MSiA-413 Introduction to Databases and Information Retrieval

Homework 1: Number representations

Name 1: Kristiyan Dimitrov

NetID 1: ktd5131

Name 2: Laurie Merrell

NetID 2: lmr8733

# Instructions

You should submit this homework assignment via Canvas. Acceptable formats are word files, text files, and pdf files. Paper submissions are not allowed and they will receive an automatic zero.

As explained during lecture and in the syllabus, assignments are done in groups. The groups have been created and assigned. Each group needs to submit only one assignment (i.e., there is no need for both partners to submit individually the same homework assignment).

Each group can submit solutions multiple times (for example, you may discover an error in your earlier submission and choose to submit a new solution set). We will grade only the last submission and ignore earlier ones.

Make sure you submit your solutions before the deadline. The policies governing academic integrity, tardiness and penalties are detailed in the syllabus.

**Due Date: Thursday October 10, 11:59pm**

## Question 1. Unsigned Integer Representation (10 points – 1 point per row)

Please fill out the blank parts of the table below to (i) express the following numbers in binary, (ii) calculate their hexadecimal representation (as shown in class), and (iii) calculate their decimal value as a sum of powers of two.

For example: 5210 = 0011 01002 = 0x34 = 1\*25 + 1\*24 + 0\*23 + 1\*22 + 0\*21 + 0\*20 = 32 + 16 + 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Decimal Number | binary | Hexadecimal | Sum of powers of two |
| **example** | 52 | **0011 0100** | **0x34** | **32 + 16 + 4** |
| **1A)** | 6 | **0110** | **0x6** | **4 + 2** |
| **1b)** | 19 | **0001 0011** | **0x13** | **16 + 2 + 1** |
| **1c)** | 22 | **10110** | **0x16** | **16 + 4 + 2** |
| **1d)** | 38 | **100110** | **0x26** | **32 + 4 + 2** |
| **1e)** | 42 | **101010** | **0x2A** | **32 + 8 + 2** |
| **1f)** | 155 | **10011011** | **0x9B** | **128 + 16 + 8 + 2 + 1** |
| **1g)** | 612 | **1001100100** | **0x264** | **512 + 64 + 32 + 4** |
| **1h)** | 1819 | **0111 0001 1011** | **0x71B** | **1024 + 512 + 256 + 16 + 8 + 2 + 1** |
| **1I)** | 2293 | **1000 1111 0101** | **0x8F5** | **2048 + 128 + 64 + 32 + 16 + 4 + 1** |
| **1J)** | 3176 | **1100 0110 1000** | **0xC68** | **2048 + 1024 + 64 + 32 + 8** |

## Question 2. Signed Integer Representation (10 points – 1 point per row)

Please fill out the blank parts of the table below as needed to calculate the 8-bit two’s complement binary representation of the following signed decimal numbers. |x| denotes the absolute value of x. If particular number(s) below cannot be represented as 8-bit signed binary integer(s), please indicate which one(s) and explain why. The explanation will then carry the points of the corresponding row(s). Please remember that:

* You may need to add zeros to the left of the number to make it an 8-bit binary number.
* In two’s complement signed integer representation, for every (positive or negative) integer x, it holds:   
  -x = ~x + 1, where ~x is the complement of x (calculated by flipping the bits of x).
* If you don’t need a column to calculate the 8-bit signed integer binary representation of a number, you can leave that column blank (as in example 1).

For example: 8210 = 0101 00102. Note that we added a zero at the front to make it an 8-bit binary number. The sign bit (most significant bit of its 8-bit binary representation) is 0, indicating a positive integer, so the 8-bit signed integer binary representation of 8210 is 0101 00102.

