# CS 278 Study Guide #3

## NAME: Aaron Borjas

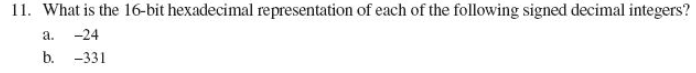
**GRADE:**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **POINTS** |  |
| EX02\_01 |  | 70 |
| EX02\_02 |  | 15 |
| EX02\_03 |  | 15 |
| **TOTAL** |  | 100 |

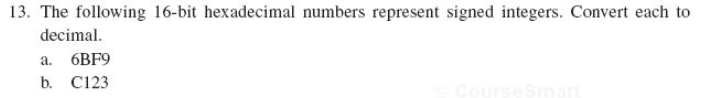
## EXERCISES:

NOTE: Please write your answers directly in this document and save it in your folder on CS1. For EX02\_02 and EX02\_03, make a folder for each in your CS-278-1 folder on CS1.

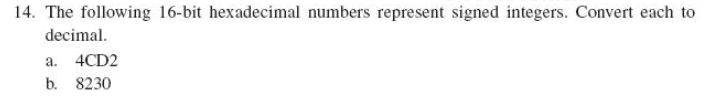
**EX03\_01 –** Do problems 11, 13, 14, 17, 24 in Section 1.7.1 of your textbook.



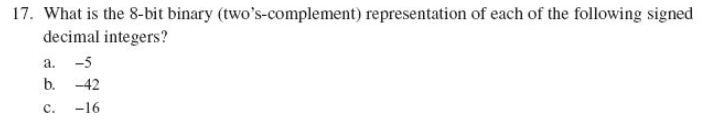
1. E8
2. FEB5



1. 27651
2. 49443 OR -16093 (2’s complement)



1. 19666
2. 33328 OR -32208 (2’s complement)



1. 1111 1011
2. 1101 0110
3. 1111 0000



2^85 -1; 38,685,626,227,668,133,590,597,631

**EX03\_02 – Write a C++ or Python Program** that prompts the user for an integer. The program then will check that integer to see if it satisfies or DOES NOT satisfy the following properties:

* **There is a sequence of 2 or more bits immediately next to each other in the binary representation of the number**.
* For example: 00**11**0101 would satisfy the rules, but 01010101 would not.

**111**00000 would satisfy the rules, but, 10101000 would not.

000010**11** would satisfy the rules, but, 00100000 would not.

00**11111**0 would satisfy the rules, but, 00100100 would not.

* **Output whether or not the rules are satisfied for the integer that was entered.**

**EX03\_03 – Write a C++ or Python Program**Prompt the user for a max integer, count the number of integers (0…max\_integer) that have the properties as defined in EX03\_02. Your program should OUTPUT the number of integers less than max that satisfy these properties.

# CS278 Study Guide 3: Data Representation

## Reading Assignment for Next Class Period

* Start Reading Chapter 2 in Irvine

## Background Material for this Session

* Chapter 1 in Irvine

## Learning Goals for this Study Guide

* Understand how to convert to/from two’s complement notation.
* Learn how to perform binary subtraction using two’s complement notation.
* Gain an intuitive understanding of why two’s complement works.
* Understand how to perform two’s complement on hexadecimal values.
* Learn to convert signed binary to decimal
* Learn to convert signed decimal to binary
* Learn to convert signed decimal to hexadecimal
* Learn to convert signed hexadecimal to decimal
* Understand how to calculate the range of an n-bit signed binary number
* Understand the different character storage formats

## Data Representation

### Positional Number Systems

Donald Knuth in the ***Art of Computer Programming*** Vol. 2, describes how “**the way we do arithmetic**” is intimately related to “**the way we represent the numbers we deal with**”. So, it is appropriate to begin our study of how we do computer arithmetic with a discussion of the principal notation we use to represent numbers.

### Representing Numbers

5

4

9

.

