Multiplexers

- Combinational circuit that has data inputs, select lines and a single output.
- One input is passed through to the output based on the control lines.
- For n data inputs, we need $\lceil \log(n) \rceil$ control lines.

2-to-1 multiplexer

- Two data inputs x_0 and x_1. One select line s. One output f.
- Truth table for a 2-to-1 multiplexer:

x_0	x_1	s	f
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

$\rightarrow$

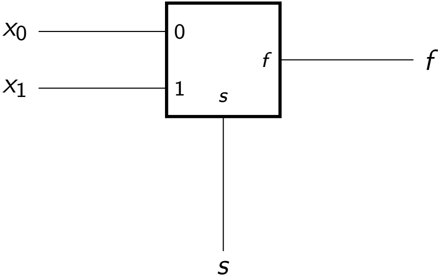
s	f
0	x_0
1	x_1

- Implements (basically) the "if-else" function in hardware. We can write an equation easily:

$$f = \overline{s}x_0 + sx_1$$

2-to-1 multiplexer

- Multiplexers have their own symbol:



- **NOTE: THIS SYMBOL IS NOT QUITE RIGHT... I AM STILL TRYING TO FIGURE OUT HOW TO DRAW IT USING THE PROGRAM THAT I USE...**

4-to-1 multiplexer

- A 4-to-1 multiplexer has 4 inputs, 2 select lines and 1 output.
- Truth table...

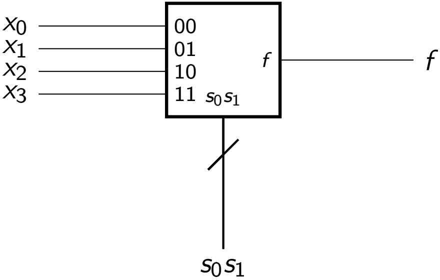
s_0	s_1	f
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

- Equation...

$$f = s_0^{'} s_1^{'} x_0 + s_0^{'} s_1 x_1 + s_0 s_1^{'} x_2 + s_0 s_1 x_3$$

4-to-1 multiplexer

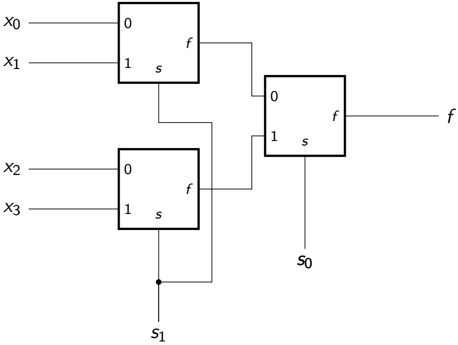
- Multiplexers have their own symbol:



- **NOTE: THIS SYMBOL IS NOT QUITE RIGHT... I AM STILL TRYING TO FIGURE OUT HOW TO DRAW IT USING THE PROGRAM THAT I USE...**
- Similarly, we can have symbols for larger multiplexers.

Multiplexer trees

- We can build larger multiplexers from smaller multiplexers; e.g., a 4-to-1 multiplexer from 2-to-1 multiplexers.



- Checking...

$$\begin{aligned} f &= s_0'(s_1'x_0 + s_1x_1) + s_0(s_1'x_2 + s_1x_3) \\ &= s_0's_1'x_0 + s_0's_1x_1 + s_0s_1'x_2 + s_0s_1x_3 \end{aligned}$$

Function implementation; *n*-input function with multiplexer with $n-1$ select lines

- You can implement logic functions using only multiplexers.
- You can implement an *n*-input function with a single multiplexer as long as the multiplexer has $n-1$ select lines.
- Example... implement

$$f = x_0^{'}x_1 + x_0x_1^{'}x_2 + x_0x_1x_2^{'}$$

  using a single multiplexer.
- For this 3 input function, we require a multiplexer with 2 select lines; i.e., a 4-to-1 multiplexer.

Function implementation; n-input function with multiplexer with $n-1$ select lines

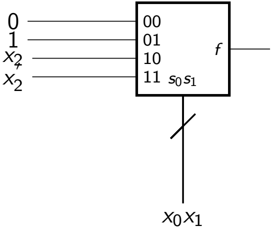
- Truth table...

x_0	x_1	x_2	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

- Use the $n-1$ leftmost inputs to connect to the select lines and divide the truth table into pieces.
- In each piece, compare the right most input to the value of f and connect the correct value to the appropriate input of the multiplexer.
  - The value to connect will always be one of four choices: 0, 1, x_{n-1} or $\overline{x_{n-1}}$.

