EE 2720

Handout #20

EE 2720

• Multiplexers:

A multiplexer, (also called sometimes data selector, abbreviated as MUX), has a group of data inputs and a group of control (select) inputs. It also has an enable input (E) and a data output. The control inputs are used to select only one of the data inputs and connect it to the output terminal. Figure 1 below, shows a 4-to-1 MUX (multiplexer). It has four data inputs named I_0, I_1, I_2, I_3, two control inputs named S_1, S_0, one active high enable input named E and one output named Y. (Reminder: Active high enable E means that when E=1, the multiplexer is enabled; otherwise not. If the multiplexer is not enabled, its output is zero). The truth table of the multiplexer of fig. 1 is shown in table 1.

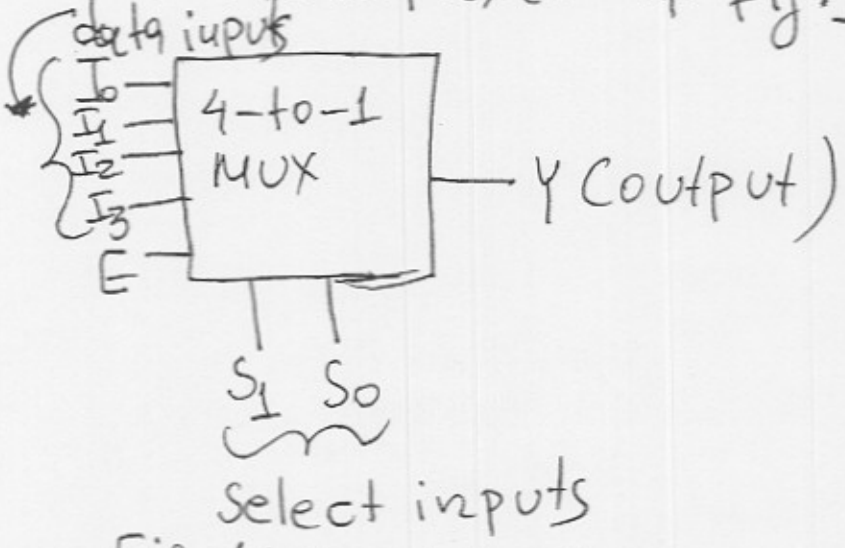

Fig. 1: Symbol for a 4-to-1 MUX

disabled.

E	S1	S0	Y
0	x	x	0
1	0	0	I0
1	0	1	I1
1	1	0	I2
1	1	1	I3

Table 1: Truth table for a 4-to-1 multiplexer (here I denoted 0,1,2,3 with capitals, but I guess it is ok with you)!!

From the truth table of table 1 of the previous page, we get the following logic equation for the output Y of the 4-to-1 multiplexer.

$$Y = S_1' \cdot S_0' \cdot I_0 \cdot E + S_1' \cdot S_0 \cdot I_1 \cdot E + S_1 \cdot S_0' \cdot I_2 \cdot E + S_1 \cdot S_0 \cdot I_3 \cdot E \quad (1)$$

As another example, I will show you an 8-to-1 MUX (multiplexer). It is shown in figure 2 while its truth table is shown in table 2.

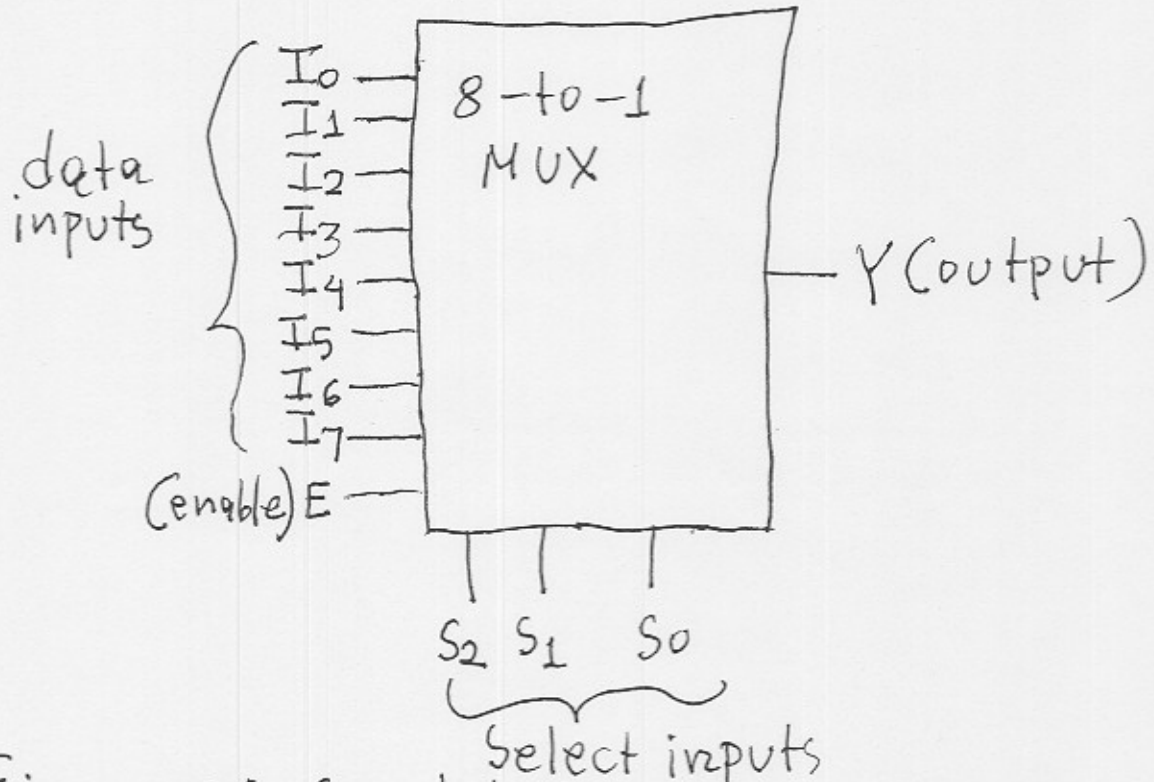

Figure 2: Symbol for an 8-to-1 MUX (multiplexer)

The table 2 (truth table for the 8-to-1 MUX) is shown on the next page.

disabled.

E	S_2	S_1	S_0	Y
0	X	X	X	0
1	0	0	0	I_0
1	0	0	1	I_1
1	0	1	0	I_2
1	0	1	1	I_3
1	1	0	0	I_4
1	1	0	1	I_5
1	1	1	0	I_6
1	1	1	1	I_7

Table 2: Truth table for an 8-to-1 multiplexer.

From the above truth table, we can easily get the logic equation for the output Y of the 8-to-1 multiplexer which is shown below:

$$Y = S_2' \cdot S_1' \cdot S_0' \cdot I_0 \cdot E + S_2' \cdot S_1' \cdot S_0 \cdot I_1 \cdot E + S_2' \cdot S_1 \cdot S_0' \cdot I_2 \cdot E + S_2' \cdot S_1 \cdot S_0 \cdot I_3 \cdot E + S_2 \cdot S_1' \cdot S_0' \cdot I_4 \cdot E + S_2 \cdot S_1' \cdot S_0 \cdot I_5 \cdot E + S_2 \cdot S_1 \cdot S_0' \cdot I_6 \cdot E + S_2 \cdot S_1 \cdot S_0 \cdot I_7 \cdot E \quad (2)$$

The figure 3 on the next page shows the circuit realization of the 8-to-1 MUX (multiplexer) presented above. As seen, it is an AND-OR logic circuit.

