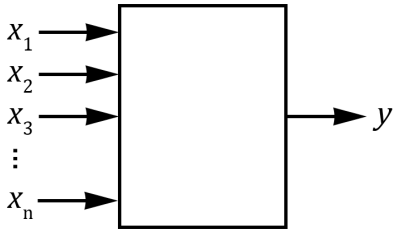
# Digital systems and basics of electronics

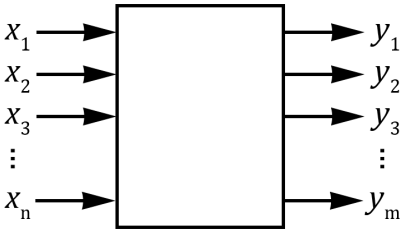
Adam Szmigielski
aszmigie@pjwstk.edu.pl
materials: $ftp(public) : //aszmigie/SYC/ENG$

# Combinational functional blocks - lecture 8

# Boolean function and combinational functional block




funkcja
boolowska




kombinacyjny
blok funkcjonalny

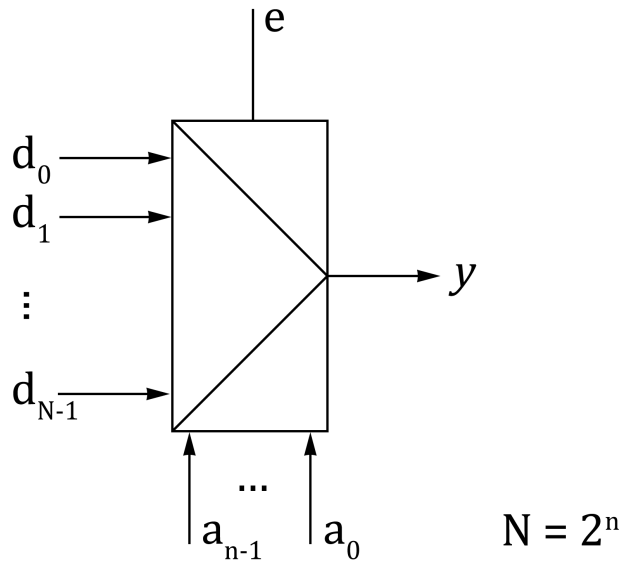
- *Combinational functional block* in digital technique is an combinational circuit with $n$ inputs and $m$ outputs, where $m, n = 1, 2, \ldots$ are natural numbers.
- *Boolean function* is a special case of *combinational functional block* - with only one output $m = 1$.

## Combinational functional blocks

Examples of *combinational functional blocks*:

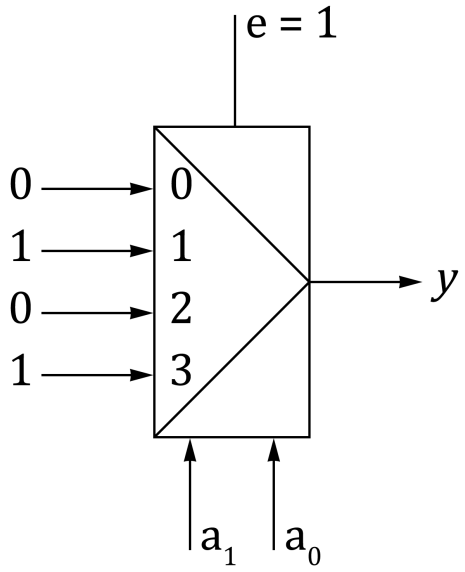
- Commutational (switching) circuit:
  - Multiplexers (MUX),
  - Demultiplexers (DMUX),
  - decoders, transcoders (DEC),
- arithmetical and logic circuits:
  - adders,
  - comparators,
  - ...
- others.

