The science of computing: Study questions

first edition

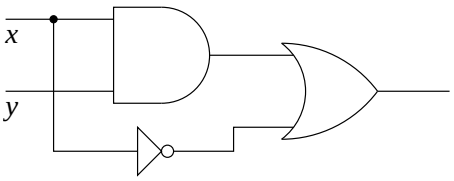
by Carl Burch

This document contains study questions to help in studying the material covered in the textbook, The science of computing. *Most questions come from previous tests given by the author.*

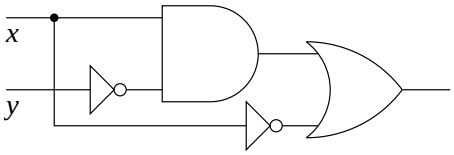
Each question's label has two parts separated by a dash. Question 3.2–1, for example, is the first study question for the material covered in Section 3.2 of The science of computing.

Question 2.1–1: (Solution, p S16) For each of the following circuits, write a truth table tabulating the circuit's output for each combination of inputs.

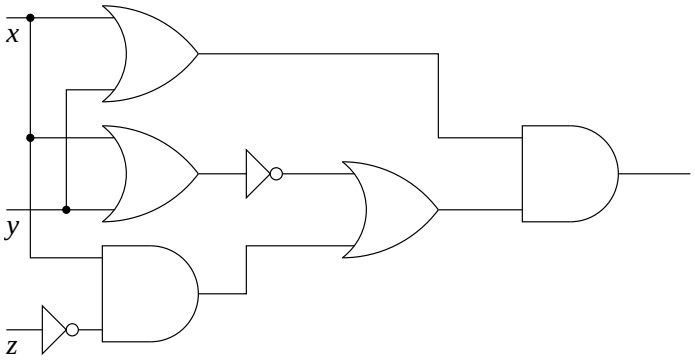
a.



b.



c.

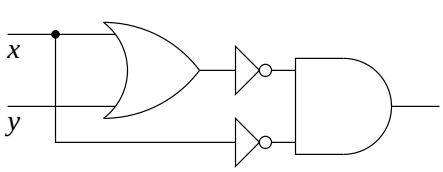


Question 2.1–2: (Solution, p S16) How deep is each of the circuits appearing in Question 2.1–1?

Question 2.2–1: (Solution, p S16) For each of the following circuits, write the Boolean expression that most closely corresponds to the circuit.

a. The circuit of Question 2.1–1(b)

b.



c. The circuit of Question 2.1–1(c)

Question 2.2–2: (Solution, p S16) Draw a circuit representing each of the following Boolean expressions.

a. $\overline{x + y} + x$

b. $\overline{xy + \bar{x}\bar{y}}$

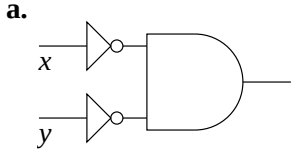
Question 2.2–3: (Solution, p S16) For each of the following Boolean expressions, write a truth table tabulating the expression's value for each combination of variable values.

a. $x + \overline{x + y}$

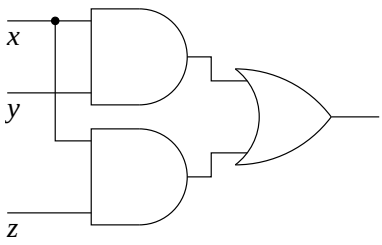
b. $xy + \overline{x + z}$

Question 2.2–4: (Solution, p S17) For each of the following, draw a smaller circuit (i.e., fewer gates) accomplishing the same task as the circuit given.

a.



b.



Question 2.2–5: (Solution, p S17) What is the *unsimplified* sum-of-products expression for the following truth tables? (Use multiplication for AND, addition for OR.)

a.

x	y	out
0	0	1
0	1	0
1	0	1
1	1	0

b.

x	y	out
0	0	1
0	1	1
1	0	0
1	1	1

c.

x	y	z	out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Question 2.3–1: (Solution, p S17) Simplify the following sum-of-products expressions. For those that cannot be simplified using the technique from Section 2.3, you may simply state this fact.

a. $xy + \bar{x}y + x\bar{y}$

b. $\bar{x}yz + x\bar{y}z + xy\bar{z} + \bar{x}\bar{y}\bar{z}$

c. $\bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xy\bar{z} + xyz$

Question 2.3–2: (Solution, p S17) Construct a *simplified* Boolean expression corresponding to the following truth table.

x	y	z	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Question 3.1–1: (Solution, p S17) How many bits do you need to represent seven different values? Nine? Twelve? Thirty?

Question 3.1–2: (Solution, p S17) How many bits are in a kilobyte of memory?

Question 3.1–3: (Solution, p S17) Perform each of the following conversions.

a. $101101_{(2)}$ to decimal

b. $1010101_{(2)}$ to decimal

c. $23_{(10)}$ to binary

d. $95_{(10)}$ to binary

Question 3.1–4: (Solution, p S17) Perform each of the following conversions.

a. $1010101_{(2)}$ to octal

b. $1010101_{(2)}$ to hexadecimal

c. $101101_{(2)}$ to hexadecimal

d. $560_{(8)}$ to binary

e. $\mathrm{CAB}_{(16)}$ to binary

f. $\mathrm{1B2}_{(16)}$ to binary

Question 3.2–1: (Solution, p S17) Represent each of the following in a sign-magnitude representation.

a. $-1_{(10)}$ in a seven-bit sign-magnitude format
b. $-20_{(10)}$ in a seven-bit sign-magnitude format
c. $20_{(10)}$ in a seven-bit sign-magnitude format
d. $-300_{(10)}$ in twelve-bit sign-magnitude format

Question 3.2–2: (Solution, p S17) Represent each of the following in a two's-complement representation.

a. $-1_{(10)}$ in a seven-bit two's-complement format
b. $-20_{(10)}$ in a seven-bit two's-complement format
c. $20_{(10)}$ in a seven-bit two's-complement format
d. $-300_{(10)}$ in twelve-bit two's-complement format

Question 3.2–3: (Solution, p S18)

a. What is the smallest (most negative) number you can represent in seven bits using sign-magnitude representation? Give both the bit pattern of the number and its base-10 translation.

b. Answer the same question for a seven-bit two's-complement representation.

Question 3.3–1: (Solution, p S18) Convert each of the following decimal numbers to the 8-bit floating-point representation described in Section 3.3.

a. 1.0
b. 0.25
c. 10.0
d. -2.5
e. 0.0625

Question 3.3–2: (Solution, p S18) According to the floating-point representation described in Section 3.3, what number does each of the following bit patterns represent? Express your answers in base-10, either as a decimal number or as a fraction.

a. 0 0110 101
b. 1 1011 001
c. 1 0111 110
d. 1 1010 101

Question 3.3–3: (Solution, p S18) Suppose we define a six-bit floating point system with one sign bit, three exponent bits (using excess 3), and two mantissa bits.

a. Represent each of the following decimal numbers in this six-bit system.

$5_{(10)}$
$-2_{(10)}$

b. For each of the following bit patterns in this six-bit floating-point system, express its base-10 numerical equivalent as a decimal number or as a fraction.

