**Chapter 4**

**The Building Blocks: Binary Numbers, Boolean Logic, and Gates**

**A Guide to this Instructor’s Manual:**

We have designed this Instructor’s Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared towards quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor’s Manual, our Instructor’s Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

|  |
| --- |
| **At a Glance** |

**Instructor’s Manual Table of Contents**

* Overview
* Learning Objectives
* Teaching Tips and Quick Quizzes
* Class Discussion Topics
* Additional Projects
* Additional Resources
* Key Terms
* Solutions to End-of-Chapter Exercises

|  |
| --- |
| Lecture Notes |

**Overview**

Chapter 4 introduces the hardware level of computer systems. It describes how binary representations of numbers and characters work, including sign-magnitude and two’s complement representations for integers, and the ASCII table for mapping characters to binary numbers. It discusses how digitized sound and images work, through sampling and representation of wave magnitudes, for sounds, and colors or intensities, for images. The chapter discusses the importance of Boolean logic and the mapping between true/false values and 1/0 values. It shows how to construct gates that implement Boolean operators from transistors. The chapter uses a specific algorithm, sum-of-products, for designing circuits and illustrates the power of such circuits by building an adder, a compare-for-equality circuit, and multiplexer and decoder control circuits.

# **Learning Objectives**

* Translate between base-ten and base-two numbers, and represent negative numbers using both sign-magnitude and two’s complement representations
* Explain how fractional numbers, characters, sounds, and images are represented inside the computer
* Build truth tables for Boolean expressions and determine when they are true or false
* Describe the relationship between Boolean logic and electronic gates
* Construct circuits using the sum-of-products circuit design algorithm, and analyze simple circuits to determine their truth tables
* Describe the purpose and workings of multiplexer and decoder control circuits

# **Teaching Tips**

**4.1 Introduction**

1. This chapter changes the focus from algorithms and an abstract notion of computing agents to focus on how digital computers work. Point out the fundamental building blocks of all computer systems—binary representation, Boolean logic, gates, and circuits. In particular, emphasize that all digital computers, including calculators, cell phones, laptops, desktops, game systems, and supercomputers, even singing get-well cards, all use the fundamentals from this chapter.

**4.2 The Binary Numbering System**

1. On the surface, computers use familiar, everyday notations for text and numbers, but internally they use a very different form. Their internal storage techniques are quite different from the way people represent information because computer designers needed a way to encode information that was clear, unambiguous, and reliable.
2. Introduce the term **binary number system** and define it as a base-2 **positional numbering system**. Use examples from our base-10 numbering to explain how our number places are given values according to powers of 10, and then introduce the base-2 approach that uses powers of 2 for each place. Note that students may become confused by the term decimal, used to thinking of it not as “base-10” but as meaning “real numbers.”
3. Discuss how to convert a binary number to a decimal one by adding up powers of two. Converting from a decimal number to a binary one requires repeatedly dividing by 2 and recording the remainder. If the remainder values are recorded right to left as the division is done, the result will be the binary number. Have students work through examples after you have demonstrated the two methods.

|  |  |
| --- | --- |
| ***Teaching Tip*** | Refer your students to the following website by Christine Wright and Sam Rebelsky at Grinnell College for a tutorial on binary numbers: <http://www.math.grin.edu/~rebelsky/Courses/152/97F/Readings/student-binary> |