# Multiplekser (MUX)



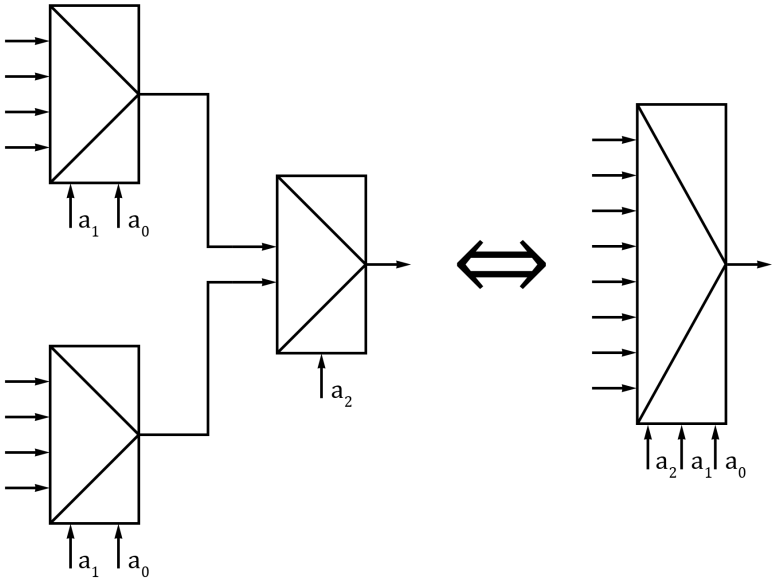

- Multiplexer has two kind of inputs -*select bits* i *data bits*,
- A multiplexer of $N = 2^n$ *data inputs* has $n$ *select bits*, which are used to select which input line to send to the output.

# Multiplexer as switching device



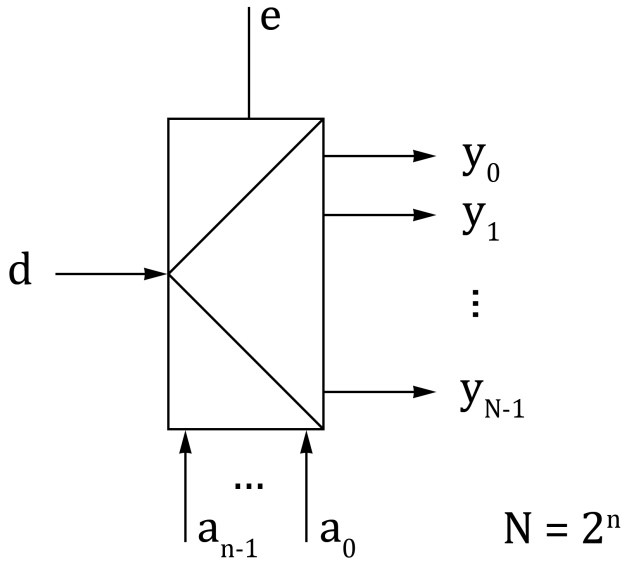

- Multiplexer works as switch,
- Given in Figure multiplexer realize Boolean function:
  $y = \overline{a_1} a_0 + a_1 a_0$
- Multiplexer sends to the output the signal on *data input* chosen by *select input*.

# Cascade configuration of MUXs



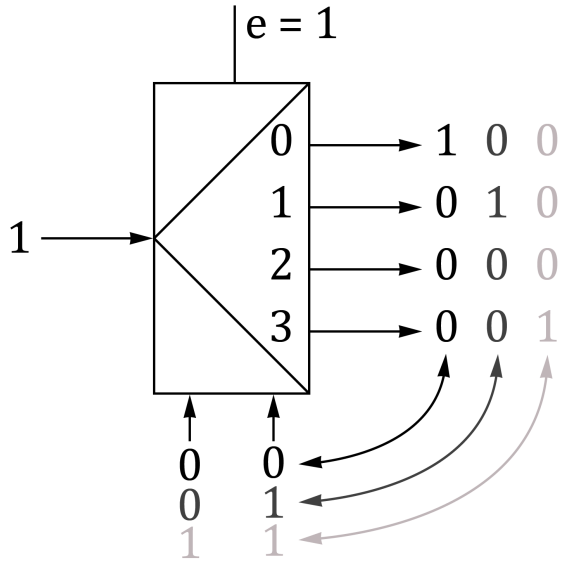

- The number of *data inputs* exponentially grows due to number of *select bits*. This is the reason why MUX with big (bigger then 4) number of select bits are not builded.
- Multiplexer with big number od select bits can be builded of smaller MUX.

# Demultiplexer (DMUX)



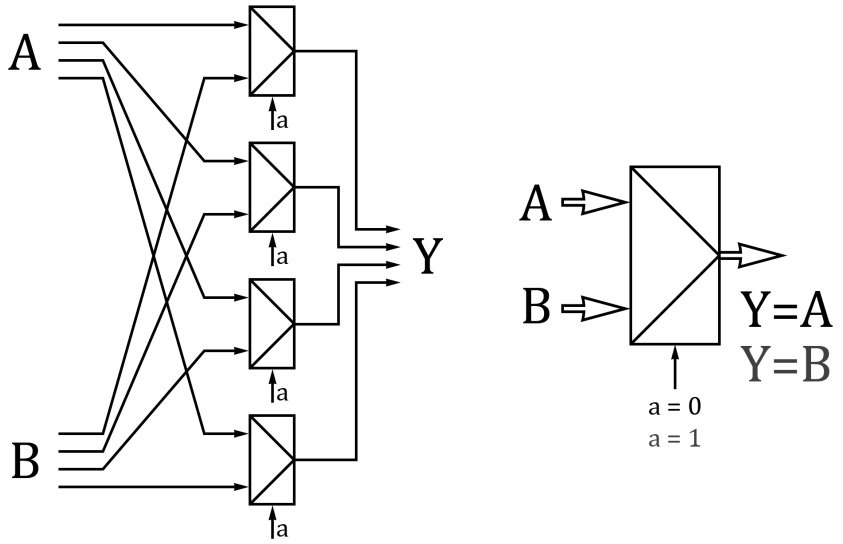

- *Demultiplexer* is a combinational circuit with one data input, $n$ select inputs, $N = 2^n$ outputs and one enable input.