0 001 10
1 110 01

Question 3.3–4: (Solution, p S18) Suppose we define a nine-bit floating point system with one sign bit, five exponent bits (using excess 15), and three mantissa bits.

a. Represent each of the following decimal numbers in this nine-bit system.

$1_{(10)}$
$-6_{(10)}$

b. For each of the following bit patterns in this nine-bit floating-point system, express its base-10 numerical equivalent as a decimal number or as a fraction.

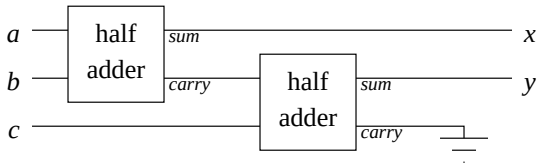
0 10000 010
1 10011 110

Question 3.4–1: (Solution, p S18) Suppose a digital camera uses a 40 MB disk to store pictures. How many 3×5-inch photographs can it store in 24-bit color at the standard printer-quality resolution of 300 pixels per inch? (You'll need a calculator for this one.)

Question 3.4–2: (Solution, p S18) We examined a compression technique called *run-length encoding* for black-and-white images, in which each byte includes four bits saying how many adjacent black pixels there are and four bits saying how many adjacent white pixels there are. As we saw, this compression technique sometimes actually *expands* a picture. What is the maximum possible expansion factor?

Question 4.1–1: (Solution, p S18) Draw two truth tables illustrating the outputs of a half-adder, one table for the sum output and the other for the $carry$ output.

Question 4.1–2: (Solution, p S18) Fill in the truth table at right for the following circuit. Ignore rows not included in the table.



a	b	c	x	y
0	1	1		
1	0	1		
1	1	0		
1	1	1		

Question 4.1–3: (Solution, p S18) What distinguishes the behavior of a half adder from that of a full-adder? (That is, are their inputs or outputs different? Is the relationship between inputs and outputs different?)

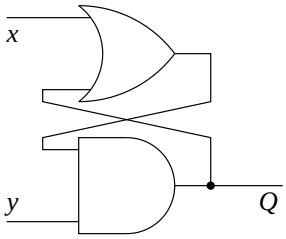
Question 4.1–4: (Solution, p S19) Draw a truth table diagramming a full adder's c_{out} output.

Question 4.1–5: (Solution, p S19) Design a full adder using only half adders. Your design must not include any logic gates, such as AND, OR, and NOT gates.

Question 4.1–6: (Solution, p S19) Using only four-bit adders, construct an eight-bit adder. Each four-bit adder has two four-bit inputs and one five-bit output. Your eight-bit adder should have two eight-bit inputs and a one eight-bit output (don't worry about the ninth output bit).

Question 4.2–1: (Solution, p S19)

For the circuit below, fill in the truth table at right to represent how the value of Q changes based on the inputs x and y. Notice that you should ignore two of the rows.

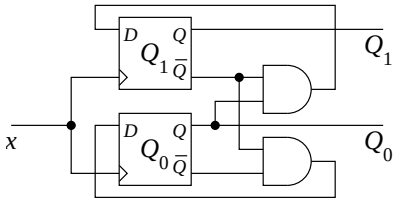


x	y	old Q	new Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	(ignore)
1	0	1	(ignore)
1	1	0	
1	1	1	

Question 4.2–2: (Solution, p S19) How is a D flip-flop's behavior different from a D latch's behavior?

Question 4.2–3: (Solution, p S20)

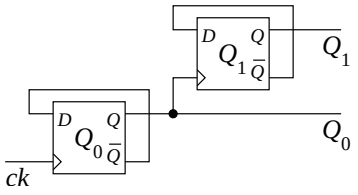
Suppose the upper D flip-flop in the below circuit were holding 1 and the lower D flip-flop held 0, while the x input were 0. At right, tabulate how the circuit's output changes as the input x toggles between 0 and 1.



x	Q_1	Q_0
0	1	0
1		
0		
1		
0		
1		
0		
1		

Question 4.2–4: (Solution, p S20)

Suppose that both of the flip-flops in the below circuit currently hold 0. Tabulate how the circuit at left changes as the input ck toggles through the inputs at left.



ck	Q_1	Q_0
0		
1		
0		
1		
0		
1		
0		
1		
0		
1		

Question 4.3–1: (Solution, p S20) Draw a circuit with a single input T and a single output, where the output toggles to a different value each time T changes from 0 to 1. (Your circuit may incorporate D flip-flops.) The following table illustrates how your circuit would chaneg.

T	Q	explanation
0	0	
1	1	T changes to 1, so Q changes
0	1	T changes to 0; Q remains at 1
1	0	T changes to 1, so Q changes
0	0	T changes to 0; Q remains at 0
1	1	T changes to 1, so Q changes
0	1	T changes to 0; Q remains at 1
1	0	T changes to 1, so Q changes

Question 4.3–2: (Solution, p S20) Design a circuit that takes a single input ck and outputs two bits $Q_1\,Q_0$, whose values cycle to the next number of the sequence $00, 01, 11, 10, 00, 01, 11, 10, \ldots$ each time ck changes from 0 to 1. (Note that this is not a two-bit counter: 11 comes after 10.)

Question 5.1–1: (Solution, p S20) Define the fetch-execute cycle as it relates to a computer processing a program. Your definition should describe the primary purpose of each phase.

Question 5.1–2: (Solution, p S20) Explain in detail what the HYMN CPU does during the fetch phase of the fetch-execute cycle. (Your explanation should describe how the computer accesses values in registers and memory.)

Question 5.1–3: (Solution, p S20) Translate each of the following HYMN instructions into machine code. Express your answers in binary.

a. `HALT`

b. `ADD` $\texttt{14}_{(16)}$

c. `LOAD` $\texttt{12}_{(16)}$

d. `JUMP` $\texttt{03}_{(16)}$

Question 5.1–4: (Solution, p S21) Suppose that the HYMN CPU begins with the following in memory.

addr	data	(translation)	
00000	100 00100	LOAD	00100
00001	110 00101	ADD	00101
00010	111 00100	SUB	00100
00011	101 00111	STORE	00111
00100	000 00101	HALT	
00101	000 00110	HALT	

a. Show the hexadecimal values taken on by the registers as this program executes. (Stop once the computer executes a HALT instruction.)

PC	00
IR	00
AC	00

b. What memory locations, if any, change? What values are stored in these locations?

Question 5.2–1: (Solution, p S21) Suppose that the HYMN CPU begins with the following in memory.

addr	data	(translation)
00000	100 11110	LOAD 11110
00001	101 11111	STORE 11111
00010	110 11110	ADD 11110
00011	101 11111	STORE 11111
00100	110 11110	ADD 11110
00101	101 11111	STORE 11111
00111	000 00000	HALT

If the user typed multiples of 25 starting at 25 (25, then 50, then 75,...) when prompted, what would the computer display?

Question 5.2–2: (Solution, p S21) Suppose that the HYMN CPU begins with the following in memory.

addr	data	(translation)
00000	100 11110	LOAD 11110
00001	110 11110	ADD 11110
00010	011 00001	JPOS 00001
00011	000 00000	HALT

If we repeatedly type the number $32_{(10)}$ when prompted, how many times would we type it before the computer halts?

