TT TEXTBOOK TESTBANKS

Solutions Manual for Digital Logic Circuit Analysis and Design 2nd Nelson

SOLUTIONS MANUAL SAMPLE PREVIEW

PUBLISHER
Pearson

COPYRIGHT
2021

RESOURCE TYPE
Solutions Manual

ISBN
9780135297070

Digital Logic Circuit Analysis and Design

Second Edition

Problem Solutions Manual

Victor P. Nelson
Auburn University

Bill D. Carroll
University of Texas at Arlington

H. Troy Nagle
North Carolina State University

J. David Irwin
Auburn University

Pearson

Contents

Chapter 1 – Number Systems and Digital Codes

1.1 Calculate $A+B$, $A-B$, $A\times B$, and $A\div B$ for the following pairs of binary numbers.

(a) 10101, 1011

```
  1  1  1  1  1          0  10  0  10
     1  0  1  0  1        1  0  1  0  1
+       1  0  1  1     −     1  0  1  1
  1  0  0  0  0  0           1  0  1  0
```

```
                 1  0  1  0  1
              ×     1  0  1  1
                 1  0  1  0  1
              1  0  1  0  1
           0  0  0  0  0
        1  0  1  0  1
  1  1  1  0  0  1  0  1
```

```
                                  1
  1  0  1  1 | 1  0  1  0  1
              -  1  0  1  1
Remainder        1  0  1  0
```

(b) 1011010, 101111

```
  1  1  1  1  1  1              10  1  10
     1  0  1  1  0  1  0     0  10  0  0  10  0  10
+       1  0  1  1  1  1     1  0  1  1  0  1  0
  1  0  0  0  1  0  0  1  −     1  0  1  1  1  1
                              1  0  1  0  1  1
```

```
                    1  0  1  1  0  1  0
                 ×     1  0  1  1  1  1
                    1  0  1  1  0  1  0
                 1  0  1  1  0  1  0
              1  0  1  1  0  1  0
           1  0  1  1  0  1  0
        0  0  0  0  0  0  0
     1  0  1  1  0  1  0
  1  0  0  0  0  1  0  0  0  0  1  1  0
```

```
                                    0  1  .  1  1
  1  0  1  1  1  1 | 1  0  1  1  0  1  0  .  0  0
                    1  0  1  1  1  1
                    1  0  1  1  0  1  0
                       1  0  1  1  1  1
                    0  1  0  1  0  1  1     0
                          1  0  1  1  1     1
                       0  1  0  0  1  1     1  0
                             1  0  1  1     1  1
Remainder                    0  1  1  1     1  1
```

(c) 101, 1011

```
    1  1  1                    10  0  10
    1  0  1                    0  1  0  1
+   1  0  1  1              −     1  0  1  1
-------------              -------------
1  0  0  0  0              (1)  1  0  0  0  0
                            Borrow
```

```
       1  0  1
×      1  0  1  1
-------------------
       1  0  1
    1  0  1
 0  0  0
1  0  1
-------------------
1  1  0  1  1  1
```

```
                  0.  0  1  1
          ------------------
1  0  1  1 | 1  0  1.  0  0  0
           - 1  0  1  1
          ------------------
              1  0  0  1  0
           - 1  0  1  1
          ------------------
Remainder     0  1  1  1
```

(d) 10110110, 01011011

```
   1  1  1  1  1  1
   1  0  1  1  0  1  1  0
+  0  1  0  1  1  0  1  1
-------------------------
1  0  0  0  1  0  0  0  1
```

```
                  10          10
   0  10  0  0  10  0  0  10
   1  0  1  1  0  1  1  0
−  0  1  0  1  1  0  1  1
-------------------------
   0  1  0  1  1  0  1  1
```

```
                        1  0  1  1  0  1  1  0
                     ×  0  1  0  1  1  0  1  1
                        ----------------------
                        1  0  1  1  0  1  1  0
                     1  0  1  1  0  1  1  0
                  0  0  0  0  0  0  0  0
               1  0  1  1  0  1  1  0
            1  0  1  1  0  1  1  0
         0  0  0  0  0  0  0  0
      1  0  1  1  0  1  1  0
   0  0  0  0  0  0  0  0                          1
-------------------------------------------------------
   0  0  0  0  0  0  1  0     1  1  0  0  1  0
```

```
                                    1  0
                  ----------------------
0  1  0  1  1  0  1  1 | 1  0  1  1  0  1  1  0
                       1  0  1  1  0  1  1
                     ---------------------
                       0  0  0  0  0  0  0  0
```

(e) 1101011, 1010

```
        1   1
    1 1 0 1 0 1 1
+       1 0 1 0
  1 1 1 0 1 0 1
```

```
    1 1 0 1 0 1 1
-       1 0 1 0
  1 1 0 0 0 0 1
```

```
          1 1 0 1 0 1 1
        ×       1 0 1 0
          0 0 0 0 0 0 0
        1 1 0 1 0 1 1
      0 0 0 0 0 0 0
    1 1 0 1 0 1 1
1 0 0 0 0 1 0 1 1 1 0
```

```
                  1 0 1 0
        ┌─────────────────
1 0 1 0 │ 1 1 0 1 0 1 1
          1 0 1 0
          ─────────
          0 0 1 1 0 1
              1 0 1 0
              ─────── 
Remainder     0 1 1 1
```

(f) 1010101, 101010

```
      1 0 1 0 1 0 1
+       1 0 1 0 1 0
    1 1 1 0 0 0 1
```

```
      0  10 0  10 0  10
      1  0  1  0  1  0  1
−     1  0  1  0  1  0
    0  1  0  1  0  1  1
```

```
          1 0 1 0 1 0 1
        ×     1 0 1 0 1 0
          0 0 0 0 0 0 0
        1 0 1 0 1 0 1
      0 0 0 0 0 0 0
    1 0 1 0 1 0 1
  0 0 0 0 0 0 0
1 0 1 0 1 0 1
1 1 0 1 1 1 1 1 0 0 1 0
```

```
                      1 0 . 1 1
        ┌─────────────────────
1 0 1 0 1 0 │ 1 0 1 0 1 0 1 . 0 0
            1 0 1 0 1 0
            ───────────
Remainder       0 1   0 0
```

(g) 10000, 1001

```
    1 0   0 0 0
+     1   0 0 1
  1 1   0 0 1
```

```
      1   1   1
  0  10  10  10  10
  1  0   0   0   0
−     1   0   0   1
  0   0   1   1   1
```

```
          1 0 0 0 0
        ×     1 0 0 1
          1 0 0 0 0
        0 0 0 0 0
      0 0 0 0 0
    1 0 0 0 0
1 0 0 1 0 0 0 0
```

```
                    1 .1  1
        ┌─────────────────
1 0 0 1 │ 1 0 0 0 0 .0  0
          - 1 0 0 1
          ─────────
            0 1 1 1 0
              1 0 0 1
              ─────── 
            0 1 0 1 0
                1 0 0 1
                ─────── 
```