1. Discuss the limitations of a finite, fixed number of digits: that a computer has a maximum integer it can represent. Introduce the term **arithmetic overflow** to describe what happens when the computer attempts to represent an integer that is too large. Demonstrate binary number addition, drawing a comparison to grade school methods for adding large numbers. Emphasize the need for a carry digit.
2. Introduce the terms **sign/magnitude notation** and **two’s complement representation**, which are two ways to represent both positive and negative integers in binary. Discuss the pros and cons of each. The circular diagram for two’s complement in the text is very useful for illuminating the details of that representation. How does a computer know how to read a signed integer? Remind students that the computer doesn’t know and that it is based on the context in which the binary number is used.
3. Introduce the term **scientific notation**: a method for representing real numbers in base 10. Then show how to apply the same notation to binary real numbers and how that is the basis for representing real numbers in the computer.
4. To represent text in binary, we need to map every character to a unique binary number. **ASCII** and **Unicode** are two standard mappings used by modern computers. Remind students that while ASCII is 8 bits and Unicode is 16 or 32 bits, 0–127 are reserved in both ASCII and Unicode to represent the same characters.
5. Discuss the terms **digital representation** and **analog representation** and note that digital representations don’t have to be binary. Keep the primary discussion to digital representations.
6. Sound waves are characterized by **amplitude**, **period**,and **frequency.** Define these terms, and discuss how we estimate the continuous sound wave in a **digitized** way through **sampling**. Quality of the sampled sound depends on **sampling rate** and **bit depth**. Use clear pictures to illustrate the effect of each of these parameters. Demonstrate how a low sampling rate will miss key elements of sound.
7. Photographs are digitized by sampling the color or intensity at fixed points equally spaced across the image. **Raster graphics** represents the grid of colors/intensities in this way. Compare the information needed for black-and-white, grayscale, and color images. Introduce the term **RGB encoding scheme**; if each color is one **byte**, have students compute how many different colors can be represented.

|  |  |
| --- | --- |
| ***Teaching Tip*** | Refer your students to the following website for a nice tutorial about digital images: <http://www.library.cornell.edu/preservation/tutorial/contents.html> |

1. Discuss the importance of **data compression**. Compare the amount of memory needed to represent a collection of 1,000 integers, a 10-page paper, a 60-second sound file, and a 480 by 640 image. How might each of these formats benefit from compression? Compare between lossy and lossless compression. Discuss applications for both.
2. Binary representations are used in the computer because of their great reliability. Discuss the four criteria for a good binary storage device and how different electronic devices fit these criteria.
3. Introduce the term **transistor** and explain how it works. The tiny size of transistors enables modern computer memories and chips (introduce the term **gigabyte**). Explain how transistors work.

**Quick Quiz 1**

1. Arithmetic overflow happens when the computer tries to represent a(n) \_\_\_\_\_\_\_ that is too large for its integer encoding system.   
   Answer: integer
2. (True or false) Sign/magnitude notation and scientific notation are two ways of representing signed integer inside the computer.

Answer: False

1. RGB encoding scheme describes the storage format of what kind of digital data?

Answer: Digital images, photographs, and graphics

1. Why did designers choose to build electronic computers using base-2 representations instead of base 10?

Answer: Reliability

1. What does the MPEG stand for in the MPEG-1 (and 2) audio formats?

Answer: Motion Picture Experts Group

**4.3 Boolean Logic and Gates**

1. **Boolean logic** describes the rules for manipulating true and false and is used to build circuits. Introduce the terms **hardware design** and **logic design**. Emphasize the connection between true/false and the 1/0 of binary representations.
2. Use examples from algorithms in earlier chapters to illustrate **Boolean expressions.** Discuss Boolean operations and their **truth tables.** Have students work through samples using AND, OR, and NOT, discussing them with each other to determine the results. Be sure to include nested Boolean operators.
3. Introduce the term **gate**, and emphasize that each gate corresponds to a Boolean operator. Discuss how gates may be constructed from transistors (or other bistable devices). The most important point is that anything we can describe with Boolean logic can be implemented as a circuit built with gates.

**4.4 Building Computer Circuits**

1. Introduce the term **circuit**. Note that every Boolean expression can be represented pictorially as a circuit diagram, and illustrate how to construct a simple circuit from a Boolean expression.
2. Discuss the importance of **circuit construction algorithms** to help design circuits. Work through a specific example of the steps of the sum-of-products algorithm: constructing a truth table, subexpression construction, subexpression combination, and circuit diagram production.
3. The compare-for-equality (CE) circuit tests two binary numbers for exact equality. It is built up from a one-bit compare-for-equality circuit, (1-CE). Illustrate the process of constructing both smaller and larger versions of the circuit, and work through examples to ensure students understand how both Boolean expressions and circuits work.
4. The binary addition circuit is one of the most amazing to students. The 1-ADD circuit is complicated; students must work through it until they understand it. The full adder has a structure similar to the binary addition algorithm discussed earlier; emphasize these similarities.