Question 5.2–3: (Solution, p S21) Suppose we want HYMN to read a number N from the user and then output $5 - N$. What should be in the computer's memory when the HYMN CPU begins? (Express your answer in bits.)

addr	data
00000	
00001	
00010	
00011	
00100	

addr	data
00101	
00110	
00111	
01000	
01001	

Question 5.2–4: (Solution, p S21) What should we place into memory so that, when started, HYMN displays the powers of two from $1_{(10)}$ to $64_{(10)}$ before halting? (Express your answer in bits.)

addr	data
00000	
00001	
00010	
00011	
00100	

addr	data
00101	
00110	
00111	
01000	
01001	

Question 5.2–5: (Solution, p S21)

Suppose that the HYMN CPU begins with memory contents at right.

a. List all new values stored in memory as the program executes. Express your answers in binary or hexadecimal.

b. What values does the AC hold in the course of executing this program? Express your answers in binary or hexadecimal.

addr	data	(translation)
00000	100 01001	LOAD 01001
00001	010 01000	JZER 01000
00010	110 01010	ADD 01010
00011	101 01010	STORE 01010
00100	100 01001	LOAD 01001
00101	110 00000	ADD 00000
00110	101 00000	STORE 00000
00111	001 00000	JUMP 00000
01000	000 00000	HALT
01001	000 00001	1
01010	000 00010	2
01011	000 00100	4
01100	000 00000	0

Question 5.3–1: (Solution, p S21) Translate the following HYMN assembly language program into machine language. Express your answer in bits.

```
      READ
top:  WRITE
      ADD one
      JPOS top
      HALT
one:  1
```

addr	data
00000	
00001	
00010	
00011	
00100	
00101	
00110	
00111	

Question 5.3–2: (Solution, p S22) Translate the following HYMN assembly language program into machine language.

```
up:   READ
      JZER done
      STORE n
      JUMP up
done: LOAD n
      WRITE
      HALT
n:    0
```

addr	data
00000	
00001	
00010	
00011	
00100	

addr	data
00101	
00110	
00111	
01000	
01001	

Question 5.3–3: (Solution, p S22) Write a HYMN assembly language program that reads a number n and displays the value $4n + 3$.

Question 5.3–4: (Solution, p S22) Write a HYMN assembly language program that repeatedly reads numbers from the user until the user types 5.

Question 5.3–5: (Solution, p S22) Write a HYMN assembly language program that displays 100 copies of the number 0.

Question 5.3–6: (Solution, p S22) Write a HYMN assembly language program that reads a number n from the user and then displays n's absolute value. (The **absolute value** of a number is that number with any negative sign removed. The absolute value of -5 is 5, while the absolute value of 3 is 3 itself.)

Question 5.3–7: (Solution, p S23) Write a HYMN assembly language program that reads a number n and displays the powers of two that are less than n. Your program may assume that n is positive.

Question 5.4–1: (Solution, p S23)
Translate each of the following pseudocode procedures into HYMN's assembly language.

a.
Read n.
while $n \neq 0$, **do:**
    Write 1.
    Read n.
end while
Stop.

b.
Initialize sum to 0.
Read n.
while $n \neq 0$, **do:**
    Increase sum by n.
    Read n.
end while
Write sum.
Stop.

Question 5.4–2: (Solution, p S23) Express in pseudocode an algorithm to read a number and display its absolute value.

Question 5.4–3: (Solution, p S23) Express in pseudocode an algorithm to read 100 numbers and then to display the maximum among the numbers typed.

Question 6.2–1: (Solution, p S24) The text describes three purposes of the operating system. Give two of them.

Question 6.2–2: (Solution, p S24) Among the three purposes of the operating system described by the text is, "The operating system abstracts computer resources." Explain what this sentence means, with an example.

Question 6.3–1: (Solution, p S24) Describe the procedure an operating system performs to switch the process running on a CPU.

Question 6.3–2: (Solution, p S24) Say our computer has a time slice of 2 ms, and it takes 1 ms to perform a context switch and 2 ms to access the disk. There are two processes in execution.

Process A	Process B
compute 1 ms	compute 1 ms
access disk	access disk
compute 3 ms	compute 1 ms

Complete the following table to show how the operating system will schedule these processes on the CPU. The front of each queue is the left end. There may be more blanks than necessary.

time	CPU	ready	disk
0		AB	

Question 6.3–3: (Solution, p S25) Suppose our computer system uses virtual memory with 12 pages, called P_0, P_1, ..., P_{11}. Further, suppose that the system contains seven page frames, where each frame contains a page as follows.

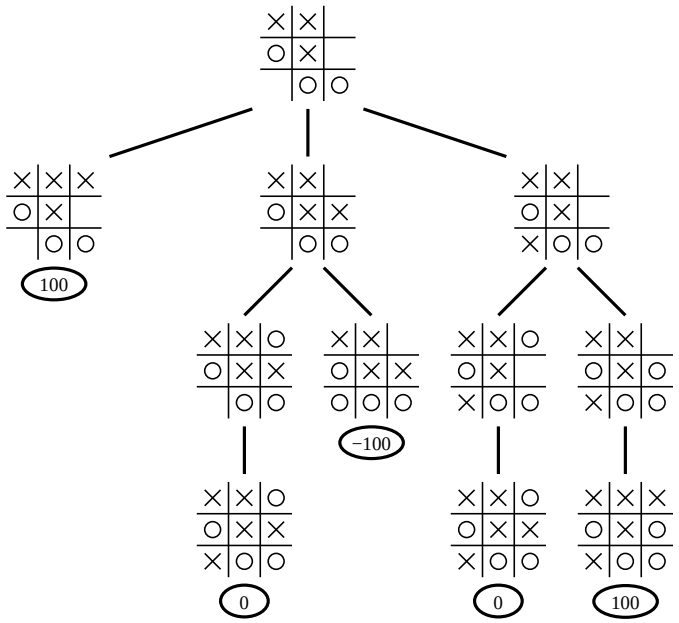
frame	page
F_1	P_5
F_2	P_4
F_3	P_2
F_4	P_7
F_5	P_{10}
F_6	P_3
F_7	P_0

Draw a diagram of the page table.

Question 6.3–4: (Solution, p S25) Suppose we have a system using paging, and a program requests to load memory occurring in page 9 of virtual memory. Suppose, moreover, that page 9 happens currently to be in page frame 3 of RAM. Outline the process the system goes through in locating the data requested.

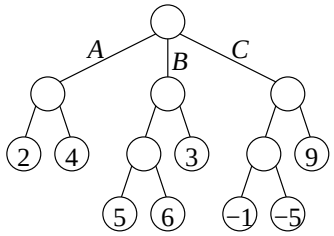
Question 7.1–1: (Solution, p S25) The game of Nim proceeds by players taking turns selecting a pile and removing stones from that pile. The player removing the last stone wins.
Draw a complete game tree for the game of Nim beginning with two piles, both containing two stones. To draw a node, list the number of stones in each pile; for example, the top node will be "2,2."
Do not include the minimax values assigned to each node in your tree.

Question 7.1–2: (Solution, p S25) Label *all* internal nodes of the following tic-tac-toe game tree with the value that minimax search would compute. I've already labeled the leaves.