Remainder 0 0 0 1

(h) 1011.0101, 110.11

```
  1  1  1  1  1
  1  0  1  1. 0  1  0  1
+    1  1  0. 1  1
---------------------
  1  0  0  1  0  0  0  0  1
```

```
     0  10    0  10
  1  0  1  1. 0  1  0  1
-    1  1  0. 1  1
---------------------
  0  1  0  0. 1  0  0  1
```

```
                 1  0  1  1. 0  1  0  1
              ×              1  1  0. 1  1
---------------------------------------
                 1  0  1  1  0  1  0  1
              1  0  1  1  0  1  0  1
           0  0  0  0  0  0  0  0
        1  0  1  1  0  1  0  1
     1  0  1  1  0  1  0  1
---------------------------------------
  1  0  1  1  1  0  0. 0  1  0  1  1  1
```

```
                          1. 1  0
                 ------------------
1  1  0. 1  1  | 1  0  1  1. 0  1  0  1
               |    1  1  0  1  1
               |    ---------------
               |    1  0  0  1  0  0
               |       1  1  0  1  1
               |       ------------
Remainder      |       0  1  0  0  1  1
```

1.2 Calculate $A+B$, $A-B$, $A\times B$, and $A\div B$ for the following pairs of octal numbers.

(a) 372, 156

```
  1  1
  3  7  2
+ 1  5  6
---------
  5  5  0
```

```
     6  12
  3  7  2
- 1  5  6
---------
  2  1  4
```

```
        3  7  2
     ×  1  5  6
---------------
     2  7  3  4
  2  3  4  2
  3  7  2
---------------
  6  5  5  5  4
```

```
                 2. 2  1
           ---------------
1  5  6  | 3  7  2. 0  0
          | 3  3  4
          | ---------
          |    3  6  0
          |    3  3  4
          |    ------
          |       2  4  0
          |       1  5  6
          |       ------
Remainder |          6  2
```

(b) 704, 230

```
    7 0 4
+   2 3 0
-------
    1 1 3 4
```

```
      6  10
    ~~7~~ ~~0~~ ~~4~~
-   2  3  0
-------
    4  5  4
```

```
        7 0 4
      × 2 3 0
-------
        0 0 0
    2 5 1 4
1 6 1 0
-------
2 0 6 1 4 0
```

```
                2. 7 6
        ------------
2 3 0 | 7 0 4. 0 0
        4 6 0
        -----
        2 2 4 0
        2 0 5 0
        -------
            1 7 0 0
            1 5 2 0
            -------
Remainder       1 6 0
```

(c) 1000, 777

```
    1 0 0 0
+     7 7 7
-------
    1 7 7 7
```

```
      7   7
    0  10  10  10
    ~~1~~ ~~0~~  ~~0~~  ~~0~~
-     7   7   7
-------
    0   0   0   1
```

```
        1 0 0 0
      × 7 7 7
-------
        7 0 0 0
    7 0 0 0
7 0 0 0
-------
7 7 7 0 0 0
```

```
                  1. 0 0 1
          ----------
7 7 7 | 1 0 0 0. 0 0 0
        7 7 7
        -----
        0 0 1 0 0 0
              7 7 7
              -----
Remainder     0 0 0 1
```

(d) 423, 651

```
    4 2 3
+   6 5 1
-------
    1 2 7 4
```

```
      4  11
    ~~5~~ ~~1~~
-   6
    4  2  3
-   2  2  6
```

```
        4 2 3
      × 6 5 1
-------
        4 2 3
    2 5 3 7
3 1 6 2
-------
3 4 4 2 1 3
```

```
                0. 5 1 3
          ----------
6 5 1 | 4 2 3. 0 0 0
        4 1 1 5
        -------
            1 1 3 0
            0 6 5 1
            -------
                2 5 7 0
                2 3 7 3
                -------
Remainder       1 7 5
```

1.3 Calculate $A+B$, $A-B$, $A\times B$, and $A\div B$ for the following pairs of hexadecimal numbers.

(a) 2CF3, 2B

		1		
	2	C	F	3
+			2	B
	2	D	1	E

		E	*13*	
	2	C	~~F~~	~~3~~
−			2	B
	2	C	C	8

	2	C	F	3
×			2	B
1	E	E	7	1
5	9	E	6	
7	8	C	D	1

			1	0	B
2	B	2	C	F	3
		2	B		
		0	1	F	3
			1	D	9
Remainder				1	A

(b) FFFF, 1000

		1		
	F	F	F	F
+	1	0	0	0
1	0	F	F	F

	F	F	F	F
−	1	0	0	0
	E	F	F	F

				F	F	F	F	
			×	1	0	0	0	
				0	0	0	0	
			0	0	0	0		
		0	0	0	0			
F	F	F	F					
F	F	F	F	0	0	0		

						F.	F	F	
1	0	0	0	F	F	F	F.	0	0
				F	0	0	0		
					F	F	F	0	
					F	0	0	0	
						F	F	0	0
						F	0	0	0
			Remainder				F	0	0

(c) 9A5, D17

			C	*11*
	9	A	5	
+	D	1	7	
1	6	B	C	

			~~D~~	~~1~~
−			9	A
+	9	A	5	
	3	7	2	

		9	A	5
×		D	1	7
	4	3	8	3
	9	A	5	
7	D	6	1	
7	E	3	E	D	3

					0.	B	C	9
D	1	7	9	A	5.	0	0	0
			8	F	F	D		
				A	5	3	0	
				9	D	1	4	
					8	1	C	0
					7	5	C	F
			Remainder			B	F	1

(d) 372, 156

```
    3 7 2
+   1 5 6
-------
    4 C 8
```

```
      6  12
    3  7  2
-   1  5  6
-----------
    2  1  C
```

```
      3 7 2
    × 1 5 6
    -------
      1 4 A C
    1 1 3 A
    3 7 2
  -----------
  4 9 A 4 C
```

```
                  2.  9  4
          -----------------
1 5 6 | 3  7  2.  0  0
        2  A  C
        -------
           C  6  0
           C  0  6
           -------
              5  A  0
              5  5  8
              -------
Remainder        4  8
```

1.4 Convert each of the following decimal numbers to binary, octal, and hexadecimal numbers.

(a) 27

2	27	1		LSD
2	13	1		
2	6	0		
2	3	1		
2	1	1		MSD
	0			

$(27)_{10} = (11011)_2$

8	27	3		LSD
	3	3		MSD
	0			

$(27)_{10} = (33)_8$

1	6	27	B		LSD
	1	6	3	1	MSD
		0			

$(27)_{10} = (1B)_{16}$

(b) 915

2	915	1			LSD
2	457	1			
2	228	0			
2	114	0			
2	57	1			
2	28	0			
2	14	0			
2	7	1			
2	3	1			
2	1	1			MSD
	0				

$(915)_{10} = (1110010011)_2$

8	915	3			LSD
8	114	2			
	14	6			
	1	1			MSD
	0				

$(915)_{10} = (1623)_8$

Or

$(915)_{10} = (\underline{1}\ \underline{110}\ \underline{010}\ \underline{011})_2$