**Quick Quiz 2**

1. The name of the circuit construction algorithm in this section is \_\_\_\_\_\_\_\_\_\_\_\_.  
   Answer: sum-of-products
2. What is another term for “hardware design?”

Answer: Logic design

1. (True or false) The first step of the sum-of-products algorithms is to construct a truth table.

Answer: True

1. To build a compare-for-equality circuit that takes two 8-bit binary numbers as input, the circuit would need to include how many one-bit CE circuits?
2. 1, (b) 4, (c) 8, (d) 16

Answer: (c)

**4.5 Control Circuits**

1. Introduce the term **control circuits**, and emphasize the critical importance of circuits that can make decisions, select values, and control what happens when.
2. Introduce the term **multiplexer**, which takes in 2N input lines and N selector lines and it outputs the value on one input line, the line whose number is given by the selector lines. Show students a very small example, perhaps with 2 selector lines and 4 inputs, and illustrate how each selector value maps to a single input line. Have students work through the details of a 2-input and 4-input multiplexer circuit diagram.
3. Introduce the term **decoder**, and emphasize that its purpose is the opposite of the multiplexer. The multiplexer chooses one of 2N input lines to output, and the decoder sends a 1 out along one of 2N output lines. In each case, the N input/selector lines are used to number the 2N lines.
4. Discuss Figures 4.35 and 4.38 in the text, which illustrate how control circuits contribute to the creation of a computing device. A decoder circuit can be used to select among different arithmetic operations; a multiplexer can be used to select the correct value from a collection of values.

**Quick Quiz 3**

1. A(n) \_\_\_\_\_\_\_\_\_\_\_\_ circuit uses its input lines to choose which among its many outputs to send a 1 on.  
   Answer: decoder
2. (True or false) Multiplexers can be used to select one among a set of values.

Answer: True

1. (True or false) Control circuits are primarily concerned with performing arithmetic.

Answer: False

1. A(n) \_\_\_\_\_\_\_\_\_\_ circuit takes 2N input lines and N selector lines and ouputs the value on one input line, the line whose number is given by the selector lines.

Answer: multiplexer

# **Class Discussion Topics**

1. Why do we use binary encodings to represent information in the computer? What would be the pros and cons of using base-10 instead of base-2?
2. Explain the relationship between Boolean logic and computer circuits. Why is Boolean logic so important to computer science?
3. Explain the purpose of the steps in the sum-of-products circuit design algorithm. Why must each step be done, and why in that order?

# **Additional Projects**

1. There are other Boolean operators, such as XOR and NAND. XOR stands for “exclusive OR”; it produces true if exactly one of its arguments is true and false otherwise. NAND stands for “Not and”; it produces true so long as both arguments are not true. Construct a Boolean circuit using only AND, OR, and NOT gates that implements XOR and NAND.
2. Draw a Boolean circuit using AND, OR, and NOT gates that implements the majority-rules Boolean function. In this function, it produces the output 1 if half or more of its inputs are 1. Build the circuit first for two inputs and then for three.
3. A computer’s central processing unit usually operates on 8-, 16-, or 32-bit numbers at once. How might you put together multiple multiplexers to select 8, 16, or 32 lines given a certain code, instead of just one?