8

2

5 x 10^2 = 5 x 100 = 500

7

7 x 10^3 = 7 x 1000 = 7000

9 x 10^0 = 9 x 1 = 9

4 x 10^1 = 4 x 10 = 40

8 x 10^-1 = 8 / 10 = 0.8

2 x 10^-2 = 2 / 100 = 0.02

TOTAL = **7549.82**

* Every number system has a **base**.
* The **base** represents the **maximum number of symbols** assigned to a single digit of the number.
* The **position** **of a digit** (relative to the radix point) assigns it as the **coefficient** of the **base raised to the power the digit's position**.

### An Arbitrary Base Number

Positional notation using **base b** (alternatively called **radix b**) is defined by the rule:

Radix point ( in Europe they use a comma instead of a period )

(…A3A2A1A0 **.** A-1A-2A-3 …)b =…+ A3b3 + A2b2 + A1 b1 + A0b0 + A-1b-1 + A-2b-2 + A-3 b-3 + …

Most Significant Digit(s)

Least Significant Digit(s)

By convention, the special case that **base** **b = 10** is of course our decimal number system.

Other examples of number systems:

|  |  |
| --- | --- |
| b = 2 | Binary |
| b = 3 | Ternary |
| b = 4 | Quaternary |
| b = 5 | Quinary |
| : | : |
| b = 8 | Octal |
| : | : |
| b = 10 | Decimal |
| : | : |
| b = 16 | Hexadecimal |
| : | : |
| b = 20 | Mayan  (Vigesimal) |
| : | : |
| b = 60 | Babylonian  (Sexagesimal) |

Many of these number systems are still in use today. Which ones?

### Binary Numbers

* Binary numbers are base 2, thus binary numbers have two symbols per digit.
* Digits are 1 and 0
  + 1 = true
  + 0 = false
* **MSB** – most significant bit
* **LSB** – least significant bit
* The radix point is sometimes called the binary point, (i.e. just like we call the radix point the decimal point in base 10 )
* Bit numbering in a binary number:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MSB** |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **LSB** |
| **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

* Binary digits (BITs) represent **powers of two**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2n** | **Decimal Value** |  | **2n** | **Decimal Value** |
| 20 | 1 |  | 28 | 256 |
| 21 | 2 |  | 29 | 512 |
| 22 | 4 |  | 210 | 1024 |
| 23 | 8 |  | 211 | 2048 |
| 24 | 16 |  | 212 | 4096 |
| 25 | 32 |  | 213 | 8192 |
| 26 | 64 |  | 214 | 16384 |
| 27 | 128 |  | 215 | 32768 |

### Translating Binary to Decimal

* Every binary number is a **sum of powers of two!** ( See our positional number system definition )  
    
  **Example:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  | 23 | 22 | 21 | 20 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

The binary number in this example may be represented as:

(0\*215) + … + (0\*24) + (1\*23) + (1\*22) + (0\*21) + (1\*20)

If we convert this binary number to base 10’s positional system we have:

(1\*23) + (1\*22) + (1\*20) = (8)10 + (4)10 + (1)10 = (13)10

**Example:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 215 |  |  | 212 | 211 | 210 | 29 |  | 27 |  | 25 | 24 | 23 |  |  | 20 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

The binary number in this example may be calculated as:

**(1\*215)**+(0\*214)+(0\*213)+**(1\*212)**+**(1\*211)**+**(1\*210)**+**(1\*29)**+(0\*28)+**(1\*27)**+(0\*26)+**(1\*25)**+**(1\*24)**+**(1\*23)**+(0\*22)+(0\*21)+**(1\*20)**

Dropping out the terms that have a coefficient of 0,

(1\*215) +(1\*212)+(1\*211)+(1\*210)+(1\*29) +(1\*27) +(1\*25)+(1\*24)+(1\*23)+(1\*20) = (40633)10

We can show this number in base 10’s positional system as:

40633 = 4 \* 104 + 0 \* 103 + 6 \* 102 + 3 \* 101 + 3 \* 100

### Translating Unsigned Decimal to Binary

* Repeatedly divide the decimal integer by 2.
* Each remainder is a binary digit in the translated value