$= (1\ \ 6\ \ 2\ \ 3\ )_8$

1	6	915	3			LSD
	1	57	9			
		3	3			MSD
		0				

$(27)_{10} = (393)_{16}$

Or

$(915)_{10} = (\underline{11}\ \underline{1001}\ \underline{0011})_2$

$= (3\ \ \ 9\ \ \ 3\ )_{16}$

(c) 0.375

$0.375 \times 2 = \underline{0}.750$

$0.750 \times 2 = \underline{1}.500$

$0.500 \times 2 = \underline{1}.000$

$(0.375)_{10} = (0.011)_2$

$0.375 \times 8 = \underline{3}.000$

$(0.375)_{10} = (0.3)_8$

$0.375 \times 16 = \underline{6}.000$

$(0.375)_{10} = (0.6)_{16}$

(d) 0.65

$.65 \times 2 = \underline{1}.3$

$.3 \times 2 = \underline{0}.6$

$.6 \times 2 = \underline{1}.2$

$.2 \times 2 = \underline{0}.4$

$.4 \times 2 = \underline{0}.8$

$.8 \times 2 = \underline{1}.6$

repeat

$(0.65)_{10} = (0.10\overline{1001})_2$

$.65 \times 8 = \underline{5}.2$

$.2 \times 8 = \underline{1}.6$

$.6 \times 8 = \underline{4}.8$

$.8 \times 8 = \underline{6}.4$

$.4 \times 8 = \underline{3}.2$

repeat

$(0.65)_{10} = (0.5\overline{1463})_8$

$0.65 \times 16 = \underline{10}.4$

$0.4 \times 16 = \underline{6}.4$

repeat

$(0.375)_{10} = (0.A\overline{4})_{16}$

(e) 174.25

Integer part:

2	1 7 4	0	LSD		
2	8 7	1			
2	4 3	1			
2	2 1	1			
2	1 0	0			
2	5	1			
2	2	0			
2	1	1			
	0				

$(174)_{10} = (10101110)_2$

8	1 7 4	6	LSD
8	2 1	5	
	8 2	2	MSD
	0		

$(174)_{10} = (256)_8$

Or

$(174)_{10} = (\underline{10}\ \underline{101}\ \underline{110})_2$

$= (2\quad 5\quad 6\ )_8$

1 6	1 7 4	14	LSD
1 6	1 0	10	MSD
	0		

$(174)_{10} = (AE)_{16}$

Or

$(174)_{10} = (\underline{1010}\ \underline{1110})_2$

$= (\ A\quad E\ )_{16}$

Fractional part:

$.25 \times 2 = \underline{0}.5$

$.5 \times 2 = \underline{1}.0$

$(0.25)_{10} = (0.01)_2$

$.25 \times 8 = \underline{2}.0$

$(0.25)_{10} = (0.2)_8$

$0.25 \times 16 = \underline{4}.0$

$(0.25)_{10} = (0.4)_{16}$

Therefore, combining integer and fractional parts:

$(174.25)_{10} = (10101110.01)_2 = (256.2)_8 = (AE.4)_{16}$

(f) 250.8

Integer part:

2	2 5 0	0	LSD		
1	1 2 5	1			
2	6 2	0			
2	3 1	1			
2	1 5	1			
2	7	1			
2	3	1			
2	1	1			
	0				

$(250)_{10} = (11111010)_2$

8	2 5 0	2	LSD
8	3 1	7	
	8 3	3	MSD
	0		

$(250)_{10} = (372)_8$

Or

$(250)_{10} = (\underline{11}\ \underline{111}\ \underline{010})_2$

$= (3\quad 7\quad 2\ )_8$

1 6	2 5 0	10	LSD
1 6	1 5	15	MSD
	0		

$(250)_{10} = (FA)_{16}$

Or

$(250)_{10} = (\underline{1111}\ \underline{1010})_2$

$= (\ F\quad A\ )_{16}$

Fractional part:

$.8 \times 2 = \underline{1}.6$

$.6 \times 2 = \underline{1}.2$

$.2 \times 2 = \underline{0}.4$

$.4 \times 2 = \underline{0}.8$

repeats

$(0.8)_{10} = (0.\overline{1100})_2$

$.8 \times 8 = \underline{6}.4$

$.4 \times 8 = \underline{3}.2$

$.2 \times 8 = \underline{1}.6$

$.6 \times 8 = \underline{4}.8$

repeats

$(0.8)_{10} = (0.\overline{6314})_8$

$0.8 \times 16 = \underline{12}.8$

repeats

$(0.8)_{10} = (0.\overline{C})_{16}$

Therefore, combining integer and fractional parts:
$(250.8)_{10} = (11111010.\overline{1100})_2 = (372.\overline{6314})_8 = (FA.\overline{C})_{16}$

1.5 Convert each of the following binary numbers to octal, hexadecimal, and decimal numbers using the most appropriate conversion method.

(a) 1101

001 101 = 1 5 = $(15)_8$ 1101 = D = $(D)_{16}$

$2^3 + 2^2 + 2^0 = 8 + 4 + 1 = (13)_{10}$

(b) 101110

101 110 = 5 6 = $(56)_8$ 0010 1110 = 2 E = $(2E)_{16}$

$2^5 + 2^3 + 2^2 + 2^1 = 32 + 8 + 4 + 2 = (46)_{10}$

(c) 0.101

.101 = .5 = $(.5)_8$ .1010 = .A = $(.A)_{16}$

$2^{-1} + 2^{-3} = .5 + .125 = (.625)_{10}$

(d) 0.01101

.011 010 = .3 2 = $(.32)_8$ .0110 1000 = .6 8 = $(.68)_{16}$

$2^{-2} + 2^{-3} + 2^{-5} = .25 + .125 + .03125 = (.40625)_{10}$

(e) 10101.11

010 101.110 = 2 5 . 6 = $(25.6)_8$ 0001 0101 . 1100 = 1 5 . C = $(15.C)_{16}$

$2^4 + 2^2 + 2^0 + 2^{-1} + 2^{-2} = 16 + 4 + 1 + .5 + .25 = (21.75)_{10}$

(f) 10110110.001

010 110 110 . 001 = 2 6 6 . 1 = $(266.1)_8$ 1011 0110 . 0010 = B 6 . 2 = $(B6.2)_{16}$

$2^7 + 2^5 + 2^4 + 2^2 + 2^1 + 2^{-3} = 128 + 32 + 16 + 4 + 1 + .125 = (181.125)_{10}$

1.6 Convert each of the following octal numbers to binary, hexadecimal, and decimal using the most appropriate conversion method.

(a) 65

$(65)_8 = (\underline{110}\ \underline{101})_2$
6 5
0011 0101 $= (35)_{16}$
3 5
$6 \times 8 + 5 = 48 + 5 = (53)_{10}$

(b) 371

$(371)_8 = (\underline{011}\ \underline{111}\ \underline{001})_2$
3 7 1
$\underline{1111}\ \underline{1001} = (F9)_{16}$
F 9
$3 \times 8^2 + 7 \times 8 + 1 = 192 + 56 + 1 = (249)_{10}$

(c) 240.51

$(240.51)_8 = (\underline{010}\ \underline{100}\ \underline{000} .\ \underline{101}\ \underline{001})_2$
2 4 0 5 1
$\underline{1010}\ \underline{0000} .\ \underline{1010}\ \underline{0100} = (A0.A4)_{16}$
A 0 A 4
$2 \times 8^2 + 4 \times 8 + 5 \times 8^{-1} + 4 \times 8^{-2} = 128 + 32 + .625 + .0156 = (160.6406)_{10}$

(d) 2000

$(2000)_8 = (\underline{010}\ \underline{000}\ \underline{000}\ \underline{000})_2$
2 0 0 0
0100 0000 0000 $= (400)_{16}$
4 0 0
$2 \times 8^3 = 1024 = (1024)_{10}$

(e) 111111

$(111111)_8 = (\underline{001}\ \underline{001}\ \underline{001}\ \underline{001}\ \underline{001}\ \underline{001})_2$
1 1 1 1 1 1
$\underline{1001}\ \underline{0010}\ \underline{0100}\ \underline{1001} = (9249)_{16}$
9 2 4 9
$8^5 + 8^4 + 8^3 + 8^2 + 8^1 + 8^0 = 32{,}878 + 4{,}096 + 512 + 64 + 8 + 1 = (37{,}449)_{10}$

(f) 177777

$(177777)_8 = (\underline{001}\ \underline{111}\ \underline{111}\ \underline{111}\ \underline{111}\ \underline{111})_2$
1 7 7 7 7 7
$\underline{1111}\ \underline{1111}\ \underline{1111}\ \underline{1111} = (FFFF)_{16}$
F F F F

$8^5 + 7 \times 8^4 + 7 \times 8^3 + 7 \times 8^2 + 7 \times 8 + 7 = 32{,}768 + 28{,}672 + 3{,}584 + 448 + 56 + 7 = (65{,}535)_{10}$

1.7 Convert each of the following hexadecimal numbers to binary, octal, and decimal using the most appropriate conversion method.

