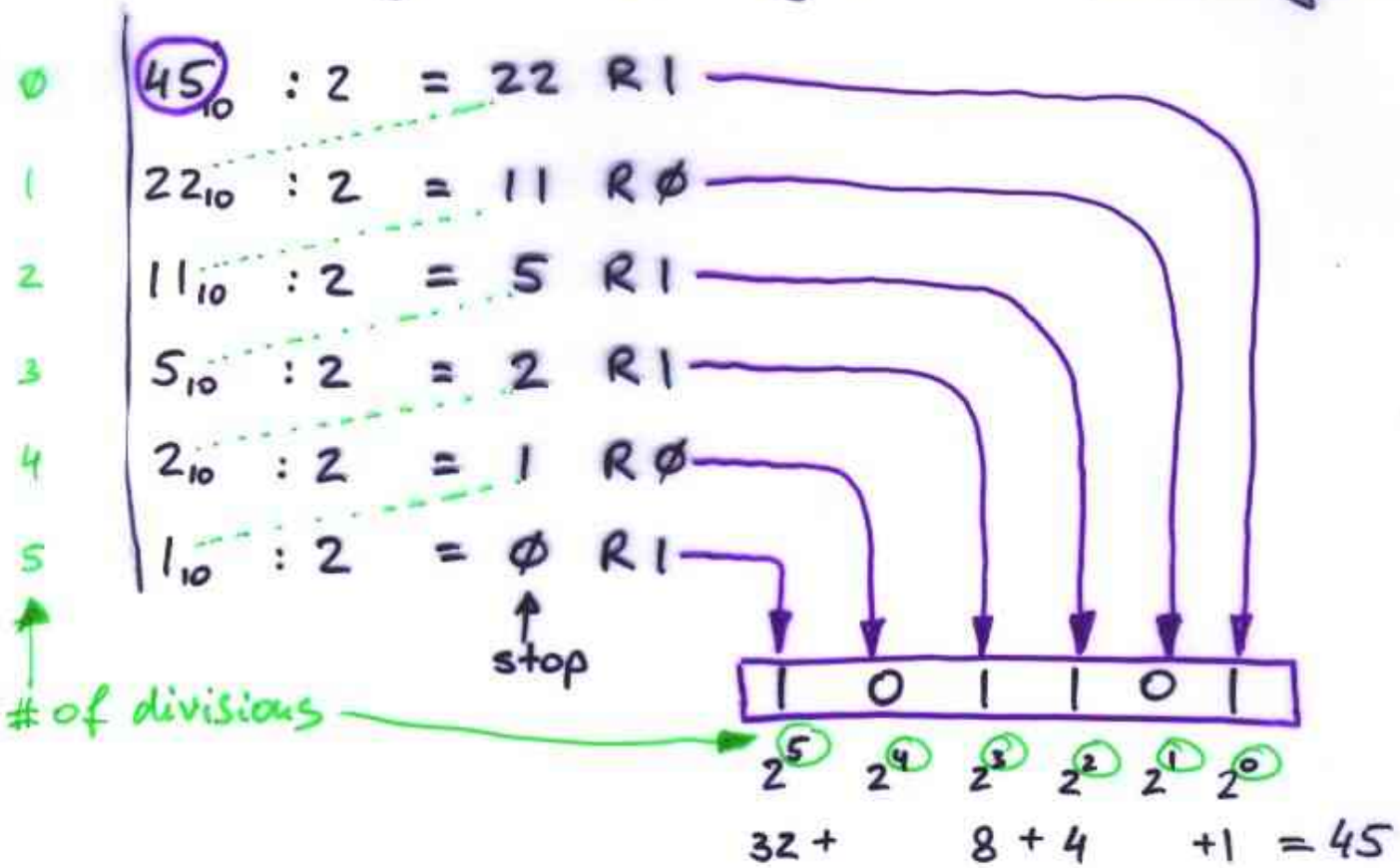
$1\,8\,A_{16} = 1\times16^2 + 8\times16^1 + A\times16^\emptyset = 256 + 128 + 10 = 394_{10}$

(16^2, 16^0)

