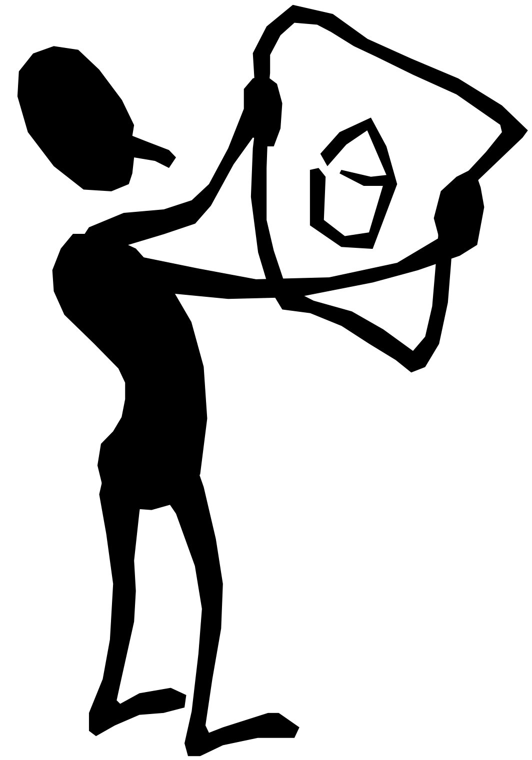
Basic circuit design and multiplexers

- In the first three lectures we learned all the fundamentals needed for making circuits.
  - Truth tables and Boolean expressions describe functions.
  - Expressions can be converted to circuits.
  - Boolean algebra and K-maps help simplify expressions and circuits.
- Today we'll apply all of these foundations to work with some larger circuits.
- We'll also begin introducing common circuits that we'll be using throughout the summer.



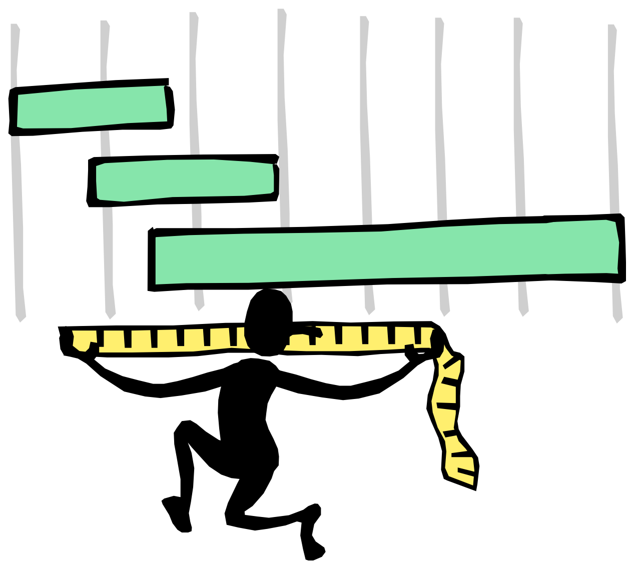
Designing circuits

- The goal in circuit design is to build hardware that solves some problem.
- The basic approach is to express the solution as a Boolean function, which can then be converted to a circuit.

1. Figure out how many inputs and outputs you need.
2. Describe the function as a truth table or a Boolean expression.
3. Find a simplified Boolean expression for the function.
4. Build the circuit based on your simplified expression.

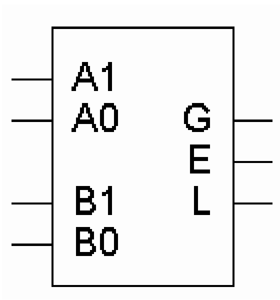
Example: comparing 2-bit numbers

- Let's design a circuit that compares two 2-bit numbers, A and B. There are three possible results: A > B, A = B or A < B.
- We will represent the results using three separate outputs.
  - G ("Greater") should be 1 only when A > B.
  - E ("Equal") should be 1 only when A = B.
  - L ("Lesser") should be 1 only when A < B.
- Make sure you understand the problem!
  - Inputs A and B will be 00, 01, 10, or 11 (0, 1, 2 or 3 in decimal).
  - For any inputs A and B, exactly one of the three outputs will be 1.



Step 1: How many inputs and outputs?

- How many inputs and outputs will this circuit have?
  - Two 2-bit numbers means a total of *four* inputs. Let’s say the first number consists of bits called A1 and A0 (from left to right), while second number has bits B1 and B0.
  - The problem specifies three outputs: G, E and L.
- Here is a block diagram that shows the inputs and outputs explicitly.



- This is like a function header or prototype in programs, which lists the inputs and outputs of a function.
- Now the hard part is to design the circuitry that goes inside the box.

Step 2: Functional specification

- For this problem, it's probably easiest to start with a truth table. This way we can explicitly show the relationship (>, =, <) between the inputs.
- A four-input function has a sixteen-row truth table. For convenience, the rows are in binary numeric order from 0000 to 1111 for A1, A0, B1 and B0.
- For example, 01 < 10, so the sixth row of the truth table (corresponding to inputs A=01 and B=10) shows that output L=1, while G and E are both 0.

A1	A0	B1	B0	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Step 3: Simplified Boolean expressions

- Let's use K-maps to simplify our circuit. There are *three* functions (each with the same inputs A1 A0 B1 B0), so we need *three* K-maps.