# Demultiplexer as switching device



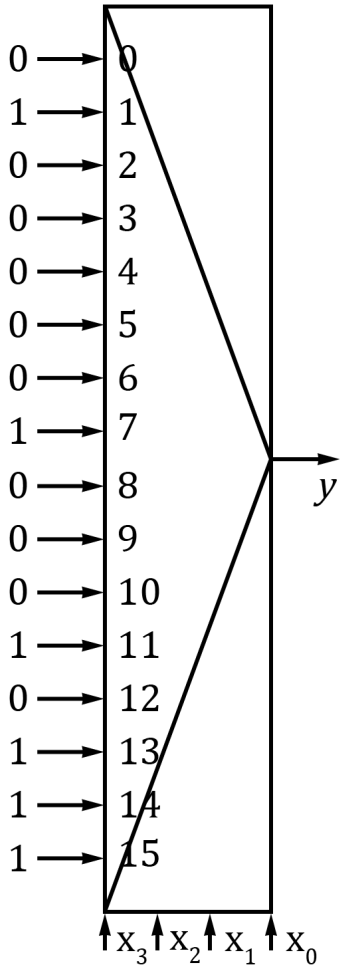

- Demultiplexer works as switch,
- Demultiplexer sends the state of input to output chosen by *select input*.

# Multiplexers and demultiplexers grouping




- Switching blocks are usually grouped from elementary multiplexers.
- Group multiplexer (in our case 4-bits) can be connected to data bus due to state of select bits.

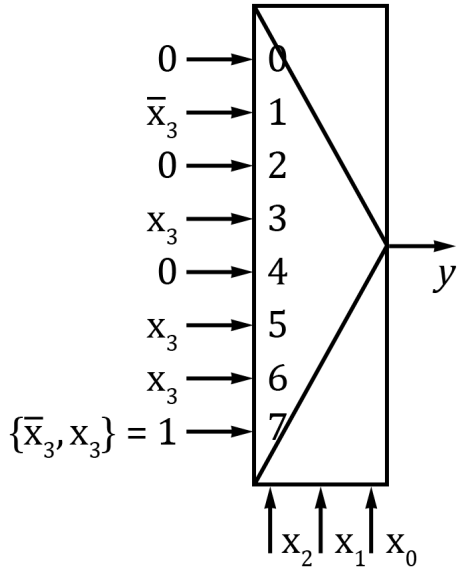
# Realization of Boolean function with multiplexer




$y = \Sigma(1, 7, 11, 13, 14, 15)$

# Realization of Boolean function with 3-select bits multiplexer

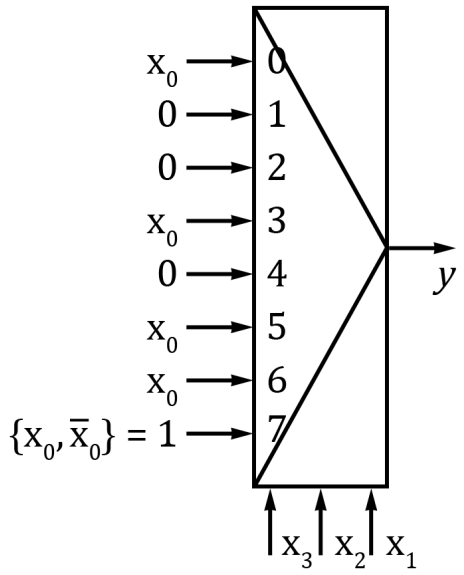
| $y$ | $x_3$ | $x_2x_1x_0$ | $x_2x_1x_0$ |
|---|---|---|---|
| 1 | 0 | 001 | 1 |
| 7 | 0 | 111 | 7 |
| 11 | 1 | 011 | 3 |
| 13 | 1 | 101 | 5 |
| 14 | 1 | 110 | 6 |
| 15 | 1 | 111 | 7 |




- $y = \sum(1, 7, 11, 13, 14, 15)$
- On firs input there is $\overline{x_3}$ - negation is needed.