(a) 4F

$(4F)_{16} = (\underline{0100}\ \underline{1111})_2$

4 F

$\underline{001}\ \underline{001}\ \underline{111} = (117)_8$

1 1 7

$4 \times 16 + 15 = 64 + 15 = (79)_{10}$

(b) ABC

$(ABC)_{16} = (\underline{1010}\ \underline{1011}\ \underline{1100})_2$

A B C

$\underline{101}\ \underline{010}\ \underline{111}\ \underline{100} = (5,274)_8$

5 2 7 4

$10 \times 16^2 + 11 \times 16 + 12 = 2,560 + 176 + 12 = (2,748)_{10}$

(c) F8.A7

$(F8.A7)_{16} = (\underline{1111}\ \underline{1000} . \underline{1010}\ \underline{0111})_2$

F 8 A 7

$\underline{011}\ \underline{111}\ \underline{000} . \underline{101}\ \underline{001}\ \underline{110} = (370.516)_8$

3 7 0 5 1 6

$15{\times}16 + 8 + 10{\times}16^{-1} + 7{\times}16^{-2} = 240 + 8 + .625 + .027343750 = (240.6523438)_{10}$

(d) 2000

$(2000)_{16} = (\underline{0010}\ \underline{0000}\ \underline{0000}\ \underline{0000})_2$

2 0 0 0

$\underline{010}\ \underline{000}\ \underline{000}\ \underline{000}\ \underline{000} = (20,000)_8$

2 0 0 0 0

$2{\times}16^3 = (8,192)_{10}$

(e) 201.4

$(201.4)_{16} = (\underline{0010}\ \underline{0000}\ \underline{0001} . \underline{0100})_2$

2 0 1 4

$\underline{001}\ \underline{000}\ \underline{000}\ \underline{001} . \underline{010} = (1001.2)_8$

1 0 0 1 2

$2{\times}16^2 + 1 + 4{\times}16^{-1} = 512 + 1 + .25 = (513.25)_{10}$

(f) 3D65E

$(3D65E)_{16} = (\underline{0011}\ \underline{1101}\ \underline{0110}\ \underline{0101}\ \underline{1110})_2$

3 D 6 5 E

$\underline{111}\ \underline{101}\ \underline{011}\ \underline{001}\ \underline{011}\ \underline{110} = (753,136)_8$

7 5 3 1 3 6

$3{\times}16^4 + 13{\times}16^3 + 6{\times}16^2 + 5{\times}16 + 14 = 196,608+2,808+1,536+80+14 = (251,486)_{10}$

1.8 Find the two's complement of each of the following binary numbers assuming $n = 8$.

(a) 101010

$$\begin{aligned} N &= 00101010 \\ &\quad 11010101 \text{ - complement} \\ &\quad +1 \text{ - add 1} \\ [N]_2 &= (11010110)_2 \end{aligned}$$

(b) 1101011

$$\begin{aligned} N &= 01101011 \\ &\quad 10010100 \text{ - complement} \\ &\quad +1 \text{ - add 1} \\ [N]_2 &= (10010101)_2 \end{aligned}$$

(c) 0

$$\begin{aligned} N &= 00000000 \\ &\quad 11111111 \text{ - complement} \\ &\quad +1 \text{ - add 1} \\ [N]_2 &= (00000000)_2 \end{aligned}$$

(d) 11111111

$$\begin{aligned} N &= 00000000 \\ &\quad 11111111 \text{ - complement} \\ &\quad +1 \text{ - add 1} \\ [N]_2 &= (00000000)_2 \end{aligned}$$

(e) 10000000

$$\begin{aligned} N &= 10000000 \\ &\quad 01111111 \text{ - complement} \\ &\quad +1 \text{ - add 1} \\ [N]_2 &= (10000000)_2 \end{aligned}$$

(f) 11000

$$\begin{aligned} N &= 00011000 \\ &\quad 11100111 \text{ - complement} \\ &\quad +1 \text{ - add 1} \\ [N]_2 &= (11101000)_2 \end{aligned}$$

1.9 Find the one's complement of each of the following binary numbers assuming $n = 8$.

(a) 110101

00110101

11001010 - l's complement

(b) 1010011

01010011

10101100 - l's complement

(c) 0

00000000

11111111 - l's complement

(d) 10000000

10000000

01111111 - l 's complement

(e) 100001

00100001

11011110 - l's complement

(f) 01111111

01111111

10000000 - l's complement

1.10 Calculate $A+B$, $A-B$, $-A+B$, and $-A-B$ for each of the following pairs of numbers assuming a two's complement number system and $n=8$. Check your results by decimal arithmetic. Explain any unusual results.

(a) 1010101, 1010

A = $(0, 1010101)_{2cns}$ $+(85)_{10}$
−A = $(0, 1010101)_{2cns}$ $-(85)_{10}$

B = $(0, 001010)_{2cns}$ $+(10)_{10}$
−B = $(1, 110110)_{2cns}$ $-(10)_{10}$

A + B

	0	1	0	1	0	1	0	1
+	0	0	0	0	1	0	1	0
	0	1	0	1	1	1	1	1

Result: $(0, 1011111)_{2cns} = +(95)_{10}$

A - B

	0	1	0	1	0	1	0	1
+	1	1	1	1	0	1	1	0
(1)	0	1	0	0	1	0	1	1

(Discard carry)

Result: $(0, 1001011)_{2cns} = +(75)_{10}$

-A+B

	1	0	1	0	1	0	1	1
+	0	0	0	0	1	0	1	0
	1	0	1	1	0	1	0	1

Result: $(1, 0110101)_{2cns} = -(75)_{10}$

-A-B

	1	0	1	0	1	0	1	1
+	1	1	1	1	0	1	1	0
(1)	1	0	1	0	0	0	0	1

(Discard carry)

Result: $(1, 0100001)_{2cns} = -(95)_{10}$

(b) 1101011, 0101010

A = $(0, 1101011)_{2cns}$ $+(107)_{10}$
−A = $(1, 0010101)_{2cns}$ $-(107)_{10}$

B = $(0, 0101010)_{2cns}$ $+(42)_{10}$
−B = $(1, 1010110)_{2cns}$ $-(42)_{10}$

A + B

	0	1	1	0	1	0	1	1
+	0	0	1	0	1	0	1	0
	1	0	0	1	0	1	0	1

Overflow condition! Invalid result

A - B

	0	1	1	0	1	0	1	1
+	1	1	0	1	0	1	1	0
(1)	0	1	0	0	0	0	0	1

(Discard carry)