			B1	B1	
	0	0	0	0	
	1	0	0	0	A0
A1	1	1	0	1	A0
A1	1	1	0	0	
		B0	B0		

G(A1,A0,B1,B0) =
A1 A0 B0' +
A0 B1'B0' +
A1 B1'

			B1	B1	
	1	0	0	0	
	0	1	0	0	A0
A1	0	0	1	0	A0
A1	0	0	0	1	
		B0	B0		

E(A1,A0,B1,B0) =
A1'A0'B1'B0' +
A1'A0 B1'B0 +
A1 A0 B1 B0 +
A1 A0'B1 B0'

			B1	B1	
	0	1	1	1	
	0	0	1	1	A0
A1	0	0	0	0	A0
A1	0	0	1	0	
		B0	B0		

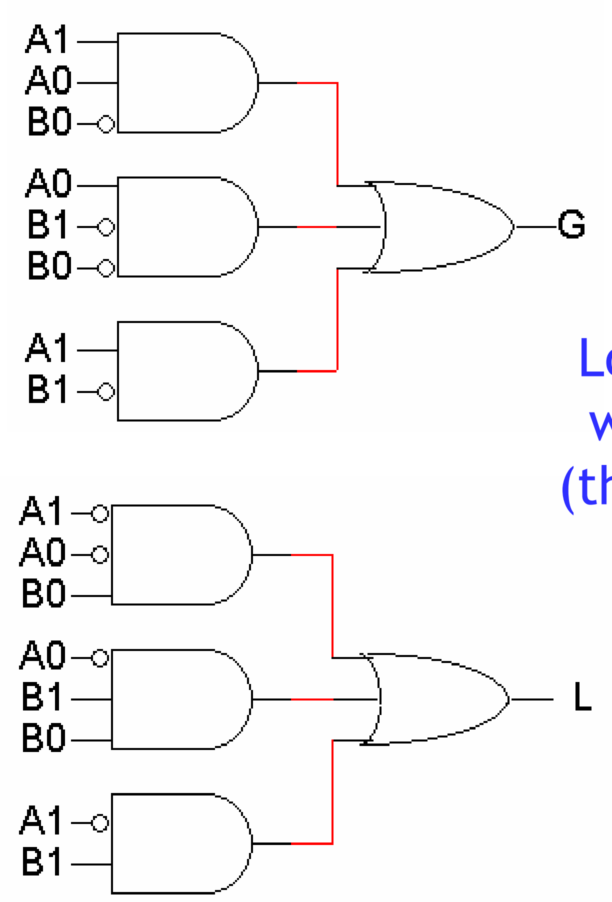
L(A1,A0,B1,B0) =
A1'A0'B0 +
A0'B1 B0 +
A1'B1

Step 4: Drawing the circuits

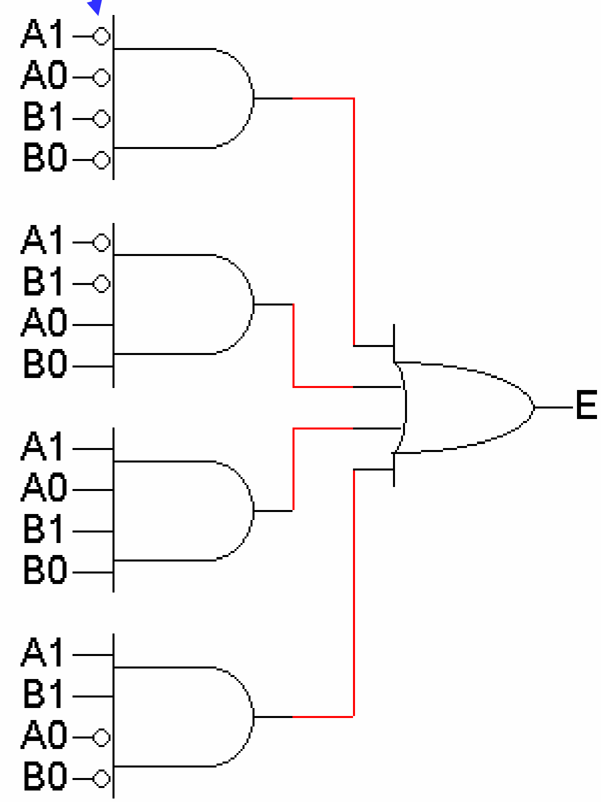
G = A1 A0 B0’ + A0 B1’B0’ + A1 B1’

E = A1’A0’B1’B0’ + A1’A0 B1’B0 + A1 A0 B1 B0 + A1 A0’B1 B0’

