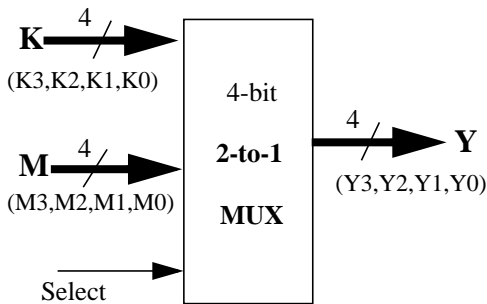


DIGITAL CIRCUIT DESIGN & ALGEBRAIC SIMPLIFICATION EXAMPLES

A 4-bit 2-to-1 Multiplexer

A 4-bit 2-to-1 MUX is a selector. It selects between two 4-bit inputs based on a select signal. As a black box, it has 9 inputs and 4 outputs as shown below. Out of the nine inputs, eight are **data** inputs and one is a **control** input. The control input is Select and the two 4-bit inputs, K and M, are data inputs. The 4-bit output is Y is a **data** output.



If Select = 0 then Y = K
else Y = M

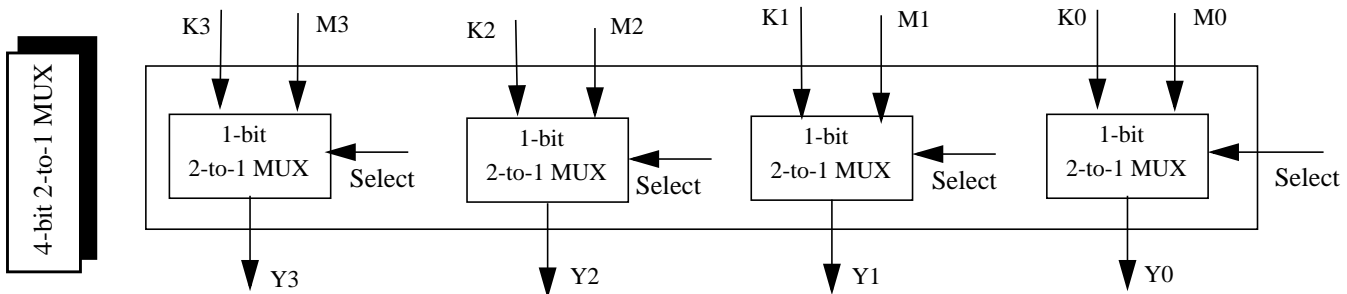
Since there are 9 inputs, we need to partition it into simpler pieces !
We have to obtain the operation table of the 4-bit 2-to-1 MUX :

Select	Operation
0	Y = K
1	Y = M

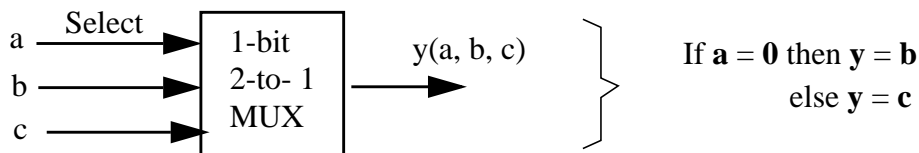
The major operations are **not** clear on the above operation table.
We need to get a **different**, more detailed operation table :

Select	Operation
0	Y3 = K3 ; Y2 = K2 ; Y1 = K1 ; Y0 = K0
1	Y3 = M3 ; Y2 = M2 ; Y1 = M1 ; Y0 = M0

There are **four** identical major operations : **Four** cases of 1-bit 2-to-1 multiplexing ! Then, we partition the 4-bit 2-to-1 MUX into **four** blocks. Each block is a 1-bit 2-to-1 MUX :