# Realization of Boolean function with 3-select bits multiplexer - choice of select inputs

| $y$ | $x_3x_2x_1$ | $x_0$ | $x_3x_2x_1$ |
|---|---|---|---|
| 1 | 000 | 1 | 0 |
| 7 | 011 | 1 | 3 |
| 11 | 101 | 1 | 5 |
| 13 | 110 | 1 | 6 |
| 14 | 111 | 0 | 7 |
| 15 | 111 | 1 | 7 |




- $y = \sum(1, 7, 11, 13, 14, 15)$
- This time negation is not needed.

## Realization of Boolean function with 2-select bits multiplexer

| $x_3x_2 \backslash x_1x_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 0 |

- $y = \sum(1, 7, 11, 13, 14, 15)$
- How to choose select inputs?

# Continue - choice of select inputs

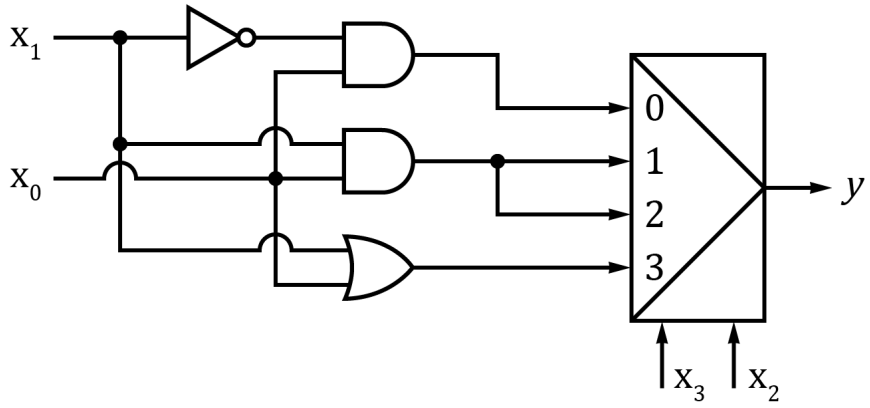
| $x_3x_2 \backslash x_1x_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 0 |

Variables $x_3x_2$ have been chosen to select bits. On data input (chosen by $x_3x_2$) we need to applied the function of the rest variables $f(x_1, x_0)$. This function is described by corresponded rows of Karnough map.

- $x_3x_2 = 00 \Longrightarrow f(x_1, x_0) = \overline{x_1}x_0$
- $x_3x_2 = 01 \Longrightarrow f(x_1, x_0) = x_1x_0$
- $x_3x_2 = 11 \Longrightarrow f(x_1, x_0) = x_1 + x_0$
- $x_3x_2 = 10 \Longrightarrow f(x_1, x_0) = x_1x_0$

# Continue - realization

- $x_3x_2 = 00 \Longrightarrow f(x_1, x_0) = \overline{x_1}x_0$
- $x_3x_2 = 01 \Longrightarrow f(x_1, x_0) = x_1x_0$
- $x_3x_2 = 11 \Longrightarrow f(x_1, x_0) = x_1 + x_0$
- $x_3x_2 = 10 \Longrightarrow f(x_1, x_0) = x_1x_0$



# Arithmetic and logic systems

# Popular number codes used in digital technology

- **Binary code** - positional code with base 2.
- **1 of N code** - the $n$-bit word includes one and only one bit with value “1”. The position of "1"determines coded value.
- **HEX** - hexadecimal code
- **Graya code** - is a binary numeral system where two successive values differ in only one bit.
- **Binary-coded decimal (BCD)** - is an encoding for decimal numbers in which each digit is represented by its own binary sequence.
- **SM** - Sign Magnitude
- **1’s compliment**
- **2’s compliment**

# Binary Code

| position | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| value | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| value | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| bits | $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |

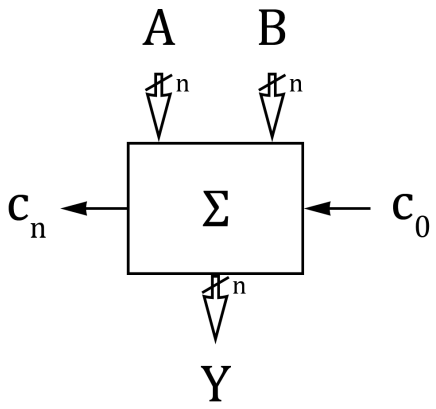
- Positional code with base 2,
- Numbers without sign,
- Value of binary number (N- length of coded world),
  $Value = \sum_{i=0}^{N-1} 2^i \cdot b_i$,
- Digit's value depends on position - $b_i = 2^i$ (index starts from 0),
- $2^N$ different code's value (code is full).