L = A1’A0’B0 + A0’B1 B0 + A1’B1

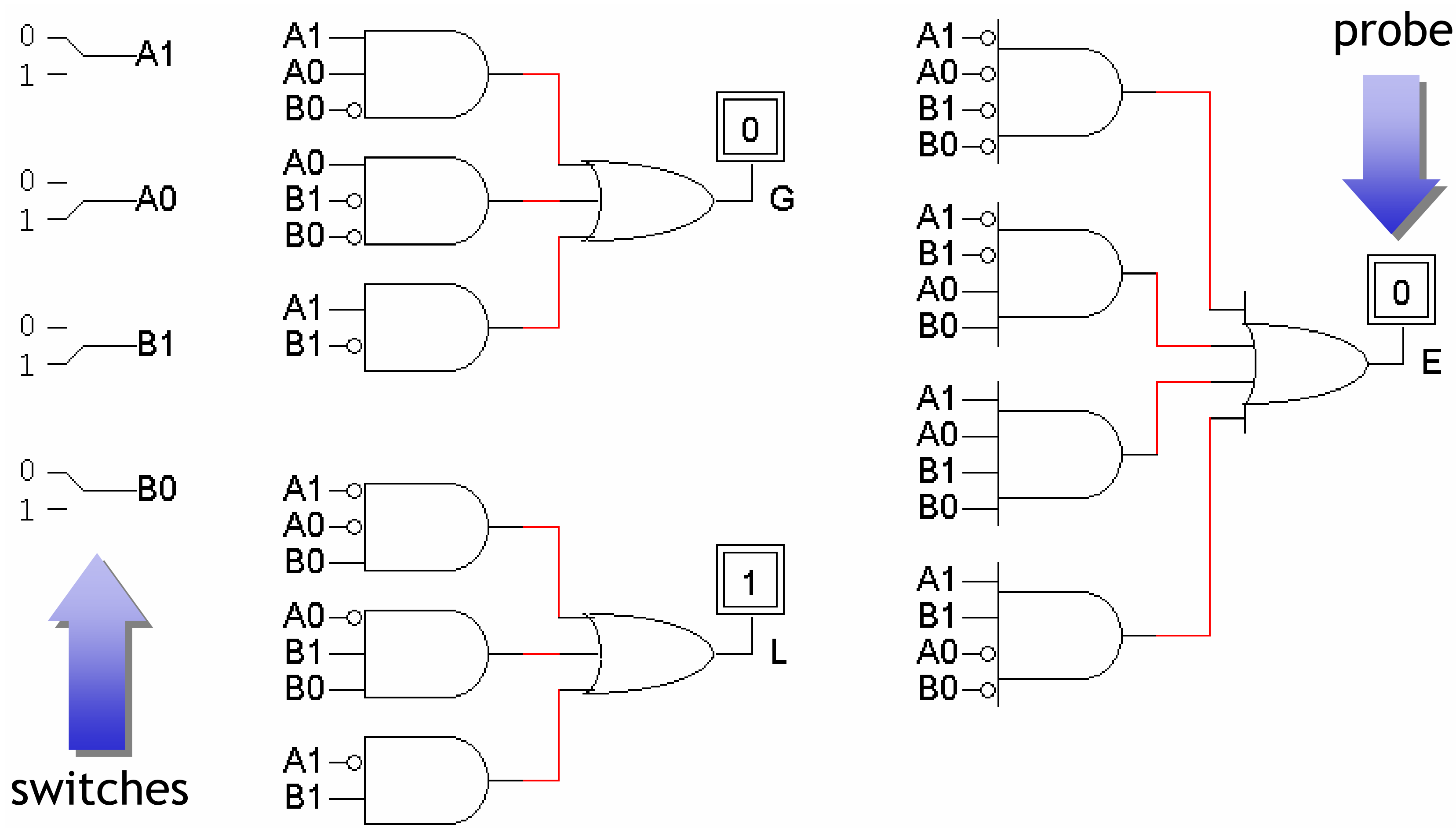


LogicWorks has gates with inverted inputs (the small bubbles) for clearer diagrams.



Testing this in LogicWorks

- In LogicWorks, binary switches provide inputs to your circuit, and binary probes display the outputs.