**Example:**

Convert (183) 10 to binary:

|  |  |  |
| --- | --- | --- |
| **Division** | **Quotient** | **Remainder** |
| 183 / 2 | 91 | 1 |
| 91 / 2 | 45 | 1 |
| 45 / 2 | 22 | 1 |
| 22 / 2 | 11 | 0 |
| 11 / 2 | 5 | 1 |
| 5 / 2 | 2 | 1 |
| 2 / 2 | 1 | 0 |
| 1 / 2 | 0 | 1 |

Thus (183) 10 = (10110111) 2

**Question 1:** **Why does this work?**

Look at the positional number system for binary numbers. **Division by 2** **shifts** **the binary digits to the right by one position**. The “bits” slide off the end as a remainder. For example:

½ \* (1\*27) + (0\*26) + (1\*25) + (1\*24) + (0\*23) + (1\*22) + (1\*21) + (1\*20) =

(1\*26) + (0\*25) + (1\*24) + (1\*23) + (0\*22) + (1\*21) + (1\*20) + (1\*2-1) Note the remainder of 1

½ \* (1\*26) + (0\*25) + (1\*24) + (1\*23) + (0\*22) + (1\*21) + (1\*20) =

(1\*25) + (0\*24) + (1\*23) + (1\*22) + (0\*21) + (1\*20) + (1\*2-1) Note the remainder of 1

½ \* (1\*25) + (0\*24) + (1\*23) + (1\*22) + (0\*21) + (1\*20)=

(1\*24) + (0\*23) + (1\*22) + (1\*21) + (0\*20) + (1\*2-1) Note the remainder of 1

½ \* (1\*24) + (0\*23) + (1\*22) + (1\*21) + (0\*20) =

(1\*23) + (0\*22) + (1\*21) + (1\*20) + (0\*2-1) Note the remainder of 0

**Question 2**: If shifting a bit pattern of binary digits to the right corresponds to dividing the value of the number by 2, how does shifting a pattern of binary digits to the left affect the value of the number?

### Binary Addition

Starting with the LSB, add each pair of digits, include the carry if present:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |  | Carry Bits |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |  | (7)10 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |  | +(13)10 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |  | (20)10 |

**Question 3:** You should be able to immediately write down the value of 20 after inspecting the answer above. Why?

### Integer Storage Sizes

|  |  |  |
| --- | --- | --- |
| **Storage Type** | **Range** | **Powers of 2** |
| Unsigned byte | 0 to 255 | 0 to (28 – 1 ) |
| Unsigned word | 0 to 65,535 | 0 to (216 – 1 ) |
| Unsigned doubleword | 0 to 4,294,967,295 | 0 to (232 – 1 ) |
| Unsigned quadword | 0 to 18,446,744,073,709,551,615 | 0 to (264 – 1 ) |

**Question 4:** What is the largest unsigned integer that may be stored in 20 bits?

### Hexadecimal Integers

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Binary** | **Decimal** | **Hexadecimal** |  | **Binary** | **Decimal** | **Hexadecimal** |
| 0000 | 0 | 0 |  | 1000 | 8 | 8 |
| 0001 | 1 | 1 |  | 1001 | 9 | 9 |
| 0010 | 2 | 2 |  | 1010 | 10 | A |
| 0011 | 3 | 3 |  | 1011 | 11 | B |
| 0100 | 4 | 4 |  | 1100 | 12 | C |
| 0101 | 5 | 5 |  | 1101 | 13 | D |
| 0110 | 6 | 6 |  | 1110 | 14 | E |
| 0111 | 7 | 7 |  | 1111 | 15 | F |

### Translating Binary to Hexadecimal

* Each hexadecimal digit corresponds to 4 binary bits.
* Example: Translate the binary integer 000101101010011110010100 to hexadecimal:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 6 | A | 7 | 9 | 4 |
| 0001 | 0110 | 1010 | 0111 | 1001 | 0100 |