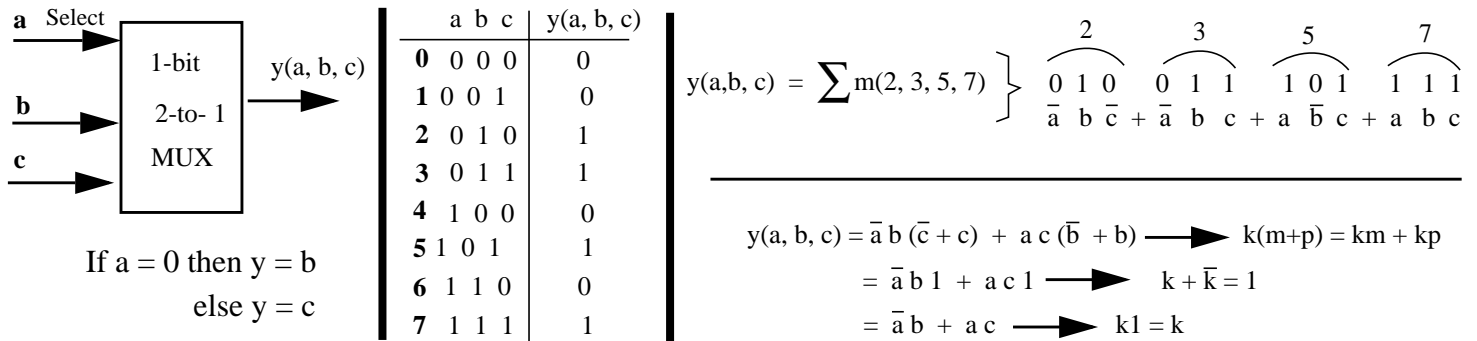


A 1-bit 2-to-1 MUX can be immediately implemented by using Switching Algebra since it has three (3) inputs. The MUX selects between two of its data inputs based on its select signal :

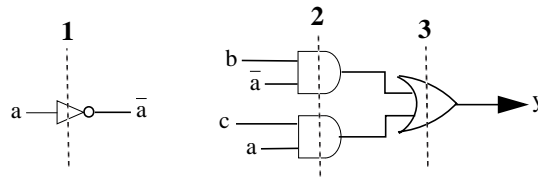


Two inputs (b and c) are **data inputs**.. The third input (a) is the **control input**, the select input. The MUX is a 1-bit MUX since when an input is selected, there is only one data line selected. The single output, y, is always equal to either b or c at any time.

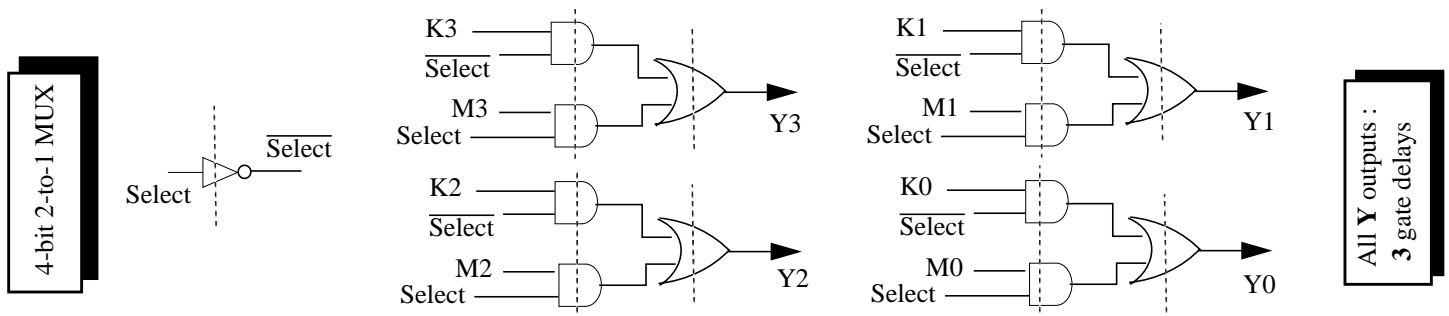
The input/output relationship (the operation, purpose) is given above textually. Below, we give the description by using a truth table, a minterm list, a canonical SOP expression and a minimal SOP expression :



The minimal 2-level AND-OR gate network is below. As the gate network shows, the MUX has three gate delays :



We can show the implementation of the 4-bit 2-to-1 MUX by using four 1-bit 2-to-1 MUXes :