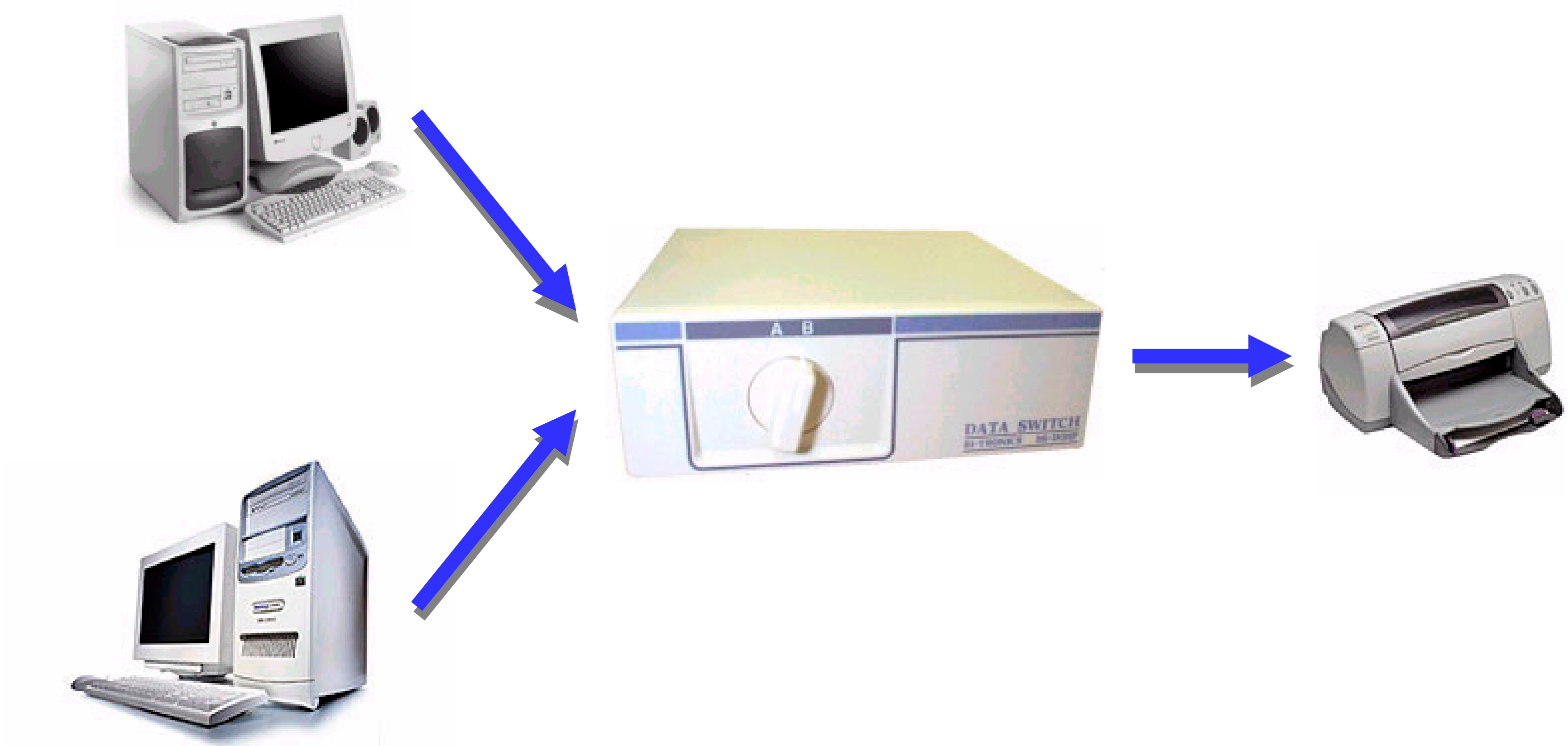


Circuit design issues

- We had to find a suitable data representation for the inputs and outputs.
  - The inputs were just two-bit binary numbers.
  - We used three outputs, one for each possibility of the numbers being greater than, equal to, or less than each other. This is called a "one out of three" code.
- K-maps have advantages but also limitations.
  - Our circuits are relatively simple two-level implementations.
  - But E(A1,A0,B1,B0) couldn't be simplified at all via K-maps. Could we do better using Boolean algebra?
- Our circuit isn't very extensible.
  - We used a brute-force approach, listing all inputs and outputs. This makes it hard to extend our circuit to compare larger numbers.
  - We'll have a better solution after we talk about computer arithmetic.
- There are always many possible ways to design a circuit!

Multiplexers

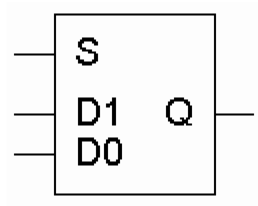
- Let's think about building another circuit, a multiplexer.
- In the old days, several machines could share an I/O device with a switch.



- The switch allows one computer's output to go to the printer's input.

A 2-to-1 multiplexer

- Here is the circuit analog of that printer switch.



- This is a 2-to-1 multiplexer, or mux.
  - There are two data inputs D0 and D1, and a select input called S.
  - There is one output named Q.
- The multiplexer routes one of its data inputs (D0 or D1) to the output Q, based on the value of S.
  - If S=0, the output will be D0.
  - If S=1, the output will be D1.

Building a multiplexer

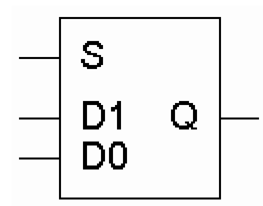
- Here is a truth table for the multiplexer, based on our description from the previous page:

> The multiplexer routes one of its data inputs (D0 or D1) to the output Q, based on the value of S.
> — If S=0, the output will be D0.
> — If S=1, the output will be D1.

S	D1	D0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- You can then find an MSP for the mux output Q.

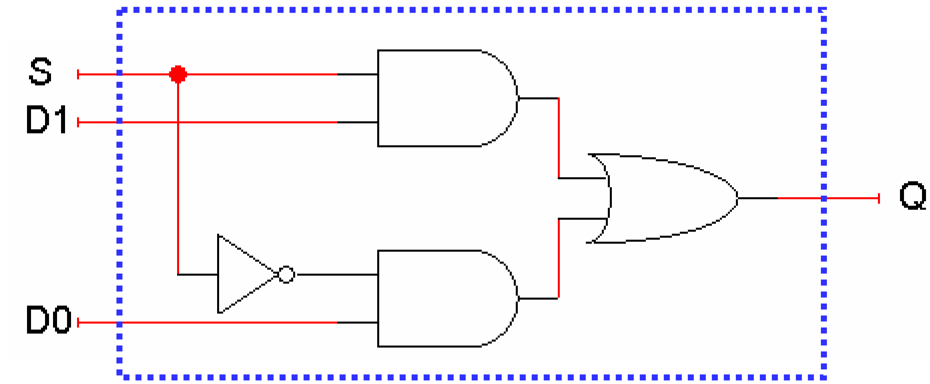
$$Q = S'D0 + S\ D1$$



- Note that this corresponds closely to our English specification above—sometimes you can derive an expression without first making a truth table.

Multiplexer circuit diagram

- Here is an implementation of a 2-to-1 multiplexer.

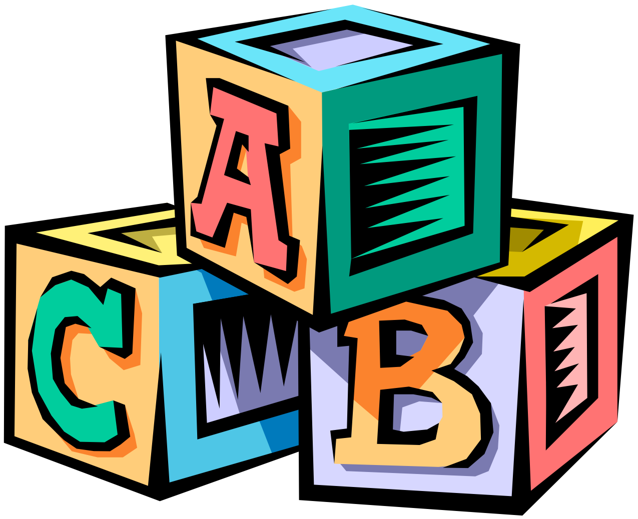


Q = S'D0 + S D1

- Remember that a minimal sum of products expression leads to a minimal two-level circuit.