# **Additional Resources**

1. A nice web applet that does conversion between binary, decimal, and hexadecimal numbers: <http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html>
2. A useful web page about subtraction and two’s complement numbers: <http://courses.cs.vt.edu/csonline/NumberSystems/Lessons/SubtractionWithTwosComplement/index.html>
3. A web page that describes how simple memory circuits work: <http://www.electronics-tutorials.ws/sequential/seq_1.html>
4. Additional information on multiplexer circuits: <http://iitg.vlab.co.in/?sub=59&brch=165&sim=904&cnt=1>
5. Additional information on decoder circuits: <http://www.electronics-tutorials.ws/combination/comb_5.html>

**Key Terms**

* **Amplitude:** The height of a periodic wave which is a measure of its loudness.
* **Analog representation:** Objects can take on any continuous value.
* **Arithmetic overflow:** An attempt to represent an integer that exceeds the maximum allowable value.
* **ASCII:** An acronym for the American Standard Code for Information Interchange; ASCII is an international standard for representing textual information in the majority of computers using 8 bits per character.
* **Binary number system:** A base-2 positional numbering system.
* **Bit:** A binary digit, 0 or 1.
* **Bit depth:** The number of bits used to encode each sample during digitization.
* **Boolean expression:** An expression that can evaluate only to true or false.
* **Boolean logic:** A branch of mathematics which operates on the values true and false.
* **Byte:** Eight bits.
* **Circuit:** A collection of logic gates (1) that transforms a set of binary inputs into a set of binary outputs and (2) where the values of the outputs depend only on the current values of the inputs—more properly called a combinational circuit.
* **Circuit construction algorithm:** An algorithm that allows us to go from a specification of what we wish to accomplish to a circuit that carries out those specifications.
* **Circuit optimization:** The process of reducing the number of gates needed to implement a circuit.
* **Compression ratio:** Measures how much a compression scheme has reduced the storage requirements of the data.
* **Control circuit:** A circuit used to make decisions and control the flow of execution.
* **Data compression:** The process of reducing the number of bits required to represent a sound or image.
* **Decoder:** A control circuit that has N input lines numbered 0, 1, 2,…, N – 1 and 2N output lines numbered 0, 1, 2, 3,…, 2N – 1.
* **Digital representation:** The values for a given object are drawn from a finite set, such as the letters {A, B, C,…, Z} or a subset of integers {0, 1, 2, 3,…, MAX}.
* **Digitized:** Converted from a continuous value to a single numeric value.
* **Fault-tolerant computing:** [in Exercise 24] The ability to continue functioning even in the presence of the failure of one or more components.
* **Frequency:** The total number of cycles per unit time measured in cycles/second, also called hertz.
* **Gate:** An electronic device that operates on a collection of binary inputs to produce a binary output.
* **Gigabyte:** One billion bytes.
* **Hardware design:** The process of designing the low-level components of a computer, including arithmetic and control circuits.
* **Logic design:** Another term for hardware design as it uses the capabilities of Boolean logic to carry out the design process.
* **Lossless compression:** No information is lost in the compress, and it is possible to reproduce exactly the original data.
* **Lossy compression:** Compress data in a way that does not guarantee that all the information in the original data can be fully and completely recreated.
* **Multiplexer:** A control circuit that has 2N input lines and 1 output line.
* **Period:** The time it takes for a single wave in a periodic wave function.
* **Positional numbering system:** A numbering system in which each position of a number represents a value times the radix to a given power.
* **Raster graphics:** A method for storing an image in which a sequence of picture elements is digitized and stored one row at a time, from left to right.
* **RGB encoding scheme:** A method for encoding color that digitizes the contribution of the red, green, and blue components of each pixel.
* **Sampling:** At fixed time intervals, the amplitude of a signal is measured and stored as an integer value; the wave is then represented in the computer in digital form as a sequence of sampled numerical amplitudes.
* **Sampling rate:** The time interval between sampling points.
* **Scientific notation:** A way to represent real numbers as a mantissa times a base to an exponential power.
* **Sequential circuit:** Circuit that contains feedback loops in which the output of a gate is fed back as input to an earlier gate.
* **Sign/magnitude notation:** A way to represent signed integer values in which one bit is used to represent the sign and the remaining bits are used to represent the magnitude.
* **Transistor:** An electronic device that can be in an OFF state, which does not allow electricity to flow, or in an ON state, in which electricity can pass unimpeded; a transistor is a solid-state device that has no mechanical or moving parts.
* **Truth table:** A table that contains columns labeled inputs that list the possible combinations of true/false values.
* **Two’s complement representation:** A way to represent signed integer in which we count up from zero to represent positive values and we count down from zero to represent negative values.
* **Unicode:** Uses at least 16-bit representations for characters.