# 1 of N code

| Decimal value | Binary value | 1 of N code |
|---|---|---|
| 0 | 0000 | 0000000001 |
| 1 | 0001 | 0000000010 |
| 2 | 0010 | 0000000100 |
| 3 | 0011 | 0000001000 |
| 4 | 0100 | 0000010000 |
| 5 | 0101 | 0000100000 |
| 6 | 0110 | 0001000000 |
| 7 | 0111 | 0010000000 |
| 8 | 1000 | 0100000000 |
| 9 | 1001 | 1000000000 |

# Decoder

**Decoder** converts *binary code* into *1 of N code*.




- Special case of demultiplexer is an decoder with input $d$ equal "1". This constat input is not externally accessible.

# Binary-coded decimal (BCD)

| Decimal digit | coded decimal digit |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

eg. Decimal number 123 consists of 3 digits. If each digit would be binary coded we obtain Binary-coded decimal number: 0001 0010 0011.

# Sign-and-magnitude

*Most Significant Bit* $b_{N-1}$ represents sign (to 0 ($b_{N-1} = 0$) for a positive number, and set to 1 ($b_{N-1} = 1$) for a negative number) eg.:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $-24_{10}$ | = | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| $118_{10}$ | = | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| $-14_{10}$ | = | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

$value = (-1)^{b_{N-1}} \cdot \sum_{i=0}^{N-2} 2^i \cdot b_i$

- The sign-and-magnitude code is not weighed code,
- code's range $< -(2^{N-1} - 1), 2^{N-1} - 1 >$,
- $2^N - 1$ combinations - 0 would have two representations (combinations 10000000 (minus zero) is not valid),
- Checking the sign and magnitude operations are complicated.

## One's complement code

- *Most Significant Bit* ($b_{N-1}$) represents sign - 0 positive number, 1 negative number. The meaning of the rest bits depends of sign bit:
  - If sign bit is 0 (positive number), the rest bits represent magnitude of number the same like in *Sign Magnitude Codding*
  - When sign bit is 1 (negative number), the rest **negated** bits represent the magnitude of negative number.
- Only for negative numbers there is codding difference,
- The range of *1's compliment* and *Sign Magnitude Codding* is the same.

## 1's compliment codding

- In 1's compliment codding 0 has two representations: 000000...00 and 111111...11.
- Using 1's compliment codding negative numbers can be written following: negate bits representing magnitude of number (sign bit set as 1) eg. for 8-bit number:

| | | |
|---|---|---|
| Sign and Magnitude: | 11010110 | (decimal -86) |
| One's compliment: | 10101001 | (decimal -86) |

# Two's compliment code

The Most Significant bit is negative, the rest bits are positive

$$Value = -(2^{N-1}) \cdot b_{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i$$

- MSB identifies if number is positive or negative,
- Code range: $< -2^{N-1}, 2^{N-1} - 1 >$,
- $2^N$ combinations (full code), 0 has only one representation,
- Positive number (from interval $< 0, 2^{N-1} - 1 >$) have the same two's compliment representation like binary and sign-and-magnitude representation,
  $(0, b_{N-2}, \ldots, b_1, b_0)_{U2} = \sum_{i=0}^{N-2} 2^i \cdot b_i$
- Weighted code, the MSB has negative value

- The negative numbers are represented as follow:
  $(1, b_{N-2}, \ldots, b_1, b_0)_{U2} = -2^{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i$
- *2's compliment* is weighted code: code range is not symmetrical, negation of number $-2^{N-1}$ fails (eg. for positive number $N = 8$ number $-128$ falls in range, but number 128 does not).
- Adding overflow, eg. for $N = 8$:
  $(127)_{U2} + (4)_{U2} = (-125)_{U2}$ - error
- Incremented number 127 gives $-128$.

## Negations in two's compliment

$$-(value)_{U2} = \overline{(value)_{U2}} + 1$$

To calculate contrary number given in 2's compliment code negate all bits and add 1 to result, eg.:

| | | |
|---|---|---|
| $7_{10}$ | | $(00000111)$ |
| bit negation: | | $(11111000)$ |
| adding bit: | $+$ | $(00000001)$ |
| result: $-7_{10}$ | $=$ | $(11111001)_{U2}$ |

## Two's compliment adding and subtraction

- **Adding** like binary code, argument's sign is not important,
- Carry of MSB is ignored,
- Overflow $\Longleftrightarrow$ sum of two positive numbers is positive or sum of two negative number is negative,
- **Subtraction -** adding negative number eg.:

$$a - b = a + (-b)$$

- only operation of *negation* and *adding* is needed.