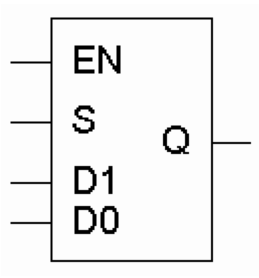
Blocks, abstraction and modularity

- Multiplexers are common enough that we often want to treat them as abstract units or black boxes, as symbolized by our block diagrams.
  - Block symbols make circuit diagrams simpler, by hiding the internal implementation details. You can use a device without knowing how it's designed, as long as you know what it does.
  - Different multiplexer implementations should be interchangeable.
  - Circuit blocks also aid hardware re-use, since you don't have to keep building a multiplexer from scratch every time you need one.
- These blocks are similar to functions in programming languages!



Enable inputs

- Many devices have an additional enable input, which "activates" or "deactivates" the device.
- We could design a 2-to-1 multiplexer with an enable input that's used as follows.
  - EN=0 disables the multiplexer, which forces the output to be 0. (It does *not* turn off the multiplexer.)
  - EN=1 enables the multiplexer, and it works as specified earlier.
- Enable inputs are especially useful in combining smaller muxes together to make larger ones, as we'll see later today.



EN	S	D1	D0	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Truth table abbreviations

EN	S	D1	D0	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

- Notice that when EN=0, then Q is always 0, regardless of what S, D1 and D0 are set to.
- We can shorten the truth table by including Xs in the input variable columns, as shown on the bottom right.

EN	S	D1	D0	Q
0	x	x	x	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Another abbr. 4 U

- Also, when EN=1 notice that if S=0 then Q=D0, but if S=1 then Q=D1.
- Another way to abbreviate a truth table is to list input variables in the output columns, as shown on the right.

EN	S	D1	D0	Q
0	x	x	x	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

EN	S	Q
0	x	0
1	0	D0
1	1	D1

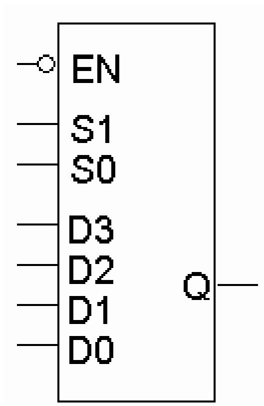
- This final version of the 2-to-1 multiplexer truth table is much clearer, and matches the equation Q = S'D0 + S D1 very closely.

A KVM switch

- This KVM switch allows four computers to share a single keyboard, video monitor, and mouse.

A 4-to-1 multiplexer

- Here is a block diagram and abbreviated truth table for a 4-to-1 mux, which directs one of four different inputs to the single output line.
  - There are four data inputs, so we need *two* bits, S1 and S0, for the mux selection input.
  - LogicWorks multiplexers have active-low enable inputs, so the mux always outputs 1 when EN’ = 1. This is denoted on the block symbol with a bubble in front of EN.