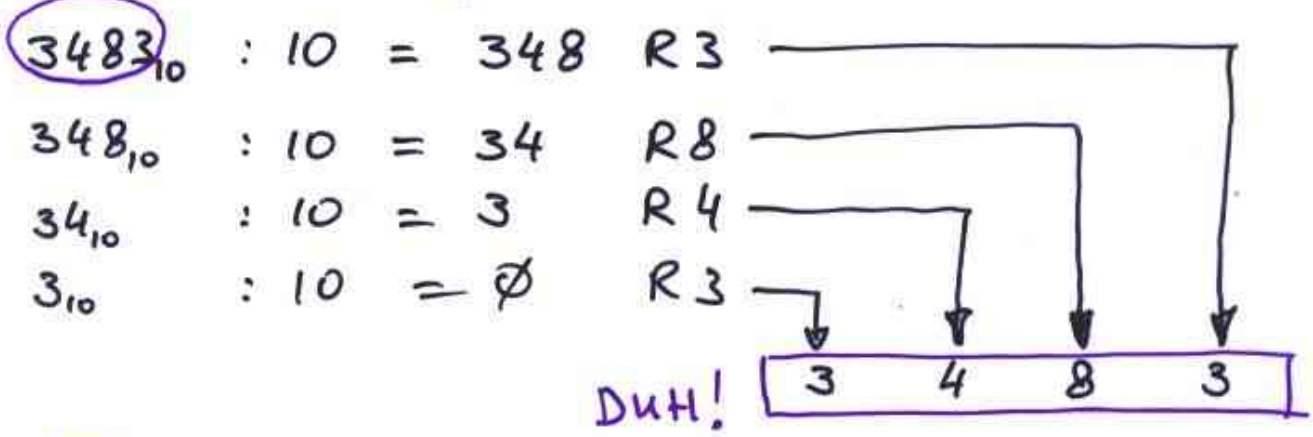
First few numbers

Dec.	Hex.	Oct.	Binary
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111
16	10	20	10000
17	11	21	10001
18	12	22	10010
32	20	40	100000
64	40	100	1000000
256	100	400	100000000
1024	400	2000	10000000000
2048	800	4000	100000000000
4096	1000	10000	1000000000000

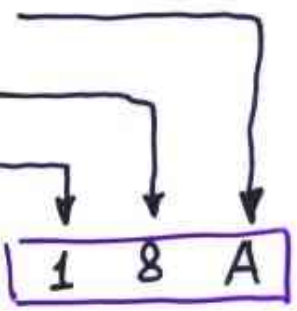
- converting into binary form: repeatedly ÷2



- works for any base!



- 394_{10} : 16 = 24 R $10_{10} = A_{16}$
- 24_{10} : 16 = 1 R $8_{10} = 8_{16}$
  1_{10} : 16 = 0 R $1_{10} = 1_{16}$



- other systems possible : e.g. with base = 3, or 4, or ...
  Why 8 & 16 ? The reason: $8 = 2^3$, $16 = 2^4$

  ⇒ every 3 binary digits correspond to 1 octal
  — " — 4 — " — " — " — 1 hexadecimal

  $1\emptyset1\,\emptyset11\,1\emptyset\emptyset\,\emptyset\emptyset1_2 = 5361_8 = AF1_{16}$

- $1\emptyset_b = b_{10}$, $\forall b$ ⇒ "$1\emptyset$" in any number system has the value = base

  $1\emptyset_2 = 2_{10}$ $\quad 1\emptyset_8 = 8_{10}$ $\quad 1\emptyset_{16} = 16_{10}$

- binary numbers add just like decimals — including the carry digits:

T 10-1

$$\begin{array}{r} {\scriptstyle (1)} \\ 36 \\ +\ 18 \\ \hline 54 \end{array} \qquad \begin{array}{r} {\scriptstyle (1)} \\ 11\emptyset \\ +\ 1\emptyset1 \\ \hline 1\emptyset11 \end{array}$$

binary addition table :

+	∅	1
∅	∅	1
1	1	1∅

carry

- same for multiplication :

$$\begin{array}{r} 10110 \\ \times\ \ 101 \\ \hline 10110 \\ 00000 \\ 10110 \\ \hline 1101110 \end{array}$$

binary multiplication table :

T 10-8

x	∅	1
∅	∅	∅
1	∅	1

• subtraction : need to borrow from the next higher bit to subtract 1 from 0:

$$\begin{array}{r} 57_{10} \\ -\ 43_{10} \\ \hline 14_{10} \end{array} \qquad \begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 111001 \\ 101011 \\ \hline 001110 \end{array}$$

"ones complement" : an alternative scheme of dealing with signed numbers.