## 2's compliment subtraction - example

$$
\begin{array}{rcl}
25 + (-1): & & \\
25: & & 00011001 \\
-1: & + & 11111111 \\
\hline
(c_7 = 1): & = & 00011000_{U2} = 24_{10}
\end{array}
$$

$$
\begin{array}{rcl}
25 + (-56): & & \\
25: & & 00011001 \\
-56: & + & 11001000 \\
\hline
(c_7 = 0): & = & 11100001_{U2} = -31_{10}
\end{array}
$$

## 2's compliment adding - example

$$
\begin{array}{rcl}
25+1 & & \\
25: & & 00011001 \\
+1: & + & 00000001 \\
\hline
(c_7=0): & = & 00011010_{U2} = 26_{10}
\end{array}
$$

$$
\begin{array}{rcl}
(-25)+(-56): & & \\
-25: & & 11100111 \\
-56: & + & 11001000 \\
\hline
(c_7=1): & = & 10101111_{U2} = -81_{10}
\end{array}
$$

## 2's compliment overflow - example

| | | |
|---|---|---|
| $112 + 113:$ | | |
| $112:$ | | $01110000$ |
| $113:$ | $+$ | $01110001$ |
| $(c_7 = 0, c_6 = 1):$ | $=$ | $11100001$ - **overflow** |
| $(-75) + (-56):$ | | |
| $-75:$ | | $10110101$ |
| $-56:$ | $+$ | $11001000$ |
| $(c_7 = 1, c_6 = 0):$ | $=$ | $01111101$ - **overflow** |

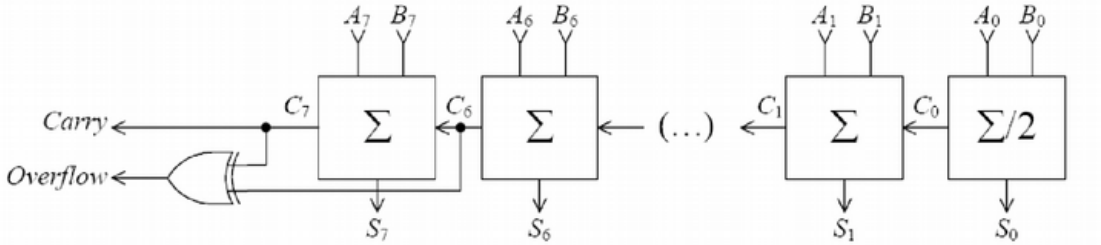
# Hardware 2's compliment overflow identification

The *Most Significant Bit* adding:

- The range overrun in U2 can be identified by analyzing the carry: incoming $C_{IN}$ and generated $C_{OUT}$ by the oldest bit,

| A | B | $C_{IN}$ | $C_{OUT}$ | S | OFL |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

$$\Rightarrow \quad OFL = C_{IN} \oplus C_{OUT}$$




- Exceeding the range occurs if and only if both transfers $C_{IN}$ and $C_{OUT}$ are the opposite sign.

## Real number representation

- *Fixed point*
- *Floating point*

## Fixed point representation

- Fixed point numbers can be interpreted as 1's compliment 2's compliment or *Sign and magnitude* numbers - the MSB has the same meaning like in those codes.
- Fixed point codding causes *truncation error*,
- Codding accuracy directly depends on world length,
- Some of rational numbers have no proper representation,
- Irrational numbers have no proper representation, always are codded with truncation error.

## Fixed point representation - example

Maintaining the assumption that 5 of the oldest bits is intended for the *integer* and the remaining 3 bits for *the fractional part*. In addition, we assume that the number is written in the 2's compliment code:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| value: | $-2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ |
| value: | $-16$ | $8$ | $4$ | $2$ | $1$ | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ |

- The sample string of bits 11001110 is then equal to: $-16 + (8 + 1 + \frac{1}{2} + \frac{1}{4}) = -6\frac{1}{4}$,
- The range of represented numbers is within the range $< -16, 15\frac{7}{8} >$,
- Real numbers are represented with an error not greater than $\frac{1}{8}$.

## Floating point representation

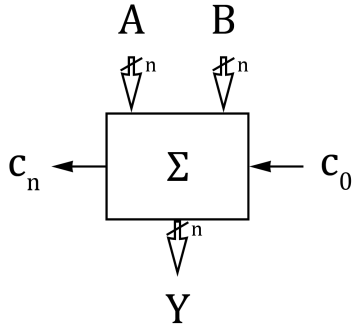
Generally: Number is represent by *mantissa* and *exponent*

| mantissa | exponent |
|---|---|

Example:

| | mantissa | exponent |
|---|---|---|
| decimal: | $2,14$ | $10^3$ |
| binary: | $0,10001$ | $2^{010}$ |