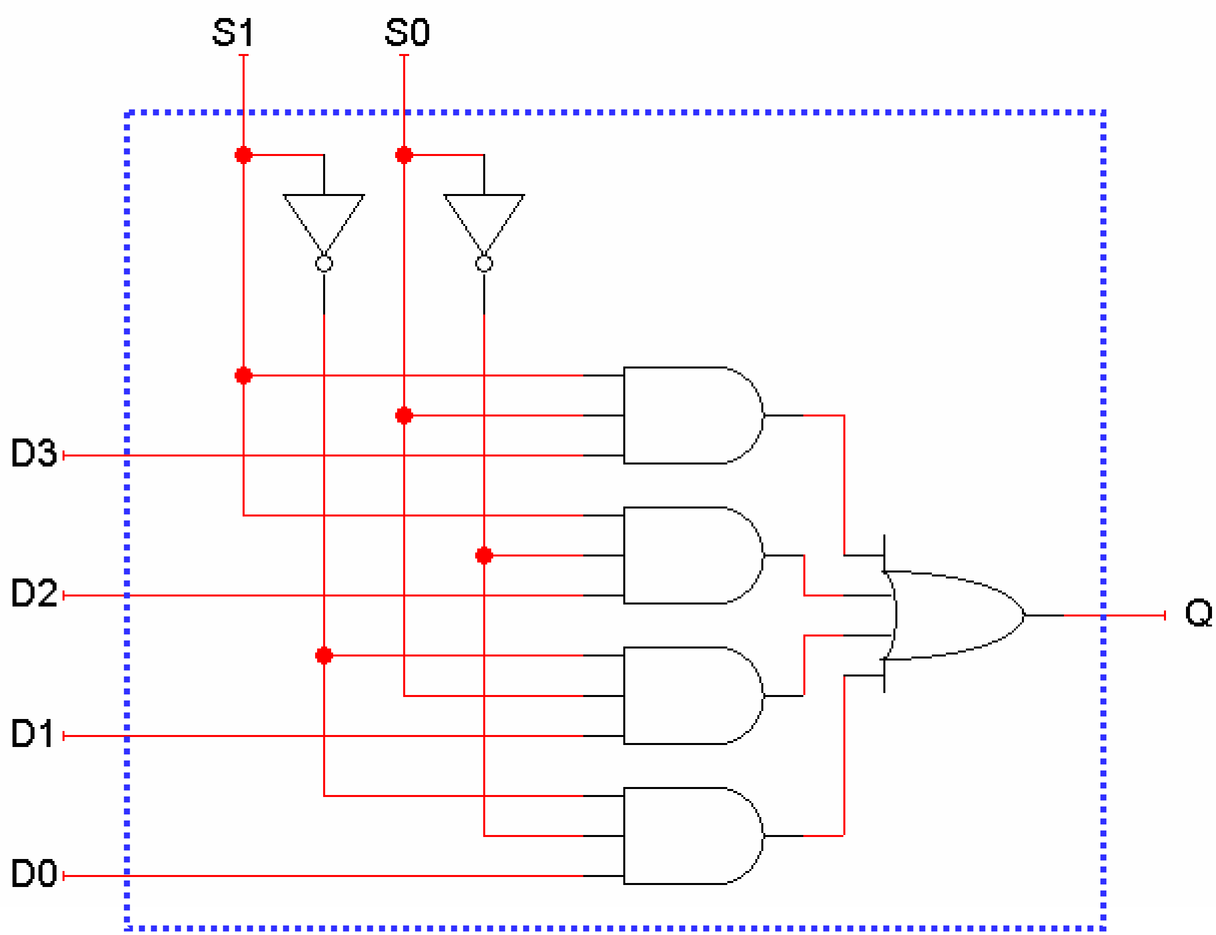


EN’	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	x	x	1

Q = S1’S0’D0 + S1’S0 D1 + S1 S0’D2 + S1 S0 D3

A 4-to-1 multiplexer implementation

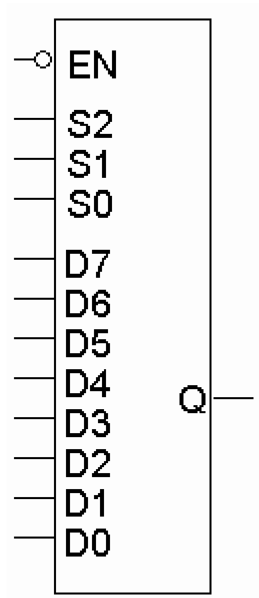
- Again we have a minimal sum of products expression, which leads to a minimal two-level circuit implementation.



$$Q = S1'S0'D0 + S1'S0\ D1 + S1\ S0'D2 + S1\ S0\ D3$$

2^n-to-1 multiplexers

- You can make even larger multiplexers, following the same pattern.
- A 2^n-to-1 multiplexer routes one of 2^n input lines to the output line.
  - There are 2^n data inputs, so there must also be n select inputs.
  - The output is a single bit.
- Here is an 8-to-1 multiplexer, probably the biggest we'll see in this class.



Example: addition

- Multiplexers can sometimes make circuit design easier.
- As an example, let's make a circuit to add three 1-bit inputs X, Y and Z.
- We'll need two bits to represent the total.
  - The bits will be called C and S, standing for "carry" and "sum."
  - These are two separate functions of the inputs X, Y and Z.
- A truth table and sum of minterm equations for C and S are shown below.

	X	Y	Z	C	S
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
0 + 1 + 1 = 10	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
1 + 1 + 1 = 11	1	1	1	1	1

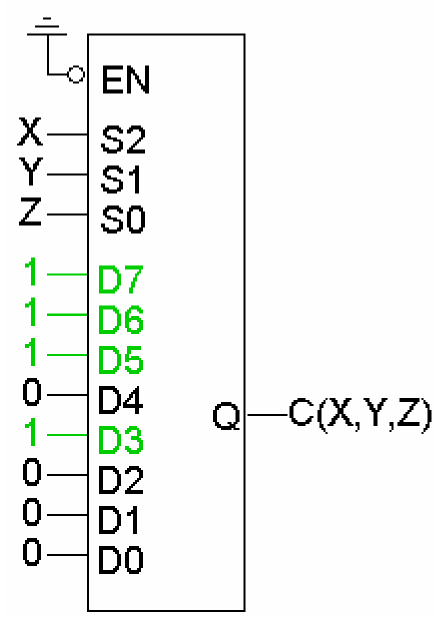
$C(X,Y,Z) = \Sigma m(3,5,6,7)$

$S(X,Y,Z) = \Sigma m(1,2,4,7)$

Implementing functions with multiplexers

- We could implement a function of *n* variables with an *n*-to-1 multiplexer.
  - The mux select inputs correspond to the function's input variables, and are used to select one row of the truth table.
  - Each mux data input corresponds to one output from the truth table. We connect 1 to data input Di for each function minterm m_i, and we connect 0 to the other data inputs.
- For example, here is the carry function, $C(X,Y,Z) = \Sigma m(3,5,6,7)$.

X	Y	Z	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Partitioning the truth table

- We can actually implement $C(X,Y,Z) = \Sigma m(3,5,6,7)$ with just a 4-to-1 mux, instead of an 8-to-1.
  - Instead of using three variables to select one row of the truth table, we'll use two variables to pick a *pair* of rows in the table.
  - The multiplexer data inputs will be functions of the remaining variable, which distinguish between the rows in each pair.
- First, we can divide the rows of our truth table into pairs, as shown on the right. X and Y are constant within each pair of rows, so C is a function of Z only.
  - When XY=00, C=0
  - When XY=01, C=Z
  - When XY=10, C=Z
  - When XY=11, C=1

X	Y	Z	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A more efficient adder

- All that's left is setting the multiplexer inputs.
  - The two input variables X and Y will be connected to select inputs S1 and S0 of our 4-to-1 multiplexer.
  - The expressions for C(Z) are then connected to the data inputs D0-D3 of the multiplexer.

