Exercise 1 – Multiplexer

C The "M" output column of each row in the truth table, from top to bottom, corresponds to the D0, D1, D2, … D7 inputs of the multiplexer. The top row of the truth table shows that the "M" result is supposed to be TRUE, so the "D0" input of the multiplexer must be connected to 5 Volts (because 5V=TRUE and 0V=FALSE). The second row of the truth table shows a FALSE result, so the D1 input of the multiplexer must be connected to 0 Volts – and so on for the remaining rows.

Exercise 2 – Decoder Circuit

B The only AND gate that will have a TRUE output is the one that has two TRUE inputs. The left input of the circuit is FALSE, but it's connected to the top two AND gates through an inverter, so both those AND gates have one TRUE input. The right input of the circuit is TRUE, and it's connected directly to the second AND gate, so that gate has two TRUE inputs and therefore has a TRUE output.

Exercise 3 – Programmable Logic Array

C The first four 6-input AND gates correspond to the product terms in the sum-of-products expression. The fuse marked "A" is correct because it connects the C input and corresponds to the "C" in "(NOT-A•B•C). The fuse that's incorrect is the one marked "C", because it's connected to the NOT-B input even though it corresponds to the "B" in (A•B•NOT-C). The "E" fuse doesn't have to be connected because there are only four product terms in the Boolean expression, so we only need to use four of the AND gates and the fifth gate doesn't have to be connected to the output.

Exercise 4 – Full Adder

B This AND gate is showing a TRUE output even though both the inputs are not TRUE. It should in fact have a FALSE output.

Exercise 5 – Carry Select Adder

C The design of this circuit is similar to the shifter in that it uses a series of AND gates to block everything while allowing only the correct signal to pass through, followed by an OR gate to pass the signal through from whichever AND gate is "open". Answer "C" is correct because it's the one where when the "Carry In" signal is TRUE, this is passed unchanged to the AND gate connected to the "Cn" input, and when "Carry In" is FALSE the inverted (TRUE) signal is passed to the AND gate with the "NCn" (No-Carry-n) input.

Exercise 6 – ALU Function Codes

E – **none of the above**

The F0/F1 inputs are selecting the "A and B function", but only the "A" input is enabled. This means that all of the "B" inputs will be forced to zeros. With the AND function, the result for each output bit can only be TRUE if both the A and the B input bits are TRUE – but since the B inputs are all FALSE (because ENB is 0), this means all of the output bits are FALSE too. So the value that comes out of the ALU is zero (every output bit will be FALSE).

Exercise 7 – Signed Numbers

1. B – 1 1 1 1 1 0 1 0

To find the signed number that's equivalent to -6, we start with the value for +6 and go through the two's complement operation (flip the bits and then add one). +6 is a number with the bits for the values "2" and 4" turned on:

+ 6:	0	0	0	0	0	1	1	0
reverse bits:	1	1	1	1	1	0	0	1
add one:	1	1	1	1	1	0	1	0

2. F – 7

If the value "**1 1 1 1 1 0 0 1**" is a signed number, then it must be negative because the high-order (first, or leftmost) bit is "1". So to determine it's value we must first convert it to a positive value so we can add up the value of the bits:

original number:	1	1	1	1	1	0	0	1
flip bits:	0	0	0	0	0	1	1	0
add one:	0	0	0	0	0	1	1	1

Counting from the low-order (rightmost) bit, the values are 1, 2 and 4 – these add up to "7". So the original number must be "-7" (it's negative because the original number has the sign bit set to TRUE).

3. A – 1 1 0 0 0 1 0 1

The two numbers are added just like any other binary number without worrying about whether they're positive or negative:

(carries)	1	1	1	1	1	1		
– 42	1	1	0	1	0	1	1	0
– 17	1	1	1	0	1	1	1	1
sum:	1	1	0	0	0	1	0	1

If we've added the numbers correctly, the result should be equal to (-42) + (-17) = -59. We can check this by doing the two's-complement operation on the result and seeing if the values of the 1-bits add up to 59:

result:	1	1	0	0	0	1	0	1
reverse bits:	0	0	1	1	1	0	1	0
add one:	0	0	1	1	1	0	1	1

Starting at the low-order (right) end, the result bits are worth: 1 + 2 + 8 + 16 + 32 = 59. Therefore we have correctly added the two numbers.