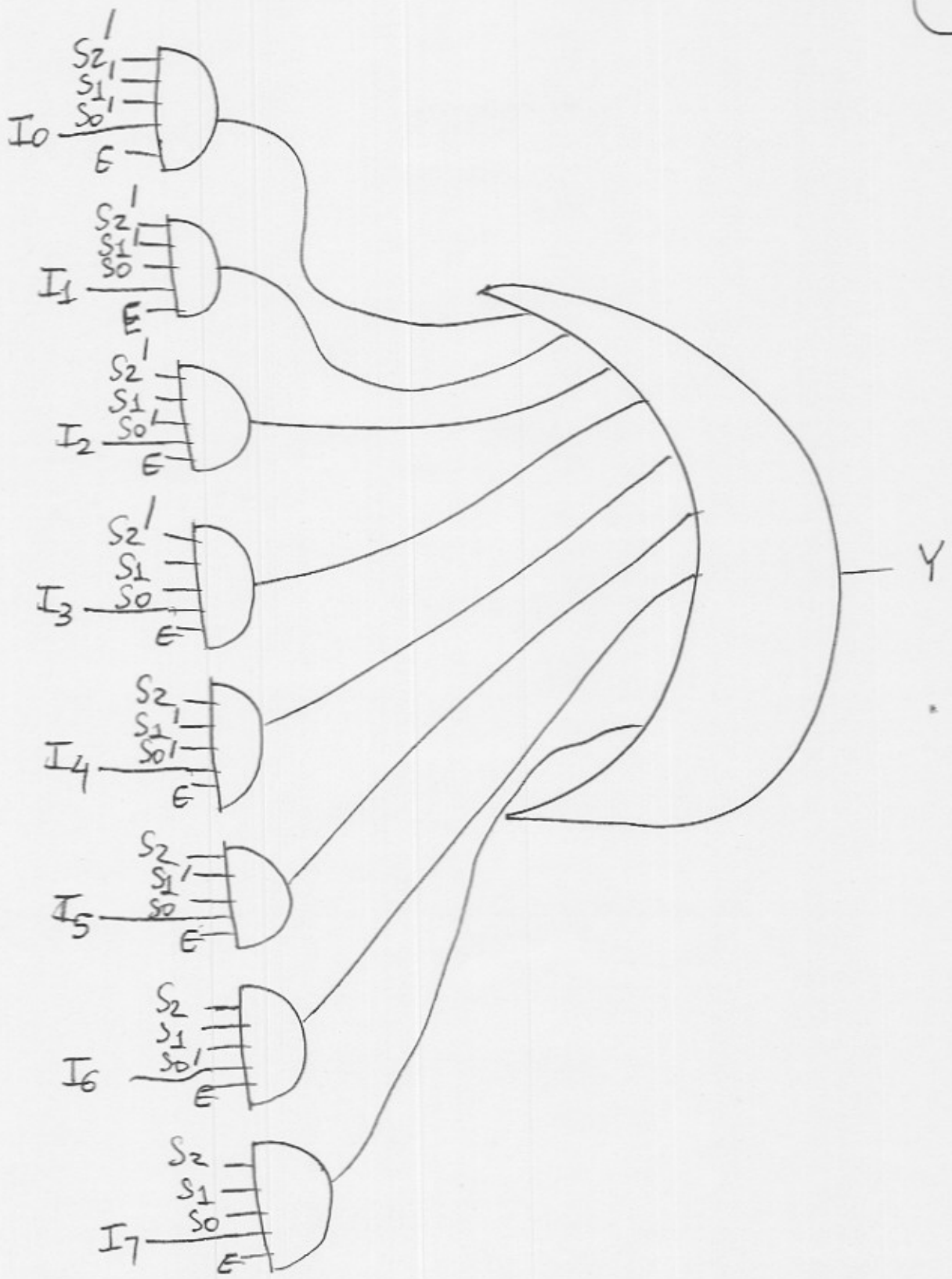

Fig. 3: Circuit realization of an 8-to-1 MUX (multiplexer). It is an AND-OR logic circuit. You can easily transform it into a realization that relies on only NAND gates.

Generalizing, we have the 2^n-to-1 MUX (multiplexer). It has 2^n data inputs named I_0, I_1, I_2, ..., I_{2^n-1}, n control (or select) input named S_0, S_1, ..., S_{n-1}, one enable input named E (it is active high but could as well be active low), and one output named Y. Its symbol is shown in figure 4 below:

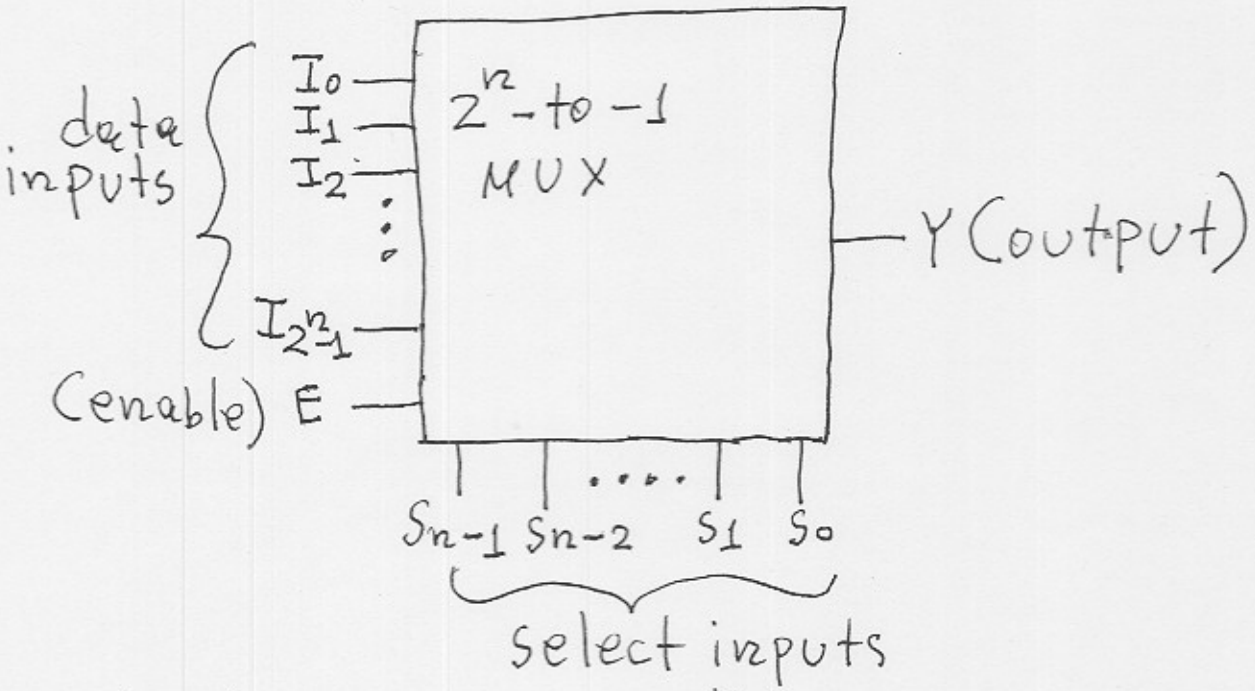

Figure 4: Symbol for a 2^n-to-1 multiplexer.

Some explanations follow about the multiplexer of figure 4. Although it should be selfexplanatory by now, I still provide some explanations

→ Go to next page →

- If the control inputs are $S_{n-1}S_{n-2}\cdots S_1S_0 = 00\cdots00$, then $Y = I_0$.
- If the control inputs are $S_{n-1}S_{n-2}\cdots S_1S_0 = 00\cdots01$, then $Y = I_1$.
- If the control inputs are $S_{n-1}S_{n-2}\cdots S_1S_0 = 00\cdots10$, then $Y = I_2$.
- If the control inputs are $S_{n-1}S_{n-2}\cdots S_1S_0 = 00\cdots11$, then $Y = I_3$

⋮

etc

⋮

- If the control inputs are $S_{n-1}S_{n-2}\cdots S_1S_0 = 11\cdots110$, then $Y = I_{2^n-2}$.
- If the control inputs are $S_{n-1}S_{n-2}\cdots S_1S_0 = 11\cdots111$, then $Y = I_{2^n-1}$.

