CSci370 Computer Architecture: Homework 3

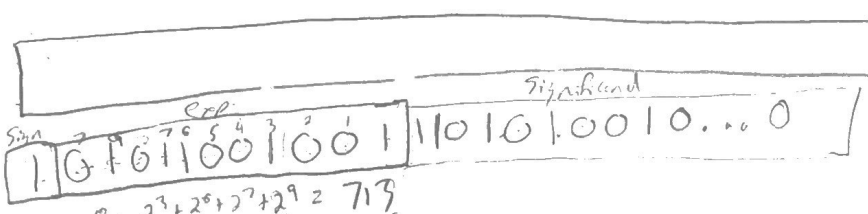
Due date: On or before Monday, April 06, 2020 in class
Absolutely no copying others' works

Name: Rich Trom

- The purpose of homeworks is for students to practice for the exams without others' help, so the penalty of mistakes will be minor.
- Without practicing for the exams properly, students would not be able to do well on the exams.

---

1. **(Floating-point representation: 20%)** What decimal number does the hexadecimal number (AC9D 4800 0000 0000)$_{16}$ represent if it is a floating point number? Use the IEEE 754 standard.



Exp $2^0 + 2^3 + 2^6 + 2^7 + 2^9 = 713$
(1 2 4 8 16 32 64 128 256 512)

Ebias = $713 - 1023 = -310$

Significand = $1.1101010010\ldots0_2$

$1 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-9} = -1.830078125 * 2^{-310}$

$= -8.773456149 \times 10^{-94}$

Ans? -3.619201775

A = 1.0011110610×2^0

B = 1.0100001100×2^1

C = -1.1010111110×2^2

Convert A, B, C to binary keep 10 bit number

2. **(Floating-point arithmetic: 80%)** IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Assuming the three numbers, A=1.237219, B=2.524723, and C=6.742837, are stored in the 16-bit IEEE 754-2008 format, calculate A×B-C by hand. Assume 1 guard bit, 1 round bit, and 1 sticky bit, and round to the nearest even (only on the Step 4. Rounding the Significand). Show all the steps in normalized floating point numbers with 11 bits of precision (or 10 bits of fraction); i.e., $(-1)^S \times (1.F)_2 \times 2^E$, where S is the sign bit, $|F|=10$, and E is without including the bias. Note that
   - have to show all five steps step-by-step,
   - the calculation should be similar to those in Slide 9.11 and Slide 9.16; i.e., the numbers should use a base-2 scientific notation and include 3 extra bits each (e.g., IEEE format and bias need NOT be used),
   - to answer this question, simulate the hardware by including 3 extra bits for each operand and result and using the 3 extra bits as possible as you could,
   - if the number has to be shifted to the right on the Step 1. Making Exponents Equal, round (just using the regular rounding method) the number to 10 bits of fraction and 3 extra bits before calculation,
   - rounding to the nearest even will be applied only once after finishing all calculation because the 3 extra bits need to be kept for further calculation, and
   - 2's complement representation has to be used if needed.

Work attached

-1.1100111110×2^1

$= -3.62109375_{10}$

Converting to Floating Binary

Q2 A 1.237219

0.237219 x 2 = 0.474438 = 0
0.474438 x 2 = 0.948876 = 0
0.948876 x 2 = 1.897752 = 1
0.897752 x 2 = 1.795504 = 1
0.795504 x 2 = 1.591008 = 1
0.591008 x 2 = 1.182016 = 1
0.182016 x 2 = 0.364032 = 0
0.364032 x 2 = 0.728064 = 0
= 1.456128 = 1
= 0.912256 = 0

A = 1.00111100102^0

B 2.524723

10.100000110010 = 1.01000001100 x 2^1

0.524723 x 2 = 1.049446 = 1
.049446 x 2 = 0.098892 = 0
.098892 x 2 = 0.197784 = 0
.197784 x 2 = 0.395568 = 0
.395568 x 2 = 0.791136 = 0
= 1.582272 = 1
= 1.164544 = 1
0.329088 = 0
0.658176 = 0
1.316352 = 1
0.632704 = 0