Question 7.1–3: (Solution, p S26)

Suppose a game player has constructed a game tree as given at right. In this tree, high numbers represent good boards for X, and it is currently X's move. (As you can see, X has three possible moves from which to choose, labeled A, B, and C.)



a. Fill in *all* empty circles with the values assigned them according to the minimax evaluation algorithm.

b. Which move will X choose?

Question 7.2–1: (Solution, p S26) Describe the Turing Test and why Turing proposed it (i.e., its purpose).

Question 7.3–1: (Solution, p S26) Suppose we have a perceptron with three inputs, and the perceptron's current weights are $\langle 0.5, 0.2, -0.5 \rangle$.

a. What would the perceptron predict given the input $\langle 1, -1, -1 \rangle$?

b. Suppose this prediction were wrong. How would the perceptron update its weights if the learning rate were $r = 0.1$?

Question 7.3–2: (Solution, p S26) Suppose we have a three-input perceptron, where input x_1 is permanently wired to a 1 input.

a. Suppose the weights of the perceptron were $\langle 0.1, 0.5, -0.4 \rangle$, and we were to give it the inputs $\langle 1, 1, 1 \rangle$. What would the perceptron output?

b. Suppose we wanted to select weights so that our perceptron would behave like the AND of the inputs x_2 and x_3.

x_1	x_2	x_3	o
1	−1	−1	−1
1	−1	1	−1
1	1	−1	−1
1	1	1	1

Give a combination of perceptron weights w_1, w_2, and w_3 that would result in the desired behavior.

Question 8.1–1: (Solution, p S27) Consider the following context-free grammar.

$$\mathbf{S} \rightarrow \varepsilon \mid a\,\mathbf{S}\,b\,\mathbf{S} \mid b\,\mathbf{S}\,a\,\mathbf{S}$$

Give a derivation of the string *aabbab* using this grammar.

Question 8.1–2: (Solution, p S27) Consider the following context-free grammar.

$$\begin{aligned} \mathbf{S} &\rightarrow \mathbf{V}\,\mathbf{V} \mid \mathbf{V} \\ \mathbf{V} &\rightarrow (\,\mathbf{S}\,) \mid x \mid y \mid z \end{aligned}$$

Each of the following sentences is either described by this grammar or not. If it is, give either a parse tree with **S** at its root or a derivation from **S**. If it is not, simply say so.

a. *x (y)* **b.** *x y z* **c.** *x (y z)*

Question 8.1–3: (Solution, p S27) Consider the following context-free grammar.

$$\begin{aligned} \mathbf{S} &\rightarrow \mathbf{T} + \mathbf{S} \mid \mathbf{T} \\ \mathbf{T} &\rightarrow \mathbf{F}\,\mathbf{T} \mid \mathbf{F} \\ \mathbf{F} &\rightarrow x \mid y \mid (\,\mathbf{S}\,) \end{aligned}$$

Draw a parse tree for the sentence *x (x + y)*.

Question 8.1–4: (Solution, p S27) Write a context-free grammar describing each of the following languages.

a. the set of strings of *a*'s and *b*'s beginning and ending with the letter *a*

b. the set of strings containing either only *a*'s or only *b*'s

c. the set of strings containing only *a*'s and *b*'s with at least one *a* and at least one *b*.

d. the set of strings of left-brackets and right-brackets where the brackets match (you can pair all brackets so that each pair includes a left bracket and a right bracket occurring after it in the string). Examples include *[[]][]* and *[[[]]]]*.

Question 8.2–1: (Solution, p S28) For each of the following, say whether the regular expression describes a language including the sentence. Your answer will be either "yes" or "no."

ab* includes *abab*? ____
(a|b)a includes *aba*? ____
a(a|b)*b includes *abbbaaab*? ____
(a|b)(ba|ab)* includes *ababab*? ____
(a|b)(ba|ab)* includes *abababa*? ____

Question 8.2–2: (Solution, p S28) Write a regular expression describing each of the following languages.

a. strings containing only *a*'s and *b*'s where all *a*'s come before all *b*'s.

b. strings containing only *a*'s and *b*'s in which "*aab*" somewhere occurs as an adjacent sequence (like *aabaa* or *baaba* but not *abbab*).

c. strings containing an even number of *a*'s (and no other letters).

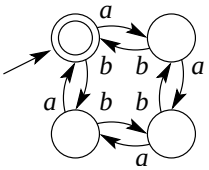
d. strings that contain either only *a*'s or only *b*'s.

e. binary representations of positive even integers.

f. binary representations of positive integers that are at least 4.

Question 8.2–3: (Solution, p S28) Give an English description of a language that can be described by a context-free grammar but not by a regular expression.

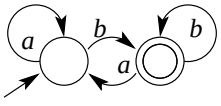
Question 9.1–1: (Solution, p S28) Consider the following finite automaton.



Check the strings that are within the language accepted by this finite automaton.

__ *ab*
__ *bbb*
__ *baaa*
__ *abba*
__ *aabbb*
__ *bbbabb*
__ *aabaabaa*

Question 9.1–2: (Solution, p S28) Consider the following finite automaton.



a. Check the strings that the automaton will accept.

__ *b*
__ *a*
__ *aba*
__ *bab*
__ *bbbb*
__ *baba*

b. Give an English description of the set of strings accepted by this automaton.

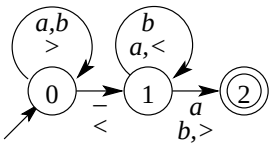
Question 9.1–3: (Solution, p S28) Draw a finite automaton that accepts the strings of *a*'s and *b*'s beginning with an *a*.

Question 9.1–4: (Solution, p S28) Design a finite state automaton that will recognize the language of strings of *a*'s and *b*'s containing at least 3 *b*'s. Examples include *bbbbb* and *bbab*, but not *bba*.

Question 9.1–5: (Solution, p S29) Design a finite automaton that accepts the language of all strings that contain *abaa* (such as *ab*<u>*abaa*</u>*ba* or *a*<u>*abaa*</u>*aaa*). You need not worry about strings containing letters other than *a*'s and *b*'s.

Question 9.2–1: (Solution, p S29)

Consider the following Turing machine. (Note that the underscore represents a blank on the tape.)



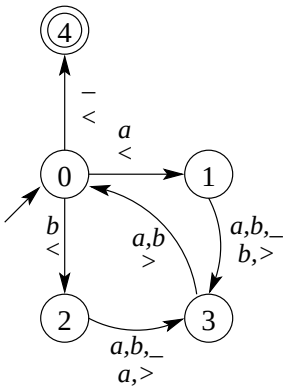
At right, diagram this Turing machine's computation as it goes through the string *ababb*. If you run out of blanks in the table, stop.

To represent the machine's initial position in the table at right, we write "$\underline{0}$ *ababb* ". This represents a tape containing " *ababb* " (with blanks extending infinitely both ways), where the Turing machine is currently in state 0 of its finite automaton, and its head is pointing to the initial *a*.

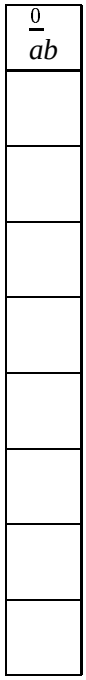
$\underline{0}$ *ababb*

Question 9.2–2: (Solution, p S29)

Consider the following Turing machine. (Note that the underscore represents a blank on the tape.)