I hope this is enough explanations.
The logic equation for the output Y of the 2^n-to-1 MUX (multiplexer) of fig. 4 is obviously the one shown below:

$$Y = \sum_{k=0}^{2^n-1} I_k \cdot m_k \cdot E \quad (3)$$

In the above equation (3), m_k is the k^{th} minterm of the variables $S_{n-1}, S_{n-2}, \ldots, S_1, S_0$.

• Designing large multiplexers from smaller ones:

In this section, I will show you how to build large multiplexers using smaller ones.

Example 1: Using five 4-to-1 multiplexers and nothing else, design a 16-to-1 multiplexer.

Answer: The figure 5 below shows the design.

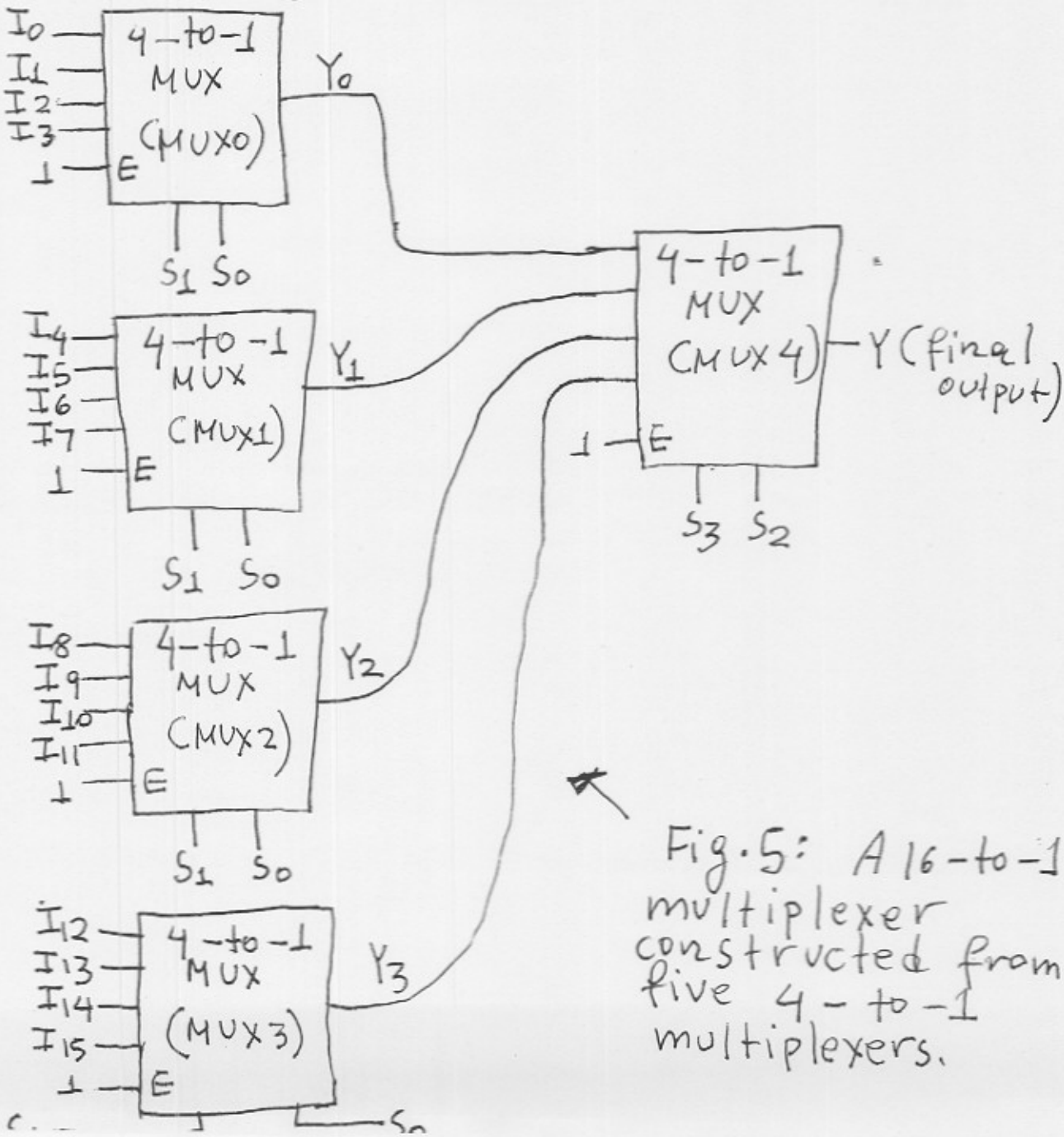

Fig. 5: A 16-to-1 multiplexer constructed from five 4-to-1 multiplexers.

• Explanations about fig. 5 of the previous page:

Although fig. 5 of the previous page should be self-explanatory) I will still provide some explanations about it. I named the five multiplexers of fig. 5 as MUX0, MUX1, MUX2, MUX3, MUX4. Each of the five 4-to-1 multiplexers has an enable input E. I connected all the enable inputs to 1 and this way I enabled all five 4-to-1 multiplexers; (see that each E is active high). The data inputs to MUX0 are I_0, I_1, I_2, I_3, the data inputs to MUX1 are I_4, I_5, I_6, I_7, the data inputs to MUX2 are I_8, I_9, I_{10}, I_{11}, the data inputs to MUX3 are I_{12}, I_{13}, I_{14}, I_{15}. The select inputs for all four multiplexers MUX0, MUX1, MUX2, MUX3 are S_1, S_0. The outputs of the multiplexers MUX0, MUX1, MUX2, MUX3, are respectively, Y_0, Y_1, Y_2, Y_3 and these outputs Y_0, Y_1, Y_2, Y_3 become data inputs to the multiplexer MUX4. The select inputs to MUX4 are S_3, S_2 while its output is Y. To summarize, before I provide more explanations, the data inputs to the overal 16-to-1 multiplexer are I_0, I_1, ..., I_{14}, I_{15}, the select inputs to the 16-to-1 multiplexer are S_3, S_2, S_1, S_0 while its output is Y. It must

be noted that S_3 is the most significant select line while S_0 is the least significant select line. This way, if $S_3S_2S_1S_0$ represents the number i, then the data input I_i is selected to appear at the output Y of the 16-to-1 multiplexer. I will now provide detailed explanations about how the 16-to-1 multiplexer works. I will do this by using truth tables. Just see below.

S_1	S_0	Y_0
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 3: Truth table for MUX0

S_1	S_0	Y_1
0	0	I_4
0	1	I_5
1	0	I_6
1	1	I_7

Table 4: Truth table for MUX1

S_1	S_0	Y_2
0	0	I_8
0	1	I_9
1	0	I_{10}
1	1	I_{11}

Table 5: Truth table for MUX2

S_1	S_0	Y_3
0	0	I_{12}
0	1	I_{13}
1	0	I_{14}
1	1	I_{15}

Table 6: Truth table for MUX3

S_3	S_2	Y
0	0	Y_0
0	1	Y_1
1	0	Y_2
1	1	Y_3

Table 7: Truth table for MUX4.

On the next page you can see the truth table of the entire 16-to-1 Multiplexer.

S_3 S_2 S_1 S_0	Y	
0 0 0 0	I_0	Y_0
0 0 0 1	I_1	
0 0 1 0	I_2	
0 0 1 1	I_3	
0 1 0 0	I_4	Y_1
0 1 0 1	I_5	
0 1 1 0	I_6	
0 1 1 1	I_7	
1 0 0 0	I_8	Y_2
1 0 0 1	I_9	
1 0 1 0	I_{10}	
1 0 1 1	I_{11}	
1 1 0 0	I_{12}	Y_3
1 1 0 1	I_{13}	
1 1 1 0	I_{14}	
1 1 1 1	I_{15}	

Table 8: Truth table of the 16-to-1 multiplexer of figure 5.

Note: I hope that it should be clear by now. If not, ask me in class.

Example 2: Using two 4-to-1 multiplexers and one 2-to-1 multiplexer design an 8-to-1 multiplexer.

Answer: Figure 6 on the next page shows the design.