Result: $(0, 1001011)_{2cns} = +(65)_{10}$

-A + B

	1	0	0	1	0	1	0	1
+	0	0	1	0	1	0	1	0
	1	0	1	1	1	1	1	1

Result: $(1, 0110101)_{2cns} = -(65)_{10}$

-A - B

	1	0	0	1	0	1	0	1
+	1	1	0	1	0	1	1	0
(1)	0	1	1	0	1	0	1	1

Overflow condition! Invalid result

(c) 11101010, 101111

$A = (1, 1101010)_{2cns}$ $-(22)_{10}$ $\quad B = (0, 0101111)_{2cns}$ $+(47)_{10}$

$-A = (0, 0010110)_{2cns}$ $+(22)_{10}$ $\quad -B = (1, 1010001)_{2cns}$ $-(47)_{10}$

A + B

	1	1	1	0	1	0	1	0
+	0	0	1	0	1	1	1	1
(1)	0	0	0	1	1	0	0	1

(Discard carry)

Result: $(0,0011001)_{2cns} = +(25)_{10}$

A - B

	1	1	1	0	1	0	1	0
+	1	1	0	1	0	0	0	1
(1)	1	0	1	1	1	0	1	1

(Discard carry)

Result: $(1, 0111011)_{2cns} = -(69)_{10}$

-A + B

	0	0	0	1	0	1	1	0
+	0	0	1	0	1	1	1	1
	0	1	0	0	0	1	0	1

Result: $(0, 1000101)_{2cns} = +(69)_{10}$

-A - B

	0	0	0	1	0	1	1	0
+	1	1	0	1	0	0	0	1
	1	1	1	0	0	1	1	1

Result: $(1, 0111011)_{2cns} = -(25)_{10}$

(d) 10000000, 01111111

$A = (1, 0000000)_{2cns}$ $-(22)_{10}$ $\quad B = (0, 1111111)_{2cns}$ $+(127)_{10}$

$-A$: can't do $+(128)_{10}$ with 8 bits $\quad -B = (1, 0000001)_{2cns}$ $-(127)_{10}$

A + B

	1	0	0	0	0	0	0	0
+	0	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1

(Discard carry)

Result: $(1,1111111)_{2cns} = -(128)_{10}$

A - B

	1	0	0	0	0	0	0	0
+	1	0	0	0	0	0	0	1
(1)	0	0	0	0	0	0	0	1

Overflow condition! Invalid result

Cannot do ($-A + B$) or ($-A - B$) since $-A$ cannot be represented!

1.11 Repeat Problem 1.10 for the following numbers using a one's complement number system.

(a) 101011, 1101

$A = (0, 0101011)_{1cns}$ $+(43)_{10}$ $\quad$ $B = (0, 0001101)_{1cns}$ $+(13)_{10}$

$-A = (1, 1010100)_{1cns}$ $-(43)_{10}$ $\quad$ $-B = (1, 1110010)_{1cns}$ $-(13)_{10}$

A + B

	0	0	1	0	1	0	1	1
+	0	0	0	0	1	1	0	1
	0	0	1	1	1	0	0	0

Result: $(0, 0111000)_{1cns} = +(56)_{10}$

A - B

	0	0	1	0	1	0	1	1
+	1	1	1	1	0	0	1	0
(1)	0	0	0	1	1	1	0	1
							+	1
	0	0	0	1	1	1	1	0

(Carry added in)

Result: $(0, 1001100)_{1cns} = +(30)_{10}$

-A + B

	1	1	0	1	0	1	0	1
+	0	0	0	0	1	1	0	1
	1	1	1	0	0	0	1	0

Result: $(1, 0110101)_{1cns} = -(30)_{10}$

-A-B

	1	1	0	1	0	1	0	1
+	1	1	1	1	0	0	1	0
(1)	1	1	0	0	0	1	1	1
							+	1
	1	1	0	0	1	0	0	0

(Discard carry)

Result: $(1, 0100001)_{1cns} = -(56)_{10}$

(b) 10111010, 11010

$A = (1, 0111010)_{1cns}$ $-(69)_{10}$ $\quad$ $B = (0, 0011010)_{1cns}$ $+(26)_{10}$

$-A = (0, 1000101)_{1cns}$ $+(69)_{10}$ $\quad$ $-B = (1, 1100101)_{1cns}$ $-(26)_{10}$

A + B

	1	0	1	1	1	0	1	0
+	0	0	0	1	1	0	1	0
	1	1	0	1	0	1	0	0

Result: $(0, 0111000)_{1cns} = -(43)_{10}$

A - B

	1	0	1	1	1	0	1	0
+	1	1	1	0	0	1	0	1
(1)	1	0	0	1	1	1	1	1
							+	1
	1	0	1	0	0	0	0	0

(Carry added in)

Result: $(0, 1001100)_{1cns} = -(95)_{10}$

-A + B

	0	1	0	0	0	1	0	1
+	0	0	0	1	1	0	1	0
	0	1	0	1	1	1	1	1

Result: $(1, 0110101)_{1cns} = +(95)_{10}$

-A-B

	0	1	0	0	0	1	0	1
+	1	1	1	0	0	1	0	1
(1)	0	0	1	0	1	0	1	0
							+	1
	0	0	1	0	1	0	1	1

(Discard carry)

Result: $(1, 0100001)_{1cns} = +(43)_{10}$

(c) 1010101, 0101010

$A = (0, 1010101)_{1cns}$ $+(85)_{10}$ $\quad B = (0, 0101010)_{1cns}$ $+(42)_{10}$

$-A = (1, 0101010)_{1cns}$ $-(85)_{10}$ $\quad -B = (1, 1010101)_{1cns}$ $-(42)_{10}$

A + B

	0	1	0	1	0	1	0	1
+	0	0	1	0	1	0	1	0
	0	1	1	1	1	1	1	1

Result: $(0, 1111111)_{1cns} = +(127)_{10}$

A - B

	0	1	0	1	0	1	0	1
+	1	1	0	1	0	1	0	1
(1)	0	0	1	0	1	0	1	0
							+	1
	0	0	1	0	1	0	1	1

(Carry added in)

Result: $(0, 0101011)_{1cns} = +(43)_{10}$

-A+B

	1	0	1	0	1	0	1	0
+	0	0	1	0	1	0	1	0
	1	1	0	1	0	1	0	0

Result: $(1, 1010100)_{1cns} = -(43)_{10}$

-A-B

	1	0	1	0	1	0	1	0
+	1	1	0	1	0	1	0	1
(1)	0	1	1	1	1	1	1	1
							+	1
	1	0	0	0	0	0	0	0

(Carry added in)

Result: $(1, 0000000)_{1cns} = -(127)_{10}$

(d) 10000000, 01111111

$A = (1, 0000000)_{1cns}$ $-(127)_{10}$ $\quad B = (0, 1111111)_{1cns}$ $+(127)_{10}$

$-A = (0, 1111111)_{1cns}$ $+(127)_{10}$ $\quad -B = (1, 0000000)_{1cns}$ $-(127)_{10}$

A + B

	1	0	0	0	0	0	0	0
+	0	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1

Result: $(1, 1111111)_{1cns} = -(0)_{10}$

A - B

	1	0	0	0	0	0	0	0
+	1	0	0	0	0	0	0	0
(1)	0	0	0	0	0	0	0	0
							+	1
	0	0	0	0	0	0	0	1