**Question 5:** What is the binary value of 3BA4?

### Translating Hexadecimal to Decimal

* Multiply each digit by its corresponding power of 16:  
  dec = (D3 × 163) + (D2 × 162) + (D1 × 161) + (D0 × 160)
* Hex 1234 equals (1 × 163) + (2 × 162) + (3 × 161) + (4 × 160), or decimal 4,660.
* Hex 3BA4 equals (3 × 163) + (11 \* 162) + (10 × 161) + (4 × 160), or decimal 15,268.
* Hexadecimal digits represent **powers of sixteen**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **16n** | **Decimal Value** |  | **16n** | **Decimal Value** |
| 160 | 1 |  | 164 | 65,536 |
| 161 | 16 |  | 165 | 1,048,576 |
| 162 | 256 |  | 166 | 16,777,216 |
| 163 | 4096 |  | 167 | 268,435,456 |

**Question 6:** What is the decimal value of Hex DAD?

### Converting Decimal to Hexadecimal

Repeatedly divide the decimal integer by 16. Each remainder is a hexadecimal digit in the translated value:

|  |  |  |
| --- | --- | --- |
| **Division** | **Quotient** | **Remainder** |
| 422 / 16 | 26 | 6 |
| 26 / 16 | 1 | A |
| 1 / 16 | 0 | 1 |

Thus (422) 16 = (1A6) 10

**Question 7:** What is the hexadecimal value of 234?

### Hexadecimal Addition

Divide the sum of two digits by the number base (16). The quotient becomes the carry value, and the remainder is the sum digit.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 1 | 1 |
| 36 | 28 | 28 | 6A |
| 42 | 45 | 58 | 4B |
| 78 | 6D | 80 | B5 |

21 / 16 = 1, remainder 5

**Question 8:** What is the sum of DADh and FADh ?

**Important skill:** Programmers frequently add and subtract the addresses of variables and instructions. Calculators can do this fairly easily – most have alternate bases built in!

### Hexadecimal Subtraction

When borrowing from the digit to the left, add 16 (i.e. 10 hex ) to the current digit's value:

Focus on the right column:

10h + 5h = 15h

15h – 7h = Eh

|  |  |
| --- | --- |
|  | -1 |
| C6 | 75 |
| A2 | 47 |
| 24 | 2E  Now, focus on the left column of the first example:  7h -1h - 4h = 2h |

**Question 9:** The address of **var1** is **00400020**. The address of the next variable after **var1** is **0040006A**. How many bytes are used by **var1**?

## Representing Signed Integers in Binary

Now you have seen how **unsigned integers** may be represented using binary, we need to consider how to represent **signed integers**.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MSB**  The **highest** **possible** **bit** represents the sign. 1 = **negative** and 0 = **positive** |  |  |  |  |  |  | **LSB** |
| **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MSB** |  |  |  |  |  |  | **LSB** |
| **0** | **0** | **1** | **1** | **1** | **0** | **0** | **1** |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

* The highest bit indicates the sign. 1 = negative, 0 = positive
* If the **highest digit** of a **signed** **hexadecimal integer** is > 7, the value is **negative**.   
    
  **Examples**: 8A, C5, A2, 9D

## Two’s Complement Notation for Signed Integers

* Negative numbers are stored in two's complement notation
* Represents the additive Inverse

|  |  |
| --- | --- |
| **Steps to Form the 2’s Complement of a Value** | **Value** |
| 1. Load starting value | 00000001 |
| 2. Reverse the bits | 11111110 |
| 3. Add 1 to the value from step 2 | 11111110  + 1 |
| 4. Result is **two’s complement** of the starting value | 11111111 |

### Binary Subtraction Using Two’s Complement

* When subtracting A – B, convert B to its two's complement
* Add A to (–B)

**Example: 1101 - 0101**

* Convert **0101** to a two’s complement number **1011** (this represents -5)
* Add the two’s complement **1011** to **1101**
* The result is your answer:

Form the 2’s complement of the second number before adding

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **1** | **0** | **1** |  |  | **1** | **1** | **0** | **1** |
| **-** | **0** | **1** | **0** | **1** |  |  | **1** | **0** | **1** | **1** |
|  |  |  |  |  |  |  | **1** | **0** | **0** | **0** |

The final “carry bit” is discarded when adding using 2’s complement

### Why Two’s Complement? How Does It Really Work?

* Two’s complement makes the design of microprocessor hardware much easier.
* No separate circuits are required for subtraction.
* All addition and subtraction are implemented using addition!
* How does it really work? See the next exercise:

#### HANDS ON EXERCISE EX03\_A: Understanding Two’s Complement

The purpose of this exercise is to help you understand intuitively how two’s complement actually works:

Add the binary, unsigned, and 2’s comp X values from the first table to the corresponding binary, unsigned, and 2’s comp Y values of the second table. Write the answers in the corresponding spaces provided in the third table.