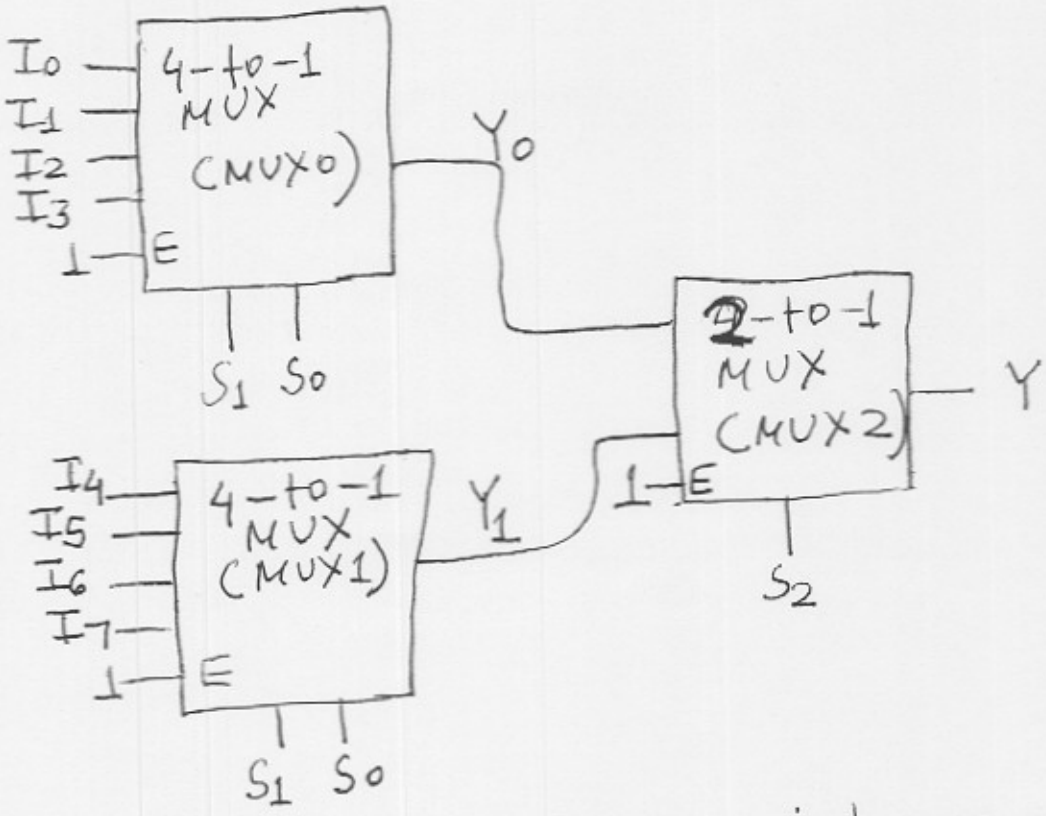

Figure 6: An 8-to-1 multiplexer constructed from two 4-to-1 multiplexers and one 2-to-1 multiplexer and nothing else.

- Explanations about figure 6:

Although by now fig. 6 should be self explanatory, I will still provide explanations about it. As you see, I named the three multiplexers by the names MUX0, MUX1, MUX2. Each of the multiplexers has an active high enable input E. I connected all the enable inputs to 1 and this way I enabled all three multiplexers. The data inputs to MUX0 are I_0, I_1, I_2, I_3, the data inputs to MUX1 are I_4, I_5, I_6, I_7. The select inputs for both multiplexers MUX0 and MUX1 are S_1, S_0. The

outputs of the multiplexers MUX0, MUX1 are, respectively, Y_0, Y_1 and these outputs Y_0, Y_1 become data inputs to the 2-to-1 multiplexer MUX2. The select input to MUX2 is S_2 while its output is Y. To summarize, before I provide more explanations, the data inputs to the overal 8-to-1 multiplexer are I_0, I_1, ..., I_7, the select inputs to the 8-to-1 multiplexer are $S_2S_1S_0$ while its output is Y. It must be noted that S_2 is the most significant select input, while S_0 is the least significant select input. I will now provide explanations about how the 8-to-1 multiplexer works. I will do this by using truth tables. Just see below.

S_1	S_0	Y_0
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 9: Truth table for MUX0.

S_1	S_0	Y_1
0	0	I_4
0	1	I_5
1	0	I_6
1	1	I_7

Table 10: Truth table for MUX1

S_2	Y
0	Y_0
1	Y_1

Table 11: Truth table for MUX2

The truth table for the 8-to-1 multiplexer of figure 6 is shown on the next page

$S_2S_1S_0$	Y	
000	I_0	Y_0
001	I_1	
010	I_2	
011	I_3	
100	I_4	Y_1
101	I_5	
110	I_6	
111	I_7	

Table 12: Truth table of the 8-to-1 multiplexer of figure 6.

The previous two examples of designing large multiplexers from smaller ones are two-level designs. Each of the designs, in other words, consists of multiplexers arranged in two levels; (several multiplexers in the first level and one multiplexer in the second level). Next I will show you an example of designing a large multiplexer using smaller ones that is going to be a one-level design; (consisting of multiplexers arranged in one level only). Before this, however, I will present one more multiplexer. This is an 8-to-1 multiplexer with three enable inputs, one active high and two active low. Its symbol is shown in figure 7 while its truth table is shown in table 13.

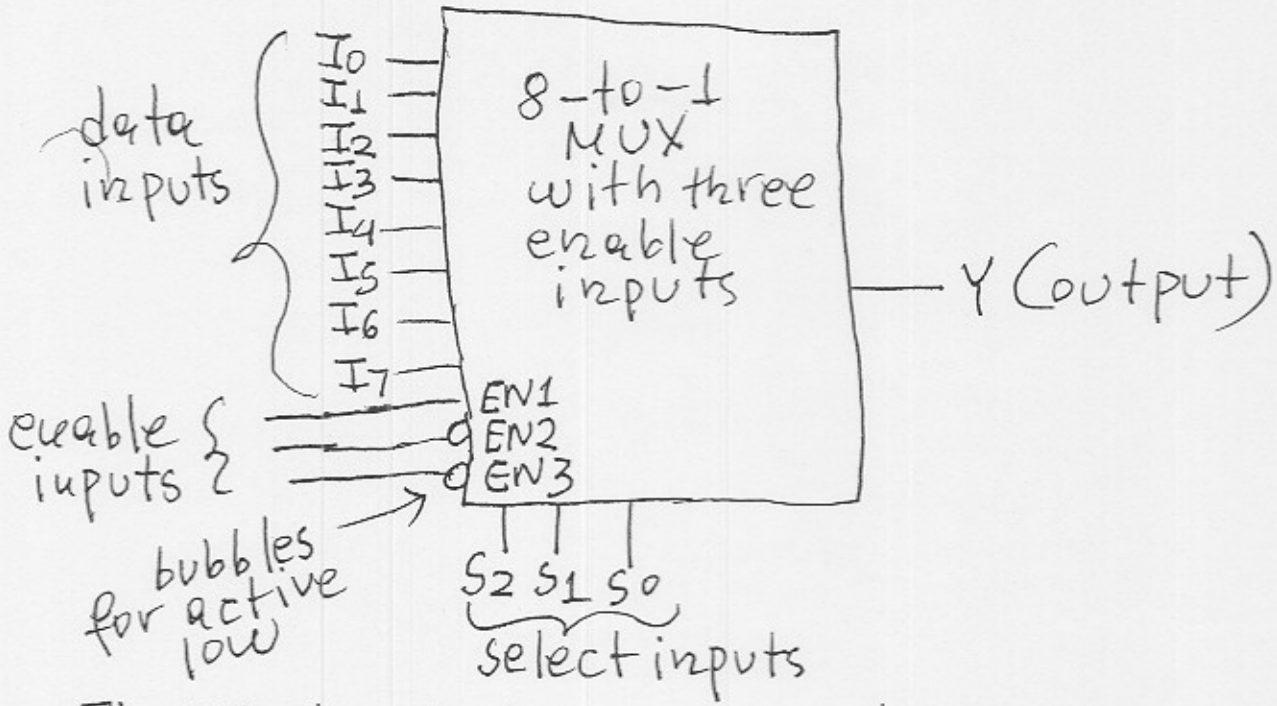

Fig 7: An 8-to-1 multiplexer with one active high and two active low enable inputs.

disabled

EN1	EN2	EN3	S2	S1	S0	Y
0	X	X	X	X	X	0
X	1	X	X	X	X	0
X	X	1	X	X	X	0
1	0	0	0	0	0	I_0
1	0	0	0	0	1	I_1
1	0	0	0	1	0	I_2
1	0	0	0	1	1	I_3
1	0	0	1	0	0	I_4
1	0	0	1	0	1	I_5
1	0	0	1	1	0	I_6
1	0	0	1	1	1	I_7

Table 13: Truth table of the multiplexer of figure 7

Note: I hope you can easily get the logic equation for the output Y of the multiplexer of figure 7 and the corresponding circuit realization for it. Do it if you want. It is very easy!