Function implementation; n-input function with multiplexer with $n-1$ select lines

- The circuit...



$f = x_0' x_1 + x_0 x_1' x_2 + x_0 x_1 x_2'$

Function implementation; *n*-input function with multiplexer with 2-to-1 multiplexers

- If we don't have a large enough multiplexer, we can work with smaller multiplexers.
- Consider having only 2-to-1 multiplexers. We can *decompose* algebraically any logic function such that it can be implemented with only 2-to-1 multiplexers.
- Say we have a function f with n inputs. Pick any input variable. Say we select the i-th input x_i. We can always write

$$\begin{aligned} f &= x_i' f(x_0, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_{n-1}) \\ &\qquad + x_i f(x_0, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_{n-1}) \end{aligned}$$

- In other words, we can collect everything involving x_i' together and everything involving x_i together and then factor out x_i' and x_i, respectively.
- This is known as cofactoring.
- Note the "structure" of f after cofactoring... It has the structure of a 2-to-1 multiplexer in which x_i is connected to the select line.

Function implementation; *n*-input function with multiplexer with 2-to-1 multiplexers

- Example... implement

$$f = x_0^{'}x_1 + x_0x_1^{'}x_2 + x_0x_1x_2^{'}$$

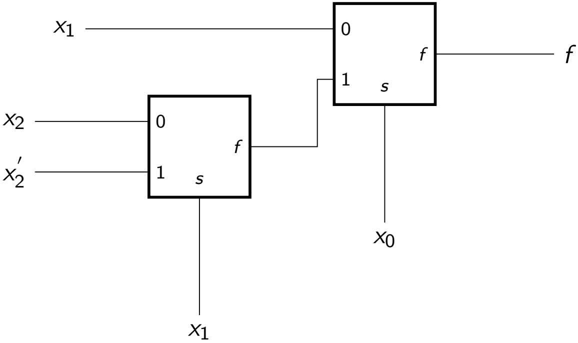
  Using only 2-to-1 multiplexers and cofactoring.

- Factoring...

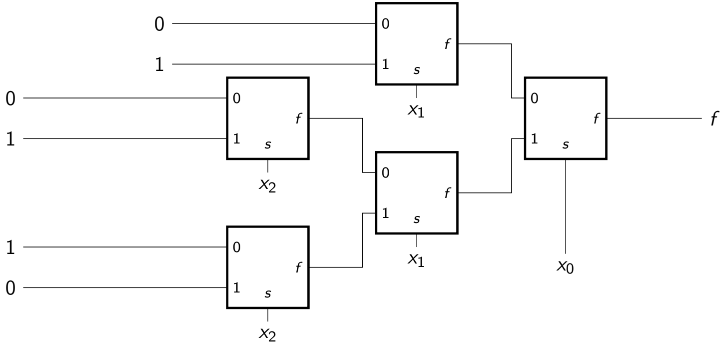
$$\begin{aligned} f &= x_0^{'}(x_1) + x_0(x_1^{'}x_2 + x_1x_2^{'}) \\ &= x_0^{'}(x_1) + x_0(x_1^{'}(x_2) + x_1(x_2^{'})) && \leftarrow \\ &= x_0^{'}(x_1^{'}(0) + x_1(1)) \\ &\quad + x_0(x_1^{'}(x_2^{'}(0) + x_2(1)) + x_1(x_2^{'}(1) + x_2(0))) && \leftarrow \end{aligned}$$

- (Several steps shown depending on "how far you want to go").

Function implementation; *n*-input function with multiplexer with 2-to-1 multiplexers



Function implementation; *n*-input function with multiplexer with 2-to-1 multiplexers



- The order in which you apply cofactoring can impact your circuit complexity (the number of multiplexers).

Demultiplexers

- A demultiplexer does the opposite operation to a multiplexer; it “switches” a single data input line onto one of several output lines depending on the setting of some control lines.
- A demultiplexer can be implemented identically to a decoder by changing the interpretation of the signals.
  - The decoder enable becomes the data input;
  - The decoder data inputs become the control signals.
- Depending on how the decoder inputs (now the control lines), the selected output will appear to follow the value on the enable input (now the data input).