

# HKN ECE 120 Midterm 1 Worksheet Solutions

---

## Binary Representations

### Problem 1

Write these conversions in decimal. Truncate if necessary.

- a. Convert  $100101_2$  to a 6-bit unsigned integer.

$$1 + 4 + 32 = 37_{10}$$

- b. Convert  $100101_2$  to a 6-bit signed magnitude integer.

$$-(1 + 4) = -5_{10}$$

- c. Convert  $100101_2$  to a 6-bit 2's complement integer.

$$1 + 4 + (-32) = -27_{10}$$

- d. Convert  $011101110_2$  to a 9-bit unsigned integer.

$$2 + 4 + 8 + 32 + 64 + 128 = 238_{10}$$

- e. Convert  $011101110_2$  to a 9-bit 2's complement integer.

$$\text{Same as above, } 2 + 4 + 8 + 32 + 64 + 128 = 238_{10}$$

- f. Convert  $100100101101_2$  to a 11-bit unsigned integer.

Since it must be 11 bits, the top bit is disregarded to get  $00100101101_2$ , which becomes  $1 + 4 + 8 + 32 + 256 = 301_{10}$

- g. Convert  $100100101101_2$  to a 9-bit 2's complement integer.

Since it must be 9 bits, the top 3 bits are disregarded to get  $100101101_2$ , which becomes  $1 + 4 + 8 + 32 + (-256) = -211_{10}$

- h. Convert  $00101101_2$  to a 12-bit unsigned integer.

Since it must be 12 bits unsigned, zero extend to the required length to get  $000000101101_2$ , which becomes  $1 + 4 + 8 + 32 = 45_{10}$

- i. Convert  $10111_2$  to a 16-bit 2's complement integer.

Since it must be 16 bits signed, sign extend to the required length to get  $1111111111110111_2$ , which becomes  $1 + 2 + 4 + 16 + 32 + 64 + 128 + 256 + 512 + 1024 + (-2048) = -9_{10}$

### Problem 2

Write these conversions in binary. Truncate if necessary.

- a. Convert  $51_{10}$  to a 8-bit unsigned integer.

$$51_{10} = 32 + 16 + 2 + 1 = 00110011_2$$

- b. Convert  $51_{10}$  to a 8-bit signed magnitude integer.  
**Same as above,  $51_{10} = 32 + 16 + 2 + 1 = 00110011_2$**
- c. Convert  $51_{10}$  to a 8-bit 2's complement integer.  
**Same as above,  $51_{10} = 32 + 16 + 2 + 1 = 00110011_2$**
- d. Convert  $-240_{10}$  to a 9-bit signed magnitude integer.  
 $-240_{10} = -1 * (128 + 64 + 32 + 16) = 111110000_2$
- e. Convert  $-240_{10}$  to a 9-bit 2's complement integer.  
 $-240_{10} = (-256) + 16 = 100010000_2$
- f. Convert  $1171_{10}$  to a 11-bit unsigned integer.  
 $1171_{10} = 1024 + 128 + 16 + 2 + 1 = 10010010011_2$
- g. Convert  $1171_{10}$  to a 11-bit 2's complement integer.
- h. Convert  $65_{10}$  to a 12-bit unsigned integer.  
 $65_{10} = 64 + 1 = 000001000001_2$
- i. Convert  $-23309_{10}$  to a 16-bit 2's complement integer.  
 $-23309_{10} = (-32768) + 8192 + 1024 + 128 + 64 + 32 + 16 + 2 + 1 = 1010010011110011_2$

## Other Representations

### Problem 1

Convert these binary values to hexadecimal.

- a. 0010101101010110  
 $0010 = x2, 1011 = xB, 0101 = x5, 0110 = x6$   
**So  $0010101101010110 = x2B56$**
- b. 1001010010001111  
 $1001 = x9, 0100 = x4, 1000 = x8, 1111 = xF$   
**So  $1011010010001111 = x948F$**
- c. 0011110000010010  
 $0011 = x3, 1100 = xC, 0001 = x1, 0010 = x2$   
**So  $0011110000010010 = x3C12$**
- d. 1011111011101111  
 $1011x = xB, 1110 = xE, 1110 = xE, 1111 = xF$   
**So  $1011111011101111 = xBEEF$**
- e. 1111000000001101  
 $1111 = xF, 0000 = x0, 0000 = x0, 1101 = xD$   
**So  $1111000000001101 = xFOOD$**

## Problem 2

Convert these hexadecimal values to binary.