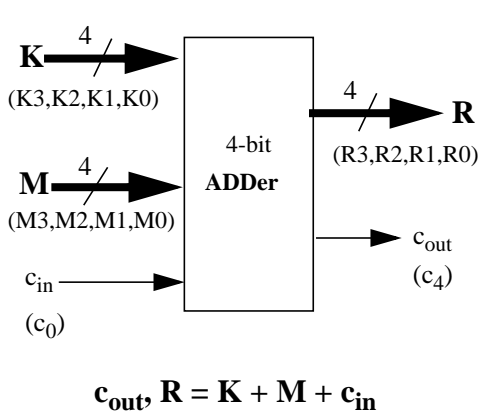


One can develop any **k-bit** 2-to-1 MUX from 1-bit 2-to-1 MUXes. For example, 2-bit 2-to-1 MUXes, 8-bit 2-to-1 MUXes, etc. are implemented by using 1-bit 2-to-1 MUXes. The above 4-bit 2-to-1 MUX will be implemented in the **lab** by only changing the input and output names.

Note also that there are **k-bit 4-to-1** MUXes, **k-bit 8-to-1** MUXes, etc.

A 4-bit ADDer

A 4-bit adder adds two 4-bit numbers. It has 9 inputs and 5 outputs as shown below. The 1-bit input is carry_{in} and the 4-bit inputs are K and M. The outputs are a 4-bit R and a 1-bit carry_{out} :



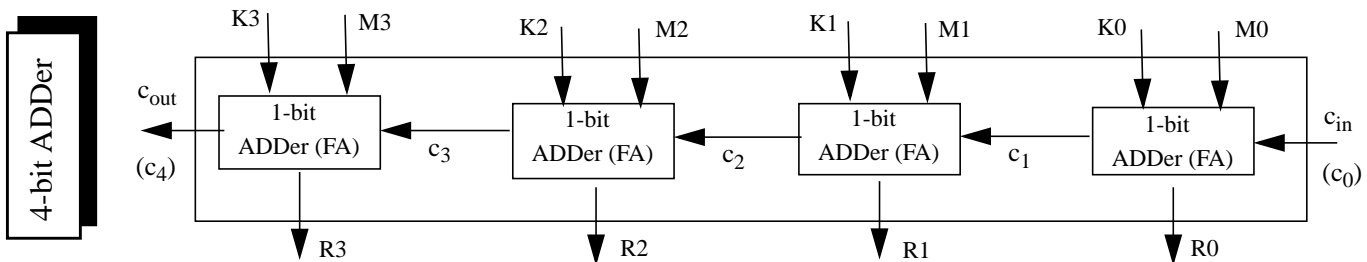
Since there are 9 inputs, we need to partition it into simpler pieces !
We have to obtain the operation table of the 4-bit ADDer :

$$\begin{array}{c} \text{Operation} \\ \hline c_{out}, R = K + M + c_{in} \end{array}$$

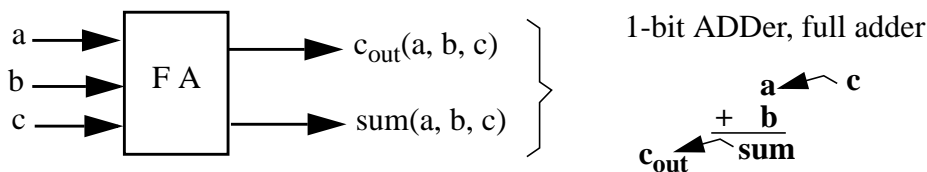
The major operations are not clear on this operation table.
We need to get a **different**, more detailed operation table :

$$\begin{array}{c} \text{Operation} \\ \hline \begin{array}{l} c_4, R_3 = K_3 + M_3 + c_3 \\ c_3, R_2 = K_2 + M_2 + c_2 \end{array} \quad \Bigg| \quad \begin{array}{l} c_2, R_1 = K_1 + M_1 + c_1 \\ c_1, R_0 = K_0 + M_0 + c_0 \end{array} \end{array}$$

There are **four** identical major operations : **Four** 1-bit ADDitions ! Each 1-bit ADDer is known as a full adder (FA). Then, we partition the 4-bit ADDer into **four** blocks. Each block is a 1-bit ADDer :



A 1-bit ADDer, a full adder, can be immediately implemented by using Switching Algebra since it has three (3) inputs. The full adder adds the three inputs :



All three inputs (a, b and c) are **data** inputs. There are two outputs, sum and c_{out}, which are **data** outputs. Outputs sum and c_{out} are always equal to the addition of a, b and c at any time.