I will now present the third example.

Example 3: Design a 32-to-1 multiplexer using four 8-to-1 multiplexers ~~arranged~~ as in figure 7 arranged in one level and the minimum number of gates.

Answer: Before I present the answer to this problem, I am telling you that you can provide a two-level design like I did in examples 1 and 2. It will consist of four 8-to-1 multiplexers in the first level and one 4-to-1 multiplexer in the second level. The select inputs to all four multiplexers of the first level will be S_2, S_1, S_0. The outputs of the first-level multiplexers will be data inputs to the 4-to-1 multiplexer of the second level. The second-level multiplexer will have as select inputs S_4, S_3. Here, S_4 is the MSB of the select inputs while S_0 is the LSB of the select inputs; (I am sure you can provide such a design by now). However, as I said, I will provide a design with multiplexers arranged in one level. This is shown in figure 8 on the next page; (it doesn't fit here).

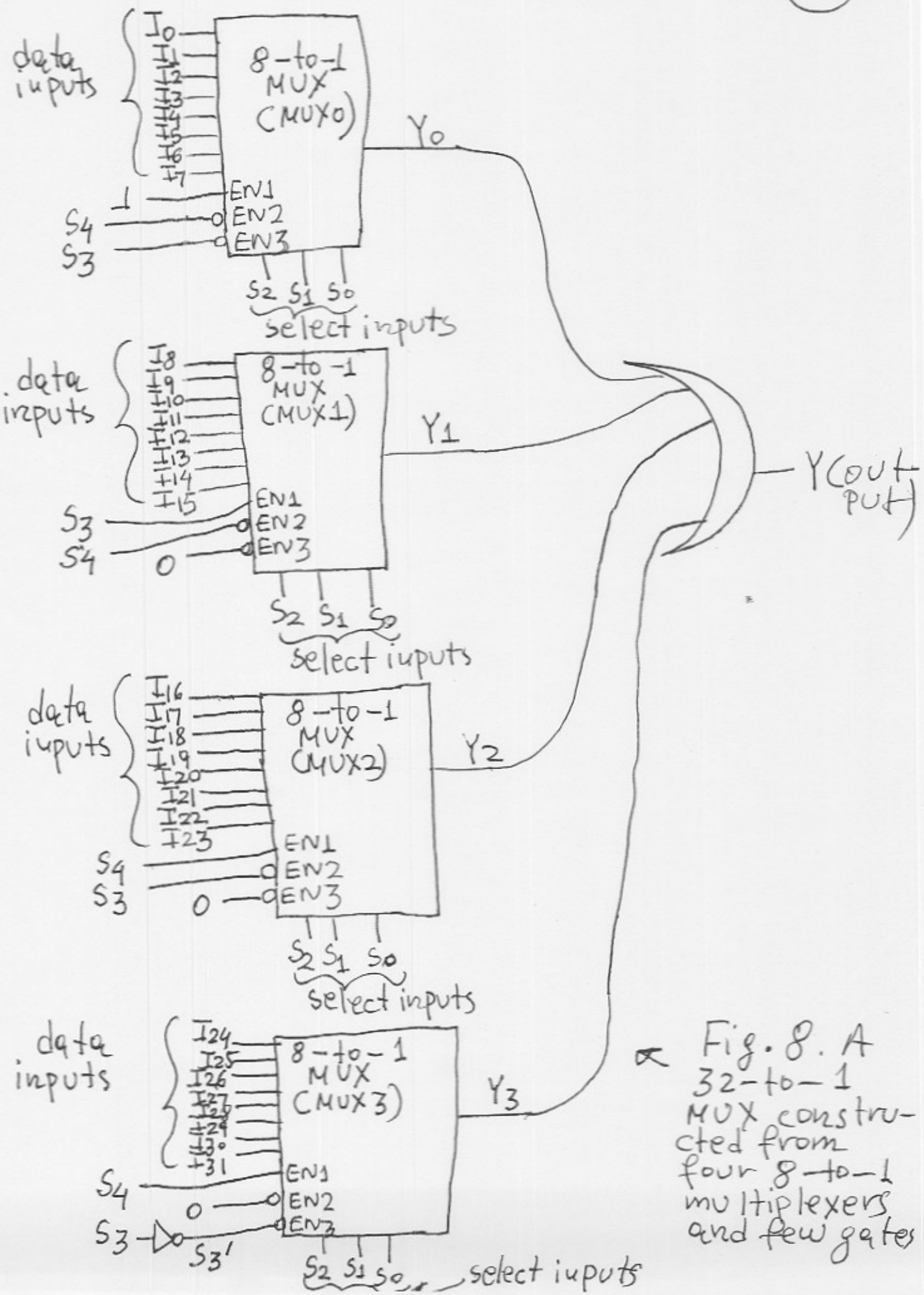

Fig. 8. A 32-to-1 MUX constructed from four 8-to-1 multiplexers and few gates

Explanation about the design of figure 8:

Although figure 8 should be self explanatory, I still provide some explanations about it. fig.8 shows the 32-to-1 multiplexer. Its data inputs are $I_0, I_1, \ldots, I_{30}, I_{31}$. Its select inputs are S_4, S_3, S_2, S_1, S_0. The bit S_4 is the MSB of the select inputs while S_0 is the LSB of the select inputs. The output of the 32-to-1 MUX is Y (the output of the OR gate). The select inputs S_4, S_3 are connected to some of the enable inputs of the 8-to-1 multiplexers which named MUX0, MUX1, MUX2, MUX3. The other three select inputs S_2, S_1, S_0 are the select inputs to all four 8-to-1 multiplexers MUX0, MUX1, MUX2, MUX3.

- when $S_4S_3 = 00$, MUX0 is enabled while MUX1, MUX2, MUX3 are disabled, so their outputs are $Y_1 = 0$, $Y_2 = 0$, $Y_3 = 0$.
- when $S_4S_3 = 01$, MUX1 is enabled while MUX0 MUX2, MUX3 are disabled, so their outputs Y_0, Y_2 Y_3 are $Y_0 = Y_2 = Y_3 = 0$
- when $S_4S_3 = 10$, MUX2 is enabled while MUX0, MUX1, MUX3 are disabled, so $Y_0 = Y_1 = Y_3 = 0$
- when $S_4S_3 = 11$, MUX3 is enabled while MUX0, MUX1, MUX2 are disabled, so $Y_0 = Y_1 = Y_2 = 0$

The truth table of table 14 further clarifies

the design:

S_4 S_3 S_2 S_1 S_0	Y_0 Y_1 Y_2 Y_3	$Y = Y_0 + Y_1 + Y_2 + Y_3$
0 0 0 0 0	I_0 0 0 0	I_0
0 0 0 0 1	I_1 0 0 0	I_1
⋮	⋮ ⋮	⋮
0 0 1 1 1	I_7 0 0 0	I_7
0 1 0 0 0	0 I_8 0 0	I_8
0 1 0 0 1	0 I_9 0 0	I_9
⋮	⋮ ⋮ ⋮	⋮
0 1 1 1 1	0 I_{15} 0 0	I_{15}
1 0 0 0 0	0 0 I_{16} 0	I_{16}
1 0 0 0 1	0 0 I_{17} 0	I_{17}
⋮	⋮ ⋮ 0	⋮
1 0 1 1 1	0 0 I_{23} 0	I_{23}
1 1 0 0 0	0 0 0 I_{24}	I_{24}
1 1 0 0 1	0 0 0 I_{25}	I_{25}
⋮	⋮ ⋮	⋮
1 1 1 1 1	0 0 0 I_{31}	I_{31}

Table 14: Truth table for the 32-to-1 multiplexer of figure 8.
I hope it is clear now.