At right, diagram this Turing machine's computation as it goes through the string *ab*. If you run out of blanks in the table, stop.



To represent the machine's initial position in the table at right, we write "$\underline{0}\ ab$ ". This represents a tape containing "*ab* " (with blanks extending infinitely both ways), where the Turing machine is currently in state 0 of its finite automaton, and its head is pointing to the initial *a*.

Question 9.2–3: (Solution, p S30) Design a Turing machine that transforms a string containing only *a*'s, *b*'s, and *c*'s by replacing each letter preceding an *a* to a *b*. (Do not worry about the case when the string begins with an *a*.) Thus, *bccb* would remain unchanged while *caccaa* would change to *bacbba*. The Turing machine should always eventually enter an accepting state to terminate.

Question 9.2–4: (Solution, p S30) Restate the Church-Turing thesis in your own words.

Question 9.3–1: (Solution, p S30) Define the halting problem language (the language that we have seen Turing machines cannot solve).

Question 9.3–2: (Solution, p S30) Consider the following choices.

A. the first is contained within the second

B. the second is contained within the first

C. neither is contained within the other

D. the languages are identical

Choose which of these choices best describes each of the following pairs of language classes.

a. languages described by regular expressions; languages accepted by finite automata

b. languages described by context-free grammars; languages described by regular expressions

c. languages described by context-free grammars; languages accepted by Turing machines

d. languages accepted by finite automata; languages accepted by Turing machines

Solution 2.1–1: (Question, p S1)

a.

x	y	output
0	0	1
0	1	1
1	0	0
1	1	1

b.

x	y	output
0	0	1
0	1	1
1	0	1
1	1	0

c.

x	y	z	output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Solution 2.1–2: (Question, p S1)

a. 2 (all paths from x or y to the output go through two gates)

b. 3 (the longest path goes from y through a NOT gate, an AND gate, and an OR gate)

c. 4 (the longest path goes from z through a NOT gate, an AND gate, an OR gate, and an AND gate)

Solution 2.2–1: (Question, p S1)

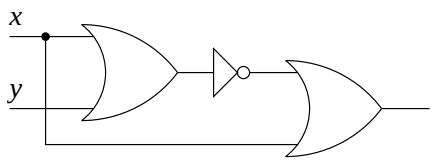
a. $x\overline{y} + \overline{x}$

b. $(\overline{x + y})\,\overline{x}$

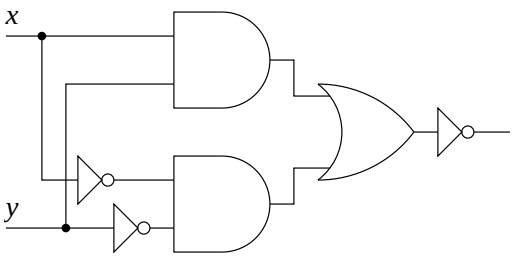
c. $(x + y)\,((\overline{x + y}) + x\overline{z})$

Solution 2.2–2: (Question, p S1)

a.



b.



Solution 2.2–3: (Question, p S1)

a.

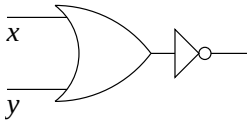
x	y	output
0	0	1
0	1	0
1	0	1
1	1	1

b.

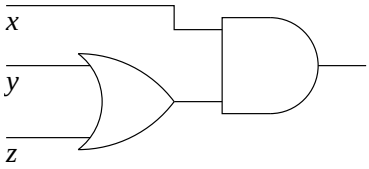
x	y	z	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Solution 2.2–4: (Question, p S1)

a. This involves an application of the DeMorgan's law $\overline{x}\overline{y} = \overline{x + y}$.



b. This involves an application of the distributive law $xy + xz = x(y + z)$.



Solution 2.2–5: (Question, p S2)
a. $\overline{x}\overline{y} + x\overline{y}$
b. $\overline{x}\overline{y} + \overline{x}y + xy$
c. $\overline{x}\overline{y}\overline{z} + \overline{x}\overline{y}z + x\overline{y}z$

Solution 2.3–1: (Question, p S2)
a. $x + y$
b. cannot be simplified using the technique from Section 2.3.
c. $yz + x$

Solution 2.3–2: (Question, p S2) $\overline{x}\overline{z} + \overline{y}\overline{z} + xyz$ is good. (One could go further, though, and write $(\overline{x} + \overline{y})\overline{z} + xyz$ or even $\overline{x}\overline{y}\overline{z} + xyz$.)

Solution 3.1–1: (Question, p S2) You need 3 bits for seven values, 4 for nine or twelve, and 5 for thirty values.

Solution 3.1–2: (Question, p S2) There are 8,096 bits in a kilobyte:

$$\frac{8 \text{ bits}}{\text{byte}} \times \frac{1{,}024 \text{ bytes}}{\text{KB}} = \frac{8{,}096 \text{ bits}}{\text{KB}}$$

Solution 3.1–3: (Question, p S2)
a. $101101_{(2)} = 45_{(10)}$
b. $1010101_{(2)} = 85_{(10)}$
c. $23_{(10)} = 10111_{(2)}$
d. $95_{(10)} = 1011111_{(2)}$

Solution 3.1–4: (Question, p S2)
a. $1\,010\,101_{(2)} = 125_{(8)}$
b. $101\,0101_{(2)} = 55_{(16)}$
c. $10\,1101_{(2)} = 2\text{D}_{(16)}$
d. $560_{(8)} = 101\,110\,000_{(2)}$
e. $\text{CAB}_{(16)} = 1100\,1010\,1011_{(2)}$
f. $1\text{B}2_{(16)} = 1\,1011\,0010_{(2)}$

Solution 3.2–1: (Question, p S3)
a. $-1_{(10)}$ 100 0001
b. $-20_{(10)}$ 101 0100
c. $20_{(10)}$ 001 0100
d. $-300_{(10)}$ 1001 0010 1100

Solution 3.2–2: (Question, p S3)
a. $-1_{(10)}$ 111 1111
b. $-20_{(10)}$ 110 1100
c. $20_{(10)}$ 001 0100
d. $-300_{(10)}$ 1110 1101 0100

Solution 3.2–3: (Question, p S3)
a. **Sign-magnitude:** 111 1111 represents $-63_{(10)}$
b. **Two's-complement:** 100 0000 represents $-64_{(10)}$

Solution 3.3–1: (Question, p S3)
a. 0 0111 000 $(1.0_{(2)} = 1.0_{(2)} \times 2^{0})$
b. 0 0101 000 $(0.01_{(2)} = 1.0_{(2)} \times 2^{-2})$
c. 0 1010 010 $(1010_{(2)} = 1.010_{(2)} \times 2^{3})$
d. 1 1000 010 $(-10.1_{(2)} = -1.01_{(2)} \times 2^{1})$
e. 0 0011 000 $(0.0001_{(2)} = 1.0_{(2)} \times 2^{-4})$

Solution 3.3–2: (Question, p S3)
a. $0.8125_{(10)}$ or $\frac{13}{16}$ $(1.101_{(2)} \times 2^{6-7} = 0.1101_{(2)})$
b. $-18_{(10)}$ $(-1.001_{(2)} \times 2^{11-7} = -10010_{(2)})$
c. $-1.75_{(10)}$ $(-1.110_{(2)} \times 2^{7-7} = -1.110_{(2)})$
d. $-13_{(10)}$ $(-1.101_{(2)} \times 2^{10-7} = -1101_{(2)})$