However, to calculate the 8-bit signed integer representation of -82 we need to calculate its two’s complement. We have |-8210| = 8210 = 0101 00102. Flipping the bits gives 1010 1101 2. Finally, adding one gives 1010 1110 2. Thus, the 8-bit signed binary representation of -8210 is 1010 1110 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | value of integer x | |x|  in 8-bit binary | ~|x| if needed | 8-bit signed integer representation |
| example 1 | 82 | **0101 0010 = 8210** |  | **0101 0010 = 8210** |
| example 2 | -82 | **0101 0010 = 8210** | **1010 1101** | **1010 1110 = -8210** |
| 2A) | -1 | **0000 0001** | **1111 1110** | **1111 1111** |
| 2b) | -19 | **0001 0011** | **1110 1100** | **1110 1101** |
| 2c) | 22 | **0001 0110** |  | **0001 0110** |
| 2d) | -38 | **0010 0110** | **1101 1001** | **1101 1010** |
| 2e) | -42 | **0010 1010** | **1101 0101** | **1101 0110** |
| 2f) | 68 | **0100 0100** |  | **0100 0100** |
| 2g) | -100 | **0110 0100** | **1001 1011** | **1001 1100** |
| 2h) | 127 | **0111 1111** |  | **0111 1111** |
| 2I) | 128 | \* | \* | \* |
| 2J) | -129 | \*\* | \*\* | \*\* |

If you found any numbers that you cannot represent as 8-bit signed integers, indicate which ones and why they cannot be represented in the space below.

\* 128 cannot be represented as an 8-bit signed integer. Its representation would be: 1000 0000, which would be read as -128 since the first digit indicates a negative sign. More generally, we know that the largest positive integer which can be represented as a signed integer with k bits is 2k-1-1, which for k=8 is 127, so 128 > 127 cannot be represented.

\*\* Similarly, -129 cannot be represented as a signed integer with 8 bits. It would need additional bits, because the smallest negative number which we can represent would take the form 1000 0000, which is -128, and we have no space to represent the additional -1 to get to -129. In general, we know that the smallest negative integer which can be represented as a signed integer with k bits is -2k-1, which for k=8 is -128, so -129 < -128 cannot be represented.

## Question 3. Binary Pattern Representation (10 points)

|  |  |
| --- | --- |
| Bit pattern 1 | Bit pattern 2 |
| 1101 1110 1010 1101 1011 1110 1110 1111 | 0100 1100 0100 1111 0100 1100 0010 0001 |

What value does bit pattern 1 represent when interpreted as a

What value does bit pattern 1 represent when interpreted as a

1. **(2 points)** 32-bit signed integer in two’s complement arithmetic?

1101 1110 1010 1101 1011 1110 1110 1111 =

**(-)** 1\*231 + 1\*230 + 1\*228 + 1\*227 + 1\*226 + 1\*225 + 1\*223 + 1\*221 + 1\*219 + 1\*218 + 1\*216 + 1\*215 + 1\*213 + 1\*212 + 1\*211 + 1\*210 + 1\*29 + 1\*27 + 1\*26 + 1\*25 + 1\*23 + 1\*22 + 1\*21 + 1\*20 = 1,588,444,911 - 2,147,483,648 = **- 559,038,737**

1. **(2 points)** 32-bit unsigned integer?

We performed problem 3c) first. From there we have that Bit Pattern 1 = **0xDEADBEEF**

From there, we can calculate the value as an unsigned decimal integer:

D\*167 + E\*166 + A\*165 + D\*164 + B\*163 + E\*162 + E\*161 + F\*160 =

13\*167 + 14\*166 + 10\*165 + 13\*164 + 11\*163 + 14\*162 + 14\*161 + 15\*160 = **3,735,928,559** (using a calculator)

1. **(1 point)** 32-bit unsigned hexadecimal number?

Bit Pattern 1: 1101 1110 1010 1101 1011 1110 1110 1111

We convert each 4 bits into their respective hexadecimal representation:

        0x - D -  E - A - D  - B - E -   E - F = **0xDEADBEEF**

What value does **bit pattern 2** represent when interpreted as a

1. **(2 points)** 32-bit signed integer in two’s complement arithmetic?

We performed problem 3e) first. The value of the signed 32-bit integer will be the same as the unsigned one, because the first bit contains a 0, thereby signifying that our number is positive. The result we got in 3e) is positive and is **1,280,265,249.**

1. **(2 points)** 32-bit unsigned integer?

Bit Pattern 2: 0100 1100 0100 1111 0100 1100 0010 0001:

We convert each 4 bits into their respective hexadecimal representation:

      0x - 4  - C - 4  - F - 4   - C - 2 - 1  =0x4C4F4C21=4\*167 + 12\*166 + 4\*165 + 15\*164 + 4\*163 + 12\*162 + 2\*161 + 1\*160 = **1,280,265,249**

1. **(1 point)** UTF-8 text?