The input/output relationship (the operation, purpose) is given above textually. Below, we give the description by using a truth table, two minterm lists and two canonical SOP expressions.

a	b	c	$c_{out}(a, b, c)$	$sum(a, b, c)$
0	0	0	0	0
1	0	0	0	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	1	0
7	1	1	1	1

$$c_{out}(a,b,c) = \sum m(3, 5, 6, 7) \left\{ \begin{array}{l} \overline{0} \overline{1} \overline{1} \\ \overline{1} \overline{0} \overline{1} \\ \overline{1} \overline{1} \overline{0} \\ \overline{1} \overline{1} \overline{1} \end{array} \right. = \overline{a} b c + a \overline{b} c + a b \overline{c} + a b c$$

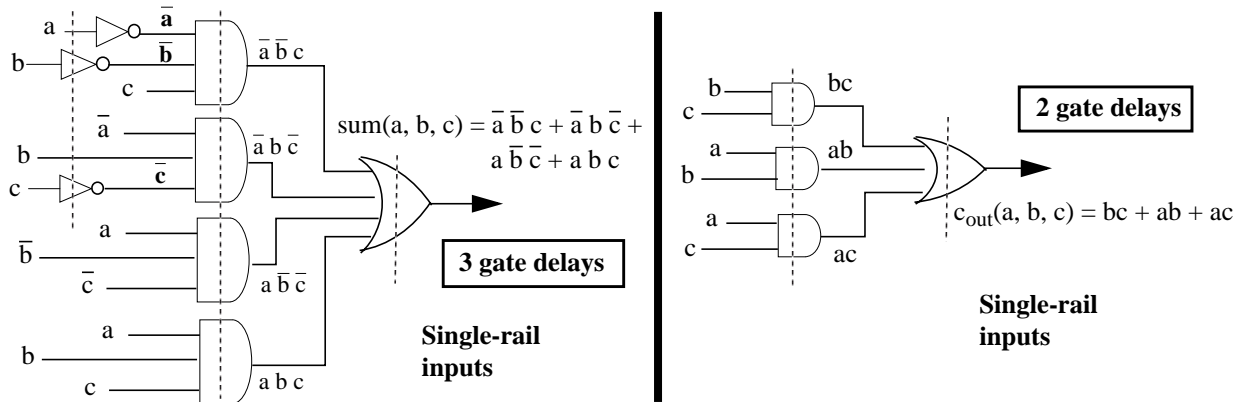
$$sum(a,b,c) = \sum m(1, 2, 4, 7) \left\{ \begin{array}{l} \overline{0} \overline{0} \overline{1} \\ \overline{0} \overline{1} \overline{0} \\ \overline{1} \overline{0} \overline{0} \\ \overline{1} \overline{1} \overline{1} \end{array} \right. = \overline{a} \overline{b} c + \overline{a} b \overline{c} + a \overline{b} \overline{c} + a b c$$

$sum(a, b, c) = \overline{a} \overline{b} c + \overline{a} b \overline{c} + a \overline{b} \overline{c} + a b c$ ← The canonical $sum(a, b, c)$ expression is also the minimal expression. It cannot be simplified !