(Carry added in)

Overflow condition! Invalid result

-A+B

	0	1	1	1	1	1	1	1
+	0	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	0

Overflow condition! Invalid result

-A-B

	0	1	1	1	1	1	1	1
+	1	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1

Result: $(1, 1111111)_{1cns} = -(0)_{10}$

1.12 Show how a 16-bit computer using a two's complement number system would perform the following computations.

(a) $(16850)_{10} + (2925)_{10} = (?)_{10}$

(b) $(16850)_{10} - (2925)_{10} = (?)_{10}$

(c) $(2925)_{10} - (16850)_{10} = (?)_{10}$

(c) $-(2925)_{10} - (16850)_{10} = (?)_{10}$

$(16, 850)_{10} = (0, 100000111010010)_{2cns}$

$-(16, 850)_{10} = (1, 011111000101110)_{2cns}$

$(2, 925)_{10} = (0, 000101101101101)_{2cns}$

$-(2, 925)_{10} = (1, 111010010010011)_{2cns}$

(a) $(16, 850)_{10} + (2, 925)_{10}$

	0	1	0	0	0	0	0	1	1	1	0	1	0	0	1	0
+	0	0	0	0	1	0	1	1	0	1	1	0	1	1	0	1
	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1

Result: $(0, 100110100111111)_{2cns} = (19, 775)_{10}$

(b) $(16, 850)_{10} - (2, 925)_{10}$

	0	1	0	0	0	0	0	1	1	1	0	1	0	0	1	0
+	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	1
(1)	0	0	1	1	0	1	1	0	0	1	1	0	0	1	0	1

Result: $(0, 011011001100101)_{2cns} = (13, 925)_{10}$

(c) $(2, 925)_{10} - (16, 850)_{10}$

	0	0	0	0	1	0	1	1	0	1	1	0	1	1	0	1
+	1	0	1	1	1	1	1	0	0	0	1	0	1	1	1	0
	1	1	0	0	1	0	0	1	1	0	0	1	1	0	1	1

Result: $(1, 100100110011011)_{2cns} = -(13, 925)_{10}$

(d) $-(2, 925)_{10} - (16, 850)_{10}$

	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	1
+	1	0	1	1	1	1	1	0	0	0	1	0	1	1	1	0
(1)	1	0	1	1	0	0	1	0	1	1	0	0	0	0	0	1

Result: $(1, 011001011000001)_{2cns} = -(19, 775)_{10}$

1.13 Show how each of the following numbers would be represented as floating-point numbers in IEEE 754-2008 Binary-32 and Binary-64 formats.

(a) 1101

Binary-32 Format (sign bit, 8 exponent bits, 23 mantissa bits):

$(1101)_2 = (1.101)_2 \times 2^3$

$M = +(1.101)_{2sm}$ $S_M = 0$ (leading 1 of M to be suppressed)

$E = +(3)_{10}$ bias + 3 = 127+3 = 130 = $(10000010)_{excess\text{-}127}$

1011 = $(0\ 10000010\ 10100000000000000000000)_{fp}$

Binary-64 Format (sign bit, 11 exponent bits, 53 mantissa bits):

Bias+3 = 1023+3 = 1026 = $(10000000010)_{excess\text{-}1023}$

1101 = $(0\ 10000000010\ 1010000000000000000000000000000000000000000000000000)_{fp}$

(b) 101110

Binary-32

$(101110)_2 = (1.01110)_2 \times 2^5$

$M = +(1.01110)_{2sm}$ $S_M = 0$ (leading 1 of M to be suppressed)

$E = +(5)_{10}$ bias + 5 = 127+5 = 132 = $(10000100)_{excess\text{-}127}$

1011 = $(0\ 10000100\ 01110000000000000000000)_{fp}$

Binary-64

Bias+3 = 1023+5 = 1028 = $(10000000100)_{excess\text{-}1023}$

1101 = $(0\ 10000000100\ 0111000000000000000000000000000000000000000000000000)_{fp}$

(c) 0.101

Binary-32

$(0.101)_2 = (1.01)_2 \times 2^{-1}$

$M = +(1.01)_{2sm}$ $S_M = 0$ (leading 1 of M to be suppressed)

$E = -(1)_{10}$ bias – 1 = 127 – 1 = 126 = $(01111110)_{excess\text{-}127}$

0.101 = $(0\ 01111110\ 01000000000000000000000)_{fp}$

Binary-64

Bias – 1 = 1023 – 1 = 1021 = $(01111111110)_{excess\text{-}1023}$

0.101 = $(0\ 01111111110\ 0100000000000000000000000000000000000000000000000000)_{fp}$

(d) 0.01101

Binary-32

$(0.\ 01101)_2 = (1.101)_2 \times 2^{-2}$

$M = +(1.101)_{2sm}$ $S_M = 0$ (leading 1 of M to be suppressed)

$E = -(2)_{10}$ bias – 2 = 127 – 2 = 125 = $(01111101)_{excess\text{-}127}$

0. 01101 = $(0\ 01111101\ 10100000000000000000000)_{fp}$

Binary-64

Bias – 2 = 1023 – 2 = 1021 = $(01111111101)_{excess\text{-}1023}$

0. 01101 =

$(0\ 01111111101\ 1010000000000000000000000000000000000000000000000000)_{fp}$

(e) 10101.11

Binary-32

$(10101.11)_2 = (1.010111)_2 \times 2^4$

$M = +(1.010111)_{2sm}$ $S_M = 0$ (leading 1 of M to be suppressed)

$E = +(2)_{10}$ bias + 2 = 127 + 2 = 129 = $(10000001)_{excess\text{-}127}$

$10101.11 = (0\ 01111101\ 01011100000000000000000)_{fp}$

Binary-64

Bias + 2 = 1023 + 2 = 1025 = $(10000000001)_{excess\text{-}1023}$

10101.11 =

$(0\ 10000000001\ 0101110000000000000000000000000000000000000000000000)_{fp}$

(f) 10110110.001

Binary-32

$(10110110.001)_2 = (1.0110110001)_2 \times 2^7$

$M = +(1.0110110001)_{2sm}$ $S_M = 0$ (leading 1 of M to be suppressed)

$E = +(7)_{10}$ bias + 7 = 127 + 7 = 134 = $(10000110)_{excess\text{-}127}$

$10110110.001 = (0\ 10000110\ 01101100010000000000000)_{fp}$

Binary-64

Bias + 7 = 1023 + 7 = 1030 = $(10000000110)_{excess\text{-}1023}$

10110110.001 =

$(0\ 10000000110\ 0110110001000000000000000000000000000000000000000000)_{fp}$

1.14 Encode each of the following numbers in BCD.

(a) **39** BCD: 0011 1001

(b) **1950** BCD: 0001 1001 0101 0000

(c) **94704** BCD: 1001 0100 0111 0000 0100

(d) **625** BCD: 0110 0010 0101 0101

1.15 Encode each of the following character strings in ASCII code. Represent the encoded strings by hexadecimal numbers.

(a) **1980**

	1	9	8	0
ASCII:	31	39	38	30

(b) **A = b + C**

	A	=	b	+	C
ASCII:	41	20 3D 20	62	20 2B 20	43

(e) COMPUTER ENGINEERING

	C	O	M	P	U	T	E	R		E	N	G	I	N	E	E	R	I	N	G
ASCII:	43	4F	4D	50	55	54	45	52	20	45	4E	47	49	4E	45	45	52	49	4E	47

(d) **The End.**

	T	h	e		E	n	d	.
ASCII:	54	68	65	20	45	6E	64	2E

1.16 Repeat problem 1.15 assuming that a parity bit is concatenated to the ASCII codes to yield an odd parity code.

(a) 1980

	1	9	8	0
ASCII:	31	B9	38	B0

(b) A = b + C

	A	=	b	+	C
ASCII:	C1 20	3D 20	62 20	AB 20	43

(c) COMPUTER ENGINEERING

	C	O	M	P	U	T	E	R		E	N	G	I	N	E	E	R	I	N	G
ASCII:	43	4F	CD	D0	D5	54	45	52	20	45	CE	C7	49	CE	45	45	52	49	CE	C7

(d) The End.