# **Solutions to End-of-Chapter Exercises**

**1**. (a) 1 × 42 + 3 × 4 + 3 = 31

(b) 3 × 82 + 6 × 8 + 7 = 247

(c) 1 × 162 + 11 × 16 + 10 = 442

**2**. There are only 10 available decimal digits, 0 . . . 9. To write a number in hexadecimal requires 16 digits to represent 0 . . . 15, so 0 . . . 9, A, B, . . . F are used. Notice that in hexadecimal the number 10 represents decimal 16, not decimal 10.

**3.** (a) 24; (b) 49: (c) 127; (d) 512

**4**. (a) 23 = 00010111

(b) 55 = 00110111

(c) 275 = 100010011 but this exceeds 8 bits, resulting in an arithmetic overflow.

**5.** (a) -49; (b) 408; (c) -1; (d) 0

**6.** (a) 01000111 (71 = 64 + 4 + 2 + 1)

(b) 10000001

(c) 11010001 (81 = 64 + 16 + 1)

**7.** The magnitude 200 in binary is 11001000, which is 8 bits. However, since the

sign takes up one bit we only have 7 left for the magnitude, which means that -200

is too big to represent in 8-bits. This would cause an *overflow error*.

**8.** 511

**9.**  0 ← carry digit

0011100011

+ 0001101110

1

1 ← carry digit

0011100011

+ 0001101110

01

1 ← carry digit

0011100011

+ 0001101110

001

1 ← carry digit

0011100011

+ 0001101110

0001

0 ← carry digit

0011100011

+ 0001101110

10001

1 ← carry digit

0011100011

+ 0001101110

010001

1 ← carry digit

0011100011

+ 0001101110

1010001

1 ← carry digit

0011100011

+ 0001101110

01010001

0 ← carry digit

0011100011

+ 0001101110

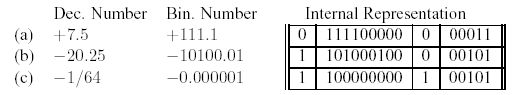
101010001

0011100011

+ 0001101110

0101010001

**10.**



**11.** (a) The mantissa is 0.111000000, which is +(1/2 + 1/4 + 1/8) = +7/8

The exponent is 0 00111 which is + (1 + 2 + 4) = +7

So, the overall value is +7/8 × 2+7 = +7/8 × 128 = +112.0

(b) The mantissa is 1.010001000 = -(1/4 + 1/64) = -(17/64)

The exponent is 1 00001 which is -1

So, the overall value is (-17/64) × 2-1 = (-17/64) × 1/2 = -17/128 = -0.132813

(Note: Although this is the correct answer, the mantissa is not normalized.

In most computers this value would be normalized so the first digit of the

mantissa is a 1, and the exponent would be adjusted accordingly.)

**12.** If we gave more bits to the mantissa we could represent our mantissa more accurately, resulting in more significant digits. However, since we have fewer digits for the exponent, we cannot represent exponents as large as before, resulting in an inability to represent larger (or smaller, if the exponent is negative) numbers. Essentially, we will get increased precision at the price of a decreased range.

**13**. (a) 01000001 01100010 01000011

(b) 01001101 01101001 01101011 01100101

(c) 00100100 00110010 00110101 00101110 00110000 00110000

(d) 00101000 01100001 00100000 00101011 00100000 01100010 00101001

**14.** There are 30 characters in “Invitation to Computer Science”, excluding the quotation marks. In ASCII, each character takes 8 bits, for a total of 240 bits. Each character in UNICODE takes 16 bits, for a total of 480 bits.