The one-level multiplexer design of fig. 8 is cheaper and faster than a two-level design like the ones I provided in examples 1 and 2. As I said earlier, a two-level design will consist

of four 8-to-1 multiplexers in the first level and one 4-to-1 multiplexer in the second level. So the cost for the two-level design will be:

cost two-level design = 4 x 8-to-1 MUX + 1 x 4-to-1 MUX (4)

The delay through the two-level design will be:

delay through two-level design = delay through an 8-to-1 MUX + delay through a 4-to-1 MUX. (5)

The cost and delay of the one-level design of figure 8 is:

cost of one-level design = 1 x inverter + + 4 x 8-to-1 MUX + 1 x 4-input OR gate (6)

The

delay through one-level design = delay through an inverter + delay through an 8-to-1 MUX + delay through a 4-input OR gate. (7)

From equations (4), (5), (6), (7) above, it is obvious that the one level design of fig. 8 is cheaper and faster than a two-level design. Just see that an inverter plus a 4-input OR gate costs less than a 4-to-1 multiplexer. Regarding the delay, just see that the delay through an inverter plus the delay through a 4-input OR gate is smaller than th

delay through a 4-to-1 multiplexer. Thus, the design of fig. 8 is better than a two-level design.

An alternative design (alternative to the design of fig. 8) is shown in figure 9.

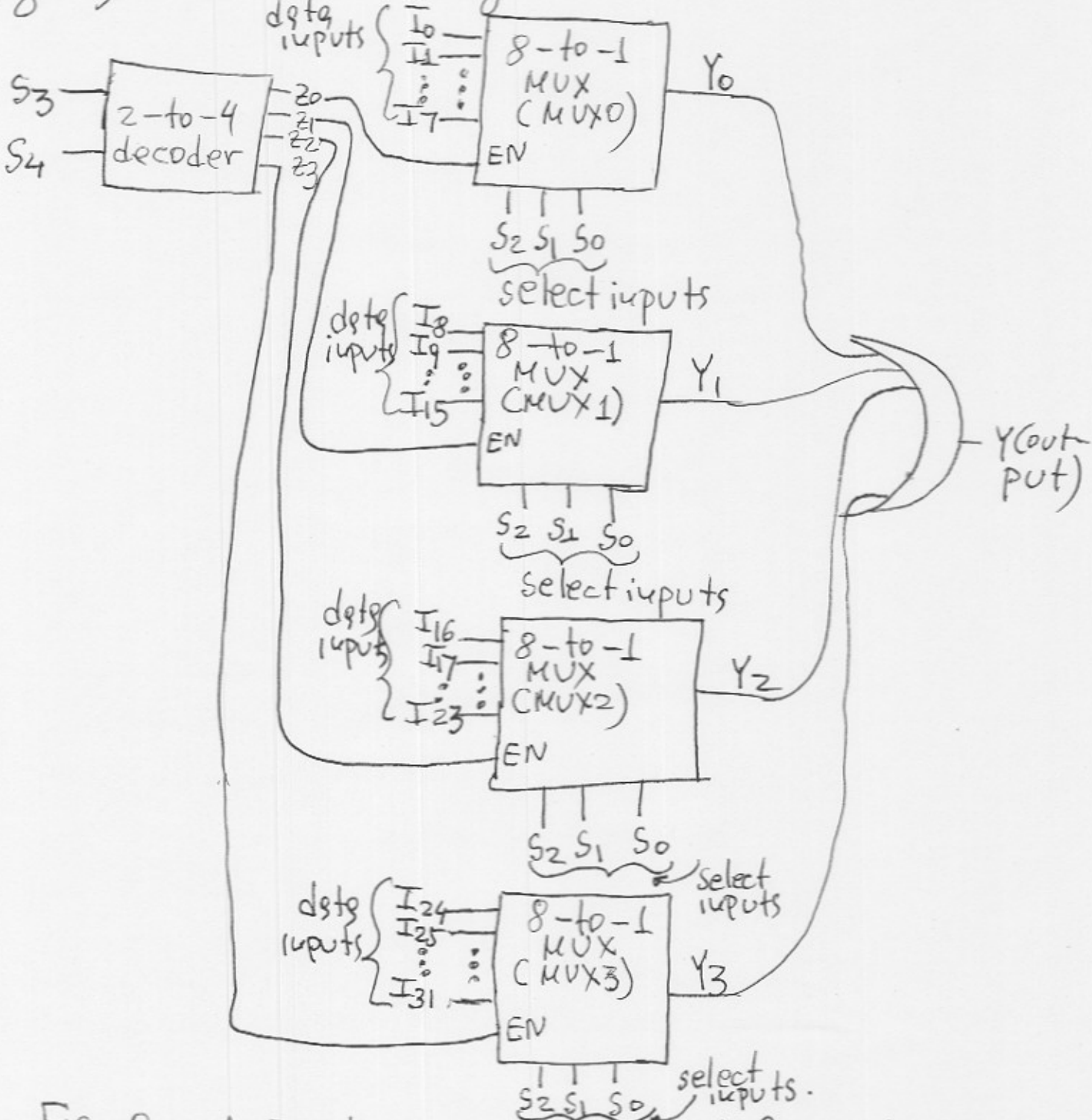

Fig. 9: A 32-to-1 MUX constructed from four 8-to-1 multiplexers, one 2-to-4 decoder and one OR gate.

Explanations about the design of figure 9:

The way the design of figure 9 operates is pretty similar to the way the design of figure 8 operates. Figure 9 shows the 32-to-1 multiplexer. Its data inputs are $I_0, I_1, \ldots, I_{30}, I_{31}$. Its select inputs are S_4, S_3, S_2, S_1, S_0, where S_4 is the MSB of the select inputs, while S_0 is the LSB of the select inputs. The select inputs S_2, S_1, S_0 are the select inputs to all four 8-to-1 multiplexers MUX0, MUX1, MUX2, MUX3. The other two select inputs S_4, S_3 are inputs to the 2-to-4 decoder which has as outputs z_0, z_1, z_2, z_3. Exactly one of the decoders output is 1 depending on the values of S_4S_3. For example, if $S_4S_3 = 00$, then $z_0 = 1$ and all other z_is are 0s, if $S_4S_3 = 11$, then $z_3 = 1$ and all other z_is are 0s. The outputs of the 2-to-4 decoder are the enable inputs to the four 8-to-1 multiplexers. This way, exactly one of MUX0, MUX1, MUX2, MUX3 is enabled at a time and all others are disabled. If $S_4S_3 = 00$, then MUX0 is enabled and all other multiplexers are disabled and thus their outputs are 0s. If $S_4S_3 = 01$, then MUX1 is enabled and all other multiplexers are disabled and thus their outputs are 0s. If $S_4S_3 = 10$, then MUX2 is enabled and all other multiplexers are disabled and thus their outputs are 0s. If $S_4S_3 = 11$, then MUX3 is enabled and all other

multiplexers are disabled and thus their outputs are 0s. This way, the 32-to-1 multiplexer of figure 9 implements the truth table of table 14. It therefore operates like a proper 32-to-1 multiplexer.