① complement all bits : $0 \to 1,\ 1 \to 0$ } ① + ② form "twos complement"

② add 1

③ add as usual, discard the overflow carry

$$\begin{array}{r} 57_{10} \\ -\ 43_{10} \\ \hline 14_{10} \end{array} \qquad \begin{array}{r} 111001 \\ 101011 \end{array} \qquad \rightarrow \begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 111001 \\ +\ 010100 \\ +\ \quad\ 1 \\ \hline (1)\,001110 \end{array}$$

$= 20_{10}$
$= 1_{10}$
} $21_{10} = 64 - 43$

i.e. the "distance" to the next higher power of 2.

A similar scheme could also work for the decimal number system:

$$\begin{array}{r} 57 \\ -\ 43 \\ \hline 14 \end{array} \rightarrow \begin{array}{r} +57 \\ +56 \\ +\ \ 1 \\ \hline (1)\,14 \end{array}$$

56 = 9's complement of 43

(56 = 99 − 43)

10's complement

next higher power of 10 (less 1)

Ex.

$$\begin{array}{rr} +25 & 0001\ 1001 \\ -16 & 1111\ 0000 \\ \hline & 0000\ 1001 = 9 \end{array} \qquad \begin{array}{rr} +16 & 0001\ 0000 \\ -25 & 1110\ 0111 \\ \hline & 1111\ 0111 = -9 \end{array}$$

2's complements:

$+25 = 0001\ 1001 \rightarrow 1110\ 0110 + 1 = 1110\ 0111 = -25$

$+16 = 0001\ 0000 \rightarrow 1110\ 1111 + 1 = 1111\ 0000 = -16$

- signed decimal encoding

	0000	0	0
	0001	1	1
	.....	...	...
	0111	7	7
sign bit	1000	8	-8
	1001	9	-7
	1010	10	-6
	.....	...	...
	1110	14	-2
	1111	15	-1

e.g. 1010 = -6
↓ complement
0101
+ 1
0110 = 6

two's complement arithmetic

(-6) $\xrightarrow{\text{2's compl.}}$ (+6)

except (-8) → (-8) !sic! ⇒ need more bits

- other encodings possible, e.g. Gray code

	binary	# bits changing	GC	# bits changing
0	0000	1	0000	1
1	0001	2	0001	1
2	0010	1	0011	1
3	0011	3	0010	1
4	0100	1	0110	1
5	0101	2	0111	1
6	0110	1	0101	1
7	0111	4	0100	1
8	1000		1100	1
9			1101	1
10			1111	1
11			1110	1
12			1010	1
13			1011	1
14			1001	1
15			1000	1

reflect

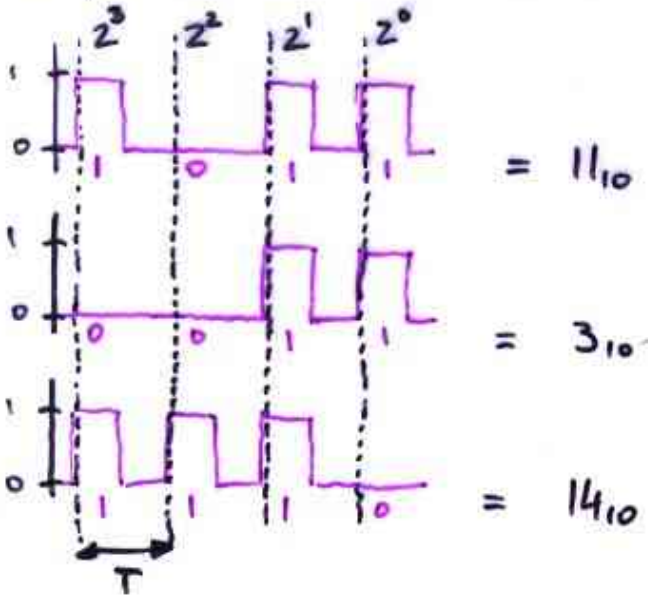
e.g. transducer

e.g. 0111 → 1000, 1-bit error can yield any number 0...15

1-bit error always ±1

- division: via several subtraction steps
  - how many times a number can be subtracted from another?
  - count gives the quotient
  - remainder gives the fractional part of the quotient

- implementation - ?



Waveforms can represent binary numbers

T ≡ clock period