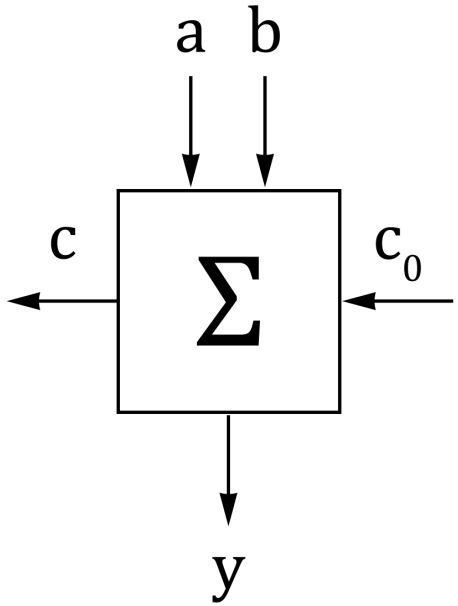
# Adder




- Adder realizes operation of arithmetical adding $Y = A + B + c_0$. The value of sum of $n$-bits natural numbers $A$ and $B$ is produced on output.
- Overload the summing operation is reported by carry bit $c_n$.
- Carry bit may be interpreted as the most significant bit of result.

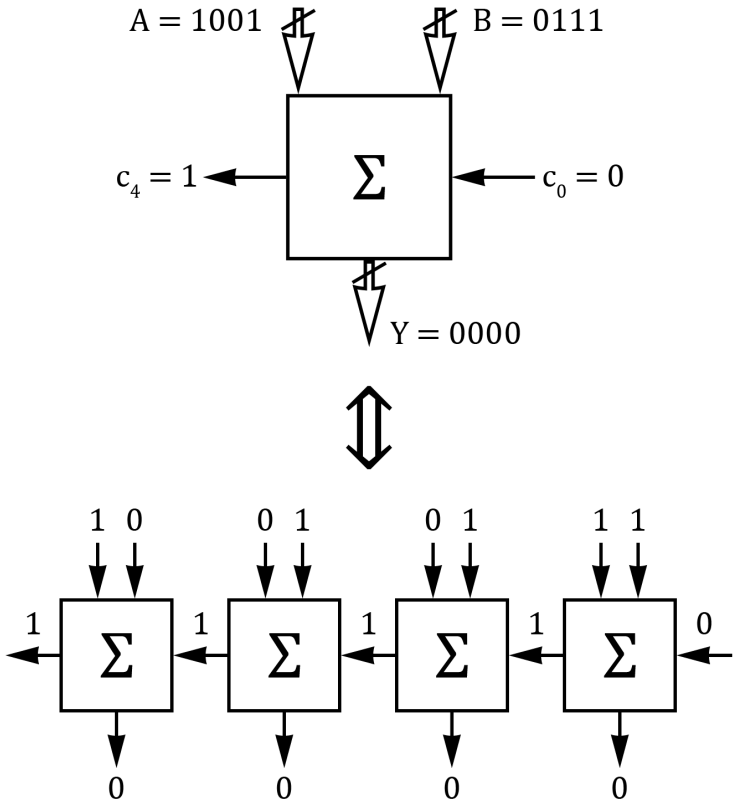
# One-bit adder

| a | b | $c_o$ | c | y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |




- $y = a \oplus b \oplus c_o$
- $c = ab + ac + bc_o$

# Cascade adder building




- Adder consists of cascade connection of one-bit adder.
- One-bit adder has inputs $a_i$, $b_i$ i $c_i$ and outputs $y_i$ i $c_{i+1}$.

## Adding

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $76_{10}$ | | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $188_{10}$ | + | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| $194_{10}$ | = | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| carry | | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

- Adding two bits $a, b$:

$$a_i, b_i, c_i \Rightarrow s_i, c_{i+1}$$

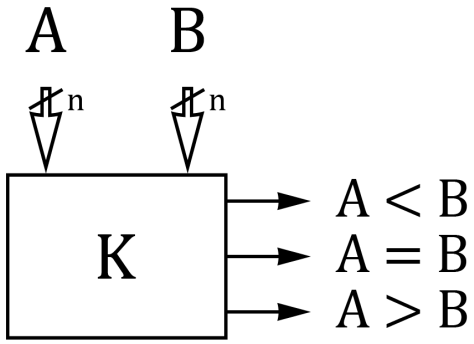
($c$ - carry, $s$ - adding result)

## Overflow

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $152_{10}$ | | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| $118_{10}$ | + | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| $14_{10}$ ? | = | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| carry | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

- Carry of Most Significant Bit ($c_{N-1} = 1$) means overflow of $N$-bit world,
- Alternatively: Overflow suggests that adding result is a $N + 1$- bit number. Carry od MSB $N + 1$ becomes the new MSB.

