**Quiz #1 Prep**

**I. Number Operations (Signed/Unsigned)**

Resources: Textbook Chapter 2 – Section 2.5, Textbook Chapter 3 – Section 3.2,

[Two’s Complement](http://sandbox.mc.edu/~bennet/cs110/tc/tctod.html#:~:text=To%20do%20this%2C%20you%20first,convert%20the%20result%20to%20decimal.) Review, [Online Converter](https://www.rapidtables.com/convert/number/hex-dec-bin-converter.html)

|  |  |  |
| --- | --- | --- |
| WORD | Unsigned | Signed |
| 8-Bit |  |  |
| 16-Bit | [0, 65536] | [-32768, +32767] |
| 32-Bit |  |  |
| n-bit | [] | [ ] |

---------------------------------------------

Worked Example 1: Add 0xFF and 0xFF (Signed)

First convert the hexadecimal 0xFF to binary where each digit corresponds to a 4-Bit binary pattern. Since 0xF = 0b1111, then 0xFF = 0b1111 1111. To convert to decimal, apply Two’s complement:

1111 1111 0000 0000 + 1 = 0000 0001 = -1

Then we perform the addition in binary (ignore the carry bit when dealing with signed operations).

1111 1111 + 1111 1111 = 1 1111 1110

Convert the answer back to hexadecimal and decimal.

0xFF -1 0b1111 1111

+

0xFF -1 0b1111 1111

Hex: 0xFE Dec: -2 Binary: 0b1111 1110

Worked Example 2: Subtract 0x7F and 0xFF (Signed)

Convert the hexadecimal:

0x7F = 0b0111 1111

Since 0x7F is positive, we convert to decimal as we normally would:

0x7F =

Convert 0xFF to binary:

0xFF =0B1111 1111

Since 0xFF is negative, we apply the Two’s complement rule:

0xFF = 0b0000 0001 = -1

To subtract a positive and negative number in binary, we make use of the negative’s two’s complement so that we actually perform (Positive) + (Two’s complement of negative).

0x7F +127 0b0111 1111

-

0xFF -1 0b1111 1111

+ 0b0000 0001

Hex: 0x80 Dec: -128 Binary: 0b1000 0000

However, we expected the result to be +128. This is a result of overflow because the maximum number in an 8-bit WORD is +127.

Example 3: Unsigned Addition

0xA9 169 0b1010 1001

+

0x3B 59 0b0011 1011

Hex: 0xE4 Dec: 228 Binary: 0b1110 0100

Example 4: Unsigned Subtraction

0xF2 242 0b1111 0010

-

0x2C 44 0b0010 1100

+ 0b1101 0100

Hex: 0xC6 Dec: 198 Binary: 0b1100 0110

Example 5: Signed Subtraction

0x0E +14 0b0000 1110

-

0xFF -1 0b1111 1111

+ 0b0000 0001

Hex: 0x0F Dec: +15 Binary: 0b0000 1111

**II. Fixed-Point Representation**

Resource: [Introduction to Fixed Point Number Representation (berkeley.edu)](https://inst.eecs.berkeley.edu/~cs61c/sp06/handout/fixedpt.html)

*\*Shifting to the RIGHT is multiplication by positions while shifting to the LEFT is multiplication by positions.*

Example 1: Below is an 8-bit (signed) fixed point number of which 3 right most are fractional (green).

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

This represents the number . To convert this into a decimal sum up the necessary terms:

Example 2: Below is an 8-bit (signed) fixed point number of which 5 right most are fractional (green).

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

This represents the number . To convert this into a decimal sum up the necessary terms:

Example 3: Below is a 10-bit (signed) fixed point number of which 9 right most are fractional (green).

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

This represents the number . To convert this into a decimal sum up the necessary terms:

Example 4: Below is a 10-bit (signed) fixed point number of which 7 right most are fractional (green).

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

This represents the number . To convert this into a decimal sum up the necessary terms:

Example 5:

**5.** Below is a 9-bit (unsigned) fixed point number of which 8 right most are fractional (green).

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

This represents the number . To convert this into a decimal sum up the necessary terms:

Example 6: Determine the *MINIMAL* number of bits required to represent -127.75 using fixed-point representation.

We need at least 8 bits to represent -127 plus an additional 2 bits to represent 0.75. In total, we need 10-bits.

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

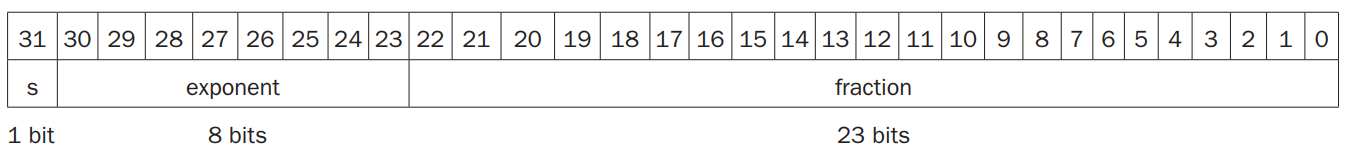
Take the result from the previous answer and shift the fixed point by 2 positions to the right and write the resulting signed decimal value.