Solution 3.3–3: (Question, p S3)
a. 0 101 01 $(101_{(2)} = 1.01_{(2)} \times 2^{2})$
1 100 00 $(-10_{(2)} = -1.0_{(2)} \times 2^{1})$
b. $0.375_{(10)}$ or $\frac{3}{8}$ $(1.10_{(2)} \times 2^{1-3} = 0.0110_{(2)})$
$-10_{(10)}$ $(-1.01_{(2)} \times 2^{6-3} = -1010_{(2)})$

Solution 3.3–4: (Question, p S4)
a. 0 01111 000 $(1.0_{(2)} = 1.0 \times 2^{0})$
1 10001 100 $(-110_{(2)} = -1.10_{(2)} \times 2^{2})$
b. $2.5_{(10)}$ $(1.010_{(2)} \times 2^{1} = 10.1_{(2)})$
$-28_{(10)}$ $(-1.110_{(2)} \times 2^{4} = -11100_{(2)})$

Solution 3.4–1: (Question, p S4) The disk can store 10 pictures:

$$\frac{3 \times 5 \text{ sq. inches}}{\text{picture}} \times \frac{300 \times 300 \text{ pixels}}{\text{sq. inch}} \times \frac{3 \text{ bytes}}{\text{pixel}} \times \frac{\text{MB}}{2^{20} \text{ bytes}} \times = \frac{3.86 \text{ MB}}{\text{picture}}$$

$$\frac{40 \text{ MB}}{\text{disk}} \times \frac{\text{picture}}{3.86 \text{ MB}} \times = \frac{10.4 \text{ pictures}}{\text{disk}}$$

We round down to 10, since it doesn't make sense to store a fraction of a picture.

Solution 3.4–2: (Question, p S4) A checkboard pattern, where every other pixel is black, would give the maximum expansion factor. For such a system, each run would be only one pixel long, and the compression system would use four bits to encode each run, for an expansion factor of four.

Solution 4.1–1: (Question, p S4)

x	y	sum
0	0	0
0	1	1
1	0	1
1	1	0

x	y	$carry$
0	0	0
0	1	0
1	0	0
1	1	1

Solution 4.1–2: (Question, p S4)

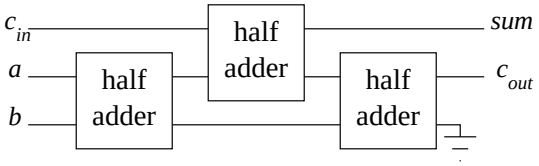
a	b	c	x	y
0	1	1	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

Solution 4.1–3: (Question, p S4) Whereas a half adder takes only *two* input bits to add, a full adder adds *three* input bits.

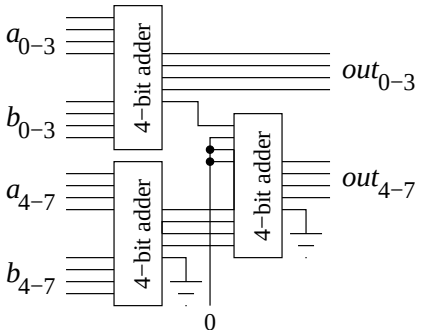
Solution 4.1–4: (Question, p S4)

c_{in}	a	b	c_{out}
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

Solution 4.1–5: (Question, p S4)



Solution 4.1–6: (Question, p S4) The design of this circuit is similar in structure to the design of a full adder using half adders.



Solution 4.2–1: (Question, p S5)

x	y	old Q	new Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	(ignore)
1	0	1	(ignore)
1	1	0	1
1	1	1	1

Solution 4.2–2: (Question, p S5) In a D flip-flop, the memory value changes only at that instant that the $clock$ input becomes 1. In a latch, however, the memory value continues adopting any values given as long as its set input is 1. (In a D flip-flop, if the D input changes while $clock$ remains 1, the remembered value doesn't change. In a latch, however, a change to the D input results in an immediate change to the remembered value.)

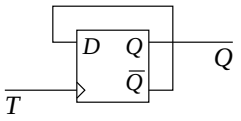
Solution 4.2–3: (Question, p S5)

x	Q_1	Q_0
0	1	0
1	0	0
0	0	0
1	0	1
0	0	1
1	1	0
0	1	0
1	0	0

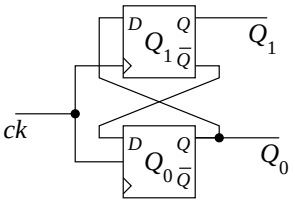
Solution 4.2–4: (Question, p S5)

ck	Q_1	Q_0
0	0	0
1	1	1
0	1	1
1	1	0
0	1	0
1	0	1
0	0	1
1	0	0
0	0	0
1	1	1

Solution 4.3–1: (Question, p S6)



Solution 4.3–2: (Question, p S6)



Solution 5.1–1: (Question, p S6) The fetch-execute cycle is the process by which a classical computer executes instructions. In the fetch phase, the computer determines the next instruction to be completed by fetching the instruction from memory. In the execute phase, the computer executes this instruction. The computer alternates between these two phases as long as it is on.

Solution 5.1–2: (Question, p S6) It looks into the PC for a memory address, requests the information at that address from RAM via the bus, and stores RAM's response in the IR.

Solution 5.1–3: (Question, p S6)

a. 000 00000

b. 110 10100

c. 100 10010

d. 001 00011

Solution 5.1–4: (Question, p S6)

a. PC 00 01 02 03 04
IR 00 84 C5 E4 A7 05
AC 00 05 0B 06

b. Memory location $00101_{(2)}$ changes to hold $06_{(16)}$.

Solution 5.2–1: (Question, p S7)

```
? 25
25
? 50
75
? 75
-106
```

(This last output is somewhat tricky: In the last ADD instruction, the CPU computes $75 + 75 = 150$, but this exceeds the maximum eight-bit two's-complement number. So the computer wraps around ends up at $150 - 256 = -106$.)

Solution 5.2–2: (Question, p S7) It would read from the user four times before halting (with the AC progressing from 32 to 64 to 96 to -128).

Solution 5.2–3: (Question, p S7)

addr	data	(translation)
00000	100 11110	LOAD 11110
00001	101 00111	STOR 00111
00010	100 00110	LOAD 00110
00011	111 00111	SUB 00111
00100	101 11111	STOR 11111
00101	000 00000	HALT
00110	000 00101	5

(Of course there are many other solutions for all questions that involve writing programs.)