**15**.  **(a)**  A 3-minute song has 180 seconds in it. Each second is sampled 40,000 times so there are 7,200,000 samples. Each sample is stored using 16 bits, thus the total number of bits (uncompressed) is 115,200,000. A compression scheme with a 5:1 ratio would reduce the size to 23,040,000 bits.

**(b)** RGB-color images use 3 bytes per pixel (one for each of the three-color contributions), a total of 24 bits per pixel. A 1,200 by 800 image has 960,000 pixels, so in total it would use 23,040,000 bits. If it uses 2.4 Mbits, that means it uses 2,400,000 bits, so the compression ratio would be 9.6, or essentially 10:1.

**16**. Run-length encoding of xxxyyyyyyzzzzAAxxxx would be (x,3) (y,6) (z,4) (A,2) (x,4). Assuming one byte per character, the original takes 19 bytes, and the encoding takes 10 bytes, for a compression ratio of 19:10, almost 2:1.

**17**. (a) The word KAI would be 11101 00 10 in the variable length code given. It takes 9 bits, compared to 12 bits in the 4-bit encoding. The savings is 4:3, or about 1.33.

(b) MAUI would be 11100 00 11110 10 in the variable length code. It takes 14 bits, compared to 16 bits in the 4-bit encoding. The savings is 8:7, or about 1.14.

(c) MOLOKAI would be 11100 0111 1111111 0111 11101 00 10 in the variable length code. It takes 29 bits, compared to 28 bits in the 4-bit encoding. In this case, the variable length code is actually longer, because it contains many less-frequent letters. The “savings” is 28:29, or 0.97.

**18.** You can store fewer numbers in binary with the same number of places than you can in decimal. For example, with 8 digits in binary you can store the numbers 0 to 255, and in decimal you can store 0 to 99999999. Other answers include the fact that the designers of the computer hardware must work in binary, which is not a familiar numbering system, and the fact that translation must be done between the external representation of numbers (in decimal) and the internal representation of numbers (in binary).

**19.** (a) True; (b) False; (c) False; (d) False; (e) False.

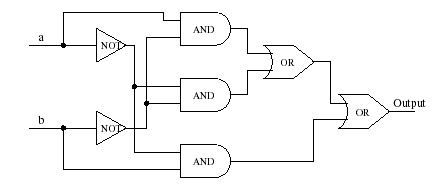
**20.** The order of operations is unclear. You can use parenthesis to indicate which is to be done first, or set the rules of precedence. Usually, AND is considered like multiplication, and takes precedence over OR, so we would consider the expression equivalent to ((*a* = 1) AND (*b* = 2)) OR (*c* = 3).

**21.** A truth table with k variables will always have 2k rows. So, when k = 5, our

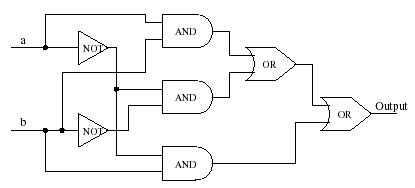
truth table will have 25 = 32 rows.

*Note: In all circuit diagrams we assume the existence of 3-input AND gates for simplicity.*

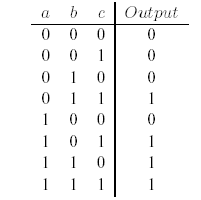
**22.** The circuit corresponds to the Boolean expression .



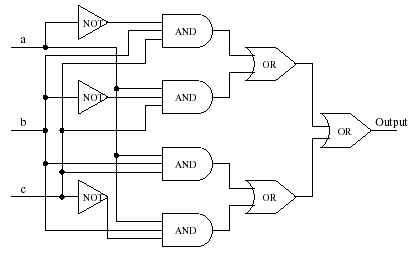
**23.** The circuit corresponds to 



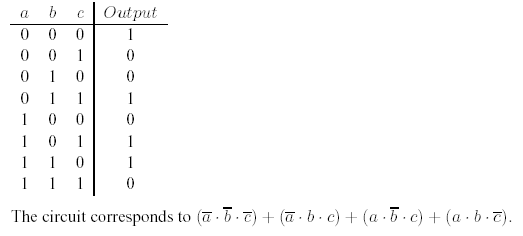
**24**. The truth table corresponding to this circuit is as follows.