Note: The textbook on page 403 offers a design pretty similar to this of fig. 9 with the following differences: The 2-to-4 decoder has an active low enable input and I don't; the outputs of the decoder are complemented; the enable inputs to the four 8-to-1 multiplexers are active low while mine are active high; the author takes the complemented outputs of the four 8-to-1 multiplexers; the OR gate is replaced with a 4-input NAND; (OR with inverted inputs). The author does that because he bases his design on real chips (integrated circuits) which you will see in EE 2731. Obviously these chips offer NAND-based implementations which are better because they are faster and cheaper. Each 8-to-1 multiplexer is a chip named 74x151, the 2-to-4 decoder is 1/2 of a chip (2 decoders in a chip) which is named 74x139 and the 4-input NAND gate is 1/2 of a chip named 74x20.

Note: I believe that we had enough examples of

designing large multiplexers from smaller ones.

• The most important application of multiplexers:

The most important application of multiplexers, (which should be obvious by now I guess), is to provide for the transmission of information from several sources over a single path; (many to one etc.). This process is known as multiplexing.

• Demultiplexers:

The function of a demultiplexer is the inverse of the multiplexer's. For example, a 1-bit n-output demultiplexer has one data input and s select inputs to select one of $n = 2^s$ data outputs. In normal operation, all the outputs except the selected one are 0's; the selected output equals the data input. A binary decoder with an enable input can be used as a demultiplexer as shown in fig. 10 The decoder's enable input is connected to the data line and its select inputs determine which of its output lines is driven with the data bit.

↳ Fig. 10 shown on next page →

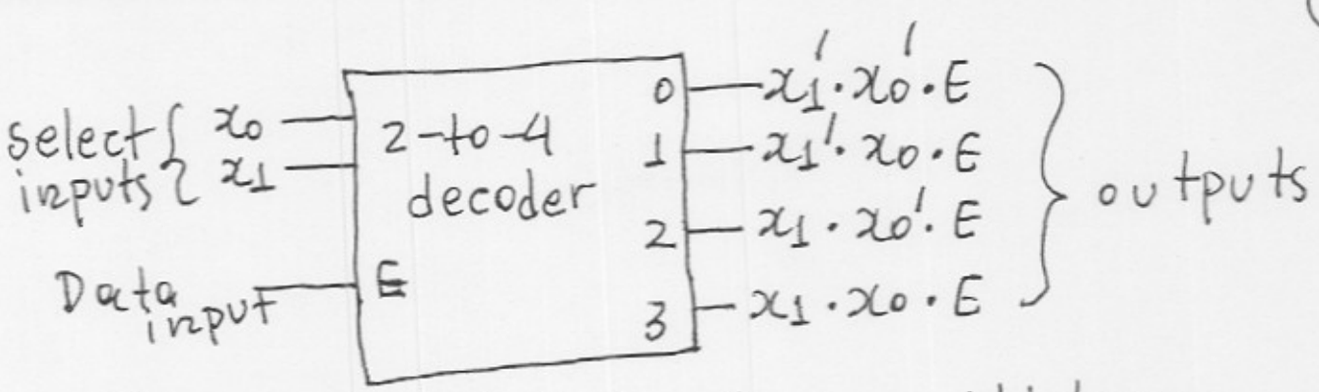

Figure 10: A 1-to-4 demultiplexer.

In the above fig. 10, if $x_1x_0 = 00$, then the output labeled 0 is described by the Boolean expression $0' \cdot 0' \cdot E = 1 \cdot E = E =$ Data input; all other outputs are 0s.

If $x_1x_0 = 01$, then the output labeled 1 is described by the Boolean expression $0' \cdot 1 \cdot E = 1 \cdot E = E =$ Data input; all other outputs are 0's.

If $x_1x_0 = 10$, then the output labeled 2 is described by the Boolean expression $1 \cdot 0' \cdot E = 1 \cdot E = E =$ Data input; all other outputs are 0's.

If $x_1x_0 = 11$, then the output labeled 3 is described by $1 \cdot 1 \cdot E = E =$ Data input; all other outputs are 0s.

Thus, the above circuit of fig. 10 provides its data input namely "Data input" to only one of the outputs labeled 0, 1, 2 or 3, and therefore functions properly as a demultiplexer.

• Combining multiplexers together with demultiplexers:

When a multiplexer is used in conjunction with a demultiplexer, (a decoder with an enable input), an efficient means is provided for connecting information from many source locations to many destination locations. This is illustrated in fig. 11 shown below:

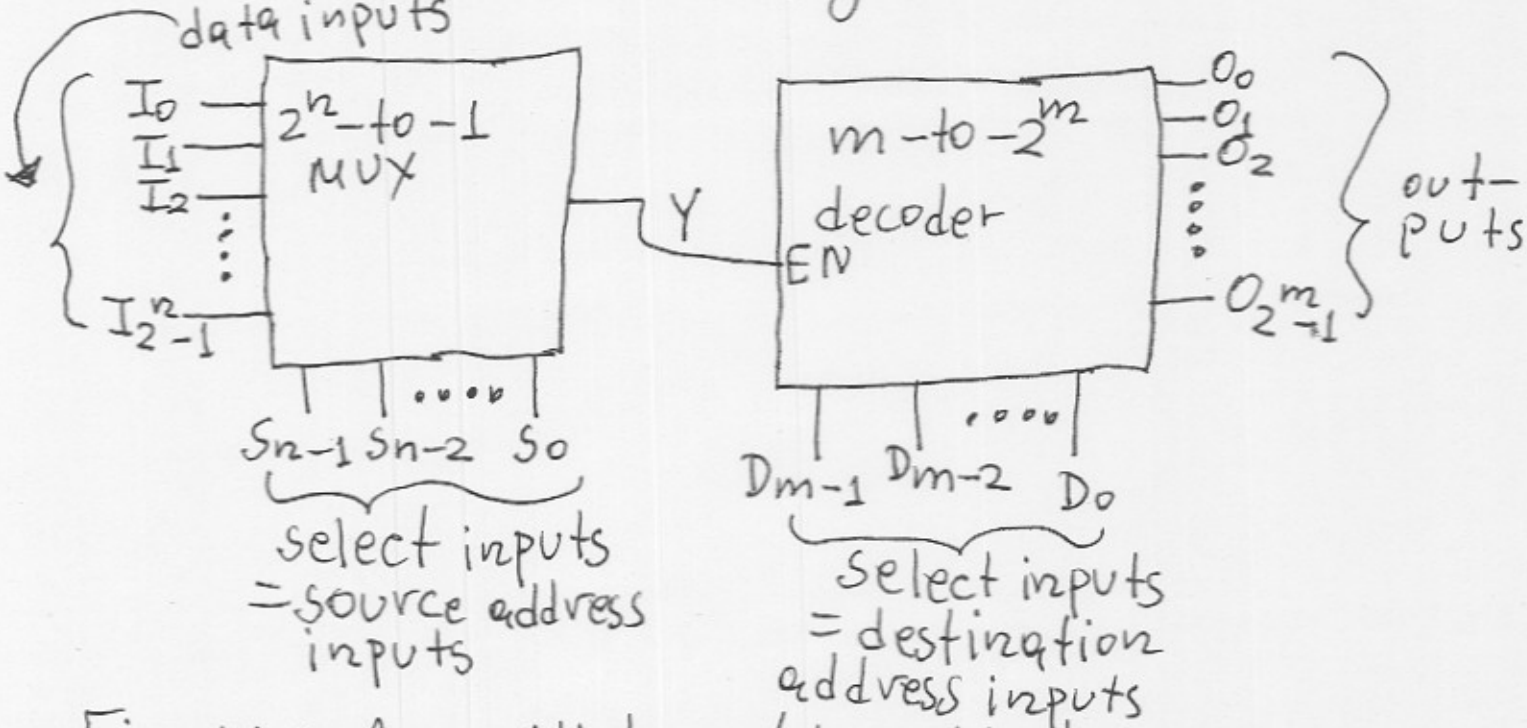

Fig. 11: A multiplexer/demultiplexer arrangement for information transmission between 2^n sources and 2^m destinations.