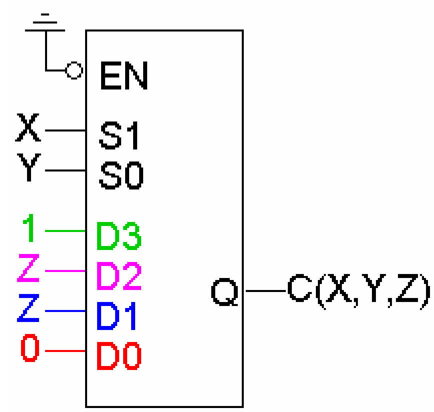
X	Y	Z	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

When XY=00, C=0

When XY=01, C=Z

When XY=10, C=Z

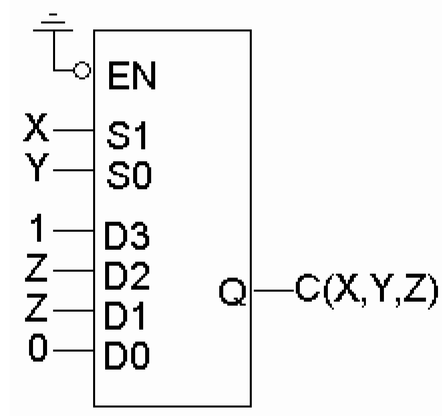
When XY=11, C=1



Verifying our adder

- Don't believe that this works? Start with the equation for a 4-to-1 multiplexer from earlier in the lecture.

$$Q = S1'S0'D0 + S1'S0\ D1 + S1\ S0'D2 + S1\ S0\ D3$$



- Then just plug in the actual inputs to our circuit, as shown again on the right: S1S0 = XY, D3 = 1, D2 = Z, D1 = Z, and D0 = 0.

$$\begin{aligned} C &= X'Y' \bullet 0 + X'YZ + XY'Z + XY \bullet 1 \\ &= X'YZ + XY'Z + XY \\ &= X'YZ + XY'Z + XY(Z' + Z) \\ &= X'YZ + XY'Z + XYZ' + XYZ \end{aligned}$$

- So the multiplexer output really is the carry function, $C(X,Y,Z) = \Sigma m(3,5,6,7)$.

Multiplexer-based sum

- Here's the same thing for the sum function, $S(X,Y,Z) = \Sigma m(1,2,4,7)$.

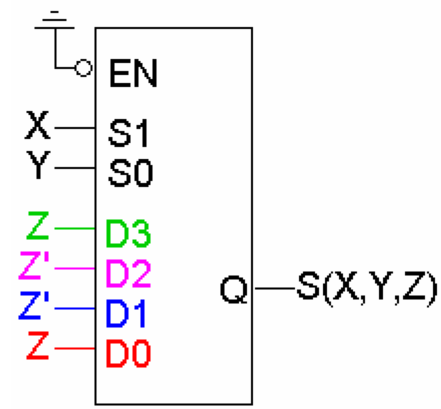
X	Y	Z	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

When XY=00, S=Z

When XY=01, S=Z'

When XY=10, S=Z'

When XY=11, S=Z

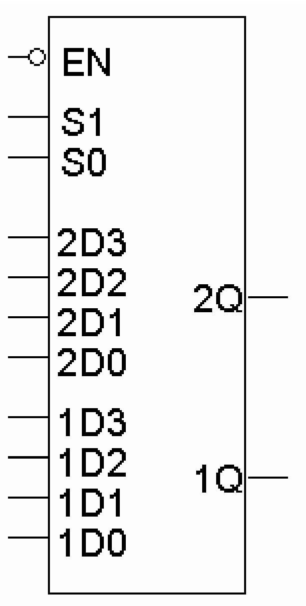


- Again, we can show that this is a correct implementation.

$$\begin{aligned} Q &= S1'S0'D0 + S1'S0\ D1 + S1\ S0'D2 + S1\ S0\ D3 \\ &= X'Y'Z + X'YZ' + XY'Z' + XYZ \\ &= \Sigma m(1,2,4,7) \end{aligned}$$

Dual multiplexers

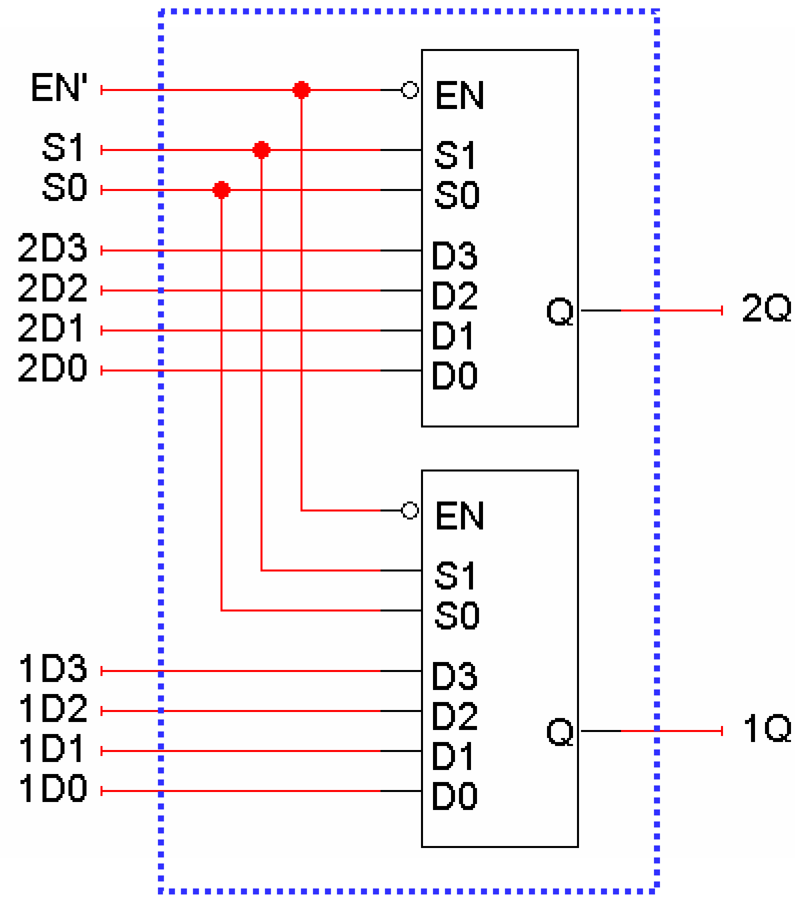
- A dual 4-to-1 mux allows you to select from one of four 2-bit data inputs.
- The Mux-4×2 T.S. device in LogicWorks is shown here.
  - The two output bits are 2Q 1Q, and S1-S0 select a *pair* of inputs.
  - LogicWorks labels the *x*-th bit of data input *y* as xDy.