## Comparator



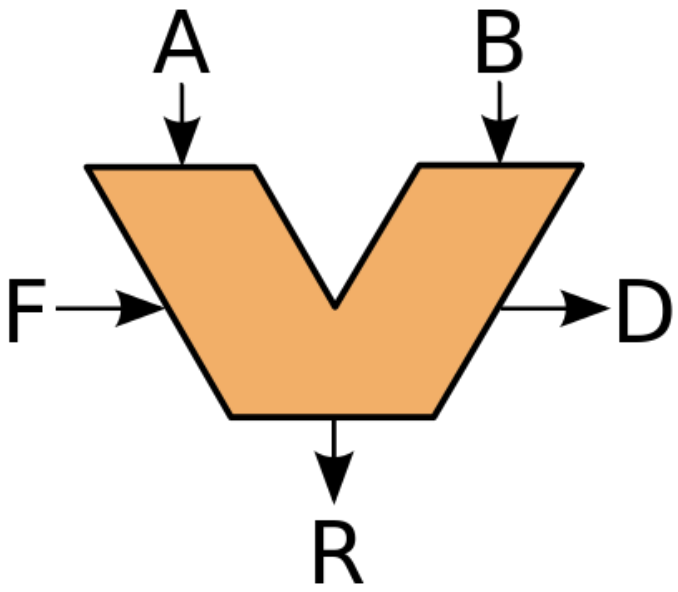

- Comparator is used to compare $n$-bits numbers,
- Outputs identify if one number equal second, bigger or less.

## Other arithmetical blocks

Others arithmetical blocks are known, eg.:

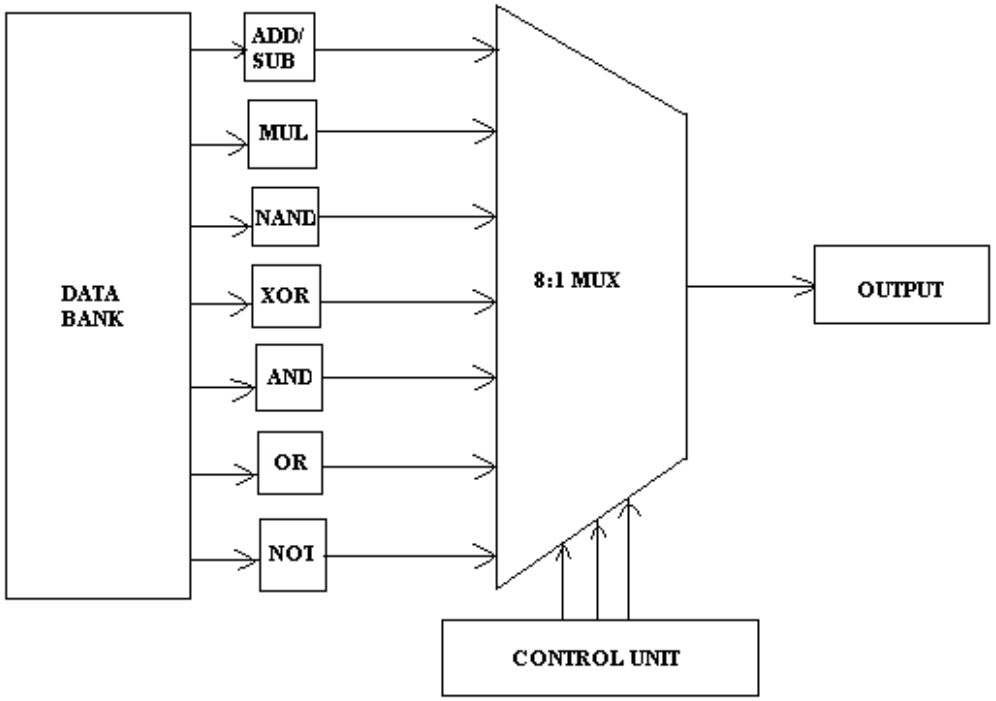
- subtractor,
- Multiplier,
- Divider,
- Negator,
- etc.

## Arithmetic Logic Unit (ALU)




An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers.

# Implementation of ALU on the multiplexer



## Tasks for laboratory

1. Using 4-selection bit multiplexer implement given function.
2. Using 3-selection bit multiplexer and at least one negation gate implement given function.
3. Implement a system that identifies the range overflow for numbers in 2's compliment code. The system should also check whether the overflow occurred as a result of adding two positive or negative numbers.
4. Implement ALU that performs 4 operations: SUB, NOT, XOR and AND. The operation number determines the order of operations. ALU should additionally set the bit Z (1 when the result of the operation is zero) and the bit C of the range overrun in 2's compliment code (1 when the range was exceeded).