	T	h	e		E	n	d	.
ASCII:	54	68	E5	20	45	6E	64	AE

1.17 Define a 4-bit code for representing the decimal digits that has the property that the code words for any two digits whose difference is 1 differ in only one bit position and that this property also holds for the digits 0 and 9.

0-	0000	5-	0111
1-	0001	6-	0101
2-	0011	7-	0100
3-	0010	8-	1100
4-	0110	9-	1000

1.18 How many bit errors can be detected in a two-out-of-five code? How many errors, if any, can be corrected in a two-out-of-five code? Prove your answers mathematically.

To correct t errors and detect s errors: $2t + s + 1 \leq d_{min}$.
For t = 0, $s = 1$: $2t + s + 1 = 2 = d_{min}$. Therefore single errors can be detected.
For t = 1, $s = 0$: $2t + s + 1 = 3 > d_{min}$. Therefore no error correction.

1.19 Examine the Gray-coded disk of Fig 1.5. Suppose the display lights give the following indications: A is off, B is on, C is on, and D is flickering on and off. Locate the position of the disk by sector numbers.

A = 0, B = 1, C = 1, D = 0 => *sector*4

A = 0, B = 1, C = 1, D = 1 => *sector*5

1.20 For the nine-track magnetic tape of Fig 1.8, the following 8-bit messages are to be recorded. Determine the parity bit to establish odd parity for each message.

(a) **P10111010** 5 1-bits => P=0
(b) **P00111000** 3 1-bits => P=0
(c) **P10011001** 4 1-bits => P=1
(d) **P01011010** 4 1-bits => P=1

1.21 Assume that the following are code words from an eight-bit even parity code. Which code words contain errors?

(a) **00000000** 0 1-bits => even parity => no error
(b) **00110100** 1 1-bits => odd parity => error
(c) **01010101** 4 1-bits => even parity => no error
(d) **10111010** 5 1-bits => odd parity => error

1.22 Develop a table that shows how check bits are defined across the information bits for Hamming Code 2 in Table 1.13.

Information Words ($i_3i_2i_1i_0$)	**Hamming Code 2** ($i_3i_2i_1i_0c_3c_2c_1c_0$)
0000	00000000
0001	00011011
010	00101101
0011	00110110
0100	01001110
0101	01010101
0110	01100011
0111	01111000
1000	10000111
1001	10011100
1010	10101010
1011	10110001
1100	11001001
1101	11010010
1110	11100100
1111	11111111

Check bits produce even parity over a subset of information bits. Looking at information bits containing a single 1 bit, if a check bit is 1, then that information bit must be included in that check bit.

- Information Word 0001: $i_0 = 1$ sets c_3, c_1, and c_0.
- Information Word 0010: $i_1 = 1$ sets c_3, c_2, and c_0.
- Information Word 0100: $i_2 = 1$ sets c_3, c_2, and c_1.
- Information Word 1000: $i_3 = 1$ sets c_2, c_1, and c_0.

Therefore, these check bits produce even parity over the following sets of information bits.

c_3 : i_2, i_1, i_0
c_2 : i_3, i_2, i_1
c_1 : i_3, i_2, i_0
c_0 : i_3, i_1, i_0

1.23 Develop a syndrome table for Hamming Code 2 that covers the cases of no error, single errors, and double errors that involved information bit i_3.

Each syndrome is the product of Hamming matrix *H* and the transpose of error vector e^T. For example, for Hamming Code 2 and error vector 10000001, which contains errors in bits i_3 and c_0, the syndrome is:

$$s = He^T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = [0 \quad 1 \quad 1 \quad 0]$$

The remaining syndromes, as listed in the following table, are determined by multiplying this Hamming matrix *H* by the transpose, e^T, of each of the 16 error patterns that contain no errors, a single error, or a double error that includes bit i_3.

Error Pattern	Syndrome	Meaning
00000000	0000	No error
00000001	0001	Error in c_0
00000010	0010	Error in c_1
00000100	0100	Error in c_2
00001000	1000	Error in c_3
00010000	1011	Error in i_0
00100000	1101	Error in i_1
01000000	1110	Error in i_2
10000000	0111	Error in i_3
10000001	0110	Errors in i_3 and c_0
10000010	0101	Errors in i_3 and c_1
10000100	0011	Errors in i_3 and c_2
10001000	1111	Errors in i_3 and c_3
10010000	1100	Errors in i_3 and i_0
10100000	1010	Errors in i_3 and i_1
11000000	1001	Errors in i_3 and i_2

1.24 Find examples that show Hamming Code 2 is not double-error correcting nor triple-error detecting.

Double-error correcting:
Since d_{min} = 4, a double error can produce a non-code word that is distance 2 from two or more valid code words.

Ex. Errors in bits c_1 and c_0 of code word 00000000 result in non-code word 00000011, which is distance 2 from code words 00000000, 00011011, 01100011, 10000111. Therefore, we cannot determine which of these four code words contains the double error that produced 00000011, so the error is not correctable.

Triple-error detecting:
Since d_{min} = 4, a triple error can produce a non-code word that is distance 1 from another code word.

Ex. Errors in bits c_3, c_2 and c_1 of code word 00000000 result in non-code word 00001110, which is distance 3 from the original code word, but distance 1 from code word 01001110. Therefore, we cannot determine whether the erroneous non-code word resulted from a single error in code word 01001110 or a triple error in code word 00000000.

1.25 The following defines the check bits of a Hamming code for encoding eight-bit information words ($i_7i_6i_5i_4i_3i_2i_1i_0$).

c_3: i_7,i_6,i_5,i_3,i_2 $\quad$ c_1: i_7,i_5,i_4,i_2,i_0
c_2: i_7,i_6,i_4,i_3,i_1 $\quad$ c_0: i_6,i_5,i_4,i_1,i_0

(a) How many information words are there?
8 information bits => 2^8 = 256 information words.

(b) How many code words?
256 (one code word per information word).

(c) Encode information words 00000000, 01010101, 10101010, 11111111.