**When adding the binary numbers, only keep 8 bits of the result** since this example is for 8-bit integers.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X Values** | | |  | **Y Values** | | |  | **Sum of X and Y Values** | | |
| **Binary** | **Unsigned**  **Decimal** | **2’s Comp Signed Dec** |  | **Binary** | **Unsigned**  **Decimal** | **2’s Comp Signed Dec** |  | **Binary** | **Unsigned**  **Decimal** | **2’s Comp Signed Dec** |
| 0000 0000 | 0 | 0 |  | 0000 0000 | 0 | 0 |  |  |  |  |
| 0000 0001 | 1 | 1 |  | 1111 1111 | 255 | -1 |  | 1111 1111 | 255 | -1 |
| 0000 0011 | 3 | 3 |  | 1111 1101 | 253 | -3 |  | 1111 1101 | 253 | -3 |
| 0000 0111 | 7 | 7 |  | 1111 1001 | 249 | -7 |  | 1111 1001 | 249 | -7 |
| 0000 1111 | 15 | 15 |  | 1111 0001 | 241 | -15 |  | 1111 0001 | 241 | -15 |
| 0001 1111 | 31 | 31 |  | 1110 0001 | 225 | -31 |  | 1110 0001 | 225 | -31 |
| 0011 1111 | 63 | 63 |  | 1100 0001 | 193 | -63 |  | 1100 0001 | 193 | -63 |
| 0111 1111 | 127 | 127 |  | 1000 0001 | 129 | -127 |  | 1000 0001 | 129 | -127 |

**Question 10:**  (a) What pattern do you notice?   
  
  
(b) Why do you think this pattern occurs?

(c) What binary number results if we DON’T IGNORE the carry bit of the binary sum?

### Memorize How to Do the Following:

* Form the two's complement of a hexadecimal integer
* Convert signed binary to decimal
* Convert signed decimal to binary
* Convert signed decimal to hexadecimal
* Convert signed hexadecimal to decimal

## Ranges of Signed Integers

|  |  |  |
| --- | --- | --- |
| **Storage Type** | **Range (low-high)** | **Powers of 2** |
| Signed byte | -128 to +127 | -27 to (27 – 1 ) |
| Signed word | -32,768 to +32,767 | -215 to (215 – 1 ) |
| Signed doubleword | -2,147,483,648 to +2,147,483,647 | -231 to (231 – 1 ) |
| Signed quadword | -9,223,372,036,854,775,808 to  +9,223,372,036,854,775,807 | -263 to (263 – 1 ) |

#### HANDS ON EXERCISE EX03\_B: Ranges of Signed Integers

What is the largest **positive** value that can be stored in a **signed** **memory location** with **20 bits?**

## Character Storage

* Character sets
  + Standard ASCII (0 – 127)
  + Extended ASCII (0 – 255)
  + ANSI (0 – 255)
  + Unicode (0 – 65,535)
* Null-terminated String
  + Array of characters followed by a null byte
* Using the ASCII table
  + Look inside cover of book

## Terminology for Numeric Data Representation

The interpretation of numbers on a computer depends greatly on the **context** in which the number appears.

* pure binary
  + raw format ready for use in a calculation
  + can be calculated directly
* ASCII binary
  + The raw format that represents an ASCII character
  + Consists of a string of digits: "01010101"
* ASCII decimal
  + The decimal value that corresponds to the binary value for an ASCII character
  + string of digits: "65"
* ASCII hexadecimal
  + The hexadecimal value that corresponds to the binary value of a particular ASCII character
  + string of digits: "9C"