a. x37A5  
 $x3 = 0011$ ,  $x7 = 0111$ ,  $xA = 1010$ ,  $x5 = 0101$   
So x37A5 = 001101110100101

b. x2009  
 $x2 = 0010$ ,  $x0 = 0000$ ,  $x0 = 0000$ ,  $x9 = 1001$   
So x2009 = 0010000000001001

c. x1F06  
 $x1 = 0001$ ,  $xF = 1111$ ,  $x0 = 0000$ ,  $x6 = 0110$   
So x1F06 = 0001111100000110

d. x2FFE  
 $x2 = 0010$ ,  $xF = 1111$ ,  $xF = 1111$ ,  $xE = 1110$   
So x2FFE = 0010111111111110

e. xDEADBEEF  
 $xD = 1101$ ,  $xE = 1110$ ,  $xA = 1010$ ,  $xD = 1101$ ,  $xB = 1011$ ,  $xE = 1110$ ,  $xE = 1110$ ,  $xF = 1111$   
So xDEADBEEF = 1101111010101101101111011101111

## Problem 3

Convert these hexadecimal values to ASCII.

a. x4A  
 $x4A = 'J'$

b. x2F  
 $x2F = '/'$

c. x0D  
 $x0D = 'CR'$  (carriage return)

d. x4045  
 $x40 = '@'$ ,  $x45 = 'E'$   
So x4045 = "@E"

e. x6E6F  
 $x6E = 'n'$ ,  $x6F = 'o'$   
So x6E6F = "no"

## Problem 4

Convert these ASCII characters to binary.

- a. 'i'  
**'i' = x69**
- b. '#'  
**'#' = x23**
- c. 'M'  
**'M' = x4D**
- d. '!'  
**'!' = x21**
- e. "bob"  
**'b' = x62, 'o' = x6F, 'b' = x62**  
**So "bob" = x626F62**

## Problem 5

True or False?

- a. An integer with 11 hexadecimal values is at most a 88-bit integer.  
**False,  $11 \times 4 = 44$  bits**
- b. The shortest hexadecimal string that we can encode any 69-bit unsigned integer into is 18 characters long.  
**True, 17 hex characters can only encode  $17 \times 4 = 68$  bits, so 18 hex characters are needed.**
- c. All uppercase letters in ASCII start with the binary string 0100.  
**False, uppercase letters start with 0100 or 0101**
- d. All lowercase letters in ASCII start with the binary string 011.  
**True**
- e. There is an ASCII character that directly corresponds to x8A.  
**False, ASCII characters only go up till x7F so x8A would actually become x0A**
- f. ASCII characters are usually stored as signed 8-bit integers.  
**True**
- g. The control characters in ASCII were originally used as special codes for teletypes, keyboards, and electrical telegraphs.  
**True**

## Binary Operations

### Problem 1

Perform the following operations.

a.  $1_2 \text{ AND } 0_2$

$$\mathbf{1_2 \text{ AND } 0_2 = 0_2}$$

b.  $1_2 \text{ OR } 0_2$

$$\mathbf{1_2 \text{ OR } 0_2 = 1_2}$$

c.  $10010010_2 \text{ AND } 01111011_2$

$$\mathbf{10010010_2 \text{ AND } 01111011_2 = 00010010_2}$$

d.  $001010_2 \text{ OR } 111101_2$

$$\mathbf{001010_2 \text{ OR } 111101_2 = 111111_2}$$

e.  $x8618 \text{ AND } x7507$

$$\mathbf{x8618 = 1000011000011000_2, x7507 = 0111010100000111_2}$$

$$\mathbf{\text{So } x8618 \text{ AND } x7507 = 0000010000010000_2 = x0410}$$

f.  $1_2 \text{ XOR } 1_2$

$$\mathbf{1_2 \text{ XOR } 1_2 = 0_2}$$

g.  $xCA09 \text{ XOR } x0990$

$$\mathbf{xCA09 = 1100101000001001, x0990 = 0000100110010000_2}$$

$$\mathbf{\text{So } xCA09 \text{ XOR } x0990 = 1100001110011001_2 = xC399}$$

h.  $\text{NOT } 1001110100110101_2$

$$\mathbf{\text{NOT } 1001110100110101_2 = 0110001011001010_2}$$

i.  $1001001101_2 \text{ NAND } 0110101110_2$

$$\mathbf{1001001101_2 \text{ NAND } 0110101110_2 = 1111110011_2}$$

j.  $100011_2 \text{ NOR } 001000_2$

$$\mathbf{100011_2 \text{ NOR } 001000_2 = 010100_2}$$

k.  $x908 \text{ XNOR } xA51$

$$\mathbf{x908 = 100100001000_2, xA51 = 101001010001_2}$$

$$\mathbf{\text{So } x908 \text{ XNOR } xA51 = 110010100110_2}$$

### Problem 2

Perform the following operations on unsigned integers. Assume the number of bits given. Indicate when there is an overflow for operations that have it.