00000000: All check bits will be 0, so code word = 00000000 0000

01010101:

$$c_3 = i_7 \oplus i_6 \oplus i_5 \oplus i_3 \oplus i_2 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0$$
$$c_2 = i_7 \oplus i_6 \oplus i_4 \oplus i_3 \oplus i_1 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 0$$
$$c_1 = i_7 \oplus i_5 \oplus i_4 \oplus i_2 \oplus i_0 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$c_0 = i_6 \oplus i_5 \oplus i_4 \oplus i_1 \oplus i_0 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1$$

Code word = 01010101 0011

10101010:

$$c_3 = i_7 \oplus i_6 \oplus i_5 \oplus i_3 \oplus i_2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$
$$c_2 = i_7 \oplus i_6 \oplus i_4 \oplus i_3 \oplus i_1 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$c_1 = i_7 \oplus i_5 \oplus i_4 \oplus i_2 \oplus i_0 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$
$$c_0 = i_6 \oplus i_5 \oplus i_4 \oplus i_1 \oplus i_0 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

Code word = 10101010 1100

11111111:

$$c_3 = i_7 \oplus i_6 \oplus i_5 \oplus i_3 \oplus i_2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$c_2 = i_7 \oplus i_6 \oplus i_4 \oplus i_3 \oplus i_1 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$c_1 = i_7 \oplus i_5 \oplus i_4 \oplus i_2 \oplus i_0 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$c_0 = i_6 \oplus i_5 \oplus i_4 \oplus i_1 \oplus i_0 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$

Code word = 11111111 1111

(d) What is the error detection and correction properties of this code?

Since information bits i_1 and i_0 are each included in only 2 check bits, the minimum weight of a code word is 3, and minimum distance d_{min} = 3. Therefore, this code is capable of correcting a single error, but cannot detect double errors.

Note that this code produces a syndrome of 4 bits, and therefore 16 unique syndromes, which is sufficient only identifying the no error case or the 12 possible single errors, as shown in the next part.

(e) Develop a syndrome table that covers no error and single-error cases.

Error Pattern	Syndrome	Meaning
000000000000	0000	No error
000000000001	0001	Error in c_0
000000000010	0010	Error in c_1
000000000100	0100	Error in c_2
000000001000	1000	Error in c_3
000000010000	0011	Error in i_0
000000100000	0101	Error in i_1
000001000000	1010	Error in i_2
000010000000	1100	Error in i_3
000100000000	0111	Error in i_4
001000000000	1011	Error in i_5
010000000000	1101	Error in i_6
100000000000	1110	Error in i_7

Chapter 2 – Logic Circuits and Boolean Algebra

2.1 Construct truth tables for the following AND and OR gates.

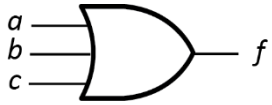
(a)

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b)

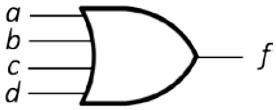
a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(c)



a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

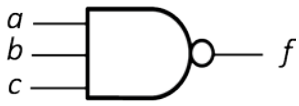
(d)



a	b	c	d	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

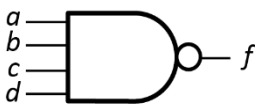
2.2 Construct truth tables for the following NAND and NOR gates.

(a)



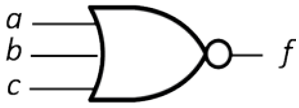
a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(b)



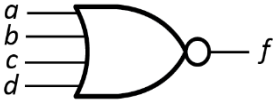
a	b	c	d	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

(c)



a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

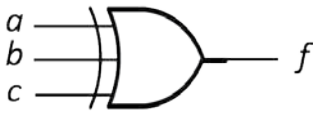
(d)



a	b	c	d	f
0	0	0	0	1
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	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

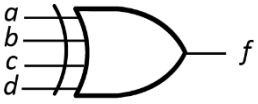
2.3 Construct truth tables for the following XOR and XNOR gates.

(a)



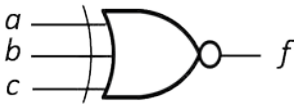
a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(b)



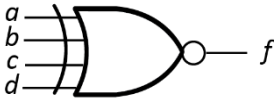
a	b	c	d	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

(c)



a	b	c	f
0	0	0	1
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	0

(d)



a	b	c	d	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

2.4 Prove that the 3-variable *XOR* function is equivalent to the *sum* function of a full-adder.

3-variable XOR function: (odd number of 1s)

a	*b*	*c*	*f*
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

Full-adder sum function: ($a + b + c$ = 01 or 11)

a	*b*	*c*	*f*
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

Identical truth tables => equivalent functions.

2.5 Prove that the 3-variable *majority* function is equivalent to the *carry* function of a full-adder.

3-variable majority function: (2 or more 1-bits)

a	*b*	*c*	*f*
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

Full-adder carry function: ($a + b + c > 1$)

a	*b*	*c*	*f*
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

Identical truth tables => equivalent functions.

2.6 Prove that a 3-variable *XOR* function is equivalent to a 3-variable *odd parity* function.

3-variable XOR (odd # 1s):

a	*b*	*c*	*f*
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

Full-adder sum ($a + b + c$ = 01 or 11)

a	*b*	*c*	*f*
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

Identical truth tables => equivalent functions.

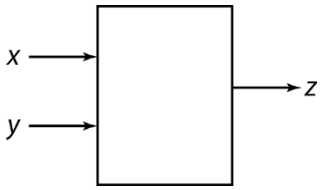
2.7 Construct a truth table for the 3-variable *even parity* function.

a	*b*	*c*	*f*
0	0	0	1
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	0

f(*a*,*b*,*c*) = 1 if even number of 1-bits (0 or 2 1-bits)

2.8 Draw a block (I/O) diagram and construct a truth table for the function defined by the following Verilog model.

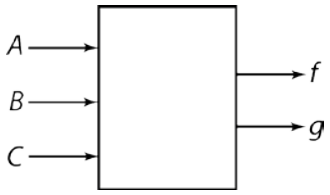
```
module problem_2_8 (x, y, z);
        input x, y;
output z;
wire a, b, c;
        nand (a, x, y);
        nand (b, a, x);
        nand (c, a, y);
        nand (z, b, c);
endmodule
```



x	y	$a = \overline{xy}$	$b = \overline{ax}$	$c = \overline{ay}$	$z = \overline{bc}$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

2.9 Repeat problem 2.8 for the following model.

```
module problem_2_9 (A,B,C,f,g);
        input A,B,C;
        output f,g;
        assign f = ~A&~B&C | B&~C | A&B;
        assign g = ~A&C | A&~B&C | A&B&~C;
endmodule
```

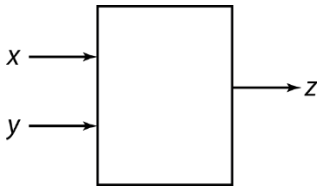


A	B	C	$\bar{A}\bar{B}C$	$B\bar{C}$	AB	$f = \bar{A}\bar{B}C + B\bar{C} + AB$	$\bar{A}C$	$A\bar{B}C$	$AB\bar{C}$	$g = \bar{A}C + AB\bar{C} + AB\bar{C}$
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0	0	1
0	1	0	0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	1	0	0	1
1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	1	0	1
1	1	0	0	1	1	1	0	0	1	1
1	1	1	0	0	1	1	0	0	0	0

2.10 Draw a block (I/O) diagram and construct a truth table for the function defined by the following VHDL model.

```
entity prob2_10 is
        port (x, y:  in bit;  z:  out bit);
end prob2_10;

architecture structure of prob2_10 is
signal  a, b, c:  bit;
begin
        process (x,y)
                begin
                a <= x nor y;
                b <= x nor a;
                c <= y nor a;
                z <= a nor b;
        end process;
end structure;
```



x	y	$a = \overline{x + y}$	$b = \overline{x + a}$	$c = \overline{y + a}$	$z = \overline{a + b}$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	1
1	1	0	0	0	1