CHAPTER 3

Number Representation

Review Questions

1. Repetitive division by 2 is used, entering the remainder (0 or 1) to the left of any previous remainders.
2. Multiply each binary digit by its weight and combine the results.
3. 2
4. 10
5. Sign-and-magnitude, one's complement, and two's complement.
6. The “maximum unsigned integer” is the largest number that can be stored in an integer location. For a given bit allocation, n, the largest storable unsigned integer would be 2^n - 1.
7. Bit allocation is the number of bits used to represent an integer.
8. The decimal 256 would require at least 9 bits for storage.
9. Unsigned integers are used for counting, addressing, or anywhere negative numbers are not used.
10. The largest positive number storable in a bit allocation of 8 bits using sign-and-magnitude representation is 127; therefore, overflow would occur.
11. The representation of positive numbers in sign-and-magnitude, one's complement, and two's complement are all identical.
12. Negative numbers in all three representations will all start with '1' (the left-most bit will be '1'). The rest of the numbers will vary depending on the representation.
13. There are 2 values for zero in both sign-and-magnitude (00000000 and 10000000) and one's complement (00000000 and 11111111) representations. Two's complement representation has only 1 value for zero (00000000).
14. For a bit allocation of N bits: The range of numbers storable in sign-and-magnitude is from $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$; the range of numbers storable in one's complement is from $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$; the range of numbers storable in two's complement is from $-(2^{N-1})$ to $+(2^{N-1} - 1)$.

15. In each of these representations, a '1' in the leftmost bit position indicates a negative number.
16. Excess_X is used primarily to store the exponential value of a fraction. 'X' represents the magic number being used: normally ($2^{N - 1}$) or ($2^{N - 1} - 1$) where "N" is the bit allocation.
17. Normalization is necessary to make calculations easier.
18. The mantissa is the bit sequence to the right of the decimal point after normalization.
19. The computer must store the sign of the number, the exponent used during normalization, and the mantissa.

Multiple-Choice Questions

20. c
21. a
22. d
23. d
24. b
25. c
26. d
27. d
28. d
29. d
30. d
31. c
32. b
33. d
34. d
35. a
36. c
37. b
38. c
39. b
40. c
41. b
42. d
43. d
44. c
45. c
46. b

Exercises

47.
   a. 00010111
   b. 01111001
   c. 00100010
   d. 101010110 (overflow)

48.
   a. 00000000 00101001
   b. 00000001 10011011
   c. 00000100 11010010
   d. 00000001 01010110

49.
   a. 00100000
   b. 11100101
   c. 00111000
   d. 10000001 (overflow)

50.
   a. 00000000 10001110
   b. 10000000 10110100
   c. 00000010 00110000
   d. 00001001 10011000

51.
   a. 01111010
   b. 10110000
   c. 11111111
   d. 10000001 (overflow)

52.
   a. 00000000 10100010
   b. 11111111 10010001
   c. 00001010 00000000
   d. 00101111 01011011

53.
   a. 11110100
   b. 10011011
   c. 00111000
   d. 10001110 (overflow)

54.
   a. 00000000 01100110
   b. 11111111 01001101

c. 00000010 00010110
d. 11110010 01101000 (overflow)

55.
a. 107
b. 148
c. 6
d. 80

56.
a. 123
b. -52
c. 99
d. -80

57.
a. 99
b. -123
c. 115
d. -63

58.
a. 119
b. -4
c. 116
d. -50

59.
a. 11110111
b. 01111100
c. 11110111
d. 01001110

60.
a. 10001000
b. 00000011
c. 10001000
d. 00110001

61.
a. 10001001
b. 00000100
c. 10001001
d. 00110010

62.
a. Original method: 10001001 New method: 10001000 + 00000001 = 10001001
b. Original method: 00000100 New method: 00000011 + 00000001 = 00000100

c. Original method: 10001100 New method: 10001011 + 00000001 = 10001100

d. Original method: 00110010 New method: 00110001 + 00000001 = 00110010

63.

a. 01110111 => 10001000 => 01110111

b. 11111100 => 00000011 => 11111100

c. 01110100 => 10001011 => 01110100

d. 11001110 => 00110001 => 11001110

64.

a. 01110111 => 10001001 => 01110111

b. 11111100 => 00000100 => 11111100

c. 01110100 => 10001100 => 01110100

d. 11001110 => 00110010 => 11001110

65.

a. 9743

b. 9887

c. 9874

d. 9888

66. We can represent numbers from -99 (900) to +99 (099). We can determine the sign of the number by the leftmost digit. If the leftmost digit is a '9', the number is negative; if the leftmost digit is a '0', the number is positive. We do have 2 zeros: +0 (000) and -1 (999).

67.

a. Already normalized

b. 2^5 x 1.111111

c. 2^0 x 1.01110011

d. 2^0 x 1.0110100000110011000

68.

a. **1** 01111111 10001000000000000000000

b. **0** 10000010 11111100000000000000000

c. **0** 01111011 01110011000000000000000

d. **1** 01111010 01101000000000000000000

69.

a. 1**01111111111**10001000000000000000000000000000000000000000000000000000

b. 0**10000000010**11111100000000000000000000000000000000000000000000000000

c. 0**01111111101**10111001100000000000000000000000000000000000000000000000

d. 1**01111111101**00110100000000000000000000000000000000000000000000000000

70.

a. 11.101 or 2^1 x 1.1101

b. 1100.00011 or 2^3 x 1.10000011

c. 100.001101 or 2^2 x 1.00001101

d. 1100.0000101 or 2^3 x 1.1000000101

71.

a. 1/8 + 1/16 or 3/16 (.001 + .0001 or .0011)

b. 1/2 + 1/8 + 1/64 or 41/64 (.1 + .001 + .000001 or .101001)

c. 1/4 + 1/8 + 1/32 or 13/32 (.01 + .001 + .00001 or .01101)

d. 1/4 + 1/8 or 3/8 (.01 + .001 or .011)

72.

a. 111.0011 or 2^2 x 1.110011

b. 1100.101001 or 2^3 x 1.100101001

c. 1011.01101 or 2^3 x 1.01101101

d. 0.011 or 2^{-2} x 1.1

73.

a. 0 10000001 11001100000000000000000

b. 0 10000010 10010100100000000000000

c. 1 10000010 01101101000000000000000

d. 1 01111101 10000000000000000000000

74.

a.

```
  0000 0001 0010 1010 0000 0000
+ 0001 0010 1010 1010 1111 1111
-----------------------------------------
  0001 0011 1101 0100 1111 1111
 =>  0 00000000 00100111101010011111111
```

b.

```
  0000 0000 0000 0000 0000 0001 0001
+       1000  0010 0000 0000 0000 0000
-------------------------------------------------
  0000 1000 0010 0000 0000 0001 0001
=> 0 00000001 00000100000000000010001
```

c.

```
   1001 0001 0001 0001 0001 0001 0001
 +       0010 0001 0001 0001 0001 0001
   -----------------------------------------
   1001 0011 0010 0010 0010 0010 0010
  => 0 00010010 01100100010001000100010
```

d.

```
1111 0001 0001 0001 0001 0001 0001
+     0111 0111  0111 0111  0111 0111
-----------------------------------------
1111 1000 1000 1000 1000 1000 1000
=> 0 00011111 00010001000100010001000
```