a.  $100100_2 + 010101_2$

$$\mathbf{100100_2 + 010101_2 = 111001_2, \text{ no overflow}}$$

- b.  $011101_2 + 111011_2$   
 $\mathbf{011101}_2 + \mathbf{111011}_2 = \mathbf{1} \ 011000_2$ , overflow
- c.  $1111000_2 \ll 2$   
 $\mathbf{1111000}_2 \ll \mathbf{2} = \mathbf{1100000}_2$
- d.  $1111000_2 \gg 2$   
 $\mathbf{1111000}_2 \gg \mathbf{2} = \mathbf{0011110}_2$
- e.  $000100_2 \gg 2$   
 $\mathbf{000100}_2 \gg \mathbf{2} = \mathbf{000001}_2$

### Problem 3

Perform the following operations on signed integers. Assume the number of bits given. Indicate when there is an overflow for operations that have it.

- a.  $110010_2 + 110001_2$   
 $\mathbf{100100}_2 + \mathbf{010101}_2 = \mathbf{111001}_2$ , no overflow
- b.  $11011010_2 + 11010110_2$   
 $\mathbf{11011010}_2 + \mathbf{11010110}_2 = \mathbf{1} \ 10110000_2$ , no overflow
- c.  $1001_2 - 1010_2$   
 $\mathbf{1001}_2 - \mathbf{1010}_2 = \mathbf{1001}_2 + \mathbf{0110}_2 = \mathbf{1111}_2$ , no overflow
- d.  $011101_2 - 111011_2$   
 $\mathbf{011101}_2 - \mathbf{111011}_2 = \mathbf{011101}_2 + \mathbf{000101}_2 = \mathbf{100111}_2$ , overflow
- e.  $1111000_2 \ll 2$   
 $\mathbf{1111000}_2 \ll \mathbf{2} = \mathbf{1100000}_2$
- f.  $1111000_2 \gg 2$   
 $\mathbf{1111000}_2 \gg \mathbf{2} = \mathbf{1111110}_2$
- g.  $000100_2 \gg 2$   
 $\mathbf{000100}_2 \gg \mathbf{2} = \mathbf{000001}_2$

## IEEE-754 Floating Point

### Problem 1

Convert the following decimal representations to IEEE-754 floating point.

- a. 3.625  
**0 10000000 11010000000000000000000000000000**
- b. -18.5  
**1 10000011 00101000000000000000000000000000**
- c. 42.3125  
**0 10000100 010100101000000000000000**

### Problem 2

Convert the following IEEE-754 floating point representations to decimal.

- a. 0 10000001 11100100000000000000000000000000  
**7.5625**
- b. 0 10000011 00001000000000000000000000000000  
**16.5**
- c. 1 10000011 10010100000000000000000000000000  
**-25.25**

## C Basics

### Problem 1

Declare the following variables:

- a. The signed integer -10 named *x*.  
**int x = -10;**
- b. The character ‘p’ named *P*.  
**char P = ‘p’;**
- c. The decimal 0.536 as a float named *y*.  
**float y = 0.536;**
- d. The unsigned integer 235 named *ux*.  
**unsigned ux = 235;**
- e. The decimal 0.46668 as a double named *dy*.  
**double dy = 0.46668;**

## Problem 2

Evaluate the following expressions in C. Assume that the variable a has been declared as 0xECEB and b has been declared as 0x2345.

a.  $a \& b$   
**0x2041**

b.  $a ^ b$   
**0xCFAE**

c.  $\sim a$   
**0x1314**

d.  $a | b$   
**0xEFEEF**

# C Programming

## Problem 1

Write code in C for the following tasks. Assume that age is already initialized to 0 and is of type int.

- a. Print a prompt message asking the user to input their age.

```
printf("Input your age: ");
```

- b. Store the input in the variable age.

```
scanf("%d", &age);
```

- c. Print twice of the age you received as an input to the console.

```
printf("%d", 2 * age);
```

## Problem 2

Consider the following C code.

```

int main() {
    for (int i = 0; i < 10; i++) {
        printf("%d\n", i);

        if (i == 10) {
            printf("Now i is 10.");
        }
    }
    return 0;
}

```

- a. How many times does the program print to the console?

**10 times. The loop stops at i = 10, but does not execute.**

- b. What is the output of this program?

**0  
1  
2  
3  
4  
5  
6  
7  
8  
9**

### Problem 3

What does the following C code print?

```

int main() {
    int x = 10;
    if (x = 5) {
        printf("x is 5.");
    } else {
        printf("x is not 5.");
    }
    return 0;
}

```

**x is 5. If if statement uses the "==" assignment operator, not the "==" comparison operator.**

## Problem 4

What does the following C code print?

```
int main() {
    int i = 90;
    while (i >= 3) {
        printf("%d ", i);
        i = i/3;
    }
    return 0;
}
```

**90 30 10 3**