Regarding fig. 11, for example, if we want to transmit information I_2 to destination O_5, we make the selects $S_{n-1}S_{n-2}\cdots S_1S_0 = 00\cdots10$ and the selects $D_{m-1}D_{m-2}\cdots D_2D_1D_0 = 00\cdots101$. This way, the 2^n-to-1 multi-

plexer selects I_2 to be connected to its output Y, while Y appears (is routed) in the decoder's output O_5; (all other O_i's are 0's). I hope that this is enough explanation about the multiplexer/demultiplexer.

Note: We can use b of the structures shown in figure 11 operating in parallel; (each such structure is a 1-bit structure). This way a b-bit information can be connected from any one of 2^n sources to any one of 2^m destinations. I guess I don't have to provide a figure for this.

• More applications of multiplexers:

Like decoders, multiplexers can be used to realize logic functions. The following examples demonstrate the situation.

Example 4: Implement $F = \Sigma_{x,y,z}(0,2,3,5)$ using a 8-to-1 multiplexer and nothing else.

Answer: I will provide, below, the truth table for F.

X Y Z	F
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	0

Table 15: Truth table for $F = \Sigma_{x,y,z}(0,2,3,5)$

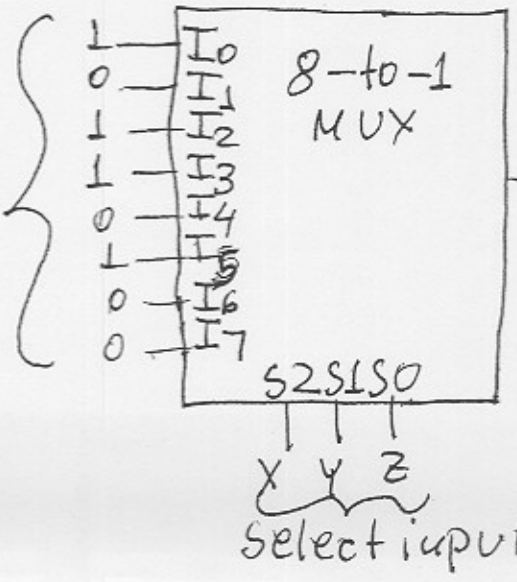

Fig. 12: Realization of F of example 4

Example 5: Implement $F = \Sigma_{I_2, I_1, I_0}(1, 2, 4, 7)$ and $G = \Sigma_{I_2, I_1, I_0}(3, 5, 6, 7)$ using two 4-to-1 multiplexers and one inverter.

Answer: I will provide truth tables for F and G at first. They are shown below:

I_2 I_1 I_0	F		G	
0 0 0	0	I_0	0	0
0 0 1	1		0	
0 1 0	1	I_0'	0	I_0
0 1 1	0		1	
1 0 0	1	I_0'	0	I_0
1 0 1	0		1	
1 1 0	0	I_0	1	1
1 1 1	1		1	

Table 16: Truth tables for F and G of example 5.

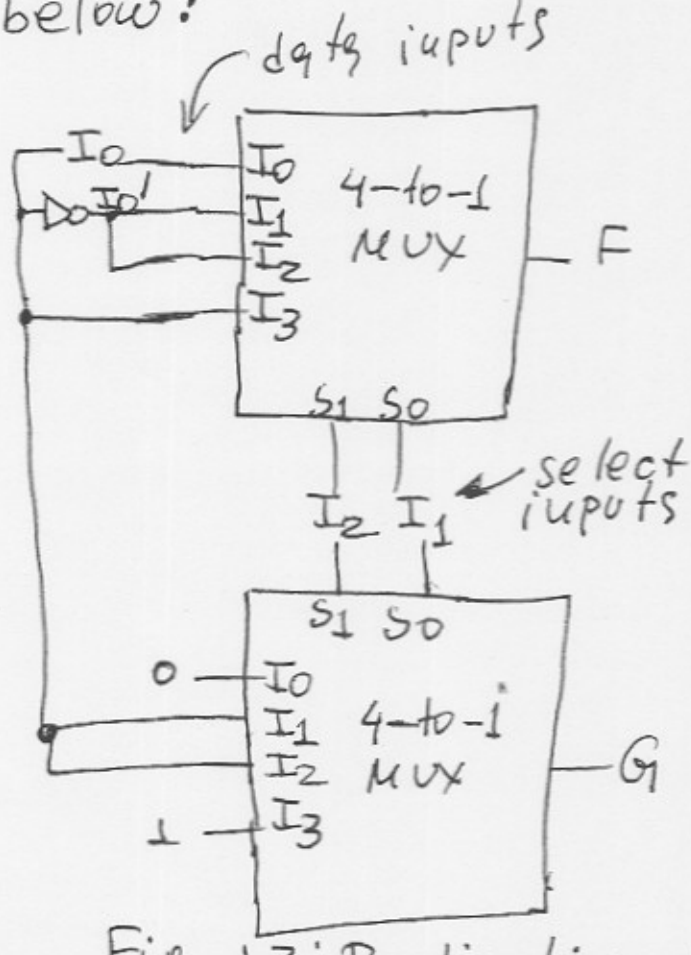

Fig. 13: Realization of F and G of example 5.

Example 6: Implement a full adder, C I told you what the function of the full adder is in HW#6; just review problem 3 of HW#6 for this matter), using two 4-to-1 multiplexers and one inverter.

Answer: I will provide the truth table of the full adder first. This is shown on the next page.

X	Y	Cin	S	Cout
0	0	0	0 } Cin	0 } 0
0	0	1	1 } Cin	0 } 0
0	1	0	1 } Cin'	0 } Cin
0	1	1	0 } Cin'	1 } Cin
1	0	0	1 } Cin'	0 } Cin
1	0	1	0 } Cin'	1 } Cin
1	1	0	0 } Cin	1 } 1
1	1	1	1 } Cin	1 } 1

Table 17: Truth table for a full adder

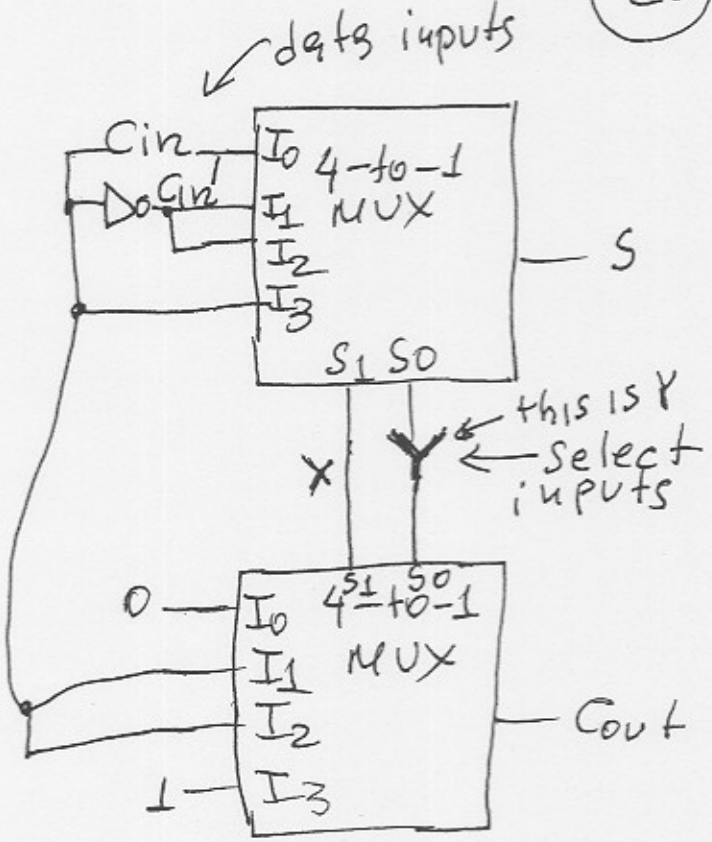

Fig. 14: Realization of a full adder.

Note: Realizing logic functions using multiplexers is not always as easy as in examples 4, 5, 6. I might assign some more difficult problems in HW#7 (not to be returned I think, of course, I will provide solutions. What about that!!!

This is the end of the subject on multiplexers. I covered everything that somebody could possibly cover on this subject!!! There is nothing more left. This is all about it.