Solution 5.2–4: (Question, p S7)

addr	data	(translation)
00000	100 00110	LOAD 00110
00001	101 11111	STOR 11111
00010	101 00111	STOR 00111
00011	110 00111	ADD 00111
00100	011 00001	JPOS 00001
00101	000 00000	HALT
00110	000 00001	1

Solution 5.2–5: (Question, p S8)

a. **address 00000:** 8A 8B 8C
address 01010: 03 06 0A

b. **AC:** 01 03 01 8A 03 06 01 8B 04 0A 01 8C 00

Solution 5.3–1: (Question, p S8)

addr	data	(translation)
00000	100 11110	LOAD 11110
00001	101 11111	STORE 11111
00010	110 00101	ADD 00101
00011	011 00001	JPOS 00001
00100	000 00000	HALT
00101	000 00001	1

Solution 5.3–2: (Question, p S8)

addr	data	(translation)
00000	100 01010	LOAD 11110
00001	010 01000	JZER 00100
00010	101 01011	STORE 00111
00011	001 01011	JUMP 00000
00100	100 00000	LOAD 00111
00101	101 01010	STORE 11111
00110	000 00000	HALT
00111	000 00000	0

Solution 5.3–3: (Question, p S8)

```
        READ
        STORE n
        ADD n
        ADD n
        ADD n
        ADD v3
        WRITE
        HALT
n:      0
v3:     3
```

Solution 5.3–4: (Question, p S8)

```
top:    READ
        SUB v5
        JZER done
        JUMP top
done:   HALT
v5:     5
```

Solution 5.3–5: (Question, p S9)

```
top:    LOAD v0
        WRITE
        LOAD i
        SUB one
        STORE i
        JPOS top
        HALT
v0:     0
i:      100
```

Solution 5.3–6: (Question, p S9)

```
      READ
      JPOS ok
      STORE n
      SUB n
      SUB n
ok:   WRITE
      HALT
n:    0
```

Solution 5.3–7: (Question, p S9)

```
       READ
       STORE n
   up: LOAD i      # display i
       WRITE
       ADD i       # double i
       STORE i
       LOAD n      # repeat if n - i > 0
       SUB i
       JPOS up
       HALT
    n: 0
    i: 1
```

Solution 5.4–1: (Question, p S9)

a.

```
        READ         # Read n
        STORE n
 while: LOAD n       # while n /= 0, do:
        JZER done
        LOAD v1      #     Write 1.
        WRITE
        READ         #     Read n.
        STORE n
        JUMP while   # end while
        HALT         # Stop.
 v1:    1
 n:     0
```

b.

```
       LOAD zero     # Initialize sum to 0
       STORE sum
       READ          # Read n
       STORE n
 up:   LOAD n        # while n /= 0, do:
       JZER done
       LOAD sum      #     Increase sum by n.
       ADD n
       STORE sum
       READ          #     Read n
       STORE n
       JUMP up       # end while
 done: LOAD sum      # Write sum.
       WRITE
       HALT          # Stop.
 zero: 0
 sum:  0
 n:    0
```

Solution 5.4–2: (Question, p S9)

Read n.
if $n > 0$, **then:**
    Write n.
else:
    Write $-n$.
end if
Stop.

Solution 5.4–3: (Question, p S9)

Initialize max to -128.
repeat 100 **times:**
    Read n.
    if $n > max$, **then:**
        Change max to be n.
    end if
end repeat
Write max.
Stop.

Solution 6.2–1: (Question, p S9)

- The operating system abstracts computer resources.
- The operating system provides hardware compatibility.
- The operating system protects the overall computer system.

Solution 6.2–2: (Question, p S9) Computer resources tend to have complex interfaces, which are difficult to use. The operating system creates a simpler interface, and programs running in the system use this interface instead. When a program uses this simpler interface, the operating system translates the request to the proper commands for the hardware.
For example, a computer display knows nothing about windows, but it is a convenient concept for programs to use. A program might tell the operating system to create a window, and the operating system would tell the display to draw each individual pixel within that window's rectangle white.

Solution 6.3–1: (Question, p S9) Suppose the process currently using the CPU is process A.

1. The operating system stores the contents of all registers, including the program counter (which holds the address of the next instruction A wishes to execute), into memory the OS has dedicated to remembering A's registers.
2. The operating system selects which process to run next. Suppose this is process B.
3. The operating system restores the contents of all registers from memory the OS dedicated to remembering B's registers.
4. The operating system jumps to the current instruction within B.

Solution 6.3–2: (Question, p S10)

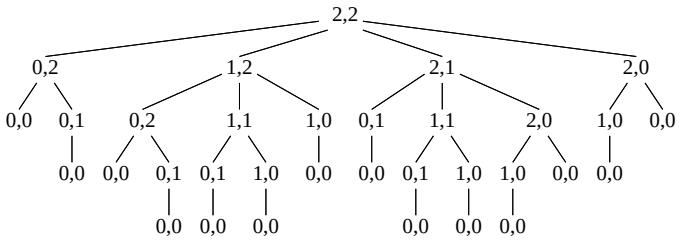
time	CPU	ready	disk
0		AB	
1	A	B	
2		B	A
3	B		A
4		A	B
5	A		B
6	A	B	
7		BA	
8	B	A	
9		A	
10	A		
11			

Solution 6.3–3: (Question, p S10)

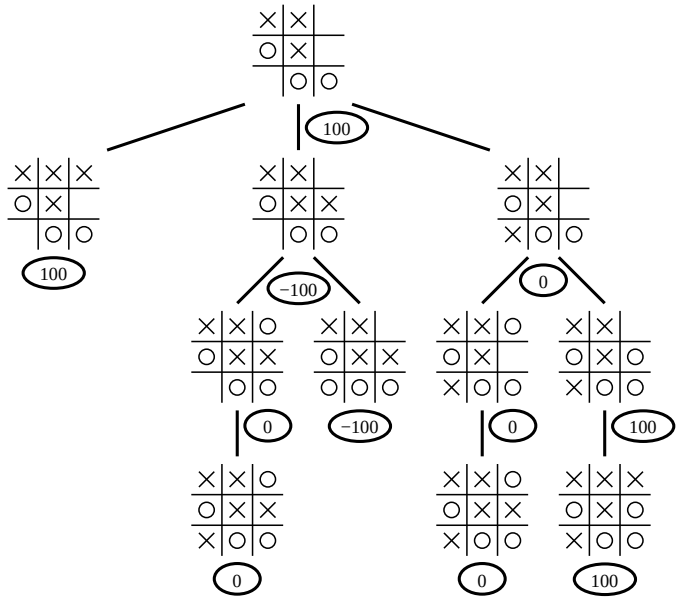
index	contents
0	7
1	—
2	3
3	6
4	2
5	1
6	—
7	4
8	—
9	—
10	5
11	—

Solution 6.3–4: (Question, p S10) The CPU first looks at entry 9 of the page table in RAM, to determine the page's location. If the entry were blank (indicating that the page isn't in RAM), the CPU would force the operating system to load page 9 into RAM. In this case, though, it would find "3" in the entry, and so the CPU would go to page frame 3 in RAM to find the requested data.

Solution 7.1–1: (Question, p S10)

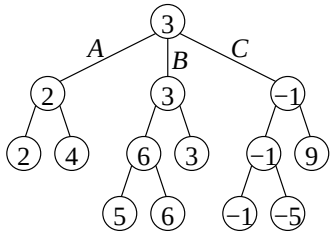


Solution 7.1–2: (Question, p S11)



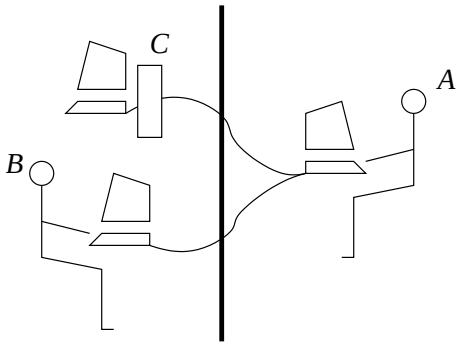
Solution 7.1–3: (Question, p S11)

a.



b. B

Solution 7.2–1: (Question, p S11) Turing proposed that a person and a computer hide behind a screen connected via a communication link to a human tester. The tester poses questions to each and tries to distinguish which is the human. If the tester can't reliably determine which is the human, the computer has "passed" the test.