We use [UTF](https://en.wikipedia.org/wiki/UTF-8)  & [ASCII](https://en.wikipedia.org/wiki/ASCII) tables from Wikipedia for reference. Below we see all the bytes can be interpreted as ASCII:

L(**100 1100**) O(**100 1111**) L(**100 1100**) !(**010 0001**) = **LOL!**

## Question 4. Accuracy of Integer and Floating Point Representations (20 points – 1 per row)

In the table below, answer “***exactly***,” “***approximately***,” or “***no way***” to indicate how the following (base ten) numbers can be represented as 32-bit signed integers, 64-bit signed integers, single precision (32-bit) floats, and double precision (64-bit) floats. If the number can be rounded to a value that is representable and the relative error is less than 10-3, then the best answer is “approximately.” For example, I would say that 2.5 is “no way” 3, but 2.9999999 is “approximately” 3. The relative error definition is in the reading material and was explained in the lab session.

For example, consider the number 1.0000000005 × 1030. It cannot be represented as 32-bit signed integer nor as a 64-bit signed integer (because it is too large). However, it can be represented approximately as a single precision float (with rounding – there are not enough bits in the mantissa of the 32-bit float to represent the 0.0000000004 fractional part of the number, but the relative error is <10-3). Moreover, it can be represented exactly as a double precision float.

You can use online floating-point conversion tools (e.g., <http://www.binaryconvert.com/convert_float.html>).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Base ten number | 32-bit *signed* Integer | 64-bit *signed* integer | 32-bit Floating point | 64-bit Floating point |
| Example | 1.0 × 1020 | **no way** | **no way** | **approximately** | **exactly** |
| 4a | -0.5 | **no way** | **no way** | **exactly** | **exactly** |
| 4b | 1/3 (one third) | **no way** | **no way** | **approximately** | **approximately** |
| 4C | 0.1 | **no way** | **no way** | **approximately** | **approximately** |
| 4d | 1/16 (one sixteenth) | **no way** | **no way** | **exactly** | **exactly** |
| 4e | 17.5 | **no way** | **no way** | **exactly** | **exactly** |
| 4f | 4,000,000,000 | **no way** | **exactly** | **exactly** | **exactly** |
| 4g | -4,000,000,001 | **no way** | **exactly** | **approximately** | **exactly** |
| 4h | \* | **no way** | **no way** | **approximately** | **approximately** |
| 4i | 2,000,000,000 | **exactly** | **exactly** | **exactly** | **exactly** |
| 4J | 2,000,000,001 | **exactly** | **exactly** | **approximately** | **exactly** |
| 4k | 20,000,000 | **exactly** | **exactly** | **exactly** | **exactly** |
| 4l | 20,000,000.25 | **no way** | **no way** | **approximately** | **exactly** |
| 4m | 33,554,432.25 | **no way** | **no way** | **approximately** | **exactly** |
| 4n | 33,554,432.9 | **no way** | **no way** | **approximately** | **approximately** |
| 4o | 9,123,000,000,000,000,000 | **no way** | **exactly** | **no way** | **exactly** |
| 4p | -9,123,123,123,000,000,000 | **no way** | **exactly** | **no way** | **approximately** |
| 4q | 9,123,123,123,123,123,123 | **no way** | **exactly** | **no way** | **approximately** |
| 4r | pi | **no way** | **no way** | **approximately** | **approximately** |
| 4s | infinity | **no way** | **no way** | **exactlya** | **exactlya** |
| 4t | zero | **exactly** | **exactly** | **exactlyb** | **exactlyb** |

\*Used R to generate an approximate value for reference.

a: We are saying “exactly” because there exists a specific reserved value in 32- and 64-bit floating point representation which is read as “infinity”. However, saying that a value “exactly” represents infinity may be misleading since infinity can be understood differently in different contexts (for example, there are sets with different infinite cardinalities.)

b: We are saying “exactly” because there exist specific reserved values in 32- and 64-bit floating point representation which are read as + /- 0. We assume that the fact that this is a “signed” 0 is acceptable (calling it “approximate” does not make sense, since to calculate a relative error value we would need to divide + 0/0 or -0/0, which are not defined.)