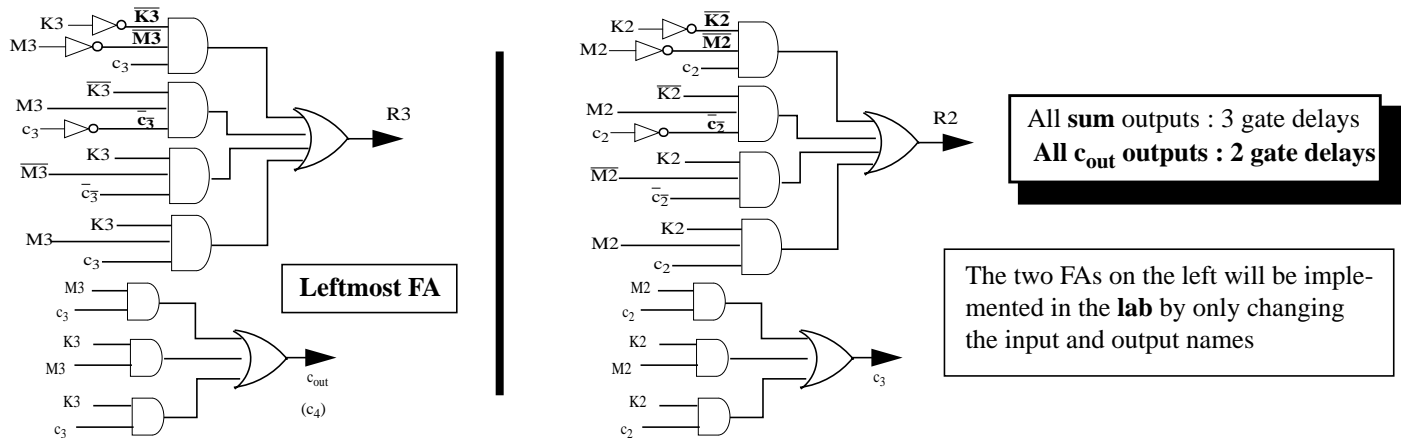
$c_{out}(a, b, c) = \overline{a} b c + a \overline{b} c + a b \overline{c} + a b c$
 $= bc(\overline{a} + a) + a \overline{b} c + a b \overline{c} \quad k(m+p) = km + kp$

 $= b(c + a) + ac \quad k + \overline{k}m = k + m$
 $= bc + ab + ac \quad k(m+p) = km + kp$

The minimal 2-level AND-OR gate networks are below :



The leftmost two FAs of the 4-bit ADDer are shown below :



A 2-bit Special-Purpose Unsigned Binary Comparator

$$z(a,b,c,d) = a(b + bc) + a(\bar{d} + d\bar{c}) + \bar{c} \bar{d}(\bar{d} + (a+\bar{a})) + abc\bar{c} + \bar{a}b\bar{c} \quad \text{a nonminimal expression}$$

$$= a(b + bc) + a(\bar{d} + \bar{c}) + \bar{c} \bar{d}(\bar{d} + (a+\bar{a})) + abc\bar{c} + \bar{a}b\bar{c} \quad k + \bar{k}m = k + m$$

$$= a(b + bc) + a\bar{d} + a\bar{c} + \bar{c} \bar{d}(\bar{d} + (\mathbf{a} + \bar{\mathbf{a}})) + abc\bar{c} + \bar{a}b\bar{c} \quad k(m+n) = km + kn$$

$$= a(b + bc) + a\bar{d} + a\bar{c} + \bar{c} \bar{d}(\bar{d} + \mathbf{1}) + abc\bar{c} + \bar{a}b\bar{c} \quad k + \bar{k} = 1$$

$$= a(b + bc) + a\bar{d} + a\bar{c} + \bar{c} \bar{d}\mathbf{1} + abc\bar{c} + \bar{a}b\bar{c} \quad k + 1 = 1$$

$$= a(b + bc) + a\bar{d} + a\bar{c} + \bar{c} \bar{d} + \mathbf{abc} + \bar{\mathbf{a}}b\bar{c} \quad k1 = k$$

$$= a(b + bc) + a\bar{d} + a\bar{c} + \bar{c} \bar{d} + bc(\mathbf{a} + \bar{\mathbf{a}}) \quad k(m+n) = km + kn$$

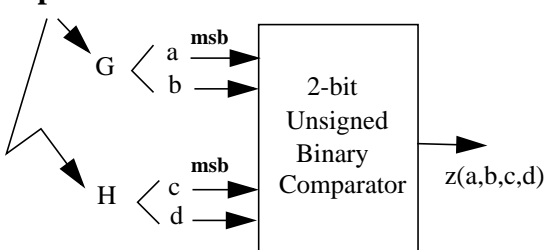
$$= a(b + bc) + a\bar{d} + a\bar{c} + \bar{c} \bar{d} + \mathbf{bc1} \quad k + \bar{k} = 1$$

$$= a(\mathbf{b + bc}) + a\bar{d} + a\bar{c} + \bar{c} \bar{d} + bc\bar{c} \quad k1 = k$$

$$= ab + a\bar{d} + a\bar{c} + \bar{c} \bar{d} + bc\bar{c} \quad k + km = k$$

$ab + a\bar{d} + a\bar{c} + \bar{c} \bar{d} + bc\bar{c}$ → the minimal SOP expression