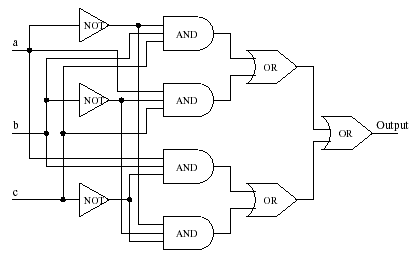


The circuit corresponds to 

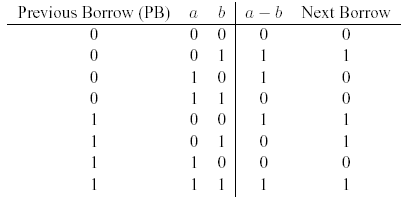


**25**. The truth table corresponding to this circuit is as follows:





**26.** This is a difficult problem that will challenge the best students in the class. You have to do several examples to determine the truth table. The truth table is:



So the sum of products expressions for *a* – *b* and Next Borrow are

*a* – *b* = 

Next Borrow = 

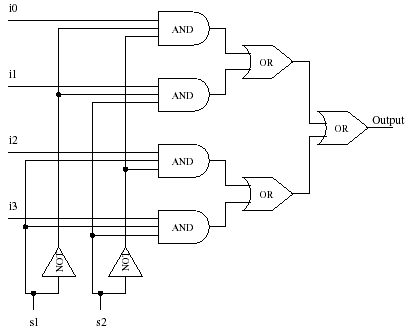
The circuit may be constructed as in the previous problems.

**27.** A multiplexor with 2*N*input lines has *N* selector lines, so a four-input multiplexor will need 2 input lines and an eight-input multiplexor will need 3 input lines.

**28.** Suppose the input lines are labeled as *i*0, *i*1, *i*2, and *i3* and the selector lines are labeled (left to right) as *s*1 and *s*2. *i0* will be the output if both *s1* and *s2* are zero, *i1* will be the output if *s1* = 0 and *s2* = 1, and so forth. The Boolean expression for a 4-input multiplexor is therefore



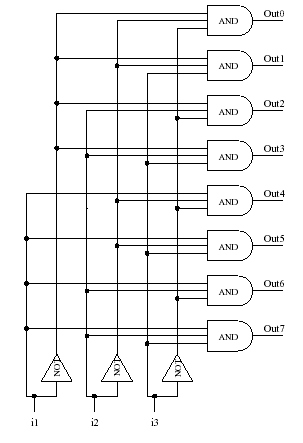
The circuit is



**29.** Suppose the input lines are labeled as *i*1, *i*2, and *i*3, and the outputs are labeled *Out*0 through *Out*7. *Out*0 is turned on when *i*1, *i*2, and *i*3are all zero, *Out*3 is on when *i*1 = 0, *i*2 = 1, and *i*3 = 1, and so forth. The circuit must have 8 subparts, one for each possible combination of input values. Each part ANDs together a particular Boolean expression like



and the result is the corresponding output line (*Out*3, in this case).



**Discussion of Challenge Work**

**1.** Changes may really only be made to the 1-bit adder. However, a small change in the 1-bit adder will multiply to a larger effect in the overall circuit. An easy first step is to have both the *si* and *ci*-1 outputs share the portion of the circuits that computes

*a ˖ b ⋅ c.* There is no other reason to have two copies of that sub-circuit. That alone will save two AND gates per 1-bit adder. For a 32-bit adder, it will save 64 × 3 transistors. As an example of further simplifications, we can simplify **in the *si* calculation to be which takes 4 gates instead of 5. This will save 3 transistors per 1-bit adder, for a total of 96 transistors saved overall. Similar simplifications may be done.

**2.** Students will need a good introductory resource for two’s complement, but otherwise it should be a straightforward problem.

**3.** Given that jpeg images are often found on the Web, students should find this an interesting topic.