CHAPTER 4

Operations On Bits

Review Questions

1. Arithmetic operations involve interpreting the bits as numbers and doing arithmetic operations on those numbers. Logical operations involve interpreting the bits as logical values (true or false) and doing operations on those values.
2. Multiplication can be looked at as repetitive addition. Y x Z can be accomplished by adding Z number of Ys together.
3. The leftmost carry is discarded.
4. The bit allocation can equal 1. This data type could represent a logical value.
5. Overflow happens when the result of an arithmetic operation is outside the range of possible values for the bit allocation being used.
6. The decimal point of the number with the smaller exponent is shifted to the left until the exponents are equal.
7. A unary operation takes a single operand. A binary operation takes two operands.
8. The logical binary operations are: AND, OR, and XOR.
9. A truth table lists all of the possible input combinations with the resulting output.
10. NOT inverts values: it change true to false and false to true.
11. The result of an AND operator is true when both of the operands are true.
12. The result of an OR operator is true when either of the operands is true.
13. The result of an XOR operator is true when both of the operands are different.
14. The inherent rule of the AND operator is that if one of the operands is false, the result is false.
15. The inherent rule of the OR operator is that if one of the operands is true, the result is true.
16. The inherent rule of the XOR operator is that if one of the operands is true, the result will be the inverse of the other operand.
17. The OR operator can be used to set bits. Set the desired positions in the mask to 1.
18. The AND operator can be used to clear bits. Set the desired positions in the mask to 0.

19. The XOR operator can be used to invert bits. Set the desired positions in the mask to 1.

Multiple-Choice Questions

20. c
21. d
22. c
23. c
24. c
25. b
26. d
27. b
28. a
29. c
30. d
31. c
32. d
33. c
34. b
35. a
36. a
37. c
38. a
39. b

Exercises

40.
   a. 00010011 + 00010111 = 00101010 = 42
   b. 00010011 - 00010111 = 000010011 + (-00010111) = 00010011 + 11101001 = 11111100 = -00000100 = -4
   c. (-00010011) + 00010111 = 11101101 + 00010111 = 00000100 = 4
   d. (-00010011) - 00010111 = (-00010011) + (-00010111) = 11101101 + 11101001 = 11010110 = -00101010 = -42

41.
   a. 00000000 10100001 + 00000011 11111111 = 00000100 10100000 = 1184
   b. 00000000 10100001 - 00000011 11111111 = 00000000 10100001 +
   (-00000011 11111111) = 00000000 10100001 + 11111100 00000001 = 11111100 10100010 = -00000011 01011110 = -862
   c. (-00000000 10100001) + 00000011 11111111 = 11111111 01011111 + 00000011 11111111 = 00000011 01011110 = 862

d. (-00000000 10100001) - 00000011 11111111 = (-00000000 10100001) + (-00000011 11111111) = 11111111 01011111 + 11111100 00000001 = 11111011 01100000 = -00000100 10100000 = -1184

42.

a. No overflow

b. No overflow

c. No overflow

d. No overflow

43.

a. Overflow

b. No overflow

c. No overflow

d. Overflow

44.

a. x012A + x0E27 = 00000001 00101010 + 00001110 00100111 = 00001111 01010001 = x0F51

b. x712A + x9E00 = 01110001 00101010 + 10011111 00000000 = 00010000 00101010 = x102A

c. x8011 + x0001 = 10000000 00010001 + 00000000 00000001 = 10000000 00010010 = x8012

d. xE12A + x9E27 = 11100001 00101010 + 10011110 00100111 = 01111111 01010001 = x7F51

45.

a.

34.075 + 23.12 =

100010.00010011 + 10111.0001111 =

2^5 x 1.0001000010011 + 2^4 x 1.01110001111 =

2^5 x (1.0001000010011 + 0.1011100011110) =

2^5 x 1.1100100110001 = 111001.00110001 = 57.19140625

b.

-12.00067 + 451.00 = 451.00 - 12.00067 =

111000011.0 - 1100.00000000001010111 =

2^8 x 1.11000011 - 2^3 x 1.10000000000001010111 =

2^8 x 1.11000011 - 2^8 x 0.0000110000000000001010111 =

2^8 x (1.1100001100000000000000000 + 2^8 x l.1111001111111111110101001) =
2^8 1.1011011011111111110101001 = 110110110.11111111110101001 = 438.99933

c.

33.677 - 0.00056 =

100001.1010110101 - .00000000001001001 =

2^5 x 1.000011010110101 - 2^{-12} x 1.001001 =

2^5 x 1.000011010110101 - 2^5 x 0.000000000000000001001001 =

2^5 x (1.000011010110101 + 1.111111111111111110110111) =

2^5 x 1.00001101011010010110111 = 100001.10101101001011 0111 =
33.6764793395996 09375

d.