Single-rail inputs

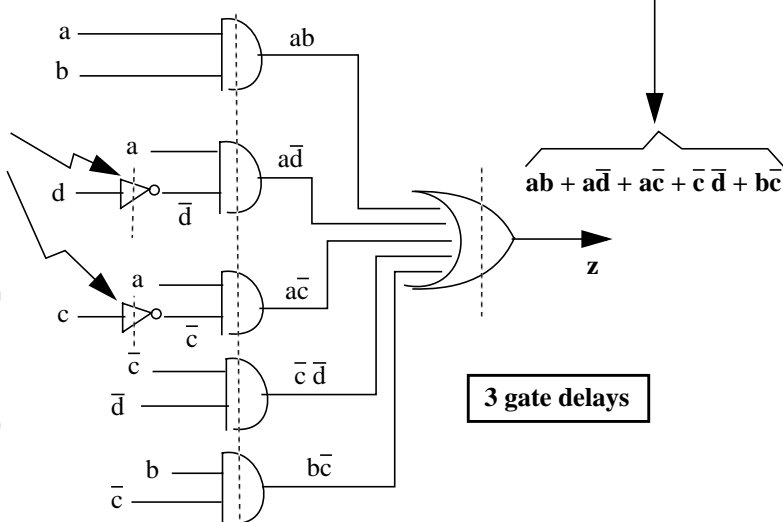


G and H are 2-bit

Unsigned Binary numbers

If G is \geq H then $z(a, b, c, d) = 1$
 else $z(a, b, c, d) = 0$

Single-rail inputs : Inverters needed



An Algebraic Simplification Example

$$w(a,b,c,d) = \mathbf{bcd} + a\bar{b}\bar{c} + \mathbf{bd}(\mathbf{ac} + \bar{\mathbf{a}}\bar{\mathbf{c}}) + c(a\bar{b} + \bar{a}\bar{d} + d) \quad \text{a nonminimal expression}$$

$$= bd(\mathbf{c} + \mathbf{ac} + \bar{\mathbf{a}}\bar{\mathbf{c}}) + a\bar{b}\bar{c} + c(a\bar{b} + \bar{a}\bar{d} + d) \quad k(m+n) = km+kn$$

$$= bd(\mathbf{c} + \mathbf{a} + \bar{\mathbf{a}}) + a\bar{b}\bar{c} + c(a\bar{b} + \bar{a}\bar{d} + d) \quad k + \bar{k}m = k + m$$

$$= bd(\mathbf{c} + \mathbf{1}) + a\bar{b}\bar{c} + c(a\bar{b} + \bar{a}\bar{d} + d) \quad k + \bar{k} = 1$$

$$= \mathbf{bd1} + a\bar{b}\bar{c} + c(a\bar{b} + \bar{a}\bar{d} + d) \quad k + 1 = 1$$

$$= bd + a\bar{b}\bar{c} + \mathbf{c(a\bar{b} + \bar{a}\bar{d} + d)} \quad k1 = k$$

$$= bd + \mathbf{a\bar{b}\bar{c}} + \mathbf{abc} + \bar{a}c\bar{d} + cd \quad k(m+n) = km + kn$$

$$= bd + a\bar{b}(\mathbf{c} + \mathbf{c}) + \bar{a}c\bar{d} + cd \quad k(m+n) = km + kn$$

$$= bd + \mathbf{a\bar{b}1} + \bar{a}c\bar{d} + cd \quad k + \bar{k} = 1$$

$$= bd + a\bar{b} + \mathbf{a\bar{c}\bar{d}} + \mathbf{cd} \quad k1 = k$$

$$= bd + a\bar{b} + \mathbf{c(d + \bar{a}\bar{d})} \quad k(m+n) = km + kn$$

$$= bd + a\bar{b} + \mathbf{c(d + \bar{a})} \quad k + \bar{k}m = k + m$$

$$\longrightarrow = bd + a\bar{b} + \mathbf{cd} + \bar{a}c \quad k(m+n) = km + kn \longrightarrow \text{It seems as if we cannot simplify anymore !}$$