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

Since we are shifting two positions to the right, we are multiplying the previous result by

**III. Floating-Point Representation**

Resource: Chapter 3 of Textbook, [Article](https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/), [Online Calculator](https://www.h-schmidt.net/FloatConverter/IEEE754.html)

The representation of a MIPS floating point number is shown below:

It requires a bit *s* (the sign of the number: 0 = positive, 1 = negative), the *exponent* (8-bit exponent field) which includes the sign of the exponent, and *fraction* (23-bits).

A floating-point number takes the decimal form:

Where *s* is the sign bit, *F* is the fraction value, and *E* is the exponent value. The *Bias* used for single precision (32-bit) floating point is 127.

Example 1: What decimal number is represented by the single precision (32-bit) float?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The sign bit is 1, the exponent field (bits 30 down to 23) contains the value 129 in decimal, and the fraction field contains . Hence, we use the following equation:

Where

Example 2: What decimal number is represented by the single precision (32-bit) float?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The sign bit is 0, the exponent field (bits 30 down to 23) contains the value 135, and the fraction field contains . Hence

Example 3: What decimal number is represented by the single precision (32-bit) float?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The sign bit is 0, the exponent field (bits 30 down to 23) contains the value 126, and the fraction field contains. Hence

Example 4: What decimal number is represented by the single precision (32-bit) float?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The sign bit is 0, the exponent field (bits 30 down to 23) contains the value 1, and the fraction field contains. Hence

Example 5: Give the 32-bit floating-point representation of -0.75

* Convert to binary by multiplying by 2 (ignore any sign)

|  |  |  |
| --- | --- | --- |
| Remainder 2 | Result | Integral Part |
|  | 1.5 | 1 |
|  | 1.0 | 1 |

* Normalize
* For binary representation, add 127 to the exponent
* Add the necessary zeros to the mantissa 0.1
* Put it all together (Note: Since the -0.75 is negative, the sign bit is 1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Example 6: Give the 32-bit floating-point representation of 85.125

* Convert integer part to binary:
* Convert fractional part to binary by multiplying by 2 (ignore any sign)

|  |  |  |
| --- | --- | --- |
| Remainder 2 | Result | Integral Part |
|  | 0.25 | 0 |
|  | 0.5 | 0 |
|  | 1.0 | 1 |

* Normalize
* For binary representation, add 127 to the exponent
* Add the necessary zeros to the mantissa
* Put it all together (Note: Since 80.125 is positive, the sign bit is 0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**IV. Infinity, NaN, & Overflow + Underflow**

Resources: [Article on Overflow](https://uweb.engr.arizona.edu/~ece369/Resources/overflow.pdf)

* *Infinity*

Consider the 32-Bit floating-point number below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The sign bit is 0, the exponent field (bits 30 down to 23) contains the value 255 in decimal, and the fraction field is . Hence, we use the following equation:

Where

The values of and are denoted with a bias exponent field of all ones and a mantissa field of all zeros where the sign bit distinguishes between and . In the case of , it is represented as or 0xFF800000. Thus, the range of single precision (32-Bit) floating-point numbers is .

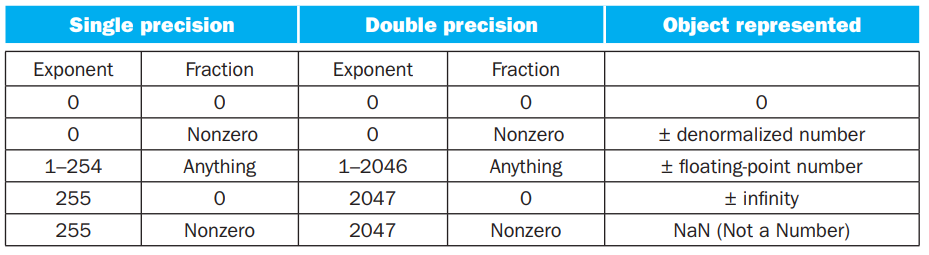
* *NaN*

A NaN (Not-a-Number) is a symbol that takes the form:

Where *s* is the sign bit, 111 1111 1 is the exponent field filled with all ones, and xxx xxxx xxxx xxxx xxxx xxxx is the mantissa of any value greater than 0. Note that and are both NaN. Here are some more examples:

There are two kinds of NaNs:

1. Signaling NaN: Throws (signals) invalid operation exception
2. Quiet NaN: Propogates through almost every arithmetic operation without signaling exception (i.e. you probably end up with weird or garbage values).



* *Overflow + Underflow*

Overflow occurs when the exponent is too large to be represented in the *Exponent (E)* field. Likewise, Underflow occurs when the exponent is too small to be represented in the *Exponent (E)* field.

**V. ASCII Representation**

*I could be wrong on this*

ASCII is an 8-bit code. That is, a character (including punctuation, symbols, and numbers) are each represented by 8-Bits of information.

Examples:

The decimal number 32 requires 8x2 = 16 bits of ASCII information

The decimal number -45.10 requires 8x6 = 48 bits of ASCII information