EN'	S1	S0	2Q	1Q
0	0	0	2D0	1D0
0	0	1	2D1	1D1
0	1	0	2D2	1D2
0	1	1	2D3	1D3
1	x	x	1	1

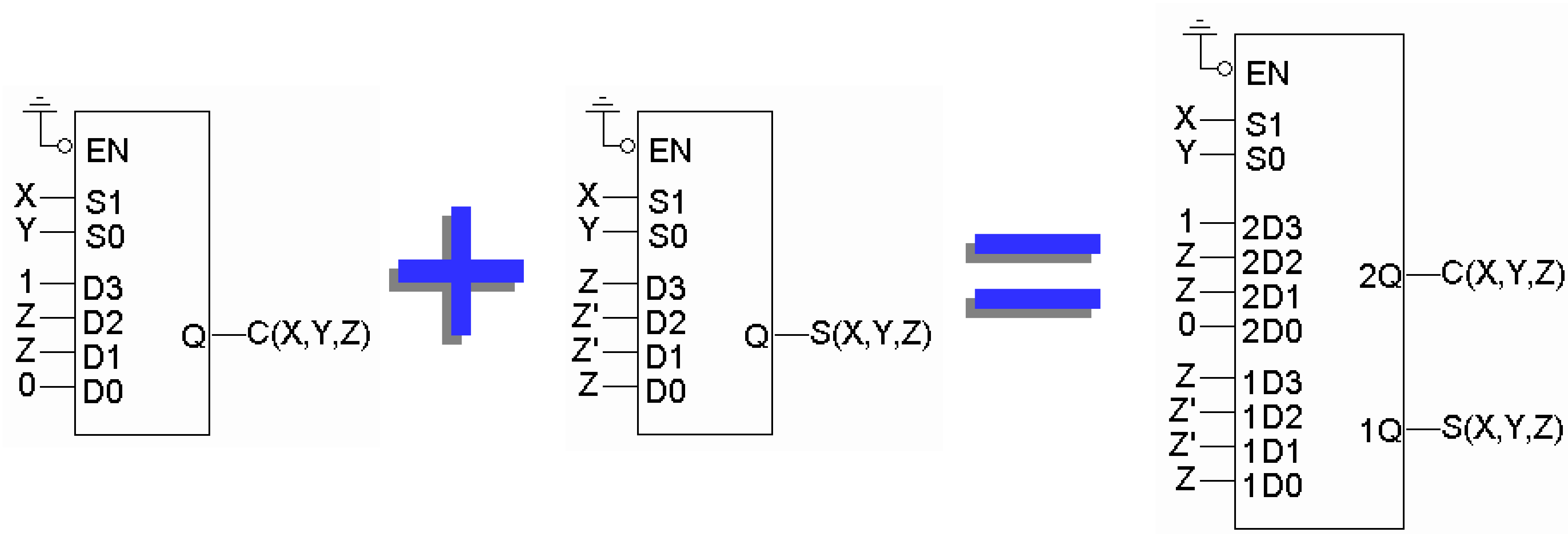
Dual muxes in more detail

- You could build a dual 4-to-1 mux from its truth table and our familiar circuit design techniques.
- It's also possible to combine smaller muxes together to form larger ones.
- You can build the dual 4-to-1 mux just by using two 4-to-1 muxes.
  - The two 4-to-1 multiplexers share the same EN', S1 and S0 signals.
  - Each smaller mux produces one bit of the two-bit output 2Q 1Q.
- This kind of hierarchical design is very common in computer architecture.



Dual multiplexer-based adder

- We can use this dual 4-to-1 multiplexer to implement our adder, which produces a two-bit output consisting of C and S.



- That KVM switch from earlier is really a "tri 4-to-1 multiplexer," since it selects from four sets of three signals (keyboard, video and mouse).

Summary

- Today we began designing circuits!
  - Starting from a problem description, we came up with a truth table to show all possible inputs and outputs.
  - Then we built the circuit using primitive gates or multiplexers.
- A 2^n-to-1 multiplexer routes one of 2^n inputs to a single output line.
  - Muxes are a good example of our circuit design techniques.
  - They also illustrate abstraction and modularity in hardware design.
  - We saw some variations such as active-low and dual multiplexers.
- Tomorrow we'll present another commonly-used device and show how it can also be used in larger circuits.