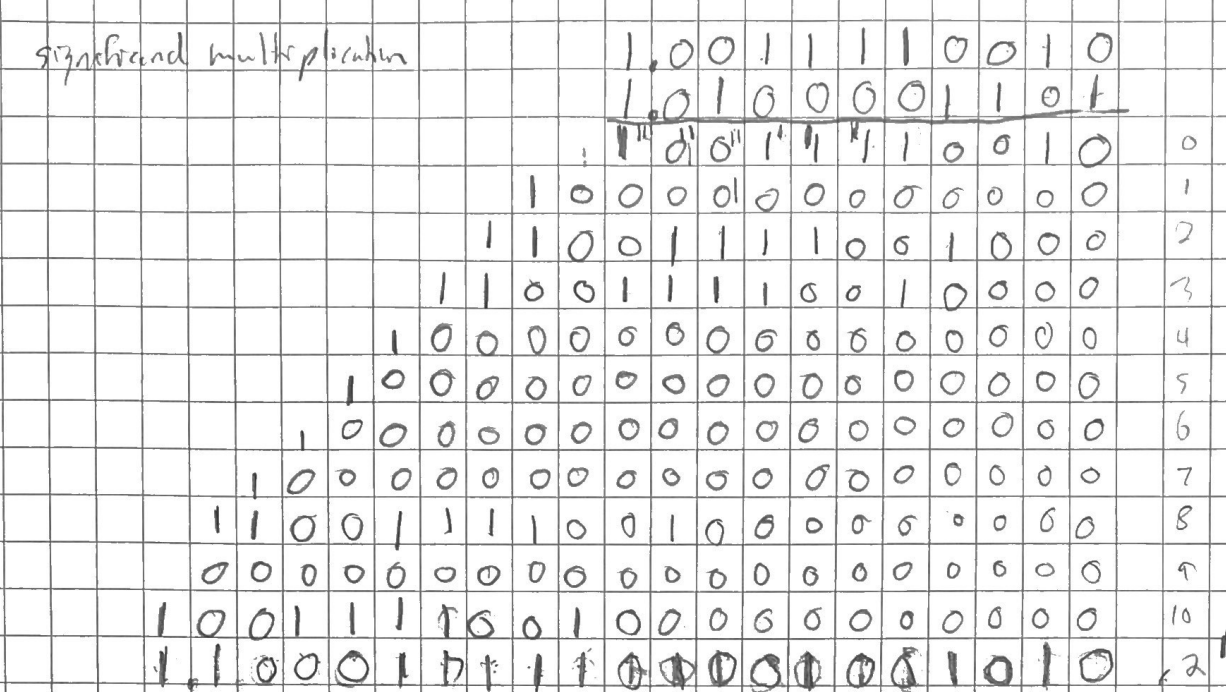
C = 6.742837

110.101111100011
1.1010111100011 x 2^2
GR 5=1

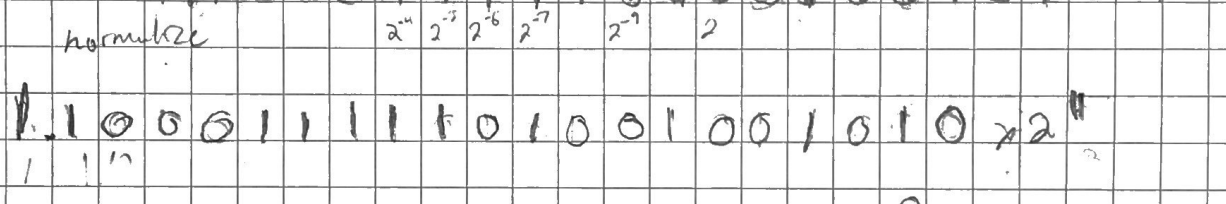
Step 1 calc exponent

0 + 1 = 1

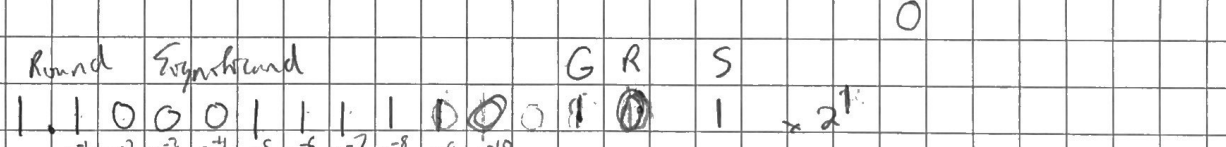
Step 2 significand multiplication



Step 3 normalize



Step 4 Round Significand

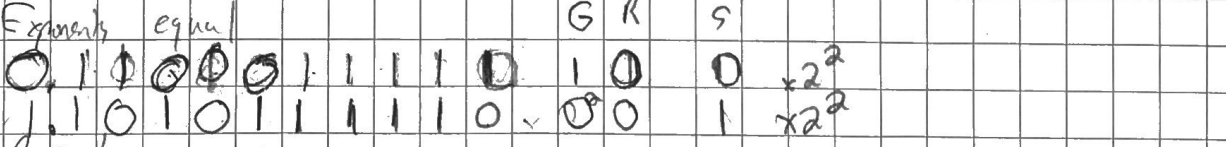


Step 5 Check overflow