The purpose of Turing's test is to be a specific, meaningful goal toward which artificial intelligence researchers can strive.

Solution 7.3–1: (Question, p S11)

a. It would predict 1. (The weighted sum is 0.8, which exceeds 0.)

b. $\langle 0.4, 0.3, -0.4 \rangle$

Solution 7.3–2: (Question, p S12)

a. 1 (The weighted sum is 0.2, which exceeds 0.)

b. $\langle -0.5, 1, 1 \rangle$ (There are many other correct answers.)

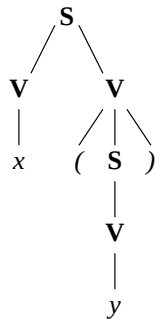
Solution 8.1–1: (Question, p S12)

$$\begin{aligned}
\mathbf{S} &\Rightarrow a\mathbf{S}b\mathbf{S} \\
&\Rightarrow aa\mathbf{S}b\mathbf{S}b\mathbf{S} \\
&\Rightarrow aab\mathbf{S}b\mathbf{S} \\
&\Rightarrow aabb\mathbf{S} \\
&\Rightarrow aabba\mathbf{S}b\mathbf{S} \\
&\Rightarrow aabbab\mathbf{S} \\
&\Rightarrow aabbab
\end{aligned}$$

[Note that it is important in a derivation to replace exactly one symbol in each step. Do not combine steps.]

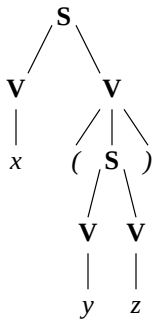
Solution 8.1–2: (Question, p S12)

a.

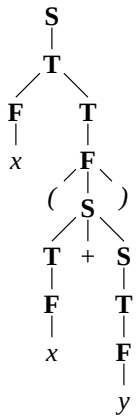


b. *x y z* is not described by the grammar

c.



Solution 8.1–3: (Question, p S12)



Solution 8.1–4: (Question, p S12)

a. $\mathbf{S} \to a\,\mathbf{T}\,a$
$\mathbf{T} \to a\,\mathbf{T} \mid b\,\mathbf{T} \mid \varepsilon$

b. $\mathbf{S} \to \mathbf{A} \mid \mathbf{B}$
$\mathbf{A} \to a\,\mathbf{A} \mid \varepsilon$
$\mathbf{B} \to b\,\mathbf{B} \mid \varepsilon$

c. $\mathbf{S} \to \mathbf{T}\,a\,\mathbf{T}\,b\,\mathbf{T} \mid \mathbf{T}\,b\,\mathbf{T}\,a\,\mathbf{T}$
$\mathbf{T} \to a\,\mathbf{T} \mid b\,\mathbf{T} \mid \varepsilon$

d. $\mathbf{S} \to \mathbf{S}\,\mathbf{S} \mid [\,\mathbf{S}\,] \mid \varepsilon$

Solution 8.2–1: (Question, p S13)

`ab*` includes *abab*?	no
`(a|b)a` includes *aba*?	no
`a(a|b)*b` includes *abbbaaab*?	yes
`(a|b)(ba|ab)*` includes *ababab*?	no
`(a|b)(ba|ab)*` includes *abababa*?	yes

Solution 8.2–2: (Question, p S13)

a. **`a*b*`**

b. **`(a|b)*aab(a|b)*`**

c. **`(aa)*`**

d. **`a*|b*`**

e. **`1(0|1)*0`**

f. **`1(0|1)(0|1)(0|1)*`** [Binary numbers that are at least 4 must have at least three bits. This regular expression describes all combinations of 0's and 1's that begin with a 1, have two more bits, and can have any number of bits thereafter.]

Solution 8.2–3: (Question, p S13) The language of strings of *a*'s and *b*'s containing the same number of each is context-free, but it is not regular.

Solution 9.1–1: (Question, p S13)

√ *ab*	___ *aabbb*
___ *bbb*	√ *bbbabb*
___ *baaa*	√ *aabaabaa*
√ *abba*	

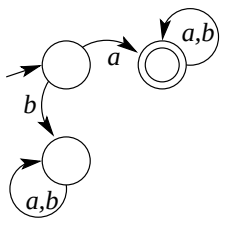
Solution 9.1–2: (Question, p S13)

a.

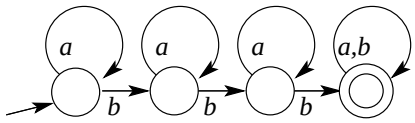
√ *b*	√ *bab*
___ *a*	√ *bbbb*
___ *aba*	___ *baba*

b. It accepts strings of *a*'s and *b*'s ending in a *b*.

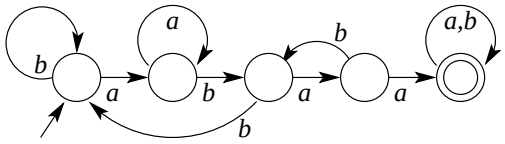
Solution 9.1–3: (Question, p S14)



Solution 9.1–4: (Question, p S14)



Solution 9.1–5: (Question, p S14)



Solution 9.2–1: (Question, p S14)

$\underline{0}$ *ababb*
a$\underline{0}$*abb*
ab$\underline{0}$*abb*
aba$\underline{0}$*bb*
abab$\underline{0}$*b*
ababb$\underline{0}$
abab$\underline{1}$
aba$\underline{1}$*ba*
ab$\underline{1}$*aa*
abb$\underline{2}$*a*

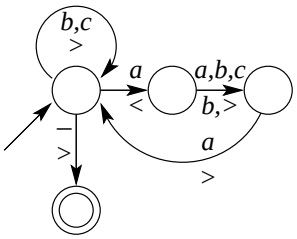
(At this point, the machine has nowhere to go, and so it stops.)

Solution 9.2–2: (Question, p S15)

$\underline{0}$ *ab*
$\underline{1}$ *ab*
$\underline{3}$ *bab*
ba$\underline{0}$*b*
$\underline{2}$ *bab*
ba$\underline{3}$*b*
bab$\underline{0}$
ba$\underline{4}$*b*

(At this point, the machine has nowhere to go, and so it stops.)

Solution 9.2–3: (Question, p S15)



Solution 9.2–4: (Question, p S15) The Turing machine model is computationally as powerful as any other computational model.

Solution 9.3–1: (Question, p S15) The halting problem is the set of strings of the form $M!x$ where M is the string representation of a Turing machine, and this Turing machine does not accept the string x as part of the language it recognizes.

Solution 9.3–2: (Question, p S15)

a. D. the languages are identical

b. B. the second is contained within the first

c. A. the first is contained within the second

d. A. the first is contained within the second