-344.23 - 123.8902 = -(344.23 + 123.8902) =

-(101011000.0011101 + 1111011.11100011111001) =

-(2^8 x 1.010110000011101 + 2^6 x 1.11101111100011111001) =

-(2^8 x (1.010110000011101 + 0.0111101111100011111001)) =

-(2^8 x (1.1101010000011101111001)) = -111010100.00011101111001 =
-468.11676025390625

46.

a.

23.125 + 12.45 =

10111.001 + 1100.0111001 =

2^4 x 1.0111001 + 2^3 x 1.1000111001 =

2^4 x (1.0111001 + 0.11000111001) =

2^4 x 10.00111001001 = 100011.1001001 = 34.4703125

b.

0.234 - 7.192 = -(7.192 - 0.234) =

-(111.0011000100 - .0011101111) =

-(2^2 x 1.110011000100 - 2^{-3} x 1.1101111) =

-(2^2 x (1.110011000100 - 0.000011101111)) =

-(2^2 x (1.110011000100 + 1.111100010001)) =

-(2^2 x (1.101111010101)) = -110.1111010101 = -6.9580078125

c.

-0.345 + 45.123 = 45.123 - 0.345 =

101101.0001111101 - .0101100001 =

2^5 x 1.011010001111101 - 2^{-2} x 1.01100001 =

2^5 x (1.011010001111101 - 0.000000101100001) =

2^5 x (1.011010001111101 + 1.111111010011111) =

2^5 x 1.011001100011100 = 101100.1100011100 = 44.7773437500

d.

-.234 - 5.345 = -(.234 + 5.345) =

-(.0011101111 + 101.0101100001) =

-(2^{-3} x 1.1101111 + 2^2 x 1.010101100001) =

-(2^2 x (0.000011101111 + 1.010101100001)) =

-(2^2 x 1.011001010000) = -101.100101 = -5.578125

47.

a. Overflow can occur.

b. Overflow can never occur because the absolute value of the result must be less than or equal to the larger of the absolute values of the operands. To put it another way, the result will be closer to zero than the operand that is farthest from zero.
c. Overflow can occur.
d. Overflow can never occur because by subtracting a negative number, you are essentially adding a positive number and we again have the case where we are adding a positive and negative number.

48.
   a. NOT x99 = NOT 10011001 = 01100110
   b. NOT xFF = NOT 11111111 = 00000000
   c. NOT x00 = NOT 00000000 = 11111111
   d. NOT x01 = NOT 00000001 = 11111110

49.
   a. x99 AND x99 = 10011001 AND 10011001 = 10011001
   b. x99 AND x00 = 10011001 AND 00000000 = 00000000
   c. x99 AND xFF = 10011001 AND 11111111 = 10011001
   d. xFF AND xFF = 11111111 AND 11111111 = 11111111

50.
   a. x99 OR x99 = 10011001 OR 10011001 = 10011001
   b. x99 OR x00 = 10011001 OR 00000000 = 10011001
   c. x99 OR xFF = 10011001 OR 11111111 = 11111111
   d. xFF OR xFF = 11111111 OR 11111111 = 11111111

51.
   a. x99 XOR x99 = 10011001 XOR 10011001 = 00000000
   b. x99 XOR x00 = 10011001 XOR 00000000 = 10011001
   c. x99 XOR xFF = 10011001 XOR 11111111 = 01100110
   d. xFF XOR xFF = 11111111 XOR 11111111 = 00000000

52.
   a. NOT (x99 OR x99) = NOT (10011001 OR 10011001) = NOT 10011001 = 01100110
   b. x99 OR (NOT x00) = 10011001 OR 11111111 = 11111111
   c. (x99 AND x33) OR (x00 AND xFF) = (10011001 AND 00110011) OR (00000000 AND 11111111) = 00010001 OR 00000000 = 00010001
   d. (x99 OR x33) AND (x00 OR xFF) = (10011001 OR 00110011) AND (00000000 OR 11111111) = 10111011 AND 11111111 = 10111011

53. Mask: 00001111
    Operation: 00001111 AND XXXXXXXX = 0000XXXX

54. Mask: 00001111
    Operation: 00001111 OR XXXXXXXX = XXXX1111

55. Mask: 11000111
Operation: 11000111 XOR XXXXXXXX = YYXXXYYY
56. Mask 1: 00011111
Mask 2: 00000011
Operation: (00011111 AND XXXXXXXX) OR 00000011 = 000XXX11
57. "abcdefgh" right shifted by 2 becomes '00abcdef', which divides the unsigned number by 4.
58. "abcdefgh" left shifted by 3 becomes 'defgh000', which multiplies the number by 8. Overflow will occur if any of the 3 leftmost bits are not zero.
59.

Mask: 00000001

Number: abcdefgh

Operations:

Step 1: Shift to the right by 3, which gives us '000abcde'.

Step 2: 000abcde AND 00000001 = 0000000e (if the result is 1, the bit was set)

Step 3: Shift once more to the right, which gives us '0000abcd'.

Step 4: 0000abcd AND 00000001 = 0000000d (if the result is 1, the bit was set)