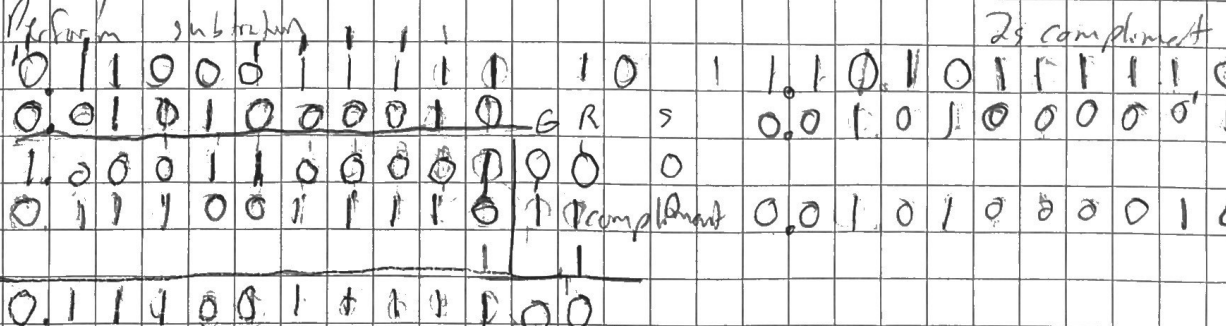
No overflow

Subtract C = 1.1010111110

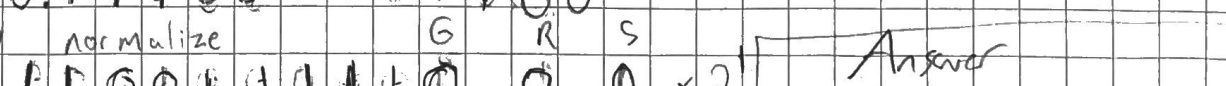
Step 1 Exponents equal



Step 2 Perform subtraction



Step 3 normalize

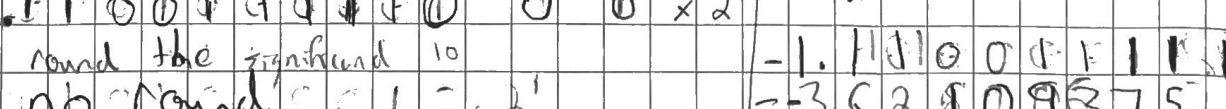


Step 4 round the significand

no round

Step 5 check overflow / underflow = None

Answer



$= -3.625_{10}$