$$\longrightarrow = bd + a\bar{b} + \mathbf{cd(a + \bar{a})(b + \bar{b})} + \bar{a}c \quad k(m + \bar{m}) = k \longrightarrow \text{We make it more complex !}$$

$$= bd + a\bar{b} + (\mathbf{acd} + \bar{\mathbf{a}}\bar{\mathbf{c}}\bar{\mathbf{d}})(\mathbf{b} + \bar{\mathbf{b}}) + \bar{a}c \quad k(m+n) = km + kn$$

$$= bd + \mathbf{a\bar{b}} + abcd + \mathbf{a\bar{b}cd} + \bar{\mathbf{a}}\bar{\mathbf{b}}cd + \bar{a}c \quad k(m+n) = km + kn$$

$$= bd + a\bar{b} + abcd + \bar{\mathbf{a}}\bar{\mathbf{b}}cd + \bar{\mathbf{a}}\bar{\mathbf{c}} \quad k + km = k$$

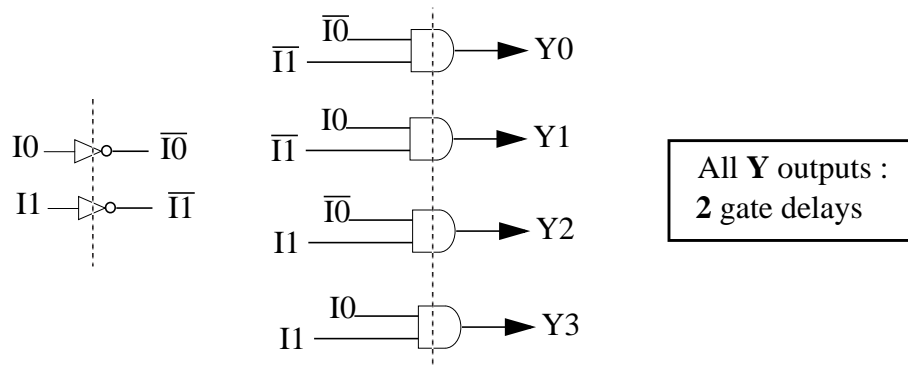
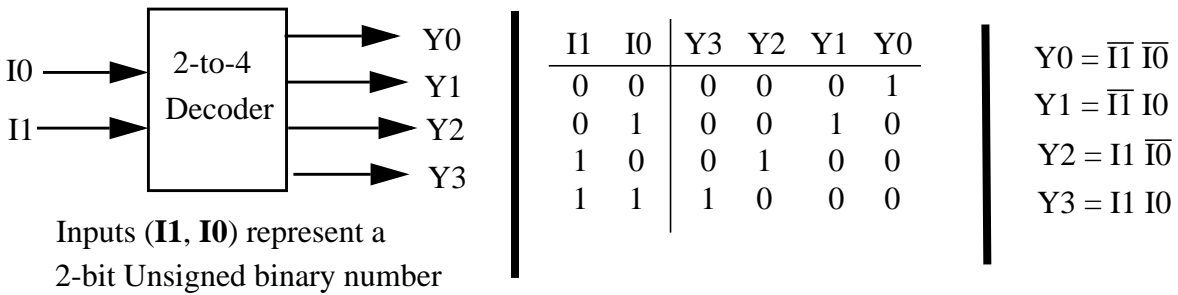
$$= \boxed{\mathbf{bd + a\bar{b} + \bar{a}c}} \quad k + km = k$$

the minimal SOP expression !

A 2-to-4 Binary Decoder

The most common decoder type is the **binary** decoder which has “k” **data** inputs and 2^k **data** outputs. If the decoder is a 2-to-4 decoder, then “k” is 2 and so there are $2^2 = 4$ outputs.

The “k” inputs represent an Unsigned binary number. The outputs decode the unsigned number represented by the “k” inputs. For example if the inputs represent $(3)_{10}$, Output 3 is 1 and the other outputs are 0. Below, the development of the 2-to-4 decoder is shown.



The decoder with four outputs requires only four gates, four AND gates. The outputs have at most two gate levels to generate the outputs. Therefore, the above decoder is **fast**.

Note that there are 3-to-8, 4-to-16, etc. binary decoders whose operation and implementation follow similarly. We will use a 3-to-8 binary decoder when we implement hardwiring in the Control Unit (Block 1) of the term project later in the semester. Today’s memory chips (DRAM, SRAM, ROM, etc.) have **very** large binary decoders.

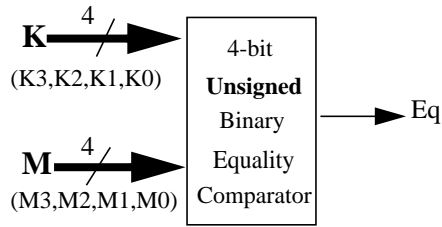
For a k-input binary decoder, there are 2^k AND gates to generate the 2^k outputs. Each input is connected to half the number of AND gates. For example, above, each input (I1, I0) is connected to two AND gates. For small size decoders, this is not a major problem. But, for large decoders, it is a major problem which is called “**fan-out**.” We will study the fan-out problem in the **lab**.

The fan-out of a gate is a number which indicates how many inputs can be connected to it. If the number is exceeded, electrically, there are problems and so the circuit may not work.

It is because of this reason that the decoders of memory chips have their gate networks with more than two levels to reduce the fan-out requirement. However, with more levels, the decoder is slower, therefore, the memory chip is slower.

A 4-bit Unsigned Binary Equality Comparator

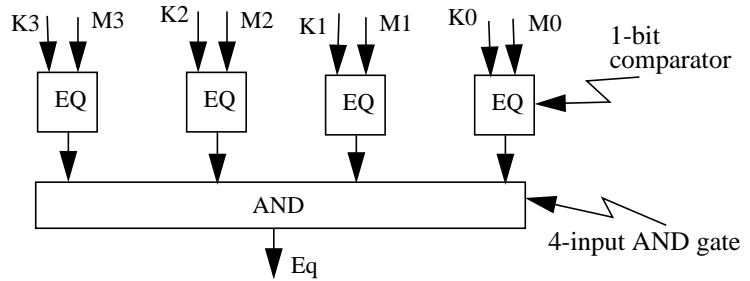
The 4-bit Unsigned Binary comparator compares two 4-bit Unsigned Binary numbers. It has 8 inputs and 1 output. All eight inputs are **data** inputs. The 4-bit inputs are K and M and the output is Eq. The comparator checks if K is equal to M :



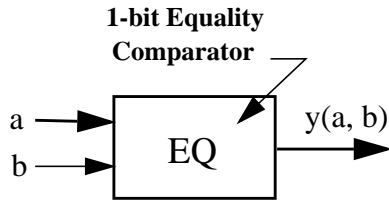
K and M are Unsigned Binary numbers
Eq = 1 if K = M else Eq = 0

Since there are 8 inputs, we need to partition it into simpler pieces !
Output Eq is 1 only if all respective bits are equal to each other :

$$K_3=M_3 \text{ AND } K_2=M_2 \text{ AND } K_1=M_1 \text{ AND } K_0=M_0$$



Each 1-bit comparator has two inputs and 1 output ! We design it by using Switching Algebra :



y(a, b) is 1 if a = b

More explicitly :

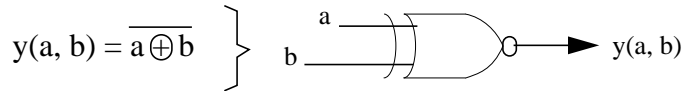
y(a, b) is 1 if
(a=0 AND b=0) OR (a=1 AND b=1)

a	b	y(a, b)
0	0	1
1	0	0
2	1	0
3	1	1

$$y(a,b) = \sum m(0,3)$$

$$\left. \begin{matrix} \overline{0} & \overline{0} & 1 & 1 \\ \overline{a} & \overline{b} & a & b \end{matrix} \right\} y(a, b) = \overline{a} \overline{b} + a b$$

We realize this is an XNOR function :



The 4-bit unsigned binary equality comparator